Zbigniew Rudnicki

Wprowadzenie do informatyki i programowania

Wydanie drugie, poprawione

WYDAWNICTWA AGH

KRAKÓW 2015

KU 0629 pozycja dydaktyczna
Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie

ISBN 978-83-7464-802-8

Redaktor Naczelny Wydawnictw AGH: *Jan Sas*

Komitet Naukowy Wydawnictw AGH:
Zbigniew Kąkol (przewodniczący),
Marek Cała,
Borys Mikułowski,
Tadeusz Sawik,
Mariusz Ziółko

Recenzenci: *prof. dr hab. Dorota Kocańda*
prof. dr hab. inż. Ryszard Tadeusiewicz

Afiliacje Autora
AGH Akademia Górniczo-Hutnicza

Druk podręcznika wykonano z materiałów oraz ze składu dostarczonych przez Autora

Projekt okładki i strony tytułowej: *Zbigniew Rudnicki*

WYDANIE 2 POPRAWIONE

Redakcja Wydawnictw AGH
al. Mickiewicza 30, 30-059 Kraków
tel. 12 617 32 28, tel./faks 12 636 40 38
e-mail: redakcja@wydawnictwoagh.pl
http://www.wydawnictwa.agh.edu.pl

SPIS TREŚCI

1. WSTĘP 9

1.1. O podręczniku 9
1.2. Podstawowe pojęcia 11
1.3. Obliczenia, modelowanie, symulacja 13
1.4. Czy warto uczyć się programowania? 16
1.5. Nieco historii 17

2. PODSTAWY PROGRAMOWANIA STRON WWW 24

2.1. Światowa pajęczyna i strony WWW 24
2.2. Języki opisu dokumentów 25
2.3. Publikowanie w Internecie 27
2.4. Podstawy języka HTML 30
2.4.1. Struktura opisu strony. Głowa dokumentu 32
2.4.2. Ciało dokumentu. 33
2.4.3. Nagłówki 33
2.4.4. Format akapitów i czcionek 34
2.4.5. Listy wypunktowane i listy numerowane 35
2.4.6. Odsyłacze hipertekstowe do innych dokumentów 35
2.4.7. Grafika 36
2.4.8. Tabele 37
2.4.9. Ramki 38
2.5. Dynamiczne strony WWW 41
2.5.1. Skrypty i makra. Języki skryptowe 42
2.5.2. Charakterystyka języka PHP 43
2.5.3. Formularze HTML 45
2.5.4. Przykład obsługi formularza w języku PHP 48
2.6. CSS – Kaskadowe arkusze stylów 50
2.7. Ćwiczenia – tworzenie strony WWW 52
2.7.1. Cel ćwiczeń 52
2.7.2. Przebieg ćwiczeń 53

3. WPROWADZENIE DO ALGORYTMÓW I STRUKTUR DANYCH 57

3.1. Algorytm 57
3.2. Przykłady „Algorytmów z życia” 58
3.2.1. Przykład 1- zakupy 59
3.2.2. Przykład 2 – przepis na masę czekoladową 60
3.3. „Algorytmy z życia” - Zadania 61
3.4. Algorytmy i metody numeryczne 61
3.5. Stałe, zmienne, struktury danych. 63
3.6. Podstawowe typy poleceń w algorytmach obliczeniowych 67

3.6.1. Wczytywanie danych....68
3.6.2. Przypisanie zmiennej wartości wyrażenia....69
3.6.3. Wyprowadzanie wyników....69
3.6.4. Rozgałęzienie warunkowe – instrukcja „Jeżeli ...”....70
3.6.5. Pętla typu "Dla ..."....70
3.6.6. Pętle o nieokreślonej liczbie cykli....71
3.6.7. Skok bezwarunkowy....72
3.6.8. Podprogramy – procedury i funkcje....72
3.7. Iteracja i rekurencja....74
3.8. Złożoność obliczeniowa algorytmów....75

4. WPROWADZENIE DO PROGRAMOWANIA....77

4.1. Etapy....78
4.2. Programowanie to nie obliczenia....78
4.3. Języki programowania....80
4.4. Paradygmaty i metodyki programowania....83
4.5. Środowiska i interfejsy programistyczne....84
4.5.1. SDK – zestaw dla programisty....85
4.5.2. IDE - zintegrowane środowisko programistyczne....86
4.5.3. RAD - Rapid Application Development....86
4.5.4. Windows API....87
4.5.5. Wirtualna maszyna Javy JVM....88
4.5.6. Platforma .NET i Visual Studio....88
4.5.7. CASE i diagramy UML....89
4.5.8. Narzędzia programowania urządzeń mobilnych....89
4.5.9. Programowanie wizualne czyli program z klocków....92
4.6. Charakterystyka języka BASIC....94
4.6.1. Od historii do teraźniejszości BASIC-a....94
4.6.2. Programowanie w środowisku QB64....96
4.6.3. Zbiór znaków języka QBASIC....97
4.6.4. Zmienne i typy wartości. Wyświetlanie objaśnień....98
4.6.5. Wprowadzanie danych z klawiatury....98
4.6.6. Instrukcja przypisania....99
4.6.7. Wyprowadzanie wyników. Kolejność poleceń....99
4.6.8. Instrukcja warunkowa „*IF* ...”....102
4.6.9. Pętla „*FOR* ...”....103
4.6.10. Skok bezwarunkowy i pętla „*WHILE* ...”....104
4.6.11. Pętla „*DO* ... *LOOP UNTIL* ...”....105
4.6.12. Funkcje standardowe i funkcje użytkownika....108
4.6.13. Podprogramy proceduralne....110
4.6.14. Komunikacja z plikami tekstowymi....110
4.6.15. Grafika w QB64....112
4.6.16. Dźwięk i muzyka w QB64....115

5. PRZYKŁADY ALGORYTMÓW I PROGRAMÓW W JĘZYKU BASIC....116

5.1. Elementarne programy obliczeniowe....116
5.1.1. Jednorazowe obliczenia z kontrolą danych....117
5.1.2. Wielokrotne obliczenia skalarne....119
5.1.3. Obliczenia iteracyjne bez użycia tablic....120
5.2. Generowanie ciągów....122
5.3. Tabelaryzacja i wykres funkcji y(x)....123
5.4. Algorytm sumowania....125
5.5. Średnia, wariancja i odchylenie standardowe....126
5.6. Algorytm selekcji elementów ciągu....128
5.7. Obliczanie wartości wielomianu....129
5.8. Rozwinięcia funkcji w szereg Maclaurina....130
5.9. Całkowanie numeryczne....131
5.9.1. Metoda prostokątów....132
5.9.2. Metoda trapezów....132
5.9.3. Metoda Monte Carlo....133
5.10. Sortowanie zbiorów liczbowych....134
5.10.1. Sortowanie bąbelkowe....134
5.10.2. Sortowanie przez wybór....136
5.11. Rozwiązywanie równań nieliniowych....137
5.11.1. Metoda bisekcji....137
5.11.2. Metoda stycznych....138
5.11.3. Metoda siecznych....139
5.12. Operacje na macierzach....140
5.12.1. Generowanie macierzy....140
5.12.2. Mnożenie macierzy....141
5.12.3. Rozwiązywanie układów równań liniowych....144
5.12.4. Warstwice funkcji dwu zmiennych....145
5.13. Optymalizacja....147
5.13.1. Prosty przykład programowania liniowego....148
5.13.2. Ekstremum nieliniowej funkcji celu....150

6. ZADANIA DO ZAPROGRAMOWANIA....153

6.1. Proponowane warianty rozwiązań....153
6.2. Zadania z geometrii....154
6.3. Zadania z fizyki....156
6.4. Algorytmy z warunkami i wyborem....157
6.5. Ciągi i iteracje....157

7. WPROWADZENIE DO PROGRAMOWANIA W VISUAL BASIC....160

7.1. Tworzenie programów dla MS Windows w języku Ms Visual BASIC....160
7.2. Twój pierwszy program w Visual BASIC-u....161
7.2.1. Koncepcja programu....161
7.2.2. Uruchomienie i obsługa Visual BASIC-a....162
7.2.3. Ustawianie cech obiektów....164
7.2.4. Procedury zdarzeniowe....165

7.2.5. Zapisanie oraz uruchomienie i testowanie działania aplikacji....166
7.3. Zadania....167

8. PODSTAWY PROGRAMOWANIA W MATLAB-IE....168

8.1. Wprowadzenie....168
8.1.1. FreeMat i inne darmowe odpowiedniki MATLAB-a....169
8.1.2. Łagodny start. Okno komend MATLAB-a....171
8.1.3. Tryby użytkowania i style programowania....174
8.2. Podstawowe elementy języka MATLAB....176
8.2.1. Słowa kluczowe....176
8.2.2. Reguły tworzenia nazw....176
8.2.3. Typy wartości....177
8.2.4. Wyrażenia....179
8.3. Wartości liczbowe, macierze, wyrażenia arytmetyczne....179
8.3.1. Typy wartości liczbowych....180
8.3.2. Zapis liczb....181
8.3.3. Zmienne. Identyfikacja i konwersja typów....181
8.3.4. Formaty wyświetlanych liczb....183
8.3.5. Macierze....183
8.3.6. Operatory działań arytmetycznych na skalarach....184
8.3.7. Przykłady wyrażeń arytmetycznych....185
8.3.8. Ważniejsze funkcje elementarne MATLAB-a....185
8.3.9. Ćwiczenia – obliczenia w trybie bezpośrednim....186
8.4. Wartości i wyrażenia logiczne....187
8.5. Wartości i wyrażenia tekstowe....188
8.6. Pisanie i uruchamianie programów....190
8.7. Podstawowe instrukcje....192
8.7.1. Instrukcja przypisania....192
8.7.2. Wprowadzanie danych z klawiatury....193
8.7.3. Wyświetlanie wyników – *DISP(), FPRINTF()*....194
8.7.4. Instrukcja *IF*....195
8.7.5. Instrukcja wyboru *SWITCH*....197
8.7.6. Pętla *FOR*. Wykres typu XY....198
8.7.7. Pętla *WHILE*....199
8.8. Operacje na plikach....200
8.8.1. Zapisywanie i odczytywanie zmiennych z przestrzeni roboczej....200
8.8.2. Otwieranie i zamykanie plików....201
8.8.3. Wyprowadzanie wyników do pliku....202
8.8.4. Wczytywanie danych z pliku....203
8.9. Wprowadzenie do wykresów typu XY....205
8.10. Definiowanie i wywoływanie funkcji użytkownika....208
8.10.1. Przykłady....209
8.10.2. Funkcje o zmiennej liczbie argumentów....211
8.10.3. Definiowanie funkcji INLINE....212
8.10.4. Zadania – definiowanie i wywoływanie funkcji....213

9. MATLAB – OPEROWANIE NA TABLICACH 214

9.1. Deklarowanie tablic 215
9.2. Sposoby wprowadzania wektorów i macierzy 217
9.2.1. Wektor generowany jako postęp arytmetyczny 217
9.2.2. Wprowadzanie z klawiatury 218
9.2.3. Rola dwukropka w wybieraniu elementów macierzy 218
9.2.4. Generowanie macierzy 219
9.2.5. Wczytywanie macierzy z pliku 220
9.2.6. Podstawowe operacje na macierzach 220
9.2.7. Układ równań liniowych. Odwracanie oraz dzielenie macierzy 222
9.2.8. Ćwiczenia 223
9.3. Wykresy funkcji dwu zmiennych 224
9.4. Tablice komórek 226
9.5. Tablice struktur 228

10. MATLAB - OBIEKTY I PROGRAMOWANIE W TRYBIE GRAFICZNYM 230

10.1. Okno graficzne - *figure* 232
10.2. Wybrane obiekty i funkcje graficzne 233
10.3. Elementy sterujące – *uicontrol's* 235
10.4. Przykład programu z interfejsem graficznym 236
10.5. Ćwiczenia 238
10.6. Samodzielne okienka dialogowe 240
10.6.1. Komunikat - *msgbox* 241
10.6.2. Wprowadzanie danych - *inputdlg* 242
10.6.3. Zapytanie – *questdlg* 242
10.6.4. Funkcja *menu* 243
10.6.5. Dialogowy wybór folderów i plików - *uigetfile* 243
10.7. GUIDE – środowisko typu RAD 244

11. ELEMENTY JĘZYKA C I PORÓWNANIE Z MATLABEM 248

11.1. Język C i jego następcy 248
11.2. Struktura programu w języku C 249
11.3. Dołączanie plików nagłówkowych 251
11.4. Stałe, zmienne, struktury danych 251
11.4.1. Stałe 251
11.4.2. Nazwy 251
11.4.3. Typy wartości i deklaracje zmiennych 251
11.4.4. Wskaźniki 253
11.4.5. Zmienne lokalne i globalne 253
11.4.6. Tablice numeryczne i ich deklarowanie 254
11.5. Operatory 255
11.5.1. Podstawowe operatory arytmetyczne 255
11.5.2. Inne operatory 256
11.6. Funkcje wejścia i wyjścia 257

11.7. Wyrażenia logiczne oraz instrukcja *IF* 258
11.8. Pętla *FOR* 258
11.8.1. Pętla *WHILE* oraz pętla *DO* 259
11.8.2. Instrukcje zmieniające działanie pętli 260
11.9. Funkcje użytkownika 260

12. BAZY DANYCH 262

12.1. Modele i ewolucja struktur baz danych 262
12.2. Architektura i ewolucja SZBD 263
12.3. Relacyjne bazy danych 265
12.4. Proste bazy w arkuszu kalkulacyjnym 266
12.5. Wymagane cechy baz danych i SZBD 267
12.6. Podstawowe operacje 268
12.6.1. Projektowanie i budowa bazy 268
12.6.2. Wprowadzanie i edycja danych 269
12.6.3. Przeglądanie, wyszukiwanie, drukowanie 270
12.7. Języki programowania SZBD 270
12.7.1. Charakterystyka języka SQL 271
12.7.2. dBase, Clipper, CA VO, Harbour 273
12.8. Bazy danych w programie Ms Access 275
12.8.1. Projektowanie i normalizacja tabel 276
12.8.2. Definiowanie tabel 278
12.8.3. Wprowadzanie danych do tabel słownikowych 280
12.8.4. Ustalanie relacji między tabelami 280
12.8.5. Tworzenie relacji „Kreatorem odnośników” 281
12.8.6. Tworzenie relacji w oknie „Relacje” 283
12.8.7. Kwerendy czyli zapytania 285
12.8.8. Kwerendy wybierające 285
12.8.9. Kwerendy obliczeniowe 286
12.8.10. Kwerendy funkcjonalne 288
12.8.11. Tworzenie formularza 288
12.8.12. Generowanie raportów 289
12.9. Przykład małej lokalnej bazy danych 290

13. ZARYS METOD SZTUCZNEJ INTELIGENCJI 298

13.1. Systemy ekspertowe 300
13.2. Sztuczne sieci neuronowe 301

LITERATURA 305

1. WSTĘP

1.1. O PODRĘCZNIKU

Podręcznik ten napisałem głównie dla studentów Studiów Niestacjonarnych AGH, dla których od wielu lat prowadzę wykłady i zajęcia laboratoryjne z „Informatyki". Mam jednak nadzieję, że będzie on przydatny dla tych wszystkich, którzy chcą bliżej poznać idee działania komputerów, podstaw programowania, budowania stron internetowych oraz tworzenia i wykorzystywania baz danych.

Podręcznik niniejszy jest kontynuacją dwutomowego podręcznika „Techniki informatyczne" [1], [2] – przeznaczonego dla pierwszego semestru studiów, zawierającego podstawowe informacje o komputerach i innych urządzeniach cyfrowych oraz uczącego korzystania z gotowych narzędzi jak: edytor tekstu, edytor grafiki, arkusz kalkulacyjny a także Mathcad i Autocad.

Przedmiot „Informatyka", w semestrze drugim, wprowadza kolejny „stopień wtajemniczenia". Oprócz wykorzystywania gotowego oprogramowania, studenci mają poznać narzędzia i sposoby tworzenia programów, a w szczególności nauczyć się konstruowania algorytmów i posługiwania wybranymi językami programowania. Uzyskają dzięki temu swobodę realizowania różnorodnych własnych pomysłów i przekształcania komputera w posłuszne narzędzie. Z jednej strony związane to będzie z trudnościami większymi niż w pierwszym semestrze, ale z drugiej - pozwoli rozwijać kreatywność i dostarczać wiele radości i dumy z realizacji własnych dzieł.

Podręcznik rozpoczyna się od wprowadzenia (lub przypomnienia) podstawowych pojęć związanych z komputerami oraz zarysowania historii komputerów na Świecie, w Polsce, a także na Wydziale Inżynierii Mechanicznej i Robotyki AGH w Krakowie.

Dalej, pod względem tematyki, można wyodrębnić pięć części.

Pierwszą z nich jest rozdział 2, dostarczający podstawowych informacji o językach opisu dokumentów, a następnie skupia się na zagadnieniu tworzenia stron internetowych w języku HTML a także opisuje inne narzędzia, jak systemy CMS, oraz języki PHP i CSS.

Część druga to rozdziały 3, 4, 5, 6. Rozdział 3 zawiera podstawowe informacje o strukturach danych i algorytmach. W szczególności - objaśnia cechy zmiennych oraz algorytmów i podstawowych poleceń z których są one konstruowane.

W rozdziale 4 scharakteryzowano różne narzędzia i metody programistyczne, skupiając się ostatecznie na dokładniejszym przedstawieniu najprostszego z tych narzędzi, jakim jest język BASIC. Najłatwiej bowiem i najprzyjemniej – zdaniem autora - można nauczyć się elementarnego programowania właśnie w języku BASIC, specjalnie opracowanym z myślą o początkujących. Rozwinięcia języka BASIC zyskują coraz większą popularność, między innymi jako narzędzia tworzenia aplikacji na smartfony, tablety czy współczesne internetowe telewizory (Smart-TV). Dlatego w podręczniku zamieszczono skrócony opis współczesnej wersji języka BASIC, o nazwie QB64, oraz szereg przykładów programów w tym języku, obrazujących zarówno środki języka jak i wybrane metody obliczeniowe.

Rozdział 5 to zbiór wielu przykładowych algorytmów, potencjalnie przydatnych dla studenta studiów technicznych, a także stanowiących materiał do tworzenia programów. Niektóre z tych algorytmów przedstawiono w postaci programów w języku BASIC. Mogą one ułatwić naukę tego języka, ale także stanowić zadania do programowania w innych językach. Tematy zadań do samodzielnego wykonania są w rozdziale 6. W całym podręczniku zamieszczono ok.100 przykładów programów i algorytmów oraz podobną liczbę zadań do samodzielnego wykonania.

Część trzecia, najobszerniejsza, złożona z rozdziałów 7-11 to podstawy programowania w wybranych językach: Visual BASIC, MATLAB oraz C.

Visual BASIC (składnik Ms Visual Studio) – to nowoczesna kontynuacja BASIC-a, umożliwiająca programowanie obiektowo-zdarzeniowe z wykorzystaniem dialogowych elementów graficznych, co pokazano na bardzo prostym przykładzie w rozdziale 7.

Najwięcej miejsca, bo aż trzy rozdziały (8,9,10) poświęcono językowi i pakietowi oprogramowania matematycznego MATLAB, który dla początkujących jest stosunkowo łatwy, a dla zaawansowanych stanowi potężne narzędzie wyposażone w wiele specjalistycznych pakietów oprogramowania przeznaczonych dla różnorodnych gałęzi nauki i techniki. I tak: rozdział 8 przedstawia elementy języka MATLAB w zakresie potrzebnym do tworzenia elementarnych programów w sposób strukturalny, analogiczny jak w języku BASIC. Rozdział 9 dotyczy głównie macierzy i operacji macierzowych, charakterystycznych dla MATLAB-a. Rozdział 10 rozwija informacje o obiektach i funkcjach graficznych w MATLAB-ie oraz tworzeniu programów dialogowych z interfejsem graficznym.

MATLAB jest kosztownym oprogramowaniem komercyjnym, dlatego podano także informacje o jego darmowych alternatywach, jak: FreeMat, Scilab czy Octave.

Aby studenci mieli nieco szerszy pogląd na języki programowania, rozdział 11 zawiera wprowadzenie do języka C i porównanie tego języka z MATLABem. Język C jest jednak trudniejszy i mniej przyjazny dla początkujących, dlatego wiadomości o nim potraktowano jako pewien dodatek i ograniczono do minimum.

Część czwarta - czyli rozdział 12 - to wprowadzenie do baz danych oraz zasad projektowania i programowania systemów zarządzania bazami danych (SZBD) - bardzo ważnej i rozpowszechnionej dziedziny zastosowań komputerów. Omówiono m.in. ewolucję SZBD, najważniejsze wymagania jakie powinny one spełniać, scharakteryzowano modele struktury baz danych a w szczególności model relacyjny. Przypomniano narzędzia tworzenia i obsługi tabel baz danych w arkuszu kalkulacyjnym, a następnie przedstawiono podstawowe polecenia języka SQL oraz scharakteryzowano języki dBase i Clipper. Obszerniej omówiono sposób definiowania i obsługiwania najprostszej relacyjnej bazy danych w systemie Ms Access. Przedstawiono też przykładowy SZBD dla małej bazy danych.

Ostatna część - rozdział 13 – to krótki opis wybranych metod sztucznej inteligencji.

Za względu na szczególny charakter studiów niestacjonarnych - starałem się przedstawić zagadnienia w sposób maksymalnie zrozumiały, choć często może zbyt uproszczony. Mam jednak nadzieję, że przynajmniej część czytelników zafascynuje programowanie komputerów - dostarczające wiele radości i satysfakcji, proporcjonalnie do włożonej pracy.

Przekazując ten podręcznik czytelnikowi, serdecznie dziękuję wszystkim, którzy wspierali jego powstanie.

Autor

1.2. PODSTAWOWE POJĘCIA

Zanim zajmiemy się zagadnieniami programowania komputerów, warto przede wszystkim przypomnieć niektóre podstawowe pojęcia, wprowadzone w poprzednim semestrze i zamieszczone w podręczniku [1].

Komputer to uniwersalny automat do pobierania, gromadzenia, przetwarzania i prezentowania **danych**, działający na podstawie opracowanych wcześniej **programów**.

Komputer gromadzi, porządkuje i przetwarza różne **typy danych**, nie tylko **liczbowe** ale także **tekstowe**, **logiczne**, **graficzne**, dźwiękowe i inne.

Ponieważ wszelkie typy informacji są w komputerze reprezentowane ciągami zer i jedynek czyli **bitów**, więc **urządzenia wejściowe** – pobierające dane do komputera - muszą dokonywać przetwarzania tekstów, liczb, obrazów, dźwięków, filmów i innych typów informacji na **postać binarną** przy zastosowaniu różnorodnych sposobów **kodowania. Urządzenia wyjściowe** spełniają funkcje przeciwne – wyprowadzają binarne postacie informacji przekształcając je (dekodując) i prezentując w czytelnych dla człowieka postaciach: wydruków, obrazów, dźwięków i t.d.

Wszelkie **działania komputera wynikają z wykonywania poleceń zawartych w programach**, **ale także mogą zależeć od wprowadzanych danych**. Na przykład, po wczytaniu współczynników równania kwadratowego komputer może sprawdzić czy dla tych współczynników wyróżnik DELTA będzie dodatni czy ujemny i w zależności od tego wyznaczy lub nie wyznaczy rozwiązania. Szczególnym rodzajem danych mogą być **zdarzenia** – na przykład takie jak kliknięcia myszką, oraz **stany** urządzeń lub badanych obiektów – na przykład brak papieru, nie włączenie drukarki lub brak miejsca na dysku.

Ponieważ program może modyfikować dane, więc także może modyfikować swoje działanie. Na tej zasadzie działają programy zaliczane do dziedziny **sztucznej inteligencji**, na przykład symulujące funkcjonowanie sieci neuronowych lub ewolucję genetyczną.

Zanim powstanie program, opracowywana jest jego **koncepcja** w postaci algorytmu. W uproszczeniu można powiedzieć, że **algorytm** to precyzyjny opis działań jakie ma realizować komputer, **program** zaś - to algorytm zapisany w jednym z formalnych **języków programowania** komputerów "zrozumiałym" dla danego komputera. Program zawiera na ogół zestaw **deklaracji** opisujących obiekty (zmienne), na których mają być wykonywane operacje, oraz ciąg **poleceń** zwanych też rozkazami, komendami (ang.: *command*) lub instrukcjami (ang. *statement*), zapisanych w odpowiedniej kolejności.

Niektóre języki programowania (np. BASIC, MATLAB), nie wymagają deklarowania zmiennych (przynajmniej w prostych programach).

Aby inżynier mógł ocenić czy dany problem może być rozwiązany na komputerze, a także aby mógł porozumieć się z informatykiem – musi znać **istotę budowania algorytmów** oraz **podstawy tworzenia programów**, przynajmniej w jednym z wielu języków programowania.

Ogólną charakterystykę języków programowania zawarto w p.4.1. niniejszego podręcznika, a oprócz tego scharakteryzowano bardziej lub mniej szczegółowo wiele języków, w tym języki opisu dokumentów: HTML (p.2.4.), XML (p.2.2.), CSS (p.2.6.), oraz języki programowania: PHP (p.2.5.2.), BASIC (p.4.6), Visual BASIC (rozdz.7), MATLAB (rozdz.8-10), SQL (p.12.7.1.), dBase i Clipper (p.12.7.2). Wszystkie wymienione języki (i wiele innych) są **językami wysokiego poziomu**, dogodnymi dla człowieka lecz wymagającymi translacji na kod procesora.

Jak już powiedziano - programy i dane różnego typu (liczbowe, tekstowe, dźwiękowe, wizualne i in.) wyrażane są w postaci binarnej (ciągów zer i jedynek) i gromadzone w **pamięciach masowych** komputera, takich jak: dyski magnetyczne, optyczne, pamięci „Flash" i inne. Wykonywanie programów i przetwarzanie danych odbywa się w szybkiej lecz nietrwałej **pamięci operacyjnej RAM** (*Random Access Memory*) a więc do niej musi być pobrany, przed uruchomieniem, program i rozmieszczone potrzebne dla niego dane.

W wykonywaniu programów główną rolę odgrywa procesor komputera. **Procesor** to najważniejszy układ scalony komputera zawierający obecnie setki milionów tranzystorów. Jego dwie zasadnicze części to **arytmometr** (*ALU - arithmetic-logic unit*) czyli jednostka arytmetyczno-logiczna zdolna do wykonywania operacji arytmetycznych i logicznych oraz **układ sterowania** (*control unit*) pobierający z pamięci operacyjnej RAM kolejne rozkazy wykonywanego programu i sterujący zgodnie z ich treścią urządzeniami komputera. Procesor posiada także szereg **rejestrów**, na zawartości których może wykonywać operacje, oraz **pamięć podręczną** zwaną ***cache***.

Każdy typ procesora ma określoną **listę rozkazów**, które jest w stanie wykonywać i ma specyficzny kod wyrażania tych rozkazów przy pomocy zer i jedynek (czyli w postaci binarnej).

Procesor komputera nie może więc realizować bezpośrednio komend programów napisanych w językach wysokiego poziomu. Wymagają one **translacji** czyli przetłumaczenia przy pomocy odpowiedniego programu zwanego **translatorem.** Translatory dzielą się na **kompilatory** - tłumaczące całość programu przed jego wykonaniem (np.: Pascal, Fortran, C) oraz na **interpretatory** (interpretery) tłumaczące i wykonujące każdą instrukcję zaraz po jej wpisaniu (np.: BASIC, dBase, MATLAB).

Po przetłumaczeniu programu przez kompilator następuje zazwyczaj jeszcze tzw. **konsolidacja** (zwana też z angielska "linkowaniem"). Jest to operacja dołączania podprogramów standardowych (tzw. bibliotecznych) i porządkowania całości, realizowana przez **konsolidator** (*linker*). W systemie Windows program zazwyczaj wykorzystuje jeszcze tak zwane **biblioteki dołączane dynamicznie** (po uruchomieniu) i zawarte **w plikach *.DLL**.W wyniku tych operacji powstaje **binarna postać wykonywalna** programu (*executable*), w kodzie procesora, zapisywana w pliku z rozszerzeniem "**.exe**" lub "**.com**".

Programy w postaci wykonywalnej nazywane są w systemie Ms Windows **aplikacjami**.

Gdy uruchamiamy taką aplikację (np.: otwieramy jej ikonę) wówczas jest ona ładowana do szybkiej (lecz nietrwałej) **pamięci operacyjnej RAM** i tam następuje wykonywanie poszczególnych rozkazów przez procesor oraz sterowane przez niego układy i urządzenia.

Jeśli dla konkretnego typu komputera (a w szczególności jego procesora) opracowano translator pewnego języka programowania to mówi się, że **zaimplementowano język** na ten komputer albo że istnieje **implementacja** tego języka dla tego typu komputera.

Jak już wspomniano, przedmiotem działań programów są **dane**. Dane mogą występować w postaci ***stałych*** liczbowych, tekstowych i innych oraz ***zmiennych*** – widocznych poprzez swoje **nazwy**

Stałe nazywane są czasem „literałami" gdyż w bezpośredni (literalny) sposób prezentują swoją wartość liczbową, tekstową czy innego typu (np.: logiczną) nie skrywając jej pod żadnymi symbolami.

Każda **zmienna** jest natomiast „symboliczną reprezentacją określonej cechy modelowanego obiektu lub procesu", inaczej mówiąc - jest widoczna w postaci **nazwy** (lub identyfikatora) reprezentującej obszar pamięci komputera, przechowujący **wartość** tej zmiennej.

Złożone dane są porządkowane w różnorodne **struktury** (wektory, macierze, rekordy, listy i in.) reprezentowane przez **zmienne złożone**.

Większe zbiory danych – przechowywane są w pamięciach masowych najczęściej jako relacyjne **bazy danych** – w postaci powiązanych wzajemnie tabel.

Niniejszy podręcznik ma głównie przybliżyć zagadnienia związane z **istotą danych** i **programów oraz sposobów ich opisywania** występujących przy rozwiązywaniu konkretnych zagadnień z pomocą komputera.

Zagadnienia **opisu danych** będą przewijać się przez cały podręcznik. Wystąpią zarówno na początku - w rozdziale dotyczącym opisywania postaci dokumentów internetowych czyli stron WWW**,** dalej – w rozdziale o podstawowych strukturach danych, w rozdziałach dotyczących języków programowania oraz w rozdziałach końcowych - dotyczących relacyjnych baz danych.

Zagadnienie **opisywania działań** jakie komputer ma wykonywać wystąpią zarówno w rozdziale o algorytmach jak i w rozdziałach dotyczących języków programowania.

1.3. OBLICZENIA, MODELOWANIE, SYMULACJA

W pracy inżyniera, oprócz standardowych obliczeń, często zachodzi potrzeba badania układów niedostępnych lub jeszcze nie istniejących (projektowanych) albo badania układów istniejących lecz w zakresie niemożliwym lub nieopłacalnym do zrealizowania na rzeczywistym obiekcie. W tym celu stosuje się **badania symulacyjne na** różnego rodzaju **modelach** - najczęściej realizowanych przy pomocy programów komputerowych.

Każdy model a więc i komputerowy jest hipotetycznym, przybliżonym i uproszczonym odwzorowaniem wybranych – interesujących dla badacza - cech rzeczywistego obiektu lub procesu. Można wyróżnić dwie podstawowe kategorie modeli:

- **modele strukturalne** – w tym tak lubiane modele **geometryczne** bryłowe, ale także dwuwymiarowe czyli rysunki - odzwierciedlające budowę obiektu, oraz
- **modele funkcjonalne** – najczęściej **matematyczne** - odzwierciedlające funkcjonowanie badanego układu lub określony proces (i raczej przez studentów nie lubiane).

Najdoskonalsze modele - zwane **wirtualnymi prototypami** - łączą cechy modeli geometrycznych i matematycznych, odzwierciedlając nie tylko wygląd i zdolności ruchu (kinematykę), ale i dynamikę (masy, przyspieszenia, bezwładność) oraz wzajemne oddziaływania (zderzenia, odbicia, odkształcenia). Rozbudowane, profesjonalne programy specjalistyczne, wspomagające tworzenie modeli bryłowych czy wirtualnych prototypów [6], są zazwyczaj kosztowne ale często istnieją ich wersje treningowe i edukacyjne.

Modelowanie matematyczne polega m.in. na:

- przypisaniu **zmiennych** – identyfikowanych przez **nazwy** (lub inne symbole) – istotnym cechom badanego układu,
- wyodrębnieniu **zmiennych wejściowych**, których wartości będą stanowiły dane do obliczeń, oraz **zmiennych wynikowych** (wyjściowych), których wartości i zachowanie chcemy określić, a czasem także zmiennych pośrednich między wejściowymi i wynikowymi,
- określeniu **ograniczeń** co do zakresu wartości zmiennych (w postaci nierówności),
- określeniu **zależności funkcyjnych** między zmiennymi wejściowymi i wynikowymi,
- określeniu **kolejności wyznaczania wartości** poszczególnych zmiennych – inaczej mówiąc kolejności operacji składających się na **algorytm** obliczeń.

Jak z tego wynika - obliczenia inżynierskie w których nie wprowadzono zmiennych nie są jeszcze modelowaniem matematycznym.

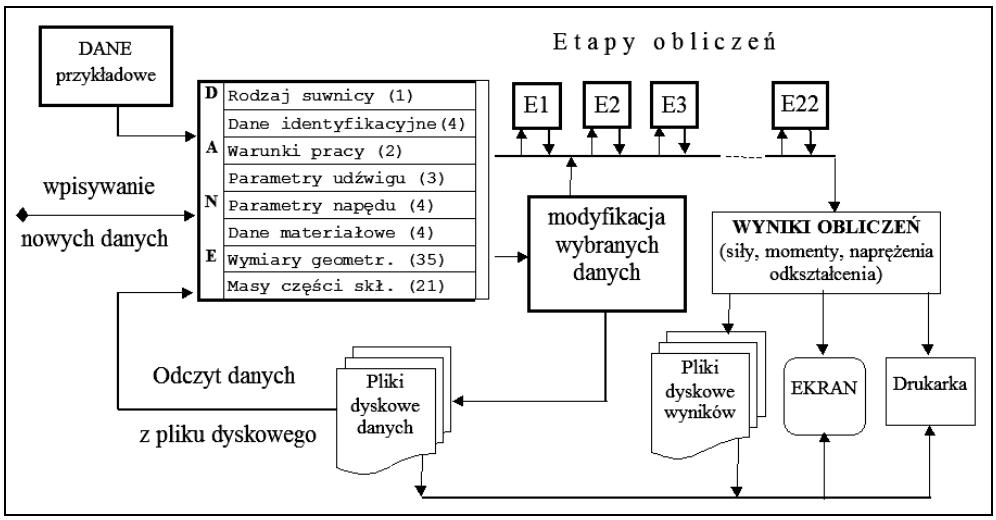
Model matematyczny ma służyć do badania wzajemnych zależności między zmiennymi. Może służyć do określania zagrożeń a także do optymalizacji względem danych kryteriów (np.: minimum zużycia materiału lub energii), przy spełnieniu innych niezbędnych wymagań i ograniczeń.

Programy obliczeniowe stanowiące realizację modeli matematycznych dają szansę znacznego zwiększenia wygody i niezawodności obliczeń inżynierskich umożliwiając wprowadzenie takich środków jak:

- wielostopniowy system pomocy – dotyczący zmiennych oraz zastosowanych metod i obsługi programu,
- kontrola typu i zakresu wprowadzanych danych i uzyskiwanych wyników,
- zastosowanie danych przykładowych umożliwiających łatwe przetestowanie działania programu,
- podzielenie długich obliczeń na etapy z możliwością powrotu do poprzednich etapów i korygowania danych,
- zastępowanie wpisywania danych przez wybór z list,
- łączenie obliczeń z wykorzystaniem baz danych,
- czytelne i efektowne sposoby prezentacji danych i wyników (wykresy, grafika).
- możliwość wyboru wyjścia obliczeń (ekran, drukarka, pamięć masowa) oraz uzyskiwania protokołu z obliczeń.

Większość z tych środków autor stosował w tworzonych przez siebie programach obliczeniowych [7]. Przykładem może być (archiwalny już) program BGR-OS (Rys. 1.1 i Rys. 1.2) opracowany we współpracy z specjalistami w zakresie wytrzymałości materiałów.

Rys. 1.1. Plansza tytułowa programu BGR-OS



Rys. 1.2. Schemat przepływu danych w programie BGR-OS

Znaczna część modeli matematycznych musi być samodzielnie opracowywana a następnie wdrażana do praktyki przez inżynierów.

Mają oni do dyspozycji dwie drogi:
- wykorzystanie uniwersalnych programów matematycznych jak Excel czy Mathcad,
- napisanie własnych programów przy wykorzystaniu jednego z istniejących języków programowania.

Pierwszej drogi dotyczył poprzedni podręcznik „Techniki Informatyczne" [1], [2], natomiast niniejszy podręcznik skupia się na tematyce drogi drugiej.

1.4. CZY WARTO UCZYĆ SIĘ PROGRAMOWANIA?

Wzrost liczby komputerów oraz ich unifikacja i standaryzacja skutkują szybkim rozwojem zastosowań komputerów i zwiększaniem liczby dostępnych programów a Internet sprzyja ich rozpowszechnianiu, także legalnie darmowemu (*freeware* lub *public domain*) lub udostępnianiu do przetestowania. Tak więc najczęściej zamiast programować samodzielnie, można wyszukać odpowiedni program w Internecie lub rozwiązać problem przy zastosowaniu jednego z wielu uniwersalnych narzędzi (na przykład w arkuszu kalkulacyjnym) - koniecznie trzeba więc być biegłym w posługiwaniu się oprogramowaniem uniwersalnym i inżynierskim, a rzadziej zachodzi potrzeba samodzielnego pisania programów.

Pisanie profesjonalnych programów działających w środowisku poszczególnych wersji systemu Ms Windows, nie jest łatwe, a odpowiednie jest raczej dla wykwalifikowanych informatyków niż dla inżynierów mechaników.

Można więc postawić pytanie: "po co inżynier mechanik ma uczyć się programowania, skoro dysponuje obecnie wielką różnorodnością gotowych programów zarówno uniwersalnych jak i wyspecjalizowanych"

Zdaniem autora, jednak warto poznać podstawy programowania, bo pozwoli nam to m.in.:

- poznać ideę działania komputera, jego możliwości i ograniczenia,
- prawidłowo formułować problemy do rozwiązywania na komputerach,
- współpracować z informatykami przy wspólnym rozwiązywaniu problemów,
- rozwiązywać nietypowe zagadnienia i budować proste w obsłudze i niezawodne narzędzia,
- rozwijać wyobraźnię i pomysłowość oraz uczyć się znajdowania błędów we własnym rozumowaniu,
- pełniej wykorzystywać uniwersalne pakiety oprogramowania, które prawie zawsze wyposażane są w języki programowania, pozwalające dostosowywać je do wyspecjalizowanych zadań.

Przykładowo: pakiety biurowe - jak Ms Office - posiadają język Visual BASIC lub jego odmianę VBA (*Visual BASIC for Applications*). Autocad dysponuje językami Lisp lub Visual Lisp oraz Visual BASIC. W MATLAB-ie język programowania jest jego istotą. Nowe wersje Mathcada zostały także wyposażone w język programowania. Języki programowania (dBase, Clipper, SQL, PHP) są używane w systemach baz danych.

Aktywne strony internetowe wykorzystują także języki programowania (np. Java Script, czy PHP). Tak więc przy wszelkich nietypowych czy bardziej zaawansowanych zagadnieniach możemy spotkać się z językami programowania i potrzebą ich użycia.

Przykładowo - umiejętność napisania programu pozwalającego gromadzić i przetwarzać w komputerze dane uzyskiwane z drogiej aparatury pomiarowej - może nam pozwolić zaoszczędzić wydatków na drogie firmowe oprogramowanie.

Jak więc widać, umiejętność programowania jest przydatna także dla inżyniera mechanika, zarówno dla poszerzenia ogólnej wiedzy jak i ze względów praktycznych.

Oprócz powyższych argumentów, warto wiedzieć, że tworzenie programów jest wspaniałym rodzajem niemal nieograniczonej twórczości pozwalającej realizować różnorodne pomysły, uczy logicznego myślenia i stanowi umysłową rozrywkę nie gorszą niż szachy a przynoszącą konkretne efekty i - po początkowych trudach - mnóstwo satysfakcji.

Przedstawiane w niniejszym podręczniku treści nie mają na celu wyszkolenia programistów lecz:

- **nauczenie elementarnych podstaw programowania** – niezależnych od języka;
- **poznanie** najważniejszych cech i podstawowych narzędzi **kilku języków**, w tym języka opisu stron WWW (**HTML**), języka do tworzenia dynamicznych stron WWW (**PHP**), języka do nauki programowania (**BASIC**), języka będącego podstawą współczesnych języków dla profesjonalistów (**C**), oraz języka operowania na bazach danych (**SQL**);
- poznanie w nieco większym stopniu niezwykle potężnego i wszechstronnego, a dla początkujących łatwego narzędzia, jakim jest **MATLAB**.

Pisanie typowych i niezbyt złożonych programów jest ułatwiane i częściowo automatyzowane przez pakiety RAD (*Rapid Application Development*) jak Visual BASIC, Delphi, C++ Builder, Java Buider. Firma Microsoft tworząca systemy operacyjne Ms Windows, i inne firmy jak choćby Google, Sun, czy Apple, udostępniają tak zwane API (*Application Programming Interface*) – interfejsy do programowania aplikacji czyli zestawy podprogramów ułatwiających tworzenie programów użytkowych z wykorzystaniem elementów dialogowych powszechnie znanych z innych programów.

1.5. NIECO HISTORII

Historia komputerów wywodzona jest często od pierwszych mechanicznych urządzeń wspomagających obliczenia, jak: liczydła (na przykład *abakus* wynaleziony kilkaset lat przed nową erą), kalkulator Wilhelma Schickarda (r.1625 - wykorzystujący koła zębate i tarcze z cyframi, ulepszony następnie przez Pascala) czy choćby niezrealizowana koncepcja mechanicznej programowanej maszyny liczącej Charlesa Babage'a (r.1832).

Jako pierwszy wyspecjalizowany komputer - wg „*timeline of computing*" zamieszczonej w [5] - wymieniana jest (skonstruowana w roku **1939** w USA) elektroniczna maszyna **ABC (*Atanasoff-Berry Computer*)** do rozwiązywania układów równań algebraicznych liniowych, jednak działała ona według tylko jednego, ustalonego programu.

Według różnych programów natomiast mógł działać komputer **Z3** (elektromechaniczny, na przekaźnikach) skonstruowany przez Niemca Konrada Zuse w roku **1941.**

Do powstania współczesnych komputerów przyczyniło się bardzo wiele odkryć i koncepcji, o czym możemy poczytać obszernie np. w[52], a między innymi wynalezienie: **lamp elektronowych** (1904-6 – John Fleming i Lee De Forest), opracowanie **metody hodowania dużych kryształów**, wykorzystywanej dziś przy produkcji układów scalonych (1916 – Polak prof. Jan Czochralski) a także skonstruowanie **tranzystora** (1947 – W.Shockley, W.Brattain, J.Bardeen) a potem **układu scalonego** (1958 - Jack Kilby). Na seryjną produkcję układów scalonych dużej skali integracji (m.in.mikroprocesorów) trzeba było jednak poczekać 13 lat.

Konstrukcje pierwszych komputerów przypadają na okres II Wojny Światowej. I tak - pierwszym elektronicznym komputerem, spełniającym podaną w p.1.1. definicję (a więc programowalnym, zbudowanym w roku **1944** z użyciem lamp elektronowych), był brytyjski **Collosus** używany do łamania szyfrów. Kolejnym - bardziej znanym – był opracowany na zlecenie armii USA w latach 1943-1946 **ENIAC**, ważący 30 ton i zbudowany z użyciem 18000 lamp elektronowych, natomiast pierwszym komputerem dostępnym w sprzedaży był amerykański **UNIVAC** (r. 1951).

Zależnie od zastosowanej technologii rozróżnia się poszczególne **generacje komputerów**. Komputery zbudowane z wykorzystaniem **przekaźników** to **generacja zerowa**, z użyciem **lamp elektronowych** - **generacja pierwsza**, z użyciem **tranzystorów -** generacja **druga**. **Trzecią** generację stanowią komputery wykorzystujące **układy scalone małej i średniej skali integracji**, a generację **czwartą** komputery wykorzystujące **układy scalone wielkiej skali integracji**.

W miarę rozwoju technologii powstawały z jednej strony coraz potężniejsze superkomputery, a równocześnie – po wprowadzeniu układów scalonych – także mini i mikrokomputery dla biur a także do użytku prywatnego.

Pierwsze trzydziestolecie od powstania ENIAC-a (1946-1976) to głównie rozwój dużych komputerów (*mainframe computers*), a jego zwieńczeniem jest powstanie w roku **1976 superkomputera trzeciej generacji Cray-1**, ale – z drugiej strony - wyprodukowanie **mikroprocesora 8-miobitowego** i powstanie pierwszych **mikrokomputerów**. W tym pierwszym trzydziestoleciu dominowały komputery firm IBM, CDC oraz DEC.

Wspomniany superkomputer Cray-1, zainstalowany w Narodowym Laboratorium Los Alamos miał szybkość 133 MFLOPS (133 miliony operacji zmienno-przecinkowych na sekundę). Wraz z freonowym systemem chłodzenia ważył on 5,5 tony, pobierał przeszło 200 kW mocy a kosztował prawie 10 milionów dolarów.

Od lat 60-tych – po skonstruowaniu układów scalonych – zaczęto także wytwarzać minikomputery. **Minikomputerami** nazywano komputery trzeciej generacji, które mieściły się w jednej niewielkiej szafie i często pełniły rolę dzisiejszych komputerów personalnych. Najbardziej popularne były minikomputery firmy DEC (*Digital Equipment Corporation*): PDP-8 (lata 1965-1990), PDP-11 (1970-1990) i VAX (1977-2005).

W Polsce w roku 1970 powstał **pierwszy polski minikomputer** (16-bitowy trzeciej generacji) **K-202** skonstruowany (w ciągu roku) przez inż. Jacka Karpińskiego wraz z zespołem zakładów MERA. Technicznie przewyższał wówczas porównywalne cenowo komputery amerykańskie. Potrafił wykonywać milion operacji na sekundę.

Wyprodukowano ok. 30 sztuk minikomputera K-202, a nowocześniejsze jego wersje nosiły nazwy **Mera-400** oraz MX-16.

Po wyprodukowaniu w firmie Intel pierwszego czterobitowego (r.1971) a następnie ośmio-bitowego **mikroprocesora** Intel 8008 w roku **1974**, już w tym samym roku powstał pierwszy prymitywny jeszcze mikrokomputerek domowy dla majsterkowiczów Altair 8800 (firmy MITS), który zamiast klawiatury miał przełączniki, zamiast monitora lampki a jego pamięć RAM miała pojemność 256 bajtów.

Mikroprocesor Intel 8008 zawierał 3500 tranzystorów a mikroprocesor Core I7 w roku 2008 już 780 milionów tranzystorów, zaś w roku 2010 opanowano technologię 32 nm pozwalającą zmieścić w jednym układzie scalonym ponad 2 miliardy tranzystorów. Wzrost złożoności układów scalonych następował w przybliżeniu wg t.zw. prawa Moore'a mówiącego o podwajaniu się ich liczby co ok. 18 do 24 miesięcy.

Prawie jednocześnie z wprowadzeniem mikroprocesorów przez firmę Intel zaczęto produkcję mikroprocesorów w firmach Motorola i MOS Technology, oraz powstał pierwszy mikrokomputer firmy Commodore o nazwie KIM-1, z klawiaturą numeryczną i wyświetlaczem z siedmiosegmentowych modułów LED.

W roku 1976 – tym samym w którym zbudowano superkomputer Cray-1 - powstała firma Apple Computer i jej pierwszy skromny mikrokomputer „Apple I" z monitorem i klawiaturą (w drewnianej obudowie starego dalekopisu) i rozpoczął się kilkunastoletni okres burzliwego rozwoju małych i tanich komputerów zwanych **mikrokomputerami domowymi lub personalnymi**, produkowanych w wielu konkurujących zażarcie firmach, które nieraz plajtowały bądź były wykupywane przez konkurencję.

W roku 1978-9 powstały 16-bitowe mikroprocesory Intel 8086, Intel 8088 - zastosowane w pierwszych komputerach personalnych PC oraz XT firmy IBM (1981-2). W roku **1982** w firmie Sinclair Research powstał ZX-Spectrum a w firmie Intel mikroprocesor 80286 – zastosowany w komputerze personalnym IBM AT.

W tym samym roku firma Motorola wyprodukowała bardzo wydajny 32-bitowy mikroprocesor MC68000 zastosowany później m.in. w komputerach: Atari ST (1985), Commodore Amiga (1985), Apple Macintosh (1984). Komputery te przewyższały znacznie inne współczesne komputery. W szczególności Amiga wyprzedzała mikrokomputery IBM z oprogramowaniem Microsoft o ok. 10 lat, takimi cechami jak wysokiej rozdzielczości (640 x 480) grafika o dużej liczbie kolorów i znacznej szybkości, czterokanałowy stereofoniczny dźwięk oraz wielozadaniowy graficzny, okienkowy system operacyjny.

Potęga i strategia marketingowa firmy IBM oraz współpracujących z nią firm Intel (mikroprocesory) i Microsoft (oprogramowanie) spowodowała, że mimo gorszych parametrów technicznych, komputery personalne IBM, a szczególnie ich tanie kopie budowane w Azji Wschodniej (dzięki dostępnej dokumentacji), stopniowo wygrały walkę konkurencyjną i wyparły z rynku większość mikrokomputerów innych firm.

Postępowała standaryzacja sprzętu, oprogramowania i procedur komunikacyjnych.

Rozpowszechnienie się mikrokomputerów „zgodnych z IBM-PC" (zwanych też klonami IBM-PC) spowodowało, że w nazwie pomija się obecnie przedrostek „mikro-" i nazywa się je „komputerami osobistymi" lub PC-tami od ang.: „*Personal Computer*".

Historia komputerów w Polsce rozpoczyna się w Zakładzie Aparatów Matematycznych PAN gdzie pod kierunkiem doc. Leona Łukaszewicza powstał w r.1958 pierwszy eksperymentalny, elektroniczny polski komputer **XYZ**. Jego ulepszone wersje pod nazwą **ZAM-2** produkowano (12 sztuk) od roku 1961.

Dalsze dzieje wiążą się głównie z Wrocławskimi Zakładami Elektronicznymi ELWRO, powołanymi do istnienia w r.1959 przez ministra przemysłu ciężkiego (a późniejszego rektora AGH) Kiejstuta Żemajtisa. W ELWRO produkowano komputery opracowywane w Polsce: UMC-1, Odra (1001, 1003, 1013, 1103, 1204, 1304, 1305, 1325) – łącznie ok. 1000 sztuk w tym większość na eksport. Od roku 1974 produkowano też R-32 – komputer t.zw. Jednolitego Systemu RIAD, krajów RWPG.

Przeszło trzydziestoletnia eksploatacja ostatnich trzech komputerów Odra 1305 - pracujących na stacjach rozrządowych PKP zakończyła się w latach 2006-2010.

W roku 1964 powstało Zjednoczenie MERA skupiające ELWRO i inne państwowe zakłady produkujące urządzenia komputerowe oraz automatyki i pomiarów. Jednym z istotniejszych były zakłady **Mera-Elzab w Zabrzu** – jedne z nielicznych, które przetrwały transformację ustrojową (produkują obecnie m.in. kasy fiskalne i sklepowe). W zakładach tych w roku **1983** skonstruowano pierwszy polski komputer domowy **Meritum** (z procesorem 8-miobitowym i pamięcią RAM 48 kB), a w r.1985 uruchomiono produkcję mikrokomputera personalnego **ComPAN-8**. Produkowano też w Polsce od r. 1986 komputery personalne „Mazovia", a w ELWRO „PC Elwro 801 AT".

Pod koniec lat osiemdziesiątych w EWRO wybudowano nowoczesną halę i zakupiono linie produkcyjne do produkcji polskich komputerów PC nowej generacji. Jednak sprzedanie w r.1993 Zakładów firmie Siemens skutkowało zlikwidowaniem produkcji komputerów, zwolnieniem kilku tysięcy pracowników i wyburzeniem 60% budynków.

Historia komputerów na Wydziale Inżynierii Mechanicznej i Robotyki AGH wiąże się z historią pracy autora tego podręcznika. Pierwsze laboratorium komputerowe na tym Wydziale (sala 322) powstało w roku 1973, wraz z powołaniem do życia **Pracowni Zastosowań Informatyki i Metrologii** (PZIiM) pod kierownictwem dr n.t. mgr **Jerzego Lasockiego** – wybitnego metrologa, a zarazem pełnomocnika rektora d/s komputeryzacji uczelni.

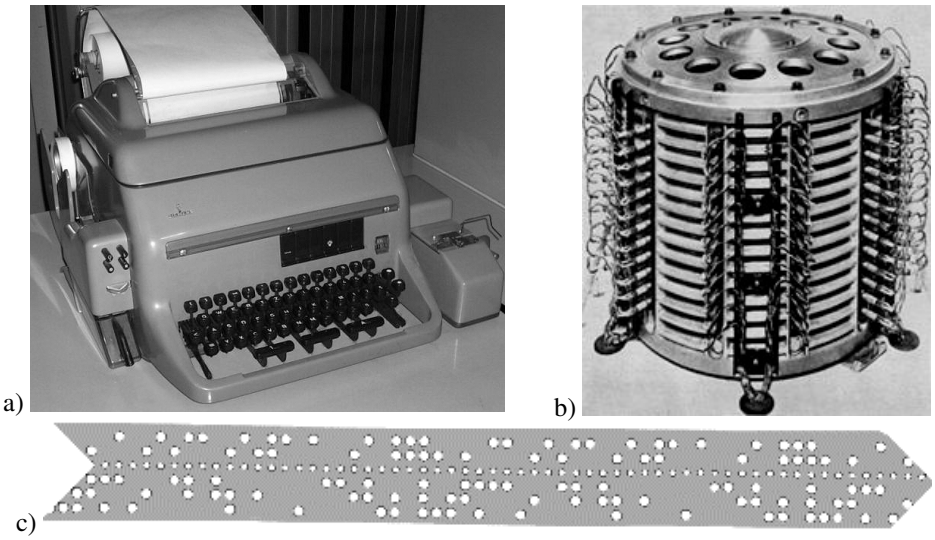
Zajęcia z ETO (Elektronicznej Techniki Obliczeniowej) odbywały się przy wykorzystaniu maszyny cyfrowej ODRA 1013 oraz kilku dalekopisów (Rys. 1.3a), przy pomocy których studenci tworzyli cyfrowy zapis opracowanych przez siebie programów na taśmie dziurkowanej (Rys. 1.3c), wczytywanej potem przez ODRĘ.

ODRA 1013 jako pamięć RAM posiadała ferromagnetyczną pamięć bębnową (Rys. 1.3b) o pojemności 8192 słowa 40 bitowe czyli 40 KB i dodatkowo niewielką pamięć na rdzeniach ferrytowych (256 słów), oraz dalekopis, czytnik i dziurkarkę taśmy.

Jeśli porównamy, to dziś komputer osobisty ma ok. 100 000 razy większą pamięć RAM (4 GB). Komputer ODRA 1013 miał szybkość 1000 dodawań na sekundę, ważył ok. pół tony i zawierał ponad 400 płytek z tranzystorami, opornikami i innymi elementami elektronicznymi. Nie było żadnych układów scalonych.

Wprawdzie pierwszy mikroprocesor, czterobitowy Intel 4004 powstał w roku 1971 ale ODRĘ wyprodukowano jeszcze przed jego wynalezieniem.

Maszyny ODRA 1003 i 1013 standardowo były programowane albo w Assemblerze albo w języku Most. Aby jednak studenci mogli programować w języku Fortran (będącym światowym standardem), autor opracował translator tego języka.



Rys. 1.3. a) dalekopis (fot. własna), b) bęben pamięci [4], c) taśma dziurkowana

W roku 1984 pozbyto się ostatniej ODRY 1013 i rozpoczęła się w PZIiM era mikrokomputerów. Zaczęło się od polskiego biurowego mikrokomputera MK-45 (produkcji MERA-KFAP w Krakowie) – wykorzystującego dyskietki o 8-mio calowej średnicy i pojemności 160 KB. Niestety oprócz systemu CP/M i kilku kompilatorów języków komputer ten nie posiadał oprogramowania użytkowego, dlatego autor opracował dla niego m.in. edytor tekstu. Następnie przybyły mikrokomputery Sinclair ZX-81 oraz dwa ZX-Spectrum 48, a potem jeszcze dwa podobne mikrokomputerki Timex. Studenci mogli więc programować w języku BASIC, standardowo wbudowanym w system.

Mikrokomputer ZX-Spectrum szybko stał się „kultowy" i do dziś w Internecie można znaleźć witryny tylko jemu poświęcone i skupiające grupy fanów. Można też pobrać z Internetu emulatory ZX-Spectrum, które symulują jego działanie na współczesnych komputerach w systemie Ms Windows i pozwalają uruchamiać tysiące programów napisanych dla ZX-Spectrum. Warto więc poświęcić nieco miejsca temu komputerkowi. Jego parametry [11], śmieszą z dzisiejszego punktu widzenia, ale wówczas były całkiem zadawalające dla tysięcy zastosowań.

ZX-Spectrum – wielkości niedużej książki - posiadał procesor Z80A taktowany zegarem 3.5 MHz, w pamięci stałej ROM (16 KB) mieścił się zawsze gotowy do pracy system operacyjny oraz interpreter języka Sinclair BASIC. Wszystkie komendy i funkcje tego języka były widoczne na klawiaturze dlatego każdy klawisz musiał mieć 5 różnych znaczeń przełączanych m.in. dwoma klawiszami Shift (białym Caps Shift i czerwonym Symbol Shift). Po początkowych trudnościach z nauczeniem się obsługi takiej klawiatury, wprowadzanie programów było łatwe, szybkie i pozbawione błędów literowych.

Pamięć RAM miała pojemność 48 KB, a mimo tego działo wiele tysięcy programów, w tym wielopoziomowe gry z biegającymi kolorowymi stworkami, edytory, arkusze kalkulacyjne i inne.

Rys. 1.4. Aparatura PZIiM w r.1975. Na pierwszym planie m.c. ODRA 1013 i pojemniki na taśmę dziurkowaną. Za nimi: czytnik taśmy, dalekopis i dziurkarka taśmy. Obok dalekopisu autor. Na regale aparatura pomiarowa. (Zdjęcie ze slajdu autorstwa dr inż. Krystyny Prync-Skotniczny.)

Rys. 1.5. Mikrokomputer ZX-Spectrum (fot. własna)

Programy firmowe, jak i samodzielnie napisane programy, można było wprowadzać z kaset magnetofonowych i na kasetach zapisywać przy użyciu zwykłego magnetofonu kasetowego, a rolę monitora pełnił zwykły telewizor. Ekran miał dwa tryby: tekstowy (32 x 24 znaki) i graficzny oferujący 8 kolorów i rozdzielczość 256 x 192 piksele. Dźwięk był jednokanałowy obejmujący 10 oktaw. ZX-Spectrum miał wymiary 23 x 14,4 x 3 cm i wagę 550g. Zasilany był z zewnętrznego zasilacza (9V, 1.4 A). Posiadał wyjścia i wejścia dźwiękowe oraz szynę systemową do podłączania urządzeń zewnętrznych. Było wiele takich urządzeń m.in. interfejs szeregowy, pamięć zewnętrzna na kasetkach Microdrive, dżojstiki, drukarka, pióro świetlne i in.

Rewolucyjnymi pomysłami – obniżającymi cenę – było wykorzystanie zwykłego magnetofonu i telewizora, a umieszczenie komend i funkcji BASIC-a na klawiaturze oraz bogata literatura i powszechność sprzyjająca rozwojowi - ułatwiały uczenie się tego języka.

W okresie „panowania" ZX-spectrum powstało m.in. szereg publikacji naukowych bazujących na programach napisanych dla tego mikrokomputerka. W ramach Koła Naukowego pod kierunkiem dr inż. Marii Zych-Porębskiej powstało szereg programów z dziedziny Podstaw Konstrukcji Maszyn opisanych i zamieszczonych w skryptach AGH:[8], [9], [10]. Głównym programistą tworzącym te programy pod kierunkiem dr Porębskiej był student Andrzej Nawalany. Znaczna część tych programów powstała w górskim schronisku koło Suchej, gdzie członkowie obozu naukowego wtaszczyli na własnych plecach telewizor i mikrokomputer ZX-Spectrum.

Ostatnim mikrokomputerem zakupionym dla PZIiM był Amstrad CPC 6128 [11] z roku 1985, wyposażony w 128 KB pamięci operacyjnej, profesjonalną klawiaturę, własny monitor, wielokanałowy dźwięk i wielokolorowy obraz (27 kolorów) o wyższej rozdzielczości (640 x 200) oraz wbudowaną stację dyskietek 3''. Amstrad posłużył m.in. do opracowania programów obliczeń łączników przegubowych [12], [13] wielkich walcarek .

Rys. 1.6. Amstrad CPC 6128 (źr.: [11])

Kolejny etap – trwający do dziś – to era tak zwanych PC-tów czyli komputerów personalnych. W latach 1987-88, autor – kierując Pracownią - utworzył Studenckie Laboratorium Komputerowe wyposażone w 8 komputerów - „zgodnych z IBM-PC" (jak wówczas mówiono), połączonych siecią lokalną NetWare Lite - oraz plotter, digitizer i drukarkę. Studenci oprócz podstaw programowania uczyli się obsługi Mathcada i Autocada. Pracownicy mieli dyżury a studenci dostęp do Laboratorium także między zajęciami.

Później – wskutek reorganizacji Wydziału – PZIiM formalnie przestała istnieć, jednak cały czas działało dalej i unowocześniało się Laboratorium Komputerowe – w ramach Katedry Konstrukcji i Eksploatacji Maszyn - aż do roku 2000 gdy powstały w jego miejsce Wydziałowe Pracownie Komputerowe. W międzyczasie powstało na Wydziale IMiR także wiele innych nowoczesnych pracowni komputerowych.

2. PODSTAWY PROGRAMOWANIA STRON WWW

Nieco dyskusyjne, jest umieszczenie w tytule słowa „programowanie", które w tym przypadku oznacza projektowanie i opisywanie postaci stron WWW, tak powszechnych w Internecie. Jednak do opisu tych stron - podobnie jak do programowania - używa się specyficznego języka, a polecenia tego języka określają sposób prezentowania treści przez komputer. Język HTML, używany do tego najpowszechniej, należy do języków opisu dokumentów czyli danych, a opis danych jest integralną częścią programowania w każdym języku.

Dodatkowo, strony WWW – podobnie jak programy komputerowe – reagują na akcje użytkownika, którymi są - w najprostszym przypadku - kliknięcia wskazywanych myszką odsyłaczy. Tak więc opracowywanie dokumentów zwanych stronami WWW można - zdaniem autora – zaliczyć do programowania.

Zagadnienia dotyczące Internetu zostały już ogólnie omówione w rozdziale 18 podręcznika [2], a w niniejszym rozdziale przypomniano i poszerzono wiadomości dotyczące możliwości publikowania w Internecie.

2.1. ŚWIATOWA PAJĘCZYNA I STRONY WWW

Pomimo, że serwis informacyjny WWW czyli „światowa pajęczyna" (*World-Wide Web*) jest tylko jednym z wielu sposobów korzystania z globalnej sieci komputerowej Internet, to jednak jest tak rozpowszechniony, że większość użytkowników utożsamia go z Internetem. W Internecie jest wiele milionów stron WWW. Mogą one zawierać:

- kolorowe lub wypełnione grafiką tła,
- sformatowane na wiele sposobów teksty oraz tabele,
- odsyłacze do innych stron (ang.: *links*),
- obrazy statyczne lub animowane,
- nagrania dźwiękowe,
- filmy i animacje,
- zmienne elementy dynamiczne - generowane na podstawie działań użytkownika.

Zawartość stron WWW zapisana jest w języku (kodzie) HTML, w postaci plików przechowywanych na **serwerach internetowych**, czyli komputerach podłączonych do Internetu i udostępniających swe zasoby przy pomocy programów, także nazywanych „serwerami internetowymi", takich jak: APACHE, STWEB lub inny. Liczba czynnych serwerów w Internecie (podawana przez Internet Systems Consortium – www.isc.org) w roku 2010 zbliżała się do 800 milionów (w lipcu 2010 ok. 769 mln.).

Liczba użytkowników Internetu, na początku roku 2012, przekroczyła 2 miliardy (według: www.internetworldstats.com), z czego ok. 45% to Azjaci (mimo, że stanowią oni tylko 26% mieszkańców Azji), 22% Europejczycy, a ok. 12% mieszkańcy Ameryki Północnej. Użytkowników Internetu w Chinach jest już przeszło 2 razy więcej niż w USA.

Użytkownik, aby oglądać strony WWW, musi mieć komputer podłączony do Internetu i uruchomić **przeglądarkę internetową** (ang.: *Internet browser*) np.: Internet Explorer, Mozilla Firefox, Google Chrome, Opera lub inną. Większość przeglądarek można darmowo pobrać z Internetu. Stare (kilkuletnie) przeglądarki na ogół nie pokażą poprawnie wszystkich stron WWW, ze względu na nie obsługiwanie najnowszych wersji języków opisu.

Strony WWW można podzielić na:

- **statyczne** – o ustalonej treści i wyglądzie, które wynikają z opisu w kodzie HTML;
- **dynamiczne** – opisane w p.2.5. - w których fragment lub całość może się zmieniać przy akcjach użytkownika gdyż jest generowany przez program (napisany na przykład w języku *Java Script* lub *PHP*) wstawiony do opisu w kodzie HTML jako tzw. skrypt.

2.2. JĘZYKI OPISU DOKUMENTÓW

Zanim przejdziemy do znacznie trudniejszych zagadnień związanych z algorytmami i programowaniem, zaczniemy od łatwych i efektownych działań, pozwalających tworzyć eleganckie strony internetowe. Problematyka ta jest fragmentem szerszej dziedziny dotyczącej **języków opisu dokumentów**, wykorzystywanych w definiowaniu składów drukarskich, sterowaniu drukarkami komputerowymi, wyświetlaniu dokumentów na ekranach komputerów oraz publikowaniu w sieci.

Ponad tysiąc lat przed powstaniem Internetu powstał wynalazek druku. Rękopisy przygotowywane do drukowania musiały mieć adnotacje i znaczniki dla drukarzy o rodzajach czcionek jakie mają w danym miejscu użyć. Systemy znaczników zmieniały się, ostatecznie stając się zaczątkiem współczesnych **języków znaczników**.

Języki znaczników pozwalają uzupełniać tekst przeznaczony do prezentowania na ekranie lub wydruku o informacje dotyczące formatu dokumentów, począwszy od hierarchicznego podziału na części, rozdziały, strony, akapity, tabele, rysunki, wzory, aż do szczegółów dotyczących kroju poszczególnych liter, symboli specjalnych, numeracji, wypunktowań, odsyłaczy, wcięć, odstępów, wyrównywania i t.p.

Najpowszechniejszym językiem znaczników jest opisywany dalej **HTML** *(Hypertext Markup Language)* używany do opisu postaci stron internetowych. Język HTML ewoluuje i co pewien czas powstają jego kolejne wersje.

Aktualne wersje współpracują z innymi językami jak:

- CSS (*Cascading Style Sheets* – kaskadowe arkusze stylów) umożliwiające stosowanie stylów i oddzielenie definicji struktury dokumentu od definicji jego wyglądu,
- wspomniane już języki do dynamicznego generowania stron jak: Java Script, VB Script, PHP, Perl i inne.

Do popularnych języków i systemów kodowania, stosowanych w edytorach, drukarkach i systemach składu wydawniczego należą także: T_EX, PDF, Postscript, PCL, RTF i in.

Internetowy system informacyjny WWW (*World-Wide Web*), oparty na języku znaczników HTML, opracowany został w latach 1980-1990 w ośrodku naukowo-badawczym fizyki CERN pod Genewą.

Twórcami HTML i WWW byli pracownicy CERN - fizyk Tim Berners-Lee i inżynier oprogramowania Robert Cailliau [4]. W roku 1991 Tim Berners-Lee opublikował pierwszą specyfikację języka HTML, który później był dalej rozwijany.

Język HTML zdefiniowano w standardzie SGML (*Standard Generalized Markup Language*) służącym właśnie do definiowania zbiorów znaczników przeznaczonych do konkretnych zastosowań. SGML jest rozbudowanym i skomplikowanym systemem dlatego częściej obecnie wykorzystuje się XML (*Extensible Markup Language* - rozszerzalny język znaczników) stanowiący podzbiór i uproszczenie SGML.

W XML można definiować dowolne znaczniki, w szczególności opisujące znaczenie i rolę poszczególnych danych. Pozwala to definiować m.in. bazy danych.

Na przykład:

```
<! PR 1:>
<?xml version="1.0" encoding="iso-8859-2"?>
<zwierzaki>
       <pies> Brytan </pies>
       <kot> Feliks </kot>
       <królik> Kicek </królik>
</zwierzaki>
```

W języku XML zdefiniowano różne języki znaczników na przykład:

- XHTML (*Extensible HyperText Markup Language*) - dla opisu stron WWW,
- OpenDocument – dla opisu dokumentów biurowych,
- SMIL (*Synchronized Multimedia Integration Language*) – dla opisu prezentacji multimedialnych,
- SVG (*Scalable Vector Graphics*) - dla grafiki wektorowej,
- MathML (*Mathematical Markup Language*) – dla opisu formuł matematycznych,
- WML (*Wireless Markup Language*) – dla opisu stron WAP.

Nieco starszym niż te języki i mocno zakorzenionym na uniwersytetach (szczególnie wśród fizyków), jest opracowany przez amerykańskiego matematyka i informatyka Donalda Knutha (w latach 1977-85 na Stanford University) system opisu postaci publikacji [4] nazwany **T_EX** (czytaj: „tech"). Program powstał, ponieważ profesor Knuth nie był zadowolony z wyglądu swojej książki *The Art of Computer Programming*. Niestety zamiast przewidywanego okresu półrocznego, prace nad systemem T_EX trwały przez 8 lat. Potem prof. Knuth ogłosił, że płaci każdemu kto znajdzie błąd w jego programie i podwajał stawkę co rok, począwszy od jednego centa.

Do głównych zalet systemu T_EX należą: stabilność działania, nieodpłatne udostępnienie oprogramowania (także w postaci źródłowej) oraz przenośność na różne platformy sprzętowe i systemowe. Prawie wszystkie prestiżowe wydawnictwa z zakresu nauk ścisłych przyjmują publikacje zapisane w T_EX-u, a czasem także udostępniają szablony, pozwalające nadawać publikacjom odpowiedni wygląd. Szablony te zazwyczaj oparte są na rozszerzeniu T_EX-a zwanym LaTeX2e.

Dla TEX-a dostępnych jest wiele zestawów makr ułatwiających pracę. Jednym z popularniejszych jest LaTeX opracowany przez Leslie Lamporta.

Powstało także szereg rozszerzeń TEX-a. Najpopularniejszym jest obecnie **PdfTeX** pozwalający tworzyć pliki w formacie **PDF**, powszechnie stosowane do publikowania artykułów i książek w Internecie. PdfTeX opracował Han The Thanh - wietnamski doktorant Uniwersytetu im. Masaryka w Brnie, w roku 2000.

Współczesne edytory tekstu często umożliwiają zapisywanie dokumentów w formacie PDF. Przykładem jest edytor z darmowego pakietu biurowego Open Office, który także potrafi wczytać i zapisać dokument w formacie Ms Word. Jeśli nasz edytor nie pozwala zapisywać dokumentów w formacie PDF to możemy pobrać, za darmo z Internetu, na przykład program CutePDF. Po zainstalowaniu tego programu zapisywanie dokumentów w formacie PDF będzie możliwe poprzez uruchomienie drukowania i wybranie z listy drukarek nazwy „CutePDF".

Współczesne **drukarki** wykorzystują języki opisu stron (tekstu i grafiki) takie jak PS czyli **PostScript** - opracowany w firmie Adobe Systems i będący równocześnie językiem programowania, oraz **PCL** (Printer Command Language). Publikacje drukowane z danego edytora są więc być automatycznie tłumaczone z języka opisu stron wykorzystywanego przez ten edytor (np.: RTF – *Rich Text Format*) na język wykorzystywany przez drukarkę.

We współczesnych stronach WWW duże znaczenie mają **animacje**. Istnieje szereg sposobów konstruowania animacji. Jednym z nich jest tworzenie animowanych plików typu GIF. Darmowe programy do tego celu można znaleźć w Internecie.

W ostatnich latach najlepszej jakości animacje, także interaktywne, tworzone są przy pomocy programu Adobe **Flash** (dawniej Macromedia Flash). Powstają pliki (z rozszerzeniem „swf"), które można odtwarzać za pomocą przeglądarki internetowej z zainstalowaną odpowiednią „wtyczką" (np. Adobe Flash Player) lub w oddzielnym programie. Od wersji Flash 5, program wyposażony został w język programowania ActionScript, umożliwiający obsługę zdarzeń (np. kliknięć myszką). Dzięki temu można tworzyć interaktywne animacje i programy - na przykład gry. Jeden z darmowych podobnych programów to Synfig Studio.

2.3. PUBLIKOWANIE W INTERNECIE

Publikowanie w Internecie jest znacznie tańsze i łatwiejsze niż wydawanie publikacji w postaci drukowanej – książek czy artykułów w czasopismach. Jak już wspomniano - autor musi w tym celu umieścić swe publikacje w postaci plików komputerowych na internetowym **serwerze WWW** a czytelnicy muszą dysponować komputerami podłączonymi do Internetu i wyposażonymi w programy zwane **przeglądarkami internetowymi**. Można wykorzystać serwer uczelniany lub serwer jednego ze znanych portali jak Onet, czy Wirtualna Polska – pod warunkiem, że mamy na tym serwerze konto z uprawnieniami do tworzenia stron WWW. Darmowe serwery (*free www servers*) łatwo znaleźć w Internecie, na przykład znalazłem ich listę w poradniku [19].

Rolę serwera WWW może także pełnić dowolny komputer podłączony **na stałe** do Internetu, jeśli zainstalowano na nim oprogramowanie realizujące funkcje serwera WWW.

Darmowe wersje oprogramowania serwera WWW można znaleźć i pobrać z Internetu. Najpopularniejszym jest program ***Apache***.

Najmniej zaawansowani autorzy internetowych publikacji, mogą skorzystać z możliwości darmowego tworzenia stron internetowych przy pomocy **kreatorów** udostępnianych przez portale takie jak Onet (onet.pl), Wirtualna Polska (wp.pl), czy Interia (miasto.interia.pl/). Najczęściej można w ten sposób tworzyć strony typu „blog" (dziennik) – przez wybór jednego z dostępnych szablonów oraz określanie elementów strony.

Kreatory tego rodzaju można zaliczyć do **systemów zarządzania treścią** określanych też angielskim akronimem ***CMS*** – *Content Management Systems*. Istnieje szereg darmowych systemów CMS (Wikipedia wymienia ich ponad 100). Systemy CMS bazują na różnorodnych **szablonach**, natomiast **treści** gromadzone są **w bazach danych**. Całość obsługiwana jest przez programy (napisane najczęściej w języku PHP), współpracujące z bazą danych (najczęściej MySQL).

Dzięki takiej organizacji, możliwe jest oddzielenie zawartości informacyjnej serwisu od sposobu prezentowania stron. System CMS generuje dynamicznie strony internetowe na podstawie treści z baz danych oraz szablonów. Dzięki temu łatwo można dokonać zmiany koncepcji graficznej serwisu lub prezentować te same dane w różnych formatach. Systemy CMS służą więc nie tylko początkującym ale także tym którzy prowadzą profesjonalne portale internetowe. Według badań z marca 2011 [24], w Polsce do najpopularniejszych systemów CMS należą: *Joomla, WordPress, Drupal* i *Moodle*.

Klasycznym sposobem tworzenia stron internetowych – dającym swobodę twórczą – jest jednak użycie wyspecjalizowanych **edytorów** ułatwiających używanie języków HTML, XHTML, XML, Java Sctipt, PHP i in., opisanych w poradnikach [19] i [21].

Aby dokładniej opisać proces publikowania w Internecie, należy odpowiedzieć na pytania:

a) w postaci **jakich typów plików** należy zapisywać dokumenty dla ich opublikowania w Internecie?

b) jak tworzyć te pliki?

c) jak umieszczać pliki na serwerach internetowych?

Przeglądając Internet łatwo zaobserwować, że najczęściej spotykane końcówki nazw plików – określające ich typy – to: „**htm**", „**html**", „**php**" i „**pdf**". Mamy więc odpowiedź na pytanie (a). Pierwsze trzy końcówki – określają pliki w języku HTML, przy czym rozszerzenie „php" sygnalizuje, że dokument w języku HTML zawiera również wstawki w języku **PHP**, na podstawie których serwer internetowy wygeneruje odpowiednie fragmenty kodu HTML. O języku PDF napisano już kilka słów w przednim podrozdziale a poniżej zajmiemy się językiem HTML.

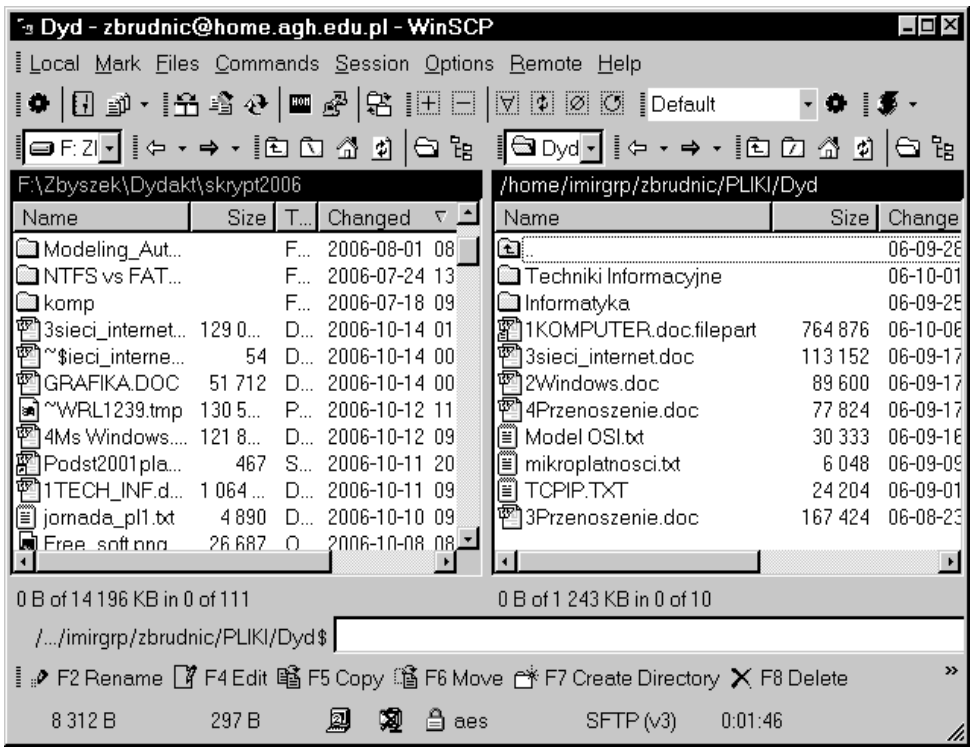
Użycie wspomnianych wyżej kreatorów stron WWW jest jedną z metod tworzenia plików HTML (pytanie b) od razu na serwerze, o czym użytkownik może nie wiedzieć.

Drugą metodą jest opracowywanie strony WWW przy pomocy zaawansowanego edytora tekstowego – jak Ms Word lub Writer z pakietu OpenOffice a następnie - przy zapisywaniu do pliku (na własnym komputerze) – wybranie opcji „zapisz jako HTML" lub „eksportuj jako PDF", gdy zdecydujemy się na taki typ pliku.

Zaletami tych metod są: brak konieczności korzystania z języka HTML oraz praca na stronie o wyglądzie docelowym czyli w trybie **WYSIWYG** (*What You See Is What You Get*) – "widzisz to co dostaniesz ostatecznie na ekranie", chociaż może to być nie w pełni prawdą, gdyż różne typy przeglądarek mogą różnie naszą stronę ukazywać.

Dla najbardziej zaawansowanych autorów stron WWW najodpowiedniejsza jest trzecia metoda - użycie jednego z **edytorów kodu HTML,** ułatwiających tworzenie nawet zaawansowanych stron WWW przy pełnym panowaniu nad szczegółami. Szereg takich edytorów – w tym polskich lub spolszczonych wymieniono w poradnikach [19] i [21]. Twórca widzi wtedy cały czas powstający kod HTML, a co pewien czas powinien sprawdzać wygląd wynikowego obrazu strony w wybranych przeglądarkach lub w przeglądarce wbudowanej w edytor.

Znając podstawy języka HTML można tworzyć proste strony WWW także korzystając z najprostszego edytora tekstu jak choćby NOTATNIK w systemie Ms Windows.



Rys. 2.1. Przykładowy ekran programu WinSCP

Odpowiedź na pytanie (c) została już udzielona w podręczniku [2] (rozdział 18.5.2), a mianowicie: mając konto na serwerze (np.: uczelnianym), umożliwiające zakładanie własnych stron WWW, można bardzo wygodnie kopiować pliki na serwer przy pomocy darmowego, programu **WinSCP**. Program **WinSCP** (Rys. 2.1), można legalnie pobrać z witryny *winscp.net*.

WinSCP służy do bezpiecznego (szyfrowanego) przesyłania plików pomiędzy lokalnym i zdalnym komputerem. Po jego uruchomieniu na ekranie pojawiają się dwa okna. W lewym - widoczna jest zawartość wybranego foldera, komputera lokalnego, a w prawym – zawartość foldera na odległym serwerze. Pliki można kopiować przez przeciąganie myszką lub użycie klawisza F5 (jak w programach Windows Commander lub Far).

Na końcu tego podrozdziału zadajmy pytanie: dlaczego w Internecie dominują pliki typu „HTML" lub „PDF" a raczej nie publikuje się dokumentów typu „doc" czy „odt" tworzonych edytorami tekstów jak Ms Word lub OpenOffice Writer?

Otóż po pierwsze dlatego, że w dokumentach takich nie zawsze istnieje możliwość wstawiania odsyłaczy, łączących dany dokument z innymi.

Po drugie, twórcy języka i systemu WWW mieli na uwadze to, że do Internetu podłączone są różnorodne komputery korzystające z różnych systemów operacyjnych i wyposażone w edytory kodujące zapis tekstów na różne sposoby. W okresie tworzenia języka HTML, niemal jedynym typem plików tekstowych czytelnych na każdym komputerze pozostały pliki typu „txt" napisane z wykorzystaniem podstawowego zbioru znaków ASCII i tworzone w najprostszych edytorach takich jak „Notatnik". Twórcy języka HTML bazując na tych najprostszych plikach tekstowych uzupełnili je o dodatkowe znaczniki HTML (ang.: *HTML tags*) czyli polecenia języka HTML - umieszczane w nawiasach kątowych **<** ... **>**, a zawierające informacje o formatowaniu tekstu i uzupełnianiu go obrazami, tabelami i innymi elementami.

Tak więc, wprawdzie można na serwerach umieszczać dokumenty utworzone na przykład w edytorze Ms Word, ale ryzykujemy, że niektórzy potencjalni czytelnicy będą mieli kłopoty z ich odczytaniem, jeśli ich komputer nie pracuje z systemem Ms Windows.

Z kolei dokumenty typu PDF mają postać przygotowaną do wydruku a ich czytanie i drukowanie jest możliwe po pobraniu i zainstalowaniu darmowego programu Adobe Acrobat Reader. Zapisując sformatowany dokument jako plik PDF możemy mieć pewność, że każdy kto ten dokument wydrukuje, otrzyma dokładnie taką postać jak w oryginale. Dokumenty PDF mogą być wprawdzie skalowane ale skala dotyczy wtedy całych stron a nie ich poszczególnych elementów. Inaczej działa skalowanie czcionek w dokumentach HTML, przy którym zmienia się rozmieszczenie tekstu w wierszach a podział na strony jest zazwyczaj dość przypadkowy. Dokumenty PDF mogą być także zabezpieczane przed kopiowaniem fragmentów tekstu czy pojedynczych obrazów i rzadko stosuje się w nich odsyłacze (choć istnieje taka możliwość).

2.4. PODSTAWY JĘZYKA HTML

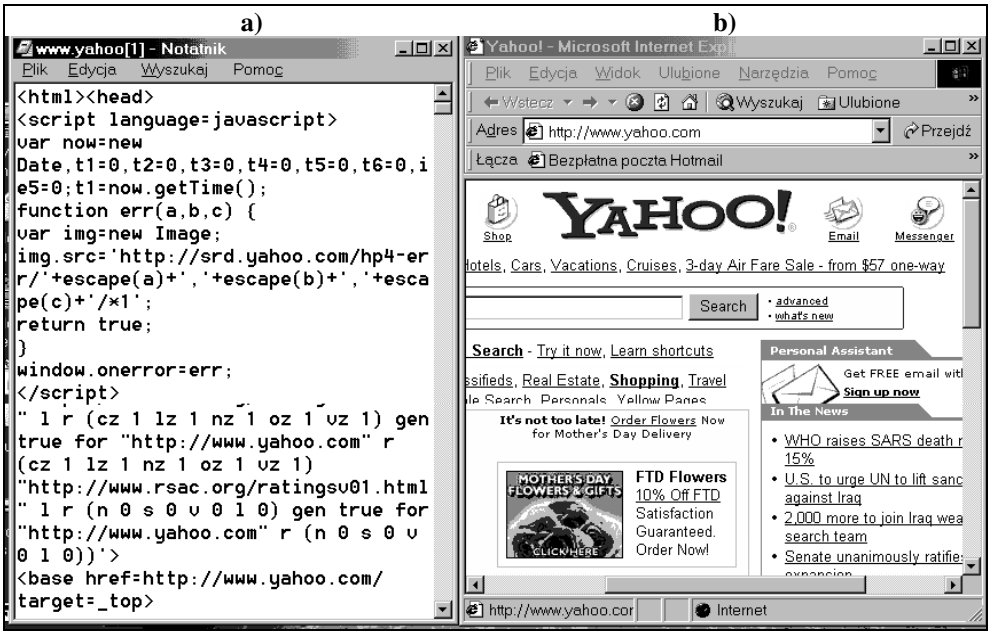
Struktura oraz wygląd większości stron internetowych serwisu WWW - czyli "światowej pajęczyny" (*World Wide Web*) - opisane są w **języku znaczników hipertekstowych HTML** *(HyperText Markup Language)***.**

Język HTML rozwija się i zmienia poprzez swoje kolejne wersje. Twórcy witryn internetowych mają także do dyspozycji inne języki, jak: XHTML, JavaScript, PHP, ASP, XML. Ale podstawą jest HTML.

W tym rozdziale opisane będą tylko najpotrzebniejsze elementy języka, w podstawowych postaciach stosowanych od najstarszych w wersji. Niektóre z nich nie są obecnie zalecane w nowszych wersjach HTML aczkolwiek są poprawnie interpretowane przez przeglądarki i są łatwe do zapamiętania ze względu na swą prostotę.

Naszym zadaniem jest więc poznanie **podstaw języka HTML** w praktyce, m.in. przez próby tworzenia i modyfikowania własnych stron WWW na ćwiczeniach laboratoryjnych lub w domu. Wszystkim zainteresowanym dokładniejszym poznaniem HTML polecam internetowe samouczki [19] i [21].

Dokumenty napisane w języku HTML – stanowiące opisy poszczególnych stron – przechowywane są jako pliki z rozszerzeniami nazw „**`.html`**” albo „**`.htm`**”.



Rys. 2.2. Język HTML: a) fragment opisu strony, b) wygląd strony wygenerowanej z tego opisu

Jeśli taki dokument otworzymy w Notatniku to będziemy widzieć jedynie nieco tajemniczy tekst stanowiący postać źródłową strony WWW (Rys. 2.2a). Jeśli natomiast ten sam dokument otworzymy w **przeglądarce internetowej** (jak Ms Internet Explorer, Mozilla Firefox, Netscape, Opera czy w innej) to pokaże się elegancka, kolorowa strona z obrazkami i formatowanym tekstem jak na Rys. 2.2b, dzięki temu, że przeglądarka reaguje na polecenia zapisane jako **znaczniki HTML,** ustawiając odpowiadające im atrybuty elementów strony.

Znaczniki HTML (ang.: *HTML tags*) to polecenia lub deklaracje zamknięte w nawiasy kątowe: **<** ... **>**. Większość znaczników występuje **w parach** obejmujących formatowane fragmenty tekstu i złożonych ze znacznika otwierającego oraz zamykającego. Znacznik zamykający różni się od otwierającego wystąpieniem kreski ukośnej (*slash*) przed nazwą znacznika. Para znaczników wraz z obejmowanym tekstem nazywana jest **elementem**.

Na przykład w tekście: „... najważniejsze jest ****zdrowie**** ...” wyraz „zdrowie” będzie wyświetlony pogrubioną czcionką, dzięki znacznikom ... .

W wielu znacznikach, po nazwie mogą wystąpić **atrybuty** (zwane czasem parametrami) oddzielane od siebie i od nazwy znacznika odstępami. Atrybut ma postać:

nazwa_atrybutu = ”wartość atrybutu”.

2.4.1. STRUKTURA OPISU STRONY. GŁOWA DOKUMENTU

Dokument HTML rozpoczyna się znacznikiem: **<html>** a kończy się zamknięciem tego znacznika: **</html>.** Między tymi znacznikami znajdują się dwie główne części dokumentu (Rys. 2.3): "głowa" oraz "ciało" (po angielsku: *head* i *body*), objęte odpowiednio parami znaczników:
<head> ... </head> oraz **<body> ... </body>**

"Głowa" zawiera informacje o dokumencie takie jak tytuł, autor, tematyka, słowa kluczowe, sposób kodowania liter i in.

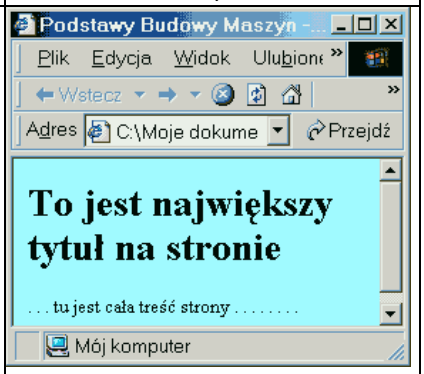
```
<html>
    <head>
    . . . .
    </head>
    <body>
    . . . .
    </body>
</html>
```

Rys. 2.3. Struktura dokumentu HTML

a)

```
<HTML>
 <Head>
  <Title> Podstawy Budowy Maszyn </Title>
  <META  HTTP-EQUIV="content-type"
  CONTENT="text/html; CHARSET=windows-1250">
  <META Name="Author" Content="Z. Kowalski">
  <META Name="Keywords" Content="maszyny">
 </Head>
 <Body bgcolor="#bbffff">
     <H1>To jest najwiekszy tytuł na stronie</H1>
     . . .  tu jest cała treść strony . . . . . .
  </Body>
</HTML>
```

b)



Rys. 2.4. Przykład opisu najprostszej strony www: a) opis w HTML-u, b) wygląd w przeglądarce

Rys. 2.4 pokazuje konkretny przykład najprostszej strony. W sekcji **<Head>** (Rys. 2.4a), umieszczenie tekstu między znacznikami **<title>...</title>**, spowoduje wyświetlenie tego tekstu w tytułowym pasku okienka przeglądarki. Następne polecenie w tej sekcji jest deklaracją systemu kodowania polskich liter:

<META HTTP-EQUIV="content-type" CONTENT="text/html; CHARSET=windows-1250">

Atrybut **CHARSET=windows-1250** określa system polskich znaków stosowany w Ms Windows, natomiast zgodność z polską normą ISO wymaga atrybutu **CHARSET=iso-8859-2**.

Polecanym obecnie systemem kodowania polskich znaków jest jednak standard Unicode: **CHARSET=utf-8.**

Kolejne dwie definicje - rozpoczynające się słowem **META** - określają autora dokumentu oraz słowo kluczowe określające tematykę dokumentu.

Podobnych informacji można dać więcej, ale tu objaśniono tylko najprostsze i najbardziej niezbędne elementy strony www. Na marginesie można dodać, że w XHTML znacznik <html> ma bardziej rozbudowaną postać i poprzedzony jest dwoma dodatkowymi liniami precyzującymi wersję XHTML.

Przeglądarki jednakowo interpretują polecenia pisane zarówno małymi jak dużymi literami, aczkolwiek zalecane jest używanie małych liter, obowiązujących już w języku XHTML.

2.4.2. CIAŁO DOKUMENTU.

W deklaracji **<body>** - rozpoczynającej "ciało" dokumentu – po słowie **BODY** może wystąpić szereg atrybutów tej sekcji opisujących tło strony oraz kolory tekstu i odsyłaczy na stronie. Jak wspomniano, składnia atrybutów ma postać: ***nazwa_atrybutu= "wartość"***.

Najczęściej spotykane atrybuty sekcji BODY pokazuje Tabela 2.1. Zamiast słowa "*kolor*" wystąpi konkretna wartość koloru, albo jako nazwa (np.: "yellow", "red", "blue", "cyan", "magenta", ...) lub jako wartość liczbowa zaczynająca się znakiem # i złożona z trzech dwucyfrowych liczb szesnastkowych określających udział kolorów składowych: czerwonego (RED), zielonego (GREEN) i niebieskiego (BLUE). Zgodnie z tym np. "#FFFFFF" to kolor biały, "#000000" – czarny, "#FFFFDD" – jasnożółty.

Tabela 2.1. Ważniejsze atrybuty sekcji BODY

Atrybut sekcji BODY:	Określa:
bgcolor = "*kolor*"	kolor tła strony
text = "*kolor*"	domyślny kolor tekstu na stronie
link = "*kolor*"	kolor tekstów będących odsyłaczami
vlink = "*kolor*"	kolor odsyłaczy już wykorzystanych
background = "*nazwa_pliku_graficznego*"	obrazek, który ma stanowić tło strony

2.4.3. NAGŁÓWKI

Jeśli chcemy aby nasza strona WWW była czytelna to jej tekst powinien mieć wyróżnione nagłowki (różnych poziomów) oraz być podzielonym na spójne tematycznie akapity.

W języku HTML mamy do dyspozycji kilka poziomów **nagłówków** (ang. *header*), a mianowicie:

```
<h1> Nagłówek poziomu 1 </h1>  (pokazany na Rys. 2.4)
<h2> Nagłówek poziomu 2 </h2>
<h3> Nagłówek poziomu 3 </h3>
<h4> Nagłówek poziomu 4 </h4>  ... i tak dalej
```

2.4.4. FORMAT AKAPITÓW I CZCIONEK

Akapity tekstu (ang.: *paragraph*) należy obejmować znacznikami ** ... .** W przeglądarce odstęp między dwoma akapitami – zwany interlinią - jest większy niż odstęp między liniami należącymi do jednego akapitu co pokazuje Rys. 2.5b, dlatego w przypadkach gdy chcemy wymusić wyświetlanie tekstu od nowej linii bez zwiększania tego odstępu możemy zastosować znacznik **
** - złamanie linii (ang.: *break*).

Na rysunku Rys. 2.5a pokazano nie tylko znaczniki ** ... ** obejmujące akapity ale także znaczniki odpowiadające za **wytłuszczanie** tekstu: ** ... ** (pochodzące od angielskiego*: bold*), podkreślanie: **<u> ... </u>** (od angielskiego: *underline*) oraz *kursywę*: ** ... ** (od angielskiego: *italic*).

a)	b)
** **Język ****HTML** (**HyperText Markup Language**)** służy **<u>**do opisu stron WWW**</u>**. Między innymi pozwala umieszczać na stronach WWW tabele, odsyłacze, grafikę i elementy multimedialne.** ** ** **Opisy stron WWW w języku HTML przechowywane są w plikach posiadających rozszerzenia nazw: HTM lub HTML** **	Język **HTML** (*HyperText Markup Language*) służy do opisu stron WWW. Między innymi pozwala umieszczać na stronach WWW tabele, odsyłacze, grafikę i elementy multimedialne. Opisy stron WWW w języku HTML przechowywane są w plikach posiadających rozszerzenia nazw: HTM lub HTML

Rys. 2.5. Postać dwu akapitów tekstu: a) w pliku HTML, b) w przegladarce

Znaczniki te łatwo zapamiętać gdyż kojarzą się z analogicznym formatowaniem w edytorach. Jednak w nowszych wersjach HTML należy, zamiast znaczników określających fizyczny wygląd, używać znaczników określających logiczną hierarchię tekstu a mianowicie:

** ... ** - „czcionka wyróżniona (emfaza)"
** ... ** - „czcionka mocno wyróżniona"

lub stosować style np.:

** ** tekst akapitu wytłuszczony** **

Dla otrzymania wykładników potęg lub indeksów można stosować znaczniki:

^{...} - superskrypt (indeks górny),
_{...} - subskrypt (indeks dolny)

Dla zmiany rozmiaru, kroju oraz koloru czcionki można wykorzystywać odpowiednie atrybuty znacznika ** ... **. Na przykład:

**** ten tekst będzie wyświetlony powiększoną czcionką Arial w kolorze czerwonym ****

Niestety znacznik ten został wycofany z wersji HTML 5 na rzecz używania stylów, które nieco komplikują zapis.

Do wyrównywania akapitów w starszych wersjach HTML można było używać znaczniki:

** ** ten akapit będzie wycentrowany ** ,**

<right> a ten akapit będzie dosunięty do prawej **</right>**,

które jednak zostały wycofane z nowych wersji. Zamiast nich można używać stylów, na przykład:

```
<p style="text-align: center; ">Akapit wyrównany do środka. </p>
<p style="text-align: right; ">Akapit wyrównany do prawej. </p>
```

2.4.5. LISTY WYPUNKTOWANE I LISTY NUMEROWANE

Postać kodów HTML pozwalających uzyskać listy wypunktowane oraz listy numerowane pokazuje Tabela 2.2.

Tabela 2.2. Kod HTML dla list wypunktowanych i numerowanych

Lista wypunktowana:	**Lista numerowana:**
`<UL>`	`<OL>`
`<LI> treść punktu`	`<LI> treść punktu`
`<LI> treść punktu`	`<LI> treść punktu`
`. . . . . .`	`. . . . . .`
`<LI> treść punktu`	`<LI> treść punktu`
`</UL>`	`</OL>`

2.4.6. ODSYŁACZE HIPERTEKSTOWE DO INNYCH DOKUMENTÓW

Termin "hipertekst" oznacza tekst zawierający wyróżnione słowa, których kliknięcie przenosi nas do innego tekstu, istniejącego albo w tym samym dokumencie albo w innym dokumencie na dowolnym internetowym serwerze WWW. Takie słowa to odsyłacze (zwane po angielsku LINKS – łączniki). Odsyłacz do dokumentu istniejącego w tym samym folderze ma postać:

<A HREF=”nazwa_pliku”**>** Tekst odsyłacza **</A>**

na przykład:

<A HREF=”zadania.htm”**>** Zadania z fizyki**</A>**

Jeśli odsyłamy do dokumentu w innym folderze to można wpisywać ścieżki dostępu podobnie jak w systemie DOS czy Windows.

Najlepiej stosować względne ścieżki dostępu, w których kropka oznacza folder bieżący a dwie kropki folder nadrzędny. Jednakże jako separatora nazw folderów nie należy stosować znaku "\" (back slash) lecz znak "/" czyli "slash", tak jak jest to przyjęte w systemie UNIX.

Na przykład, jeśli czytany dokument jest w folderze FIZYKA i jest w nim odsyłacz:

<A HREF="../matematyka/zadania.htm"**>**Zadania z matematyki**</A>**

to po jego kliknięciu komputer wycofa się z bieżącego foldera (FIZYKA) do nadrzędnego (bo to oznacza "../") a z niego dopiero wejdzie do foldera MATEMATYKA.

Jeśli natomiast z tego samego dokumentu w folderze FIZYKA klikniemy odsyłacz:

<A HREF="./odpowiedzi/zadanie_9.htm"**>**Odpowiedź do zad.9**</A>**

to znaczy, że w folderze bieżącym (bo "./") czyli „FIZYKA" musi być folder podrzędny "ODPOWIEDZI" a w nim plik "zadanie_9.htm".

Odsyłacze do dokumentów na innych komputerach (serwerach):
<A HREF="**http://nazwa_komputera/ścieżka_dostępu/nazwa_pliku**"**>** tekst odsyłacza **</A>**

Na przykład odsyłacz do dokumentu **wyklady.htm** w folderze KWP na serwerze **www.kkiem.agh.edu.pl** może wyglądać następująco:

<A HREF="http://www.kkiem.agh.edu.pl/KWP/wyklady.htm"**> KWP wykłady </A>**

2.4.7. GRAFIKA

Wstawianie obrazków:

<img src="nazwa_pliku_z_obrazkiem"**>**

IMG jest skrótem od słowa *image* - obraz,
SRC jest skrótem od słowa *source* źródło.

Na przykład:

<img src="kwiatek.gif">

Można dodawać różne **atrybuty**, jak choćby zmieniać szerokość w pikselach (np.: width=200) i wysokość (np.: height= 150) ale szczególnie przy zmniejszaniu obraz o niedużej rozdzielczości może stać się mało czytelny. Jeśli mamy duże obrazki nie mieszczące się na ekranie to jednak trzeba je na pewno zmniejszyć. Bardzo praktyczne jest wtedy **podanie tylko szerokości** (dzięki czemu będą zachowane naturalne proporcje wysokości do szerokości) ale nie w pikselach lecz w **procentach** szerokości okna, na przykład:

<img src="panorama.png" width=90% >

Jakie typy plików obrazów stosować? Dla fotografii barwnych najlepszy jest typ **JPG** natomiast dla innych rysunków i obrazków (na przykład schematów) korzystne jest zmniejszenie liczby potencjalnych kolorów do 256 lub nawet 16-tu i zapisanie ich w pliku typu **GIF** albo **PNG**.

Nie należy obrazków czarno-białych lub zawierających kilka kolorów zapisywać jako barwne 24-bitowe czy nawet 36-bitowe lecz zapiszmy je jako 4-ro bitowe czyli 16-to kolorowe dzięki czemu zajmą odpowiednio 6 lub 9 razy mniej miejsca i odpowiednio szybciej będą się ładować w przeglądarce.

2.4.8. TABELE

Definicja tabeli zawarta jest między znacznikami:

<table border = *n* ...> ... </table>

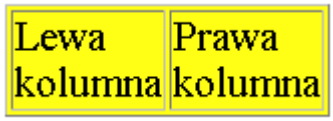
gdzie ***n*** - szerokość linii siatki,

Opis wiersza (ang.: *row*) zawarty jest między znacznikami: **<TR ...>** ... **</TR>**,
a opis komórki danych w wierszu zawarty jest między znacznikami: **<TD ...> ... </TD>**.

Aby tabela zawierała linie siatki, po słowie **table** musi wystąpić atrybut **border**. Jeśli nie podamy szerokości linii siatki, przyjmowana jest jej domyślna wartość.

Kolor tła tabeli możemy określić atrybutem **bgcolor=**"kolor".
Rys. 2.6 podaje przykład tabeli mającej 1 wiersz i 2 kolumny:

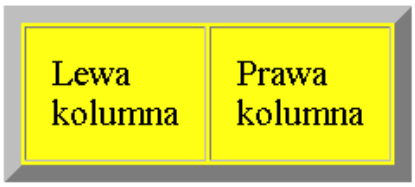
```
<Table border=1 bgcolor="yellow">
 <tr>
  <td> Lewa <br> kolumna  </td>
  <td> Prawa <br> kolumna </td>
 </tr>
</Table>
```



Rys. 2.6. Przykład definiowania tabeli w języku HTML

Jeśli uznamy, że odstęp między zawartością komórki a jej obramowaniem jest zbyt mały, możemy użyć atrybutu CELLPADDING (domyślnie wynosi on 1). Przy okazji poszerzymy ramkę poleceniem „border=10" (Rys. 2.7).

```
<Table border=10 bgcolor="yellow"
          cellpadding=15>
 <tr>
  <td> Lewa <br> kolumna  </td>
  <td> Prawa <br> kolumna </td>
 </tr>
</Table>
```



Rys. 2.7. Definiowanie szerokości obramowania i odstępu między tekstem a ramkami tabeli

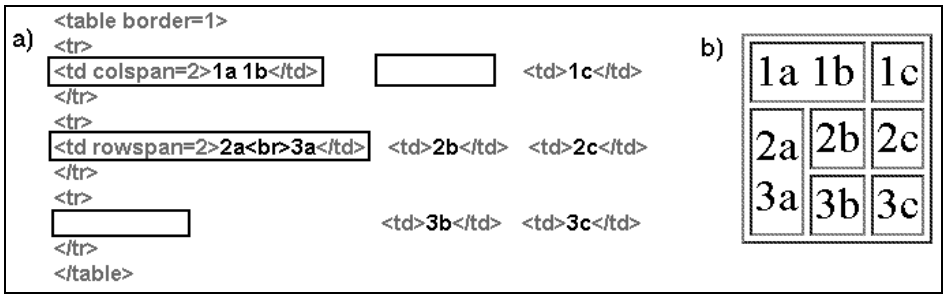
Scalenie **n** wybranych komórek z sąsiadujących kolumn (a więc w poziomie) jest możliwe przy użyciu atrybutu **colspan n**, natomiast do scalenia **n** wybranych komórek z sąsiadujących wierszy (czyli w pionie) trzeba użyć atrybutu: **rowspan n.**

Przykładowy opis i wygląd tabeli o wymiarach 3x3, przed scalaniem komórek pokazuje Rys. 2.8.

Widać jej opis w języku HTML (Rys. 2.8a) na którym zaznaczono prostokątami fragmenty dotyczące komórek (1a i 1b oraz 2a i 3a) do scalenia. Fragmenty te muszą ulec zmianie polegającej na użyciu atrybutów **colspan** i **rowspan** oraz zastąpieniu dwu opisów „<td>...</td>" jednym.

Rys. 2.8. Tabela przed scalaniem komórek

Zapis i wygląd tabeli po scaleniu pokazuje Rys. 2.9. Prostokątami zaznaczono w opisie HTML fragmenty nowe oraz puste miejsca po fragmentach usuniętych (Rys. 2.9a).



Rys. 2.9. Tabela po scaleniu wybranych komórek

2.4.9. RAMKI

Okno przeglądarki może wyświetlać dokument składający się z kilku **ramek**, których zawartości mogą zmieniać się w oddzielnych momentach. Wymiary ramek mogą być ustalone (np. jako procent całego okna) przy czym często istnieje możliwość zmiany tych wymiarów przy pomocy myszki. Ramki ułatwiają przegladanie stron WWW, jednak w wersjach HTML4 i 5 nie są zalecane.

Utworzenie strony z ramkami wymaga zaplanowania podziału okna przeglądarki na ramki, to znaczy określenia ile będzie ramek oraz czy powstaną one z podziału wysokości czy też szerokości okna. Główny dokument, którego nazwa będzie używana przy wywoływaniu strony z ramkami, musi zawierać jedynie opis podziału strony, nazwy i opis ramek, a także nazwy dokumentów, które jako pierwsze mają być w tych ramkach wyświetlone. Osobno dla każdej ramki trzeba zdefiniować jeden lub więcej dokumentów, które mają się w tych ramkach pojawiać.

Najprostszą tego rodzaju sytuacją a zarazem najczęściej spotykaną jest podzielenie ekranu na dwie sąsiadujące obok siebie ramki i umieszczenie w jednej (np. lewej) **menu** pozwalającego wybierać interesujące nas tematy (jak na Rys. 2.10).

Po każdym kliknięciu wybranego w menu tematu w drugiej ramce (prawej) powinien pojawić się dokument odpowiadający temu tematowi, przy czym menu ma pozostać dalej w (lewej) ramce.

Załóżmy, że chcemy utworzyć strony z ramkami dotyczące trzech największych miast Polski. W menu wyświetlanym w lewej ramce będą więc trzy pozycje: „Warszawa", „Kraków", „Łódź". W prawej ramce powinny pojawiać się informacje o wybranym mieście. Przykładowy ekran wykorzystujący takie dwie ramki pokazuje Rys. 2.10.

Dla wszystkich plików dotyczących naszego projektu warto utworzyć osobny folder np.: o nazwie MIASTA. Jak wspomniano, konieczne będzie utworzenie nie tylko dokumentów przeznaczonych do wyświetlania w każdej z ramek, ale także **dokumentu głównego** zawierającego opis ramek. Niech ten główny dokument nazywa się „index.html".

Rys. 2.10. Przykład podziału okna przegladarki na dwie ramki

Treść tego dokumentu głównego (dla naszego przykładu) może być następująca:

```
<HTML> <! PR 2: index.html>
<HEAD>
   <meta http-equiv="Content-Type" content="text/html;
    charset=windows-1250">
   <TITLE>Miasta</TITLE>
</HEAD>
<frameset cols="12%,*">
    <frame name="wybor" src="wykaz.htm" scrolling="no"
     marginheight="0" marginwidth="0" frameborder=0>
    <frame name="tresc" src="warszawa.htm"
     scrolling="auto" marginheight="10" marginwidth="10">
</frameset>
</HTML>
```

Definicja układu ramek (w powyższym przykładzie) rozpoczyna się od klauzuli „*frameset*" a dalej „*cols*=" sygnalizuje, że podział okna definiowany jest przez szerokość wynikowych kolumn.

W naszym przypadku pierwsza kolumna ma zajmować 12% szerokości okna, a gwiazdka umieszczona po przecinku oznacza, że druga kolumna zajmuje resztę szerokości okna.

Cechy (własności) każdej z ramek podane są w definicji rozpoczynającej się słowem *frame*. Niektóre istotne cechy to:

name	-	nazwa ramki
src	-	nazwa wyświetlanego dokumentu
scrolling	-	możliwość przewijania zawartości („*yes*" lub „*no*")
marginheight	-	wysokość marginesu
marginwidth	-	szerokość marginesu
frameborder	-	szerokość obramowania
noresize	-	polecenie uniemożliwiające zmianę rozmiarów myszką

Dokument *„wykaz.htm"* - zawierający menu i przeznaczony dla lewej ramki o nazwie *„wybor"* - może w najprostszym przypadku wyglądać następująco:

```
<HTML> <! PR 3: wykaz.htm>
<HEAD>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1250">
<TITLE>Menu</TITLE>
</HEAD>
<BODY>
<Center>
<p>Największe<br />miasta<br />Polski</p>
<p><a href= "warszawa.htm" target="tresc">Warszawa</a></p>
<p><a href= "krakow.htm" target="tresc">Kraków</a></p>
<p><a href= "lodz.htm" target="tresc">Łódź</a></p>
</Center>
</BODY>
</HTML>
```

Istotną nowość w tym dokumencie stanowi (wewnątrz odsyłaczy) komenda **target** a dokładniej **target=**"tresc", dzięki której treść odsyłacza pojawi się nie w bieżącej ramce lecz w ramce o podanej nazwie – w tym przypadku o nazwie „tresc".

Dla prawej ramki trzeba przygotować tyle dokumentów z opisami miast ile pozycji zawiera nasze menu, a więc przygotujemy dokumenty: *warszawa.htm, krakow.htm, lodz.htm*.

Treść dokumentu *warszawa.htm* odpowiadającego na Rys. 2.10 zawartości prawej ramki jest następująca:

```
<HTML> <! PR 4: warszawa.htm>
<HEAD>
<META HTTP-EQUIV="Content-type" CONTENT="text/html;
charset=windows-1250">
<TITLE>Warszawa</TITLE>
</HEAD>
```

```
<BODY bgcolor="#FFFFAA">
<table border=0>
<tr> <td width="40%">
<center> <h2>Warszawa<br>
<img src="w-wa_herb.jpg" alt="(4kB)" border=0 height=72
width=48></h2>
</center>
<img src="warsaw.jpg" alt="(107kB)" border=0 width=100%>
</td><td >
<p>
Warszawa - stolica i największe miasto Polski, leży w środkowo-wschodniej części kraju, na
Mazowszu nad Wisłą.
</p><p>
Warszawa jest największym polskim miastem pod względem liczby ludności (1 714 446
zameldowanych mieszkańców wg stanu z 31 grudnia 2009 r.) i powierzchni (517,24 km2).
</p><p>
Dużą część mieszkańców stanowi ludność niezameldowana – szacowana całkowita liczba
mieszkańców spędzająca noc w mieście wynosi w przybliżeniu 1,91 – 1,96 mln osób.
Ponadto codziennie ok. 500 tys. osób dojeżdża do Warszawy do pracy, z czego najwięcej z
Łodzi (ok. 150 tys.) i Radomia (ok. 50 tys.). Po uwzględniuniu tej liczby otrzymujemy
szacunkową wielkość 2,41-2,46 mln osób przebywających w granicach miasta w ciągu dnia
</p><p>
Warszawa jest ośrodkiem naukowym, kulturalnym, politycznym oraz gospodarczym.
Mieszczą się w niej siedziby parlamentu, Prezydenta RP, Rady Ministrów i innych władz
centralnych. Jest także stolicą województwa mazowieckiego.</p>
</td></tr></table>
</BODY> </HTML>
```

Zawartość pozostałych dwu dokumentów: *krakow.htm* oraz *lodz.htm* pozostawiam inwencji czytelnika.

W razie potrzeby można używać specjalnych wartości atrybutu target, a mianowicie:

`target="_blank"` – nowy dokument zostanie załadowany do nowego, pustego okna;

`target="_self"` – nowy dokument zamieni w ramce bieżący dokument;

`target="_top"` – usuwa ramki i wyświetla nowy dokument bez ramek.

2.5. DYNAMICZNE STRONY WWW

Język HTML pozwala opisać strukturę i wygląd **statycznych stron WWW** – o ustalonej treści i wyglądzie. Jeśli jednak chcemy uzyskać **dynamiczne strony WWW** – których elementy będą generowane na podstawie działań użytkownika wykorzystując informacje uzyskiwane od użytkowników lub z baz danych, to język HTML do tego nie wystarczy.

Dla uzyskania stron dynamicznych trzeba dokumenty HTML uzupełniać wstawianiem bloków poleceń napisanych w innych językach. Bloki te nazywane są **skryptami**.

Strony tworzone dynamicznie mogą na przykład wyświetlać wyniki wyszukania w bazie danych, takiej choćby jak internetowa książka telefoniczna czy rozkład jazdy pociągów.

Przy każdym przeglądaniu konkretnej strony internetowej mamy do czynienia z dwoma komputerami: tym przy którym siedzimy i tym na drugim końcu łącza czyli serwerem.

Istnieją różne języki programowania używane do pisania skryptów osadzanych w plikach HTML. Języki te - czyli **języki skryptowe** - dzielą się na dwie kategorie w zależności od tego czy skrypty pisane w danym języku wykonywane są na serwerze czy na komputerze klienta, a mianowicie języki w których skrypty są **wykonywane**:

a) **po stronie serwera** – strona dynamiczna jest generowana przed wysłaniem do klienta, serwer musi w tym celu współpracować z interpreterem języka skryptowego (np.:PHP)

b) **po stronie klienta** – dopiero po pobraniu pliku z serwera, na naszym komputerze generowana jest strona dynamiczna

W przypadku działań po stronie serwera, dokument na serwerze zawiera skrypty napisane najczęściej w języku PHP albo innym (np.: ASP lub Perl) i będą one wykonywane na serwerze a do użytkownika wysyłany jedynie ich wynik w postaci wygenerowanego kodu HTML. A więc użytkownik nie może wówczas oglądnąć treści tych wstawek. Jak widać, w tym przypadku nie ma ryzyka, że program coś niedobrego zdziała na naszym komputerze.

W drugim przypadku, gdy podglądniemy postać źródłową pobranego z serwera do przeglądarki pliku, zobaczymy opis strony w języku HTML a wewnątrz niego wstawione bloki programów – przeważnie w języku JavaScript (czasem także VBScript). Bloki te mają odpowiednio zaznaczone początki i końce, tak aby przeglądarka mogła wysłać je do interpretera zastosowanego języka (musi on być zainstalowany na naszym komputerze w pakiecie z przeglądarką). Interpreter wykona odpowiednie operacje i zwróci wyniki do przeglądarki.

2.5.1. SKRYPTY I MAKRA. JĘZYKI SKRYPTOWE

Słowo ***script*** oznacza w języku angielskim **scenariusz**. **Językiem skryptowym** nazywany jest język osadzony wewnątrz jakiejś aplikacji (programu użytkowego), który pozwala zastąpić zwykły tryb wydawania poleceń dla tej aplikacji przez uruchomienie wykonywania określonego, wcześniej przygotowanego scenariusza czyli **skryptu** zawierającego zbiór poleceń a ściślej mówiąc program.

Pojęcie „skrypt” jest zbliżone do pojęcia „makro” choć nie ma precyzyjnych definicji tych pojęć. Tworzenie „makra” polega na włączeniu rejestrowania operacji i poleceń wydawanych przez użytkownika a w tym naciśnięć klawiszy i kliknięć myszką. Makro jest więc zbiorem poleceń dla danej aplikacji, zapamiętanym, z możliwością wielokrotnego wywoływania (przy pomocy nadanej nazwy lub przypisanej kombinacji klawiszy) a następnie odtwarzania. W tym znaczeniu makra ułatwiają wykonywanie pewnych operacji jednak nie rozszerzając zastosowań danej aplikacji.

Skrypt jest natomiast samodzielnym programem, napisanym w języku skryptowym zagnieżdżonym w danej aplikacji, może więc rozszerzać obszar jej zastosowań.

Przykładem aplikacji wyposażonej w język skryptowy jest pakiet Microsoft Office w którym można używać języka VBA (Visual BASIC for Applications), a także AutoCAD wyposażony w język AutoLisp.

Do tworzenia dynamicznych stron WWW wykorzystywane są m.in. takie języki jak: Java Script, VB Script, PHP, Perl, Python i Ruby.

Języki skryptowe są osadzane także w grach komputerowych, gdzie mogą służyć do sterowania przebiegiem gry.

Coraz częściej języki skryptowe są kompilowane do binarnego kodu pośredniego, który jest wykonywany w interpreterze zwanym „maszyną wirtualną", dużo szybciej niż przebiegałaby jego interpretacja bezpośrednio, z tekstowej postaci źródłowej.

2.5.2. CHARAKTERYSTYKA JĘZYKA PHP

PHP [25], [27] jest obiektowym językiem programowania służącym najczęściej do generowania dynamicznych stron internetowych. Pod względem składni PHP jest podobny do języka C lecz wykazuje szereg uproszczeń. Na przykład typy zmiennych (podobnie jak w MATLAB-ie) określane są automatycznie - na podstawie przypisywanych im wartości - i nie trzeba ich deklarować. Z poziomu PHP można też obsługiwać bazy danych.

Jak wspomniano, dla uzyskania z pomocą PHP dynamicznych stron WWW, do ich opisów w języku HTML muszą zostać wstawione odpowiednie skrypty w języku PHP.

Plik HTML zawierający skrypty PHP musi być zapisany z rozszerzeniem „.php" (czasem: „.php3" lub „.php4"), a serwer na którym będą przechowywane takie pliki musi być wyposażony w program zwany „parserem PHP" i odpowiednio skonfigurowany do współpracy z tym parserem. Jeśli użytkownik kliknie w przeglądarce odsyłacz do pliku z rozszerzeniem php, wówczas ten plik na serwerze jest najpierw przetwarzany przez parser PHP a dopiero potem wysyłany do przeglądarki użytkownika.

Znaczniki początku i końca skryptu to: <?php *... skrypt ...* **?>**

Parser PHP ma 2 tryby pracy: HTML lub PHP. Początkowo jest w trybie HTML – nie zmieniając treści pliku – aż do napotkania znacznika „**<?php**" informującego o konieczności przełączenia się do trybu PHP i realizowania poleceń tego języka. Znacznik „**?>**" kończy zestaw poleceń PHP i powoduje ponowne przełączenie parsera do trybu HTML.

Zmienne: **jak w każdym języku programowania - zmienna jest pojemnikiem przechowującym wartość określonego typu i dostępnym poprzez swą nazwę.**

W języku PHP **nazwy zmiennych** muszą zaczynać się od znaku dolara „**$**" a następnie musi wystąpić litera lub podkreślnik „_". Kolejnymi znakami mogą być litery lub cyfry oraz znaki o kodzie ASCII powyżej 127. Rozróżniane są duże i małe litery (jak w językach C i MATLAB). Nie jest konieczne deklarowanie typów zmiennych gdyż zmienna jest inicjalizowana (to znaczy rezerwowany jest dla niej odpowiedni obszar pamięci) przy pierwszym nadawaniu jej wartości. Przykład: **$sila = 23.5;**

Tablica - to zmienna złożona przechowująca zbiór wartości do których dostęp mamy przez nazwę tablicy oraz indeks (numer) elementu tablicy.

Na przykład: $a[5] oznacza odwołanie do piątego elementu tablicy $a. Jak widać **indeksy elementów tablic umieszczane są w nawiasach prostokątnych**. Najmniejsza wartośc indeksu to zero. Aby wstawiać wartości do elementów tablicy można podawać ich indeksy np.:

$a***[0] = 12.5; ***$a[1] = 0.657; ***$a***[2] = 78.2;

jednak definiując kolejny element na końcu tablicy można także pomijać indeks zostawiając nawiasy prostokątne puste:

$a***[ ] = 16.1; ***$a[ ] = 3.42; ...

Tablice o większej liczbie wymiarów to „tablice tablic", czyli element tablicy wyższego rzędu zawiera tablicę niższego rzędu. Oprócz zwykłych tablic, w PHP można stosować tablice asocjacyjne, w których zamiast indeksów używa się nazw pól tablicy – pełniącej role rekordu bazy danych.

Do przekazywania wartości zmiennych z formularzy HTML do skryptów PHP służą tak zwane **tablice superglobalne** o nazwach: **$_GET** i **$_POST** – zależnie od użytej w formularzu metody GET lub POST. Użycie tablicy $_POST pokazano dalej na przykladzie.

Najprostszym i najczęściej stosowanym **poleceniem wyprowadzania wartości i tekstów** do pliku wynikowego jest użycie funkcji **echo**(...). Wyprowadzane przez tą funkcję teksty muszą być ujmowane w cudzysłowy lub apostrofy. W przypadku wyprowadzania tekstów będących poleceniami języka HTML, w których mogą występować cudzysłowy, wygodniej jest całość ująć w apostrofy np.:

```
<?php
echo('<img src="kwiatek.gif">')
?>
```

- przekopiuje do pliku wynikowego odsyłacz do obrazka "kwiatek.gif". Można jedw ramach cudzysłowów wyprowadzać także cudzysłowy ale poprzedzając je znakiem „backslash" czyli "\":

```
<?php
echo "<img src=\"kwiatek.gif\">"
?>
```

Polecenie „echo" jest naprawdę tak zwaną „niby-funkcją" gdyż nawiasy okrągłe – niezbędne dla funkcji – mogą być w tym poleceniu pominięte, jak to pokazano powyżej.

Role apostrofów i cudzysłowów nie są identyczne. W przypadku **cudzysłowów** zmienne zawarte między nimi są zamieniane na ich **wartość**, a w przypadku **apostrofów** zmienna pozostaje jako nazwa:

```
<! PR 5: polecenia przypisania i wyprowadzania>
<?php
   $zm1 = 5; // Zmiennej "$zm1" przypisano wartość 5
   $zm2 = "Ala"; // Zmiennej "$zm2" przypisano napis „Ala"
   $zm3 = $zm1; // Zmiennej "$zm3" przypisano wartość $zm1
   echo "To jest $zm2"; // wyświetli się napis "To jest Ala"
```

```
   echo '$zm2'; // wyświetli się napis "$zm2"
   echo $zm1; // wyświetli się cyfra 5
?>
```

Oddzielanie poleceń: każde polecenie musi kończyć się znakiem średnika „ **;**".

Komentarze: zamykane są między znakami: **/*** ... ***/**. Krótki komentarz na końcu linii można poprzedzić podwójnym slashem: „**//**".

Instrukcja warunkowa IF ma budową taką jak w języku C:

```
if (warunek)
{ instrukcje, które będą wykonane gdy warunek będzie spełniony }
else
{ instrukcje, które będą wykonane gdy warunek nie będzie spełniony }
```

Ważniejsze operatory:

=	- przypisanie wartości	**>**	- większy	**&&**	- koniunkcja (i)
+	- dodawanie	**<**	- mniejszy	**||**	- alternatywa (lub)
−	- odejmowanie	**>=**	- większy lub równy	**!**	- negacja (nie)
*****	- mnożenie	**<=**	- mniejszy lub równy	**.**	- sklejanie ciągów znaków
/	- dzielenie	**==**	- równy	**++**	**$a++;** *jak* **$a=$a+1;**
%	- reszta z dzielenia	**!=**	- nie równy	**−−**	**$a−−;** *jak* **$a=$a−1;**

Instrukcje pętli także mają budową zapożyczoną z języka C, na przykład **pętla FOR**, która ma wyprowadzić wartości zmiennej **$indeks** od zera do 9 ma budowę:

```
for($indeks= 0;  $indeks<10;  $indeks++)
{ echo "$indeks";
}
```

Pętla WHILE wykonująca analogiczne działania ma budowę:

```
$indeks=0;
do
{ echo "$indeks"; $indeks++;
}
while($indeks<10)
```

2.5.3. FORMULARZE HTML

Strony internetowe przekazują zapisane na serwerach informacje dla swoich czytelników. Może być jednak potrzebny przepływ informacji w odwrotną stronę - od czytelników do serwera i do autorów lub inicjatorów stron internetowych. Takie sytuacje występują na przykład przy zamawianiu książek czy towarów w sklepach internetowych, wypełnianiu ankiet lub zgłoszeń na konferencje. Przesyłanie informacji do serwera umożliwiają strony internetowe utworzone w języku HTML z wykorzystaniem **formularzy** [22], [23] zawierających elementy dialogowe jak pola tekstowe, przyciski, listy rozwijalne, i inne - zwane potocznie „kontrolkami" (od angielskiego określenia *controls*).

Opis formularza w języku HTML musi być zawarty między parą znaczników:

<form ...*atrybuty ...* **> .... </form>**

Standard ***(X)HTML Strict*** wymaga aby opisy „kontrolek" formularza były dodatkowo umieszczone wewnątrz pary innych znaczników stanowiących pojemnik jak np.: ** .. , .. ,** czy **
...**

Znacznik **<form>,** tak jak inne znaczniki, może posiadać szereg atrybutów a w szczególności:

method="*metoda wysyłania*"	–	czyli słowo "GET" lub "POST"
action="*adres docelowy*"	–	wyrażenie określające gdzie mają być wysłane dane
enctype = "*typ kodowania*"	–	sposób kodowania

A oto przykład użycia znacznika **<form>**:

```
<form action = "mailto: kowalski@wp.pl" method="post">
<div>
        ....
        opis kontrolek
        ....
</div>
</form>
```

Atrybut `method` określa metodę przesyłania danych. Do wyboru mamy metodę GET lub metodę POST. Metoda **POST** pozwala wysłać dane z formularza do adresata poczty elektronicznej lub do skryptu na serwerze, który przetworzy te dane. Przesyłane dane nie są widoczne. Podobnie jak w poczcie elektronicznej można przesyłać także pliki.

W metodzie **GET** dane z formularza nie mogą być przesyłane pocztą elektroniczną lecz zostają doklejone do adresu http strony docelowej, a więc są widoczne, a poza tym mają ograniczenie co do długości bo adres wraz z danymi może mieć najwyżej 255 znaków. Jeśli na przykład formularz zawierał tylko pola z imieniem i nazwiskiem to nazwy i wartości tych pól zostaną do adresu doklejone po znaku pytajnika i oddzielone od siebie znakiem &.

Na przykład:

`http://www.wp.pl/wyniki.php`**`?imie=Ewa&nazwisko=Nowak`**

Atrybut `action` określa miejsce gdzie dane mają być wysłane. Dla metody POST może to być to adres e-mail np.:

action = "mailto: kowalski@wp.pl"

W przypadku obu metod może to być także adres bezwzględny strony odbierającej dane na przykład:

action = "http://www.wp.pl/wyniki.php"

lub adres względny - jeśli strona jest na tym samym serwerze co formularz np.:

action = "../wyniki.php"

Atrybut `enctype` określa typ kodowania i ma znaczenie jedynie, przy metodzie POST.

Domyślnie jego wartością jest **"application/x-www-form-urlencoded"** i taki też typ stosowany jest dla metody GET. W przypadku przekazywania pliku na serwer powinien być stosowany typ: **"multipart/form-data"**, natomiast przy przesyłaniu pocztą elektroniczną korzystne może być ustawienie zwykłego tekstu czyli typu: **"text/plain"**. Inne atrybuty znacznika **<form>** opisano m.in. w [22].

Do definiowania „kontrolek" formularza w języku HTML służy znacznik **<input..>**, z odpowiednimi atrybutami. Najważniejsze atrybuty tego znacznika to: **type, name, value.**

Atrybut **type** określa **rodzaj kontrolki** jaka ma powstać, atrybut **name** przypisuje kontrolce nazwę traktowaną jako **nazwa zmiennej** wynikowej, natomiast atrybutem **value** można przypisać tej zmiennej **domyślną wartość**.

Typy kontrolek, które można umieszczać w formularzu, podano poniżej.

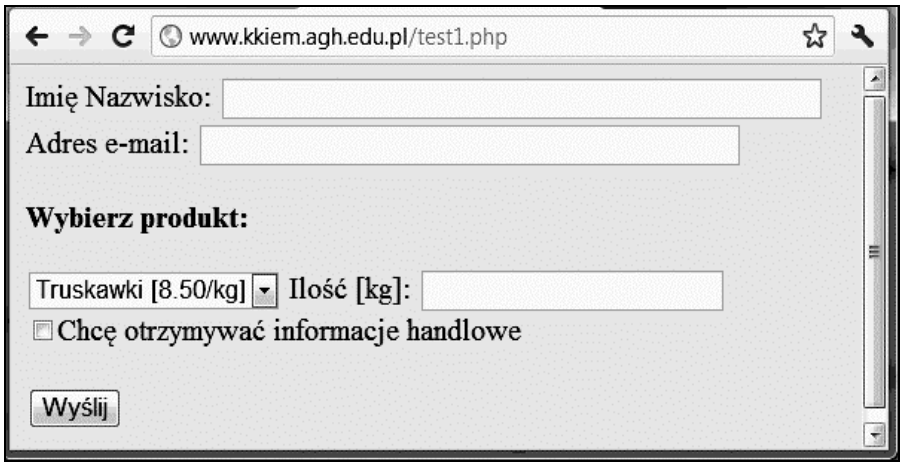
Wartość atrybutu type:		**Rodzaj „kontrolki":**
type="button"	-	klasyczny przycisk
type="checkbox"	-	wybór wielu z wielu opcji,
type="hidden"	-	element ukryty,
type="password"	-	pole do wpisywania haseł,
type="radio"	-	pole wyboru jednej z wielu opcji
type="reset"	-	przycisk „reset" czyli „od początku"
type="select"	-	lista wyboru
type="submit"	-	przycisk „submit" czyli „wyślij"
type="text"	-	pole tekstowe,
type="textarea"	-	rozszerzone pole tekstowe

Na konkretnym przykładzie pokazano poniżej możliwości zastosowania kilku istotnych typów kontrolek. Załóżmy, że potrzebujemy formularza w którym klient wpisze swoje imię i nazwisko i adres e-mail, a następnie wybierze z listy określony towar. Dodatkowo będzie mógł zaznaczyć (lub nie) pole opcji dotyczące zgody na przysyłanie informacji handlowych. Całość musi klient zatwierdzić i wysłać przyciskiem na którym będzie napis „Wyślij".

Utworzymy plik o nazwie na przykład: **test1.php**, który na początek będzie zawierał tylko projekt formularza o następującej treści:

```
<html lang="pl"> <! PR 6: test1.php>
<head>
  <meta http-equiv="Content-Type" content="text/html;
charset=Windows-1250">
  <title>Formularz</title>
</head>
<body bgcolor="yellow">
<form action="" method="post">
 <div>
  Imię Nazwisko:
  <input name="im_naz" value="" size="45" /><br />
```

```
  Adres e-mail:
  <input name="email" value="" size="40" /><br />
  <strong>Wybierz produkt:</strong>
 <select name="owoce">
   <option value="8.50"> Truskawki [8.50/kg]</option>
   <option value="9.00">Maliny [9.00/kg]</option>
   <option value="9.50">Poziomki [9.50/kg]</option>
   <option value="4.50">Wiśnie [4.50/kg]</option>
   <option value="3.50">Śliwki [3.50/kg]</option>
 </select>
Ilość [kg]:<input name="ilosc" value="" /><br />
  <input type="checkbox" value="checked" name="mailing" />Chcę
otrzymywać informacje handlowe<br /><br />
  <input type="submit" value="Wyślij" name="wyslano" />
 </div>
</form>

<?php
// ..... tu będzie wstawiony skrypt PHP
?>
</body>
</html>
```



Rys. 2.11. Przykładowy formularz

Po wysłaniu na serwer katedry i wyświetleniu w przeglądarce WWW formularz będzie wyglądał tak jak pokazano na rysunku Rys. 2.11.

2.5.4. PRZYKŁAD OBSŁUGI FORMULARZA W JĘZYKU PHP

Po wysłaniu danych ze skonstruowanego powyżej przykładowego formularza chcemy te dane wykorzystać, planując odpowiednie ich użycie i realizując ich obsługę przy pomocy skryptów w języku PHP.

Najpierw trzeba sprawdzić – przy pomocy funkcji ***isset(..)*** - czy dane już dotarły, to znaczy, czy zmienne przypisane „kontrolkom" formularza znajdują się już w tablicy superglobalnej $_POST. Następnie sprawdzamy – przy pomocy funkcji ***empty(..)*** - czy wszystkie pola formularza wypełniono. Jeśli nie to wartości niektórych zmiennych będą „puste" – wówczas trzeba wyświetlić odpowiedni komunikat: „Wypełnij wszystkie pola" oraz wyświetlić odsyłacz powrotu do pustego formularza.

Jeśli natomiast pola zostały wypełnione to ma być wyświetlony napis: „Dziękujemy za zgłoszenie" a pod spodem podana kwota do zapłaty.

Czynności te zrealizuje następujący skrypt wstawiony w miejscu zaznaczonym na poprzednim wydruku:

```
<! PR 7: skrypt PHP do obsługi formularza>
<?php
if (isset($_POST["wyslano"]))
{
 // - sprawdzamy, czy wszystkie pola zostały wypełnione:
  if (empty($_POST["im_naz"])||
       empty($_POST["email"]) ||
       empty($_POST["ilosc"]) )
  {echo "<p style=\"color:red\">Wypełnij wszystkie pola!</p>";
   echo "<p><a href=\"test1.php\">Powrót do formularza</a></p>";
  }else
  {echo "<h3>Dziękujemy za zgłoszenie! </h3>";
   $koszt=$_POST["owoce"]*$_POST["ilosc"];
   echo "<p>Do zapłaty: $koszt zł</p>"; }
}
?>
```

Zauważmy, że wewnątrz tekstów ujętych w cudzysłowy znajdują się kolejne teksty w cudzysłowach ale poprzedzanych znakiem „backslash" co umożliwia potraktowanie ich jako znaków specjalnych do wyprowadzenia.

Czytelnik może samodzielnie przetestować działanie tego bardzo prostego i niedoskonałego, przykładowego skryptu *test1.php* na stronie Katedry Konstrukcji i Eksploatacji Maszyn AGH (*www.kkiem.agh.edu.pl*), jak również innego, znacznie bardziej złożonego skryptu *matlab_test1.php*, testującego studencką znajomość elementów MATLAB-a.

Język PHP jest dość trudny a dodatkowe komplikacje wprowadza jego powiązanie z innymi językami (HTML, SQL). Dlatego nie ma miejsca aby w ramach jednego semestru poświęcić mu więcej uwagi.

Zainteresowanych odsyłam do literatury i samouczków w Internecie.

Tworzenie stron internetowych z użyciem PHP i SQL znacznie ułatwiają systemy zarządzania treścią (CMS wspomniane w p.2.3.) oraz systemy typu RAD (*Rapid Application Development*) jak na przykład TurboPHP (*www.turbophp.com*).

2.6. CSS – KASKADOWE ARKUSZE STYLÓW

Styl - podobnie jak w edytorach tekstu - pozwala pod jedną nazwą zgromadzić zespół cech (i poleceń) dotyczących formatowania tekstów, akapitów i stron – w tym przypadku stron WWW wyświetlanych w przeglądarkach.

Kaskadowe arkusze stylów (**CSS** - *Cascading Style Sheets*) stanowią ważne uzupełnienie współczesnych wersji języka HTML (a także XHTML). Ponieważ specyfikacja CSS opublikowana jest niezależnie od specyfikacji HTML więc CSS traktowany jest jako osobny język.

Jakie powinny być role obu tych języków? Otóż:

- HTML powinien - oprócz treści stron - określać ich strukturę czyli znaczenie poszczególnych elementów strony (np.: co jest nagłówkiem rozdziału czy podrozdziału, co akapitem a co elementem listy wypunktowanej),
- CSS ma służyć do definiowania formatu czcionek i innych szczegółów dotyczących **sposobu prezentowania strony WWW** w przeglądarce.

Dzięki wprowadzeniu stylów CSS, wszystkie polecenia dotyczące formatowania można umieścić w jednym miejscu (tzw. arkuszu) i powiązać je z konkretnymi elementami, wstawionymi za pomocą czystego (X)HTML. Taka koncepcja sprawia, że modyfikacja wyglądu stron może przebiegać dużo sprawniej.

Można wprawdzie stworzyć witrynę, używając jedynie czystego HTML, jednak style pomogą ją wzbogacić, oraz w łatwy sposób zmieniać sposoby jej prezentacji i dostosowywać do różnych urządzeń wyświetlających, przykładowo także do telefonów komórkowych. Jeżeli konkretna przeglądarka nie obsługuje stylów, to po prostu je pominie.

Można korzystać z wielu sposobów definiowania stylów na różnych poziomach, a mianowicie [20]:

a) Najniższy poziom stanowi definiowanie stylu „inline" czyli w tej samej linii w której znajduje się formatowany element, na przykład:

```
<p style="color: red">Ten tekst będzie wyświetlony na czerwono</p>
```

Ogólna postać takiej definicji stylu jest następująca:

<selektor style="cecha1:w1; cecha2:w2..."> *napis* </selektor>,

gdzie: **selektor** to dowolny znacznik np.: **p** (akapit), **h1** (tytuł), **td** (komórka tabeli), **w1, w2,** ... – to wartości cech

Większe fragmenty tekstu mogą być formatowane przez style definiowane w blokach **span** lub **div**, które samodzielnie nie wnoszą żadnego formatowania:

```
<span style="cecha: wartość; cecha2: wartość2..."> ... </span>
<div style="cecha: wartość; cecha2: wartość2..."> ... </div>
```

b) Wyższym poziomem jest zastosowanie **wewnętrznego arkusza stylów** umieszczonego w części nagłówkowej dokumentu - pomiędzy znacznikami **<head>** i **</head>**.

Ogólna postać takiego wewnętrznego arkusza stylów jest następująca:

```
<head> <! PR 8: wewn. arkusz stylów>
    ...
<style type="text/css">
/* <![CDATA[ */
    selektor{cecha: wartość; cecha2: wartość2... }
    selektor2{cecha: wartość; cecha2: wartość2... }
    ...
/* ]]> */
</style>
    ...
</head>
```

Taki sposób definiowania może być przydatny szczególnie wtedy, gdy elementy, które pragniemy poddać formatowaniu, występują wielokrotnie w dokumencie.

Na przykład jeśli chcemy aby wszystkie nagłówki poziomu 3 miały kolor czerwony to wystarczy w bloku nagłówkowym strony umieścić następujący wewnętrzny arkusz stylów:

```
<! PR 9: przykład wewn. arkusza stylów>
<style type="text/css">
/* <![CDATA[ */
h3{color: red }
/* ]]> */
</style>
```

c) Trzeci poziom polega na stosowaniu **zewnętrznych arkuszy stylów** – definiowanych w osobnych plikach z rozszerzeniem „.css". Ten sposób daje najwięcej możliwości, pozwala bowiem definiować format określonych elementów na wielu stronach jednocześnie. A więc, dzięki arkuszowi zewnętrznemu, wszystkie strony w obrębie całego serwisu mogą mieć jednolite cechy formatowania. Zmiany formatowania w wielu dokumentach można również łatwo i szybko uzyskać modyfikując jedynie zewnętrzny arkusz stylów, bez konieczności zmiany każdej strony osobno.
Jeśli dokument ma korzystać z zewnętrznego arkusza stylów, zdefiniowanego w pliku „*style.css*", to musi mieć w bloku nagłówkowym polecenie dołączające taki arkusz:

```
<link rel="Stylesheet" type="text/css" href="style.css" />
```

W pojedynczym dokumencie można korzystać z wielu zewnętrznych arkuszy stylów i wówczas musi być wiele poleceń **<link rel="Stylesheet" . . . />**.
W przypadku konfliktów ważniejsze będą deklaracje z arkusza dołączonego później.
Zewnętrzny arkusz stylów budowany jest z takich samych deklaracji, dotyczących poszczególnych selektorów, jak arkusz wewnętrzny. Do jego napisania wystarczy najprostszy edytor tekstu ale można też posłużyć się specjalnym edytorem CSS. Jednym z darmowych edytorów tego rodzaju jest „Balthisar Cascade".

Można także definiować kilka alternatywnych arkuszy stylów, co spowoduje, że w przeglądarce pojawi się możliwość wyboru jednej z możliwych postaci strony, np.:

```
<! PR 10: alternatywne arkusze stylów>
<link rel="Alternate stylesheet" type="text/css"
      href="style1.css" title="Nazwa 1" />

<link rel="Alternate stylesheet" type="text/css"
      href="style2.css" title="Nazwa 2" />
```

Kaskadowość arkuszy stylów określa priorytety ich ważności, a mianowicie definicje stylów z arkuszy zewnętrznych mogą być modyfikowane przez ważniejsze od nich style zapisane w nagłówku dokumentu, a te z kolei mogą być zamieniane przez najważniejsze style zdefiniowane bezpośrednio w ciele dokumentu (*inline*).

2.7. ĆWICZENIA – TWORZENIE STRONY WWW

2.7.1. CEL ĆWICZEŃ

Celem niniejszych ćwiczeń jest poznanie podstawowych **zasad budowy dokumentów WWW** oraz najważniejszych **elementów języka HTML** (*HyperText Markup Language*) i zastosowanie ich przy opracowaniu własnej witryny internetowej poświęconej wybranemu tematowi. W trakcie ćwiczeń student ma przećwiczyć i utrwalić:

1. znajomość:
   - ogólnej struktury dokumentu WWW w języku HTML,
   - elementów bloku HEAD (tytuł, deklaracja tematyki i polskich liter),
   - atrybutów tła (kolor, obraz) ,
   - stylów nagłówkowych **<h1> ...</h1>, <h2> ...</h2>** i in.,
   - akapitów ** ... ** i zmian wiersza **
**,
   - formatowania czcionek (wytłuszczenia, pochylenia i in.),
   - formatowania akapitów (wyrównanie do środka, do prawej i in.),
   - list numerowanych i wypunktowanych,
   - odsyłaczy do innych dokumentów,
   - wstawiania obrazków,
   - tworzenia i formatowania tabel;
2. umiejętność **wyszukiwania w Internecie** i otwierania dokumentów HTML w osobnych oknach lub kartach;
3. umiejętność **pracy z wieloma otwartymi aplikacjami** i kopiowania przez „schowek" adresów i tytułów znalezionych stron WWW do „Notatnika";

4. umiejętność wyświetlania i modyfikowania źródłowej postaci wzorcowego dokumentu HTML zgodnie z swoimi potrzebami (i uwzględnieniem elementów jak w p.1) oraz sprawdzania efektów w przeglądarce.

2.7.2. PRZEBIEG ĆWICZEŃ

W trakcie ćwiczeń używany będzie program „Notatnik" z grupy Akcesoria w Ms Windows oraz przeglądarka Internetu (Internet Explorer, Mozilla Firefox lub inna).

Będziesz pracować z kilkoma otwartymi oknami aplikacji a w szczególności:

- źródłową postacią strony wzorcowej (w NOTATNIKU),
- widokiem tej strony w przeglądarce (Rys. 2.12),
- innymi oknami lub kartami przeglądarki, w których będziesz wyszukiwać pokrewne strony, aby dodać odsyłacze do nich na swojej stronie.

Rys. 2.12. Wzorcowa strona WWW przeznaczona do modyfikowania

1. **Utwórz swój folder** (na pulpicie) dla plików twojej strony www**.**

2. Zależnie od tego czy masz dostęp do Internetu wykonaj punkt (a) lub (b):

   a) Jeśli posiadasz dostęp do Internetu to **otwórz** w nowej karcie przeglądarki, z podanej (przez prowadzącego zajęcia) lokalizacji w Internecie, przykładowy **dokument o nazwie „wzorzec.htm”**. Pokaże się strona www taka jak na Rys. 2.12. Następnie **otwórz postać źródłową** czyli kod HTML tej strony z menu ***Widok – Źródło*** i przejrzyj jej treść uważnie przypominając sobie poznane wcześniej znaczenie poszczególnych znaczników HTML. Przekopiuj treść źródłowej postaci strony do Notatnika i **zapisz do swojego foldera z nazwą „wzorzec.htm”.**

   b) Jeśli nie masz dostępu internetowego do tej strony to przepisz w Notatniku zamieszczony poniżej kod HTML tej strony i zapisz do pliku o nazwie „wzorzec.htm”.

```
<! PR 11. KOD HTML DOKUMENTU „WZORZEC.HTM”>
<! wykrzyknikiem rozpoczynane są komentarze>
<HTML>
<head>
 <meta http-equiv="Content-Type" content="text/html;
 charset=windows-1250">
 <meta name="Author" content="Janka Kowalska">
 <meta name="Keywords" content="roses">
 <meta name="Description" content="Informacje o różach">
 <title="I like roses">
</head>

<BODY bgcolor="#ffccdd">
<center>
<H1><font color="red">RÓŻE – ULUBIONE KWIATY</font></h1>
<IMG SRC="rose1.jpg">
</center>
<font color="blue">
<p>Nazywam się Janka Kowalska</p>
<p>To jest moja witryna internetowa na której znajdziesz nieco
informacji o pięknych kwiatach jakimi są róże a także ich
fotografie.
<br>Poniżej są odsyłacze do stron o różach, osobno w języku
polskim i osobno w języku angielskim.</p> </font>
<p>
<! tabela rozmieszczona na środku>
<center>
<Table border=10 bgcolor="#ddffff" cellpadding=15>
 <tr>
<! pierwsza kolumna tabeli>
        <td>
Strony w języku polskim:
<OL>
   <LI> <A
HREF="http://www.obi.pl/pl/porady/garten/pflanzenaussen/Rosen/
">OBI-Ogród – róże</A>
```

```
   <LI> <A
HREF="http://klub.chip.pl/rybnickie_kwiaty/strony.html">
Kwiaty w Internecie </A>
   <li> <A HREF=" "> xxx </A>
   <LI> <A HREF=" "> xxx </A>
   <li> <A HREF=" "> xxx </A>
</OL>
        </td>
<! druga kolumna tabeli:>
        <td>
Strony w języku angielskim:
<UL>
   <LI> <A HREF="http://www.weeksroses.com/">
Weeks Roses</A>
   <LI> <A HREF=" "> ... </A>
   <li> <A HREF=" "> ... </A>
   <LI> <A HREF=" "> ... </A>
   <li> <A HREF=" "> ... </A>
</UL>
        </td>
 </tr> </Table>
</center>
<! ten znacznik poniżej tworzy poziomą linię>
<hr>
Opracowała: Janka Kowalska.
E-mail: <A HREF="mailto:.....@..... "> ......@...... </A>
<! wpisz w miejsce kropek swój adres e-mail 2 razy
- jako adres odsyłacza i drugi raz jako tekst odsyłacza>
</body> </html>
```

W kolejnych krokach masz dokument „wzorzec.htm" zmodyfikować i dostosować do innej wybranej tematyki oraz zapisać do swojego foldera z nazwą „index.html". Twoja witryna ma dotyczyć konkretnego tematu, na przykład dziedziny nauki, dyscypliny sportowej lub hobby, gatunku zwierząt czy roślin, pojazdu lub urządzenia, religii, filozofii czy idei, sławnej postaci, miasta, kraju, instytucji itp. itd.

3. Wybranie tematyki i ilustracji

Aby ułatwić Ci wybór tematyki i zapewnić oprawę graficzną zaczniemy od wyboru obrazka. Otwórz w nowej karcie przeglądarki jedną z podanych poniżej witryn z obrazkami (clipartami) lub wyszukaj w google.pl inne podobne:

- http://www.barrysclipart.com/
- http://www.free-clip-art.com/angel_clipart.shtml

Następnie **wybierz obrazek** odpowiedni do tematyki jakiej poświęcisz tworzoną stronę WWW, kliknij na nim prawym przyciskiem myszy i zapisz go do swojego foldera z odpowiednią nazwą. Jeśli nie znasz angielskiego to - przy okazji przeglądania danej kategorii ilustracji – zanotuj sobie angielską nazwę tej kategorii tematycznej, co ułatwi ci wyszukanie w Internecie stron związanych z tą tematyką.

4. Wyszukiwanie tematyczne w Internecie

Wybierz jedną z wyszukiwarek internetowych (ja polecam: www.google.pl) wpisując jej adres w przeglądarce (u góry). Po zgłoszeniu się strony wyszukiwawczej wpisz szukany temat (słowo kluczowe lub kilka słów). Z otrzymanego wykazu otwórz kilka witryn w nowych oknach (klikając prawym przyciskiem myszy).

5. Tworzenie odsyłaczy

Adresy i tytuły najciekawszych witryn przekopiuj do NOTATNIKA do odsyłaczy opracowywanej strony, pamiętając, o budowie odsyłaczy:

**** Tytuł strony ****

6. Modyfikowanie opracowywanej witryny

Zmodyfikuj odpowiednio treść strony, między innymi wpisując swoje nazwisko (zamiast Janki Kowalskiej) a także swój e-mail u dołu strony (w miejsce kropek).

Zmień nazwę obrazka w deklaracji <img ...> aby wywoływała ten wybrany przez ciebie obrazek.

7. Zapis przeróbek i odświeżenie widoku

Aby zobaczyć efekt tych modyfikacji musisz zapisać ponownie opracowywaną stronę do pliku (Plik-Zapisz) i nie zamykając NOTATNIKA odszukać swój folder gdzie zapisałeś i otworzyć w przeglądarce (np. przez podwójne kliknięcie), albo jeśli już jest tam otwarty to odświeżyć widok (na ogół klawiszem F5). Otworzy się poprawnie tylko wtedy jeśli plik ma rozszerzenie .HTM lub .HTML.

Następnie przeprowadź kolejne modyfikacje i eksperymenty zmieniając na przykład kolory, rozmieszczenie czy wielkość czcionek itp.

Pracując tak z otwartym i NOTATNIKiem i przeglądarką - z dwoma postaciami tej strony - po każdej modyfikacji aby zobaczyć efekt musisz:

a) zapisać w NOTATNIKU
b) odświeżyć w przeglądarce (z menu lub klawiszem F5)

Czasem – jeśli odświeżanie nie działa - wtedy trzeba z menu wejść w Opcje i usunąć pliki tymczasowe.

3. WPROWADZENIE DO ALGORYTMÓW I STRUKTUR DANYCH

3.1. ALGORYTM

ALGORYTM to jednoznaczny opis postępowania prowadzącego w skończonej liczbie kroków do uzyskania określonego celu, zawierający:

a) **opis obiektów** na których mają być dokonywane operacje oraz narzędzi służących do tych operacji;
b) **opis procedury** – składający się ze skończonej liczby jednoznacznie określonych poleceń jakie należy wykonać w określonej kolejności dla uzyskania celu.

Polecenia muszą być zrozumiałe i możliwe do realizacji dla wykonawcy algorytmu. Wykonawca musi znać zarówno **obiekty** na których ma działać jak używane **narzędzia** oraz stosowane **operacje**. Tak więc algorytmy np.: pieczenia placka, naprawiania spłuczki czy diagnozowania przyczyny awarii samochodu muszą być zrozumiałe dla ludzi, bo oni będą ich realizatorami.

W przypadku algorytmów przaznaczonych dla komputera obiektami będą **zmienne** reprezentujące **dane wejściowe** oraz **wyniki** pośrednie i końcowe, a także zmienne pomocnicze jak indeksy, flagi bufory, objaśnienia i inne. Co jednak znaczy wówczas powiedzenie, że algorytm ma być „*zrozumiały dla komputera*"?

Działaniem komputera steruje procesor na podstawie rozkazów mających składnię określoną przez projektanta tego typu procesorów. Polecenia algorytmu będą wiec „zrozumiałe" dla komputera jeśli zostaną zapisane **w języku programowania implementowanym na danym komputerze**. Mówiąc prościej, algorytm może być zapisany w postaci programu w dowolnym języku programowania pod warunkiem, że dysponujemy translatorem, który przetłumaczy ten program na rozkazy w kodzie procesora.

Jak już wspomniano - w odróżnieniu od jednorazowych obliczeń na kalkulatorze - polecenia algorytmu powinny być formułowane tak, aby mogły być **wykorzystywane wielokrotnie**, **dla różnych danych** oraz różnych sytuacji, które mogą zaistnieć (definiowanych stanami elementów dialogowych, urządzeń peryferyjnych, czujników, mierników i t.p). Nie jest bowiem sensowne i opłacalne konstruowanie algorytmu, który byłby wykonany tylko jeden raz - na przykład rozwiązując jeden jedyny układ równań.

Algorytm powinien więc **rozwiązywać dowolne zadanie z określonej klasy zadań**. Na przykład rozwiązywać dowolne równanie wielomianowe, czy wyznaczać sumę elementów dowolnego ciągu liczb, czy rozwiązywać dowolny układ równań liniowych i t.p.

Oczywiście w praktyce wprowadza się **ograniczenia** dotyczące tej dowolności, na przykład odnośnie maksymalnego stopnia wielomianu czy maksymalnej liczby równań w układzie, czy dokładności uzyskiwanych rozwiązań.

Przed rozpoczęciem konstruowania algorytmu należy więc sprecyzować:
(1) jaką klasę zadań ma on realizować oraz
(2) jakie przyjmujemy ograniczenia.

Dobry algorytm powinien kończyć działanie po wykonaniu skończonej – być może nawet bardzo dużej, ale **skończonej** - liczby kroków. Zasada ta obowiązuje również wtedy, gdy celu algorytmu nie udaje się osiągnąć. Brak tego warunku prowadzi do tego, że program wykonujący określone działania „zapętla się" i komputer liczy bez końca.

W algorytmach można wyróżnić **trzy podstawowe struktury** a mianowicie:
- sekwencje poleceń,
- rozgałęzienia,
- pętle.

Niezależnie od tych struktur zawsze **istotna jest kolejność wykonywania poleceń**. Na zajęciach z matematyki często zdarza się, że wykładowca pisze na tablicy wzór a dopiero potem objaśnia jego składniki lub podaje wartości użytych zmiennych. W algorytmach i programach nie jest to zazwyczaj dopuszczalne. Obowiązuje kolejność umożliwiająca komputerowi wykonywanie działań, a więc najpierw muszą być definiowane obiekty (zmienne i ich wartości) a dopiero potem mogą wystąpić wyrażenia obliczeniowe definiujące operacje na tych obiektach.

Rozgałęzienie w algorytmie lub progamie składa się z dwu alternatywnych sekwencji poleceń, przy czym program działając dla konkretnych danych wykona tylko jedną z tych sekwencji – zależnie od wartości logicznej pewnego wyrażenia decyzyjnego zwanego też warunkiem.

Pętla programowa polega na wielokrotnym powtarzaniu wykonywania pewnej sekwencji rozkazów, zapisanej w programie tylko jeden raz ale w odpowiedni sposób. Aby program nie działał jednak w nieskończoność - liczba cykli pętli powinna być albo z góry określona albo uzależniona od wartości konkretnego wyrażenia logicznego. Zgodnie z tym będziemy mówić o **dwu typach pętli**.

Wyrażenia logiczne stosowane w pętlach i rozgałęzieniach pozwalają sprawdzać czy nastąpiło spełnienie interesującego nas warunku dotyczącego wartości określonych zmiennych (np.: 3<a<8 i b>0) albo zaistnienia określonych zdarzeń (naciśnięcia klawisza, kliknięcia myszką, wykrycia końca pliku danych i t.d.). W relacjach sprawdzajacych równość dwu wartości, używany jest na ogół inny symbol od symbolu przypisywania wartości, na przykład w MATLAB-ie `x=5` przypisuje wartość 5 zmiennej x , natomiast `x==5` jest relacją sprawdzającą czy x ma wartość 5.

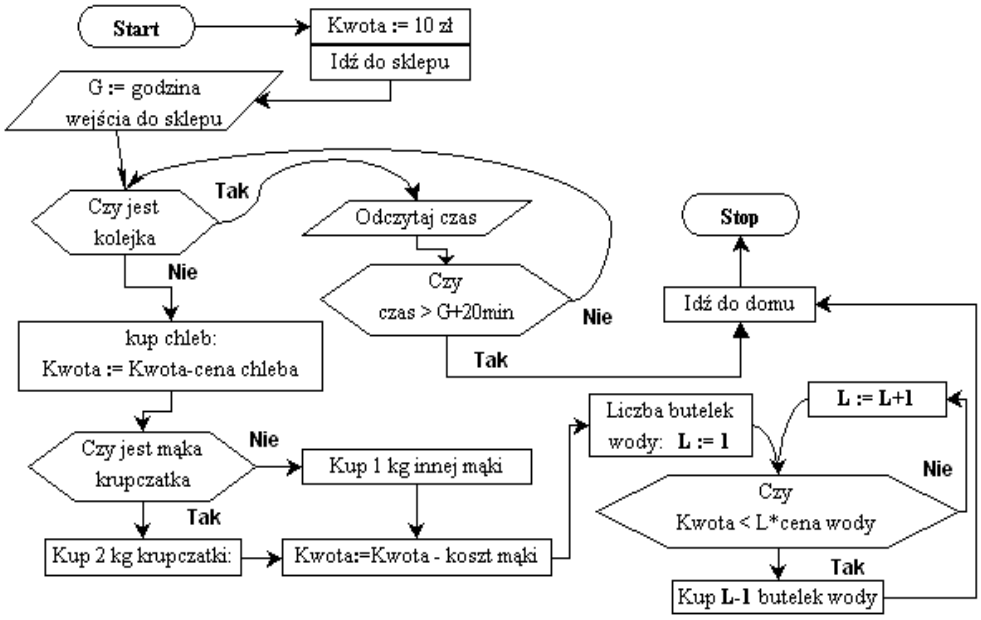
3.2. PRZYKŁADY „ALGORYTMÓW Z ŻYCIA"

Zanim zajmiemy się algorytmami dla komputerów, spróbujmy na razie przeanalizować różnego rodzaju „algorytmy" przeznaczone dla ludzi, takie jak:
- przepisy kulinarne,

- instrukcje montażu mebli lub urządzeń,
- instrukcje diagnozowania i naprawiania urządzeń,
- opisy załatwiania spraw w urzędach,
- porady dotyczące pisania CV czy innych pism

3.2.1. PRZYKŁAD 1- ZAKUPY

Małego Jasia posyłamy na zakupy i mówimy mu: *Jasiu, masz tu 15 zł, idź do sklepu i kup chleb oraz spytaj czy jest mąka "krupczatka", jeśli jest to kup 2 kilo a jeśli nie to tylko kilogram jakiejś innej mąki. Jak będzie kolejka to patrz na zegarek i nie czekaj więcej niż 20 minut. Jak starczy ci pieniędzy to kup jeszcze kilka butelek wody.*



Rys. 3.1. Schemat blokowy przykładowego algorytmu zakupów

Ten sam algorytm jako SCHEMAT BLOKOWY pokazuje Rys. 3.1. W schemacie tym zastosowano bloki analogiczne jak w algorytmach dla komputerów a mianowicie:

- początek ("Start") i koniec ("Stop"),
- wprowadzanie danych (Kwota, godzina G, czas, ..)
- modyfikowanie danych (np. zmniejszanie **Kwota := Kwota - koszt...** albo zwiększanie **L:=L+1**)
- podejmowanie decyzji zależnie od określonego warunku ("Czy jest kolejka?", "Czy jest mąka...")
- pętle - powodujące wielokrotne wracanie i powtarzanie pewnych czynności.

Zauważmy, że w poleceniach typu: „do zmiennej ... podstaw wartość ...", zastosowano znak podstawiania „**:=**" (taki znak stosowany jest w Mathcadzie oraz językach Algol i Pascal). W badanych wyrażeniach logicznych występują znaki „>" oraz „<" i podobnie mógłby wystąpić znak „**=**" w pytaniu „czy równe są wartości dwu wyrażeń", natomiast polecenie **L := L+1** oznacza „zwiększ dotychczasową wartość **L** o 1 i wstaw do **L** jako nową wartość".

W niektórych językach – jak BASIC, Fortran czy MATLAB – w tego typu instrukcjach używany jest znak „=" (zamiast znaku „**:=**") jako znak podstawiania.

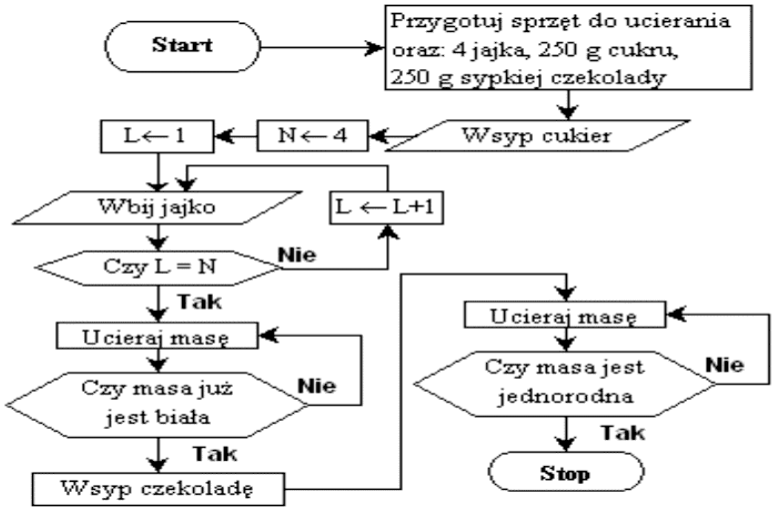
W algorytmach, w roli znaku podstawiania (zwanego też znakiem „przypisania"), bywają używane znaki: „**=**" albo „**:=**" albo „**←**". My będziemy preferować znak „**:=**".

3.2.2. PRZYKŁAD 2 – PRZEPIS NA MASĘ CZEKOLADOWĄ

Potrzebujemy („dane"): *4 jajka, 250 g cukru, 250 g sypkiej czekolady,*
Procedura: *Jajka utrzeć z cukrem do białości, wsypać czekoladę i dalej ucierać do otrzymania jednorodnej masy.*

Przeanalizuj algorytm (Rys. 3.2) i odpowiedz na pytania:

- jaki znak podstawiania zastosowano w poleceniach typu: „do zmiennej ... podstaw wartość ..." ?
- ile pętli występuje w algorytmie ?
- które pętle mają zdeterminowaną (określoną) liczbę obiegów (cykli) a które nie ?



Rys. 3.2. Algorytm przygotowania masy czekoladowej

Zauważmy, że podstawową cechą wszelkich algorytmów jest JEDNOZNACZNIE OKREŚLONA KOLEJNOŚĆ WYKONYWANIA OPERACJI.

3.3. „ALGORYTMY Z ŻYCIA" - ZADANIA

Przedstawione poniżej czynności przedstaw jako algorytmy – w postaci schematów blokowych lub opisów słownych, złożonych z ponumerowanych kroków.

Z1. **Zaliczanie semestru.** Przedstaw proces zaliczenia (lub nie) semestru (zgodnie z regulaminem studiów) przez uzyskiwanie zaliczeń z N przedmiotów z możliwością dwukrotnego poprawiania danego zaliczenia.

Z2. **Jaja sadzone na śmietanie.** Produkty: 8 jaj, szklanka śmietany, łyżka mąki.
Śmietanę mieszając z mąką zagotować na patelni. Jaja umyte wbijać kolejno do filiżanki i wlewać na patelnię. Przykryć i gotować na wolnym ogniu aż białko się zetnie.

Z3. **Masa tortowa.** Produkty: 6 jaj, szklanka cukru, 5 łyżeczek kakao.
Żółtka oddzielić od białek i ucierać z cukrem aż do białości. Nastepnie dalej ucierając dodawać po jednej łyżeczce kakao.

Z4. **Czarny ekran komputera.** Opisz co należy robić gdy ekran komputera jest czarny mimo, że komputer jest włączony.

Z5. **Poranne czynności.** Sporządź algorytm swoich porannych działań w zależności od dnia tygodnia.

Z6. **Naprawa lampy.** Sporządź algorytm wykrywania i usuwania przyczyny nie świecenia lampy.

Z7. **Przyszywanie guzika.** Sporządź algorytm przyszywania guzika.

Z8. **Sok z marchwi.** Sporządź algorytm przyrządzania soku z marchwi na sokowirówce (skrobanie, płukanie, krajanie, wirowanie).

Z9. **Przyrządzanie śniadania.** Sporządź algorytm przyrządzania śniadania

Z10. **Zdobywanie żony.** Jak wyobrażasz sobie optymalny algorytm zdobywania żony.

3.4. ALGORYTMY I METODY NUMERYCZNE

Algorytmy mogą opisywać działania z różnorodnych dziedzin, dla inżyniera jednak najbardziej przydatne będą algorytmy rozwiązywania problemów matematycznych.

W matematyce możemy używać pojęć abstrakcyjnych jak „nieskończenie mały przyrost" i oparte na nich pojęcia całek i pochodnych. Uczymy się też przekształcania wzorów czyli operacji na symbolach.

Komputery służą jednak najczęściej do operowania na konkretnych wartościach liczbowych czyli do realizacji **operacji numerycznych**.

Algorytmy rozwiązywania zagadnień działaniami arytmetycznymi i logicznymi na liczbach to **algorytmy numeryczne**. Zagadnieniami konstruowania i oceny tych algorytmów zajmuje się dziedzina zwana **analizą numeryczną** lub **metodami numerycznymi**.

Według publikacji [33]c - „metody numeryczne" zajmują się głównie budową algorytmów, natomiast „analiza numeryczna" – badaniem ich właściwości i oceną optymalności oraz dokładności uzyskanych wyników.

W [40] przytoczone są definicje pojęcia „metody numeryczne" według kilku źródeł, w tym m.in.:

- wg McGraw-Hill Dictionary of Scientific & Technical Terms: „metody przybliżonego rozwiązywania problemów matematycznych przy pomocy działań arytmetycznych",
- wg Encyclopædia Britannica: „dziedzina matematyki stosowanej zajmująca się aproksymacyjnymi metodami rozwiązywania złożonych równań przy użyciu operacji arytmetycznych, najczęściej przy użyciu komputera".

Metody numeryczne powstały wcześniej niż komputery – głownie dla problemów trudnych lub niemożliwych do rozwiązywania analitycznego. Obecnie są jednak z komputerami nierozerwalnie związane, z jednej strony ze względu na pracochłonność ręcznych obliczeń a z drugiej - ze względu na specyfikę komputera działającego na liczbach.

Cechą charakterystyczną metod numerycznych jest **dyskretyzacja** wartości zmiennych ciągłych i wynikająca z niej wielokrokowość obliczeń. W niektórych przypadkach jest to dyskretyzacja czasu. Na przykład przy rozwiązywaniu równań różniczkowych, funkcja odpowiedzi modelu na określone wymuszenie, wyznaczana będzie w określonych momentach. W innych przypadkach może to być dyskretyzacja przestrzenna – jak w metodzie elementów skończonych, gdzie model geometryczny jest podzielony na wiele drobnych elementów połączonych węzłami.

Wyniki uzyskiwane w metodach numerycznych są wprawdzie na ogół przybliżone lecz dokładność obliczeń może być z góry założona i osiągana przez zwiększenie liczby kroków procedury iteracyjnej, dokładności zapisu liczb w komputerze i dokładności dyskretyzacji wielkości ciągłych.

Popularne zagadnienia rozwiązywane metodami numerycznymi to:

- całkowanie i różniczkowanie,
- znajdowanie miejsc zerowych wielomianów,
- rozwiązywanie układów równań liniowych,
- rozwiązywanie równań i układów równań różniczkowych,
- znajdowanie wartości i wektorów własnych,
- aproksymacja i interpolacja funkcji,
- znajdowanie minimów i maksimów, używane m.in. przy optymalizacji.

Najważniejsze cechy metody numerycznej to:

- **zbieżność** czyli osiąganie rozwiązania dla różnych danych, przy możliwie małej liczbie kroków, dla zadanej określonej dokładności,
- **błąd metody** (zależny od liczby kroków lub podprzedziałów),
- **złożoność obliczeniowa** (omówiona w 3.7.) – tym większa im więcej dana metoda wymaga operacji a także zasobów pamięci - w porównaniu do innych metod dających ten sam wynik.

3.5. STAŁE, ZMIENNE, STRUKTURY DANYCH.

Tworzenie modeli matematycznych i opartych na nich algorytmów oraz programów, ale również korzystanie z gotowych narzędzi obliczeniowych jak Excel czy Mathcad, wymaga przede wszystkim zrozumienia istoty podstawowego pojęcia jakim jest **zmienna**.

Abstrakcyjne **zmienne**, występujące w modelach matematycznych w postaci symboli reprezentujących określone cechy, w informatyce mają bardzo konkretną realizację – **pełnią rolę pojemników na dane**, które mają być do komputera wprowadzone, przechowane, przetworzone i wyprowadzone.

A więc zapamiętajmy:

Zmienna:
- w dziedzinie modeli matematycznych: to **symboliczna reprezentacja cechy** modelowanego obiektu, procesu, lub samego modelu
- a w dziedzinie programów komputerowych: **zmienna** to **obszar pamięci komputera**, dostępny poprzez przypisaną mu **nazwę** i przechowujący **wartość** określonego **typu**.

Skoro zmienną w algorytmie i programie można traktować jak "pojemnik" na dane określonego typu, więc zamiast mówić: „zmiennej X przypisano wartość ...", lub „nadano wartość" będziemy także mówić: „do zmiennej X wstawiono wartość ...".

Według [33] - każda zmienna ma sześć istotnych atrybutów:

1) nazwę,
2) adres,
3) wartość,
4) typ,
5) czas życia,
6) zakres widoczności.

Każda zmienna obecna jest w algorytmie lub programie przez swoją **nazwę** z którą skojarzony jest **adres** pierwszej komórki przypisanego tej zmiennej obszaru pamięci. Przypisanie zmiennej obszaru pamięci, odpowiedniego dla **typu wartości** jakie ma ona przechowywać, nazywa się **alokacją pamięci** dla tej zmiennej i wykonywane jest przez **translator** na podstawie **deklaracji typu zmiennych** lub określonych reguł obowiązujących w danym języku. Zwalnianie przydzielonych obszarów to **dealokacja**.

Czas między alokacją i dealokacją to **czas życia** zmiennej.

Zmienne statyczne to zmienne, którym przydzielono obszary pamięci przed wykonywaniem programu i pozostają one bez zmian aż do zakończenia działania programu. Czas życia zmiennych statycznych jest więc (w przybliżeniu) równy czasowi wykonywania programu.

Zmienne dynamiczne – to zmienne których alokacja i dealokacja może odbywać się w trakcie wykonywania programu, a więc mogą mieć różny czas życia. Szczególnym typem zmiennych dynamicznych są **zmienne wskaźnikowe** istniejące m.in. w języku C (rozdział.11). Obszary pamięci przydzielone zmiennym dynamicznym mogą być zwiększane lub zmniejszane zależnie od potrzeb. Czasem możliwa jest również **zmiana typu** przechowywanych wartości w trakcie wykonywania programu.

Profesjonalne programy składają się z wielu **podprogramów**, często tworzonych przez różnych programistów. Aby mogli oni niezależnie tworzyć nazwy zmiennych dla swych podprogramów i nie musieli ich uzgadniać, stosuje się **zmienne lokalne**, których **zakres widoczności** jest ograniczony do wnętrza danego podprogramu. Inaczej mówiąc – nie istnieją one na zewnątrz tego podprogramu (nie są tam znane ani ich nazwy ani wartości). Czas życia zmiennej lokalnej jest najczęściej równy czasowi działania podprogramu po każdorazowym jego wywołaniu, a więc jest to zmienna przydzielana dynamicznie.

Przeciwieństwem zmiennych lokalnych są **zmienne globalne** – dostępne we wszystkich podprogramach z których składa się nasz program. W niektórych językach są to wszystkie zmienne zadeklarowane w głównym segmencie programu a w innych muszą być deklarowane, na przykład deklaracją *„Global”*. Zmienne globalne są z reguły statycznymi.

Pośrednim typem między lokalnymi i globalnymi są **zmienne wspólne** czyli należące do **wspólnego bloku** dla określonych podprogramów. Istnieją takie zmienne na przykład w Fortranie, jeśli zadeklarujemy je w odpowiednich podprogramach deklaracją *„Common”*. **Komunikacja podprogramu** z innymi blokami programu odbywa się więc przy pomocy **parametrów podprogramu** oraz **zmiennych globalnych i wspólnych**.

W różnych językach stosowane są różne – jawne lub niejawne, np.: domyślne – **sposoby deklarowania typu wartości zmiennych**, na przykład:

- język **BASIC** domyślnie przyjmuje, że zmienne niezadeklarowane będą typu ***single*** czyli ich wartości to liczby rzeczywiste o zakresie odpowiadającym t zw. „pojedynczej precyzji”, natomiast zmienne innych typów muszą być zadeklarowane albo przez umieszczenie na końcu nazwy jednego ze znaków: ***#*** *- double,* ***$*** *- string,* ***%*** *- integer,* ***&*** *- long*, albo przez deklarację zaczynającą się od słowa ***DIM*** ...;
- MATLAB automatycznie deklaruje typy i alokuje zmienne w sposób dynamiczny, na podstawie typu przypisywanych im wartości, przyjmując dla wartości liczbowych domyślnie typ ***double array*** – czyli tablica liczb rzeczywistych o „podwójnej precyzji”;
- języki Pascal oraz C wymagają jawnego zadeklarowania wszystkich zmiennych przy pomocy deklaracji rozpoczynających się odpowiednimi nazwami typów;
- w Fortranie wszystkie zmienne o wartościach liczbowych, których nazwy zaczynają się od liter: I, J, K, L, M, N są domyślnie traktowane jako zmienne typu INTEGER (całkowite), a inne jako typu REAL (rzeczywiste).

Można przyjąć, że **typ wartości** zmiennej jest określony przez:

- **rodzaj wartości** (np.: liczbowa, tekstowa, logiczna, ...) i przyjęty sposób jej kodowania,
- **liczbę zajmowanych bitów** pamięci i wynikający z niej dopuszczalny **zakres** wartości,
- **strukturę** (skalar, wektor , macierz, rekord, lista, ...).

W momencie użycia zmienna musi posiadać konkretną **wartość** - zgodną z typem a uzyskaną dzięki wcześniej wykonanym przez komputer poleceniom (instrukcjom, rozkazom) programu.

Zapisy konkretnych wartości - nadawanych zmiennym – nazywane są **stałymi** lub rzadziej „**literałami**” bo literalnie oznaczają to co widać,

Na przykład zmiennej ***temp*** można przypisać stałą **38.6** reprezentującą wartość temperatury konkretnego pacjenta w konkretnej chwili, a zmiennej ***naz_masz*** można przypisać stałą tekstową **"strugarka"** reprezentującą nazwę konkretnej maszyny:

temp := **38.6;** ***naz_masz*** **:= "strugarka"**

Pojęcie „stała" jest więc w informatyce nieco odmienne od stosowanego w fizyce czy matematyce, gdzie kojarzy się nam na ogół z takimi stałymi jak „pi", „e" – podstawa logarytmów, naturalnych, prędkość światła czy stała Plancka. W informatyce **stała** to najczęściej zapis konkretnej wartości określonego typu, umieszczony w treści programu, chociaż czasem używane jest też klasyczne pojęcie stałej jako predefiniowanej zmiennej o ustalonej i nie zmienianej wartości.

Sposoby zapisu (składnia) stałych i zmiennych różnych typów, w konkretnym języku programowania, są ściśle określone przez reguły tego języka. **Separatorem części ułamkowej** w zapisie liczb dziesiętnych (stałych liczbowych) z reguły **jest kropka** - zgodnie ze zwyczajami anglosaskimi. **Stałe tekstowe** ujmowane są **w cudzysłowy** lub **apostrofy**.

W odróżnieniu od stałej, reprezentującej konkretną wartość, zmienna jest abstrakcyjną reprezentacją wszystkich swych możliwych wartości z dopuszczalnego zakresu, chociaż każda operacja algorytmu dokonywana jest na konkretnych wartościach zmiennej. Myślenie o zmiennej musi więc odbywać się na dwu planach. Na etapie opracowywania ogólnej koncepcji algorytmu musimy mieć na uwadze wartości jakie ta zmienna może przyjmować, a także takie wartości jakich nie może przyjmować, bo na przykład doprowadzą do niewykonalnych działań jak dzielenie przez zero. Jeśli natomiast śledzimy działanie algorytmu, wówczas musimy rozpatrywać bieżące wartości zmiennych, wynikające z wcześniej wykonanych operacji.

Dzięki zmiennym, każdy program powinien stanowić uogólniony zapis rozwiązywania pewnej **klasy zagadnień** i powinno być możliwe wielokrotne wykorzystywanie go **dla różnych wartości danych**. Twórca programu powinien więc zaprogramować jego reakcje na różne sytuacje jakie mogą wystąpić w trakcie działania programu.

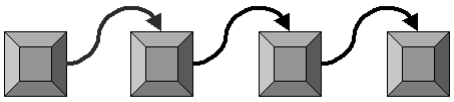
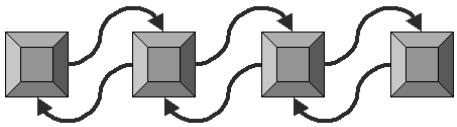
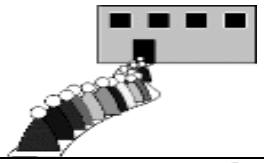
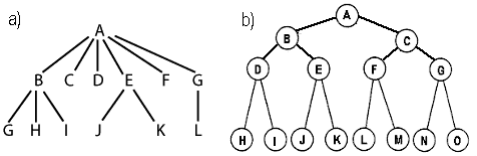
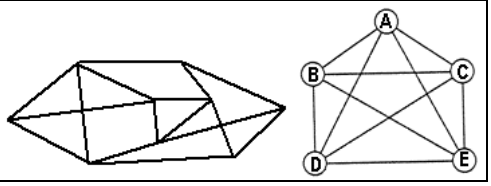
Ponieważ nie da się skonstruować algorytmu czy programu bez precyzyjnego określenia **roli każdej zmiennej** i pamiętania o tej roli, dlatego niezbędne jest **zapisanie** przez programistę **objaśnień poszczególnych zmiennych** zarówno przed lub w trakcie konstruowania programu jak i w jego **dokumentacji** oraz **systemie pomocy**.

O programie mówi się, że jest „samodokumentujący się" jeśli objaśnienia ról zmiennych oraz funkcji poszczególnych bloków programu umieszczone są w **komentarzach** wewnątrz treści programu oraz w opisach pojawiających się na ekranach roboczych oraz po wywołaniu systemu pomocy.

Zmienne w algorytmach i programach potrzebne są do reprezentowania cech modelowanego obiektu, ale często bywają potrzebne także **zmienne pomocnicze** związane z samym algorytmem lub strukturą danych, jak indeksy, flagi (znaczniki), bufory i in.

Oprócz **zmiennych prostych** czyli **skalarnych** - przechowujących pojedyncze wartości (liczbowe, znakowe, logiczne) w programach używane są **zmienne złożone** czyli **struktury danych,** zwane czasem kontenerami (ang.: *container*), zdolne do przechowywania wielu wartości. Najważniejsze struktury danych pokazuje Tabela 3.1.

Tabela 3.1. Najważniejsze struktury danych

Struktura danych	Przykłady
Tablica jednowymiarowa czyli **wektor** - ciąg ***n*** ponumerowanych składowych	a= <table><tr><td>a1</td><td>a2</td><td>a3</td><td>a4</td><td>a5</td></tr><tr><td>0.4</td><td>1.2</td><td>0.7</td><td>2.2</td><td>3.6</td></tr></table>
Tablica dwuwymiarowa - czyli **macierz** – oraz tablice o większej liczbie wymiarów	$M = \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \end{bmatrix} = \begin{bmatrix} 9.8 & 2.4 & -1 \\ 8.5 & 0 & 7.1 \end{bmatrix}$
Rekord - zbiór ***n*** danych o pojedynczym obiekcie lub zdarzeniu (np. dane personalne: nazwisko, rok urodzenia, ...), przy czym każda z tych danych może być innego typu. Tabele baz danych gromadzą rekordy w poszczególnych wierszach.	<table><tr><td>Nr</td><td>Nazwisko</td><td>Imię</td><td>Rok_ur</td><td>Wzrost</td></tr><tr><td>1</td><td>Stypuła</td><td>Marcin</td><td>1967</td><td>178</td></tr><tr><td>2</td><td>Kowal</td><td>Stefan</td><td>1986</td><td>170</td></tr><tr><td>3</td><td>Zaborek</td><td>Jan</td><td>1948</td><td>169</td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>
Lista jednokierunkowa - której każdy element oprócz wartości zawiera adres swego następnika lub znacznik końca listy	
Lista dwukierunkowa – której każdy element oprócz wartości zawiera adres swego następnika lub znacznik końca listy oraz adres poprzednika lub znacznik początku	
Stos - lista o zmiennej długości w której dołączanie i usuwanie elementów odbywa się na końcu listy. Bywa realizowana jako rejestr LIFO (ang.: Last In First Out - „ostatni przyszedł pierwszy wyjdzie")	
Kolejka - lista o zmiennej długości do której elementy są dołączane na końcu listy a usuwane na początku listy. Bywa realizowana jako rejestr FIFO (ang.: *First In First Out* - „ten kto pierwszy przyszedł ten pierwszy odejdzie")	
Drzewo – (a) struktura hierarchiczna, (b) drzewo binarne Przykłady: 1) rozdziały i podrozdziały podręcznika, 2) foldery i podfoldery plików	
Graf – struktura sieciowa Np.: dokumenty w Internecie powiązane odsyłaczami (ang.: *link* - łącze)	

Najbardziej podstawowym i najczęściej wykorzystywanym typem zmiennej złożonej jest **tablica**. Tablice dostępne są praktycznie w każdym języku programowania. Elementy tablicy identyfikowane są przy pomocy **nazwy tablicy wraz z** indywidualnymi dla każdego elementu numerami czyli **indeksami**. Liczba indeksów identyfikujących dowolny element to **wymiar** tablicy.

Matematycznym odpowiednikiem tablicy jednowymiarowej jest **ciąg (wektor)**, a tablicy dwuwymiarowej - **macierz**. Maksymalne wartości wskaźników określają **rozmiary** tablicy. W niektórych (starszych) językach programowania istnieją tylko **tablice statyczne – o stałych rozmiarach**, które trzeba zadeklarować przed pierwszym użyciem tablicy. Nowoczesne języki programowania - w tym MATLAB - dysponują zazwyczaj możliwością używania **tablic dynamicznych**, których rozmiary mogą się zmieniać w trakcie wykonywania programu. W większości języków programowania wszystkie elementy tablicy muszą być tego samego typu, ale MATLAB dysponuje tablicami komórkowymi, w których każdy element może być innego typu (również złożonego).

Inne typy danych złożonych – listy – wykorzystywane są m.in. w językach Lisp i Logo, posiadających odpowiednie dla nich narzędzia.

Do złożonych struktur danych należą też omawiane w osobnym rozdziale relacyjne bazy danych, złożone z wielu powiązanych wzajemnie tablic, a także dostępne w MATLAB-ie tablice komórkowe, w których każdy element może być innego – także złożonego – typu.

3.6. PODSTAWOWE TYPY POLECEŃ W ALGORYTMACH OBLICZENIOWYCH

Tworzenie algorytmów sprawia studentom na ogół spore trudności a wynikają one między innymi z nieznajomości sensu poleceń jakie można wydawać komputerowi. Tymczasem do konstruowania prostych algorytmów i programów obliczeniowych - niezależnie od języka programowania w którym je zapiszemy - wystarczy tylko **sześć podstawowych typów poleceń**, a mianowicie:

1) **Wczytywanie wartości** danych z urządzenia wejściowego **do zmiennych**.

2) **Przypisanie zmiennej wartości** podanego **wyrażenia** arytmetycznego.

3) **Wyprowadzanie** wartości zmiennych oraz napisów na urządzenie wyjściowe.

4) **Rozgałęzienie warunkowe IF**: *„JEŚLI podany warunek jest spełniony to wykonuj te ... operacje W PRZECIWNYM PRZYPADKU wykonuj tamte ... operacje”.*

5) **Pętla FOR:** „*Dla poszczególnych wartości zmiennej ... w zakresie od ... do ... z przyrostem ... powtarzaj wykonywanie następujących operacji ...”*

6) **Pętla WHILE**: „*Podczas gdy zachodzi ten ... warunek powtarzaj wykonywanie tych ... operacji.”*

Nieco bardziej złożone programy obliczeniowe wymagają jeszcze dwu dodatkowych typów poleceń, a mianowicie:

7) **definiowania podprogramów**, oraz
8) **wywołania** tych **podprogramów** lub gotowych podprogramów bibliotecznych.

Zapamiętanie tych naprawdę niewielu poleceń, a przede wszystkim opisanych niżej działań komputera jakie spowodują, nie jest więc trudne a równocześnie jest niezwykle ważne. Jeśli na końcu semestu ktoś zapomni szczegółów składni takich poleceń jak pętla FOR czy WHILE w MATLAB-ie, BASIC-u czy języku C, to nie szkodzi - zawsze może skorzystać z pomocy czyli HELP. Jeśli jednak nie będzie znał roli tych sześciu typów poleceń, to nie ułoży żadnego poprawnego algorytmu ani programu – gorzej – nie będzie rozumiał istoty działań komputera!

Wszystkie wymienione typy poleceń objaśniono w zamieszczonych niżej podpunktach.

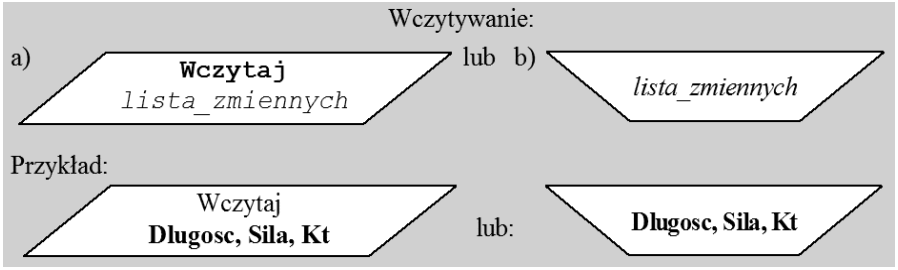
3.6.1. WCZYTYWANIE DANYCH

Z przyjętego już założenia, że każdy program obliczeniowy powinien poprawnie funkcjonować dla wielu różnych zestawów danych, wynika, że **należy unikać umieszczania DANYCH w treści programu**.

W programie potrzebne więc będą **polecenia pobierania wartości danych z urządzeń wejściowych lub z pamięci masowej i przesyłania ich do obszarów pamięci operacyjnej RAM przypisanych określonym zmiennym**. Takie operacje nazywane są „**wprowadzaniem**" lub częściej „**wczytywaniem**" danych do określonych zmiennych.

Instrukcja wprowadzania danych w językach programowania zawiera najczęściej słowo kluczowe: ***INPUT*** – wprowadź lub ***READ*** – wczytaj, musi też zawierać **listę nazw zmiennych** do których dane mają być wczytane oraz **tekst żądania danych** – jeśli dane mają być wprowadzane z klawiatury – aby użytkownik wiedział co ma zrobić.

W algorytmach norma PN-75/E-01226 zaleca użycie bloku wczytywania o kształcie pokazanym na Rys. 3.3(a), jednak norma ta określa identyczny kształt bloku także do wyprowadzania, dlatego autor proponuje bardziej kojarzący się z rolą kształt (b).



Rys. 3.3. Bloki wczytywania danych

Kolejność i typy wprowadzanych danych muszą odpowiadać kolejności i typom zmiennych na liście, tak aby pierwsza wczytana dana była przypisana pierwszej zmiennej z listy, druga z danych – drugiej zmiennej i t.d.

3.6.2. PRZYPISANIE ZMIENNEJ WARTOŚCI WYRAŻENIA

Polecenie przypisania nazywane też **instrukcją podstawiania** – ma w algorytmach ogólną postać:

```
zmienna := wyrażenie
```

Polecenie to oznacza: „**oblicz wartość *wyrażenia* i nadaj ją *zmiennej***”. *Wyrażenie*, w najprostszych postaciach może być pojedynczą zmienną lub stałą.

Instrukcja przypisania służy najczęściej do **obliczania wyrażeń** a także do nadawania początkowych wartości zmiennym kontrolnym i modyfikowania ich wartości.

Przykład: ***a :=a*b+5*** – oznacza: „oblicz wartość wyrażenia zapisanego po prawej stronie znaku podstawiania, biorąc do obliczeń dotychczasowe wartości zmiennych a uzyskany wynik przypisz zmiennej zapisanej po lewej stronie tego znaku, jako jej nową wartość”.

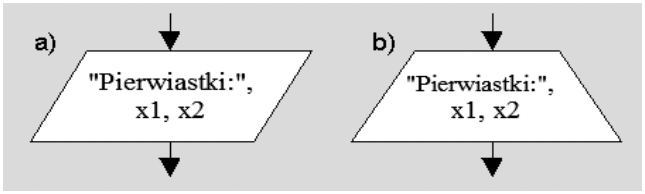
Inne przykłady: **sila := 157.56** , **moc := (sila*droga)/czas** , **i:=i+1**

Znak podstawiania (lub jak kto woli przypisania) „**:=**” jest stosowany w językach Algol, Pascal oraz w Mathcadzie i będziemy go także stosować w algorytmach dla wyraźnego odróżnienia od znaku równości, używanego przy porównywaniu.

W MATLAB-ie znakiem podstawiania jest wprawdzie „**=**” ale – dla odróżnienia - znakiem równości jest „**==**”.

3.6.3. WYPROWADZANIE WYNIKÓW

Wyprowadzanie wyników – tekstów, wartości zmiennych, wykresów i in. na urządzenie wyjściowe (np. wyświetlanie, drukowanie lub zapis do pliku), to polecenie istniejące we wszystkich językach programowania, identyfikowane najczęściej słowem kluczowym ***PRINT*** albo ***WRITE*** po którym następuje lista elementów, których wartości mają zostać wyprowadzone. Lista ta może zawierać stałe tekstowe (teksty objaśnień) i zmienne lub wyrażenia różnych typów



Rys. 3.4 Przykładowe bloki wyprowadzania wyników

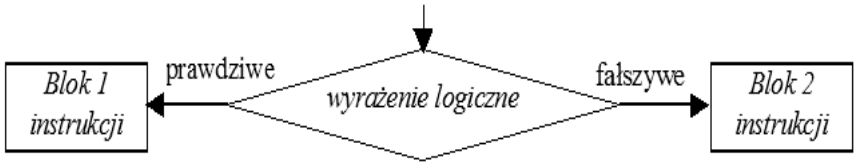
Ten typ polecenia jest niezbędny w większości języków programowania gdyż bez niego nie zobaczymy uzyskanych wyników. Jednym z niewielu wyjątków jest MATLAB, w którym wynik każdego polecenia nie zakończonego średnikiem jest automatycznie wyświetlany.

Wyświetlane lub drukowane wartości powinny być uzupełniane objaśnieniami, gdyż bez nich będą niezrozumiałe. Teksty objasnień – zależnie od języka programowania – powinny być ujmowane w cudzysłowy lub apostrofy jako stałe typu tekstowego, w odróżnieniu od zmiennych i wyrażeń numerycznych.

W algorytmach blok wyprowadzania danych - według normy PN-75/E-01226 - powinien mieć kształt równoległoboku (Rys. 3.4a), identycznie jak blok wprowadzania. Dla rozróżnienia tych bloków autor proponuje stosowanie kształtu pokazanego na Rys. 3.4b.

3.6.4. ROZGAŁĘZIENIE WARUNKOWE – INSTRUKCJA „JEŻELI ..."

Na schemacie blokowym algorytmu jest to blok o kształcie rombu, posiadający jedno wejście i **dwa wyjścia** – jako jedyny - wszystkie pozostałe typy bloków (z wyjątkiem ostatniego w algorytmie) posiadają jedno wyjście. Rozgałęzienie warunkowe - zwane instrukcją IF (Rys. 3.5) pozwala zbadać umieszczone w nim wyrażenie logiczne i na podstawie wyniku badania uaktywnić jedno z dwu wyjść.



Rys. 3.5. Blok rozgałęzienia warunkowego

Dokładniej, sens bloków algorytmu przedstawionych na Rys. 3.5 jest następujący: „jeśli *wyrażenie logiczne* jest prawdziwe to wykonaj instrukcje podane w *Bloku 1* a w przeciwnym przypadku wykonaj instrukcje podane w *Bloku 2*".

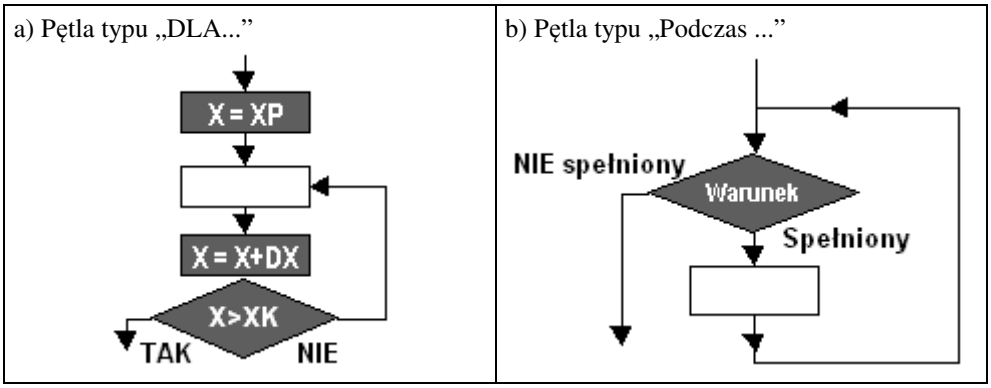
W językach programowania – przed „Blokiem 1" umieszczone byłoby słowo ***IF*** oznaczające jeżeli, a po nim wyrażenie logiczne (i czasem dodatkowo słowo ***THEN*** czyli „to"), natomiast przed „Blokiem 2" wystąpiłoby słowo **ELSE** oznaczające „w przeciwnym przypadku", a za nim element oznaczający zakończenie instrukcji IF – zależny od języka (np. słowo END albo END IF).

W niektórych językach (np. w BASIC-u) rolę prawdziwych wyrażeń logicznych mogą pełnić również wyrażenia algebraiczne o wartości różnej od zera, natomiast wartość zero jest równoznaczna z wartością logiczną „fałsz".

3.6.5. PĘTLA TYPU "DLA ..."

Pętlę typu „Dla..." (Rys. 3.6a) - o określonej liczbie cykli - tworzą na schematach blokowych algorytmów trzy bloki:

1) blok nadający początkową wartość zmiennej kontrolnej (inicjacja),
2) blok zwiększający zmienną kontrolną o stały przyrost (inkrementacja),
3) blok warunkowy – sprawdzający czy zachodzi już warunek kończący pętlę.



Rys. 3.6. Pętle programowe

W językach programowania pętla tego typu rozpoczyna się od słowa FOR czyli „dla”.

Pętla FOR przedstawiona na Rys. 3.6a ma znaczenie następujące:

"Dla wartości zmiennej kontrolnej X zmieniającej się od XP, z przyrostem DX, do XK, powtarzaj wykonywanie instrukcji zapisanych wewnątrz pętli, tyle razy ile będzie wartości zmiennej X".

3.6.6. PĘTLE O NIEOKREŚLONEJ LICZBIE CYKLI

W pętli FOR liczba cykli jest z góry określona przez wartość początkową, końcową i przyrost zmiennej kontrolnej.

Nie zawsze jednak da się z góry określić ciąg wartości zmiennej kontrolnej i wynikającą z niego liczbę cykli pętli. Czasem konieczne jest powtarzanie cykli aż do zaistnienia pewnego zdarzenia, zmieniającego związane z nim wyrażenie logiczne (lub innego typu).

Funkcje takie spełniają pętle:

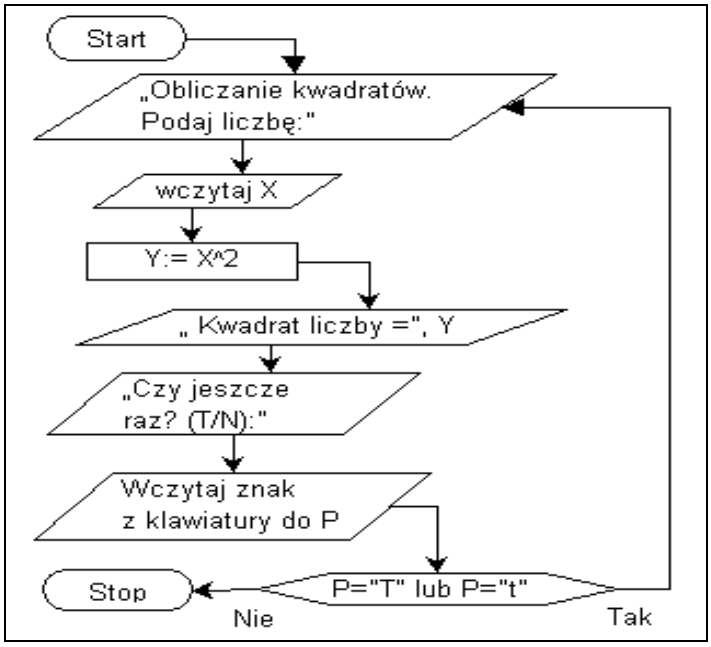
a) typu „podczas” - ang.: ***WHILE*** (Rys. 3.6b) - w której na początku każdego cyklu sprawdzany jest WARUNEK a powtarzanie wykonywania instrukcji zapisanych wewnątrz pętli następuje tak długo dopóki ten WARUNEK jest spełniony,

b) typu „wykonuj ...dopóki” – ang.: ***DO ... UNTIL*** – w której warunek sprawdzany jest na końcu pętli.

WARUNEK to wyrażenie logiczne (w najprostszym przypadku relacja) np.: X>0.

Prostym przykładem zastosowania pętli o nieokreślonej liczbie cykli może być algorytm powtarzania pewnych obliczeń dla różnych danych, tak długo aż użytkownik nie zakończy obliczeń przez naciśnięcie odpowiedniego klawisza. Rys. 3.7 przedstawia przykładowy algorytm obliczania kwadratów liczb, wprowadzanych przez użytkownika z klawiatury. Ponieważ warunek jest sprawdzany na końcu więc najlepiej byłoby zastosować pętlę typu „Do ... Until”, jednak po podstawieniu na początku **P=”T”** można zastosować pętlę typu „While ...”.

Postacie zapisu tego algorytmu w kilku językach pokazano w p. 4.3.



Rys. 3.7. Przykładowy schemat blokowy algorytmu obliczeń w pętli

3.6.7. SKOK BEZWARUNKOWY

W niektórych językach – na przykład w BASIC-u i Pascalu - istnieje polecenie skoku bezwarunkowego do określonej instrukcji programu, która – dla umożliwienia tego skoku - została oznaczona tak zwaną etykietą (liczbowym lub tekstowym identyfikatorem). Instrukcja ta ma postać: „GoTo nr" czyli „SKOCZ DO nr".

Przy pomocy instrukcji skoku wraz z instrukcją warunkową (IF) można zbudować dowolne pętle, jednak używanie wielu instrukcji skoku może bardzo zagmatwać strukturę i zmniejszyć czytelność i klarowność programu, dlatego nie jest zalecane.

Programowanie bez użycia instrukcji skoku a jedynie przy pomocy opisanych wcześniej poleceń, umieszczonych wewnątrz pętli lub wewnątrz dwu bloków ograniczonych elementami instrukcji IF – to tak zwane programowanie strukturalne.

3.6.8. PODPROGRAMY – PROCEDURY I FUNKCJE

Współczesne programy są bardzo złożone i składają się często z wielu tysięcy poleceń. Dlatego, tak jak przy realizacji dużych przedsięwzięć wyodrębnia się zadania składowe, tak i duże programy (a wcześniej algorytmy) dzieli się na poszczególne **podprogramy** (*subroutine*) **o ściśle określonej roli, nazwie** oraz **wejściach i wyjściach**.

Podobnie powinien postępować student czy inżynier programujący w danym języku. Wprawdzie ma on na ogół do dyspozycji wiele gotowych podprogramów, pogrupowanych w tematyczne biblioteki (w MATLAB-ie zwane „toolboxami") ale może i powinien definiować i wywoływać także własne podprogramy.

Zdefiniować podprogramy warto nie tylko wtedy gdy program jest duży i trzeba go podzielić na składowe, lecz również wtedy gdy wielokrotnie trzeba wykonać podobne operacje dla różnych danych wejściowych w ramach programu głównego albo gdy jasno sprecyzowana procedura może się nam przydać także w innych programach.

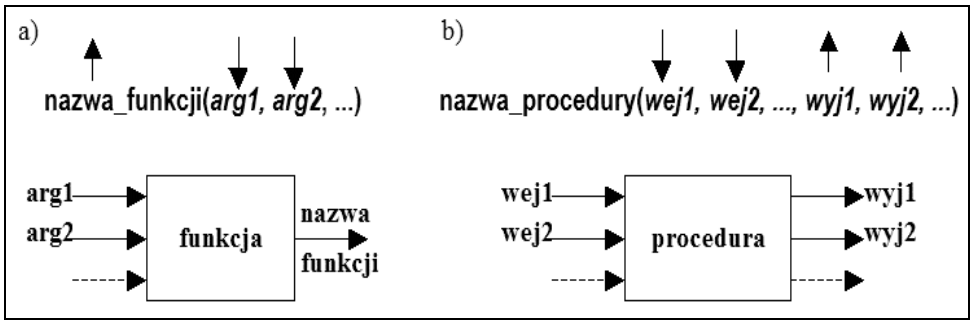
Podprogram powinien przypominać „czarną skrzynkę", której zawartość (treść) jest niezależna od reszty programu i interesuje tylko twórcę tego podprogramu a pozostałych interesuje sposób działania podprogramu oraz dane jakie należy dostarczyć i uzyskiwane wyniki. Dzięki temu praca nad dużym programem może być rozłożona na opracowanie wielu podprogramów przez różnych programistów. Podprogram powinien więc stanowić samodzielne zadanie, które można krótko i jasno zdefiniować i wielokrotnie realizować dla różnych danych.

Należy rozróżniać definicję podprogramu od jego wywołania.

Definicja podprogramu przypisuje podprogramowi identyfikującą go **nazwę** (pozwalającą go następnie wywoływać), określa **parametry formalne** czyli zmienne wejściowe i wyjściowe (nie posiadające na razie wartości), oraz **sposób przetwarzania** wartości wejściowych na wyjściowe. W niektórych językach wymagane są również deklaracje **zmiennych lokalnych** – używanych i „widzianych" tylko wewnątrz podprogramu oraz **zmiennych globalnych** – wspólnych dla określonych podprogramów i ewentualnie programu głównego.

Wykorzystanie podprogramu nazywa się jego „**wywołaniem**" i polega na umieszczeniu w programie polecenia zawierającego nazwę podprogramu a w miejsce parametrów formalnych umieszcza się **parametry aktualne:** wejściowe – wprowadzające konkretne wartości danych oraz wyjściowe - pobierające wyniki. Kolejność parametrów przy wywołaniu podprogramu musi być zgodna z kolejnością określoną przy definiowaniu podprogramu, ponieważ role oraz typy parametrów wynikają właśnie z tej kolejności.

W tradycyjnych językach programowania (jak Fortran, BASIC, Pascal) istnieje logiczny podział podprogramów na **funkcje** oraz **procedury** (Rys. 3.8).



Rys. 3.8. Wejścia i wyjścia: a) funkcji, b) procedur

Funkcja (Rys. 3.8a) może mieć **dowolną liczbę wejść** (argumentów funkcji) ale zawsze tylko **jedno wyjście**. Zmienna wyjściowa jest utożsamiona z nazwą funkcji. Przykładem może być funkcja wyznaczająca odległość między dwoma punktami o danych współrzędnych. Dzięki takim cechom, **wywołania funkcji mogą stanowić elementy wyrażeń**.

Procedura (Rys. 3.8b) może natomiast posiadać **dowolną liczbę wejść i wyjść** a w szczególności nie posiadać żadnych wejść ani wyjść – jeśli wykonuje zawsze te same ustalone czynności. Wywołanie procedury jest oddzielnym poleceniem i nie może stanowić elementu wyrażenia.

Przykłady:

a) rozwiązywanie równania kwadratowego: wchodzą współczynniki a, b, c, wychodzą pierwiastki x1, x2;

b) drukowanie nagłówka firmowego wraz z logo firmy – brak parametrów.

W innych językach (np. w języku C i językach z nim spokrewnionych) stosowane są jedynie funkcje ale definicję ich na tyle poszerzono aby mogły pełnić także rolę procedur.

W MATLAB-ie funkcje mogą dawać w wyniku wektor wielu parametrów wyjściowych, a więc pełnić rolę taką jak procedury w Fortranie czy Pascalu. Dodatkowo, liczba parametrów może być zmienna.

3.7. ITERACJA I REKURENCJA

Pętla programowa pozwala krótko zapisać operacje które będą wykonywane w wielokrotnie powtarzanych cyklach.

Bardzo często w pętli umieszczane są instrukcje które wyznaczają nową wartość zmiennej na podstawie jej dotychczasowej wartości.

Na przykład przy wyznaczaniu sumy ciągu liczb wczytywanych kolejno do zmiennej ***A***, w pętli powtarzana będzie operacja ***S:=S+A***, a przy wyznaczaniu iloczynu takiego ciągu musi być powtarzana operacja ***P:= P*A***. W takich przypadkach trzeba przed pętlą nadać wartość początkową takiej zmiennej (dla sumy zero, dla iloczynu 1).

Iteracja to metoda polegająca na wielokrotnym stosowaniu sekwencji tych samych operacji, przy czym wyniki i-tego przebiegu stanowią dane wejściowe dla kolejnego przebiegu o numerze (i+1).

Innym pojęciem, nie związanym z obliczeniami w pętli lecz związanym z definiowaniem funkcji, jest **rekurencja** zwana też krócej **rekursją**.

Funkcja jest rekurencyjna [34][37], jeśli w jej definicji użyto odwołanie do niej samej, lub inaczej mówiąc odwołuje się do siebie samej.

Zaletą rekurencji jest zwięzłość zapisu, zaś wadą może być duże zapotrzebowanie na pamięć i czas przetwarzania (przy wielu poziomach rekurencji).

Przykład:

```
Silnia(n):
        jeśli n jest równe 1 to:
                Silnia:=1
        w przeciwnym przypadku:
                Silnia:=n*Silnia(n-1)
        koniec
```

Nie należy stosować rekurencji, jeśli nie można zdefiniować warunku końcowego (w naszym przypadku ***1!***). Brak tego warunku powoduje ciąg niekończących się wywołań.

Rekurencję powinno stosować się do problemów, które można zdefiniować z zastosowaniem tej samej procedury do coraz mniejszych zagadnień – np.: ***Silnia(n-1)*** - prowadzących do warunku końcowego.

Rekurencję można zastąpić obliczeniami iteracyjnymi z wykorzystaniem stosu.

3.8. ZŁOŻONOŚĆ OBLICZENIOWA ALGORYTMÓW

Dla komputerowej realizacji postawionego zadania możemy zazwyczaj opracować wiele algorytmów różniących się szybkością działania i wykorzystaniem zasobów komputera. Najlepsze będą oczywiście algorytmy najszybsze i wykorzystujące najmniej zasobów. O takich algorytmach mówi się, że charakteryzują się „najmniejszą złożonością obliczeniową".

Przez **złożoność obliczeniową algorytmu** [4], [33], [37] rozumiana jest **ilość zasobów** niezbędnych do jego realizacji. Jest to jedna z najistotniejszych cech algorytmu decydująca o efektywności budowanego na jego podstawie programu.

Uwzględniane zasoby to przeważnie **czas** oraz wymagana **objętość pamięci** operacyjnej i w związku z tym rozróżnia się **czasową złożoność obliczeniową** oraz **pamięciową złożoność obliczeniową**. Za jednostkę złożoności pamięciowej przyjmuje się na ogół bajt.

Na złożoność czasową składają się dwie wartości: pesymistyczna, czyli taka, która charakteryzuje najgorszy przypadek działania oraz oczekiwana. Złożoność czasowa nie może zależeć od szybkości komputera a więc nie jest określana jednostkami czasu. Oszacowuje się natomiast **f(n)** - **zależność liczby wykonywanych operacji** podstawowych, **od liczby danych**, oznaczonej jako ***n***. Operacjami podstawowymi mogą być na przykład: podstawienie, porównanie lub prosta operacja arytmetyczna.

Teoria złożoności obliczeniowej to rozbudowana i dość trudna dziedzina matematyki.

Złożoność algorytmów określana jest przez t.zw. **asymptotyczne tempo wzrostu**, czyli tendencję wzrostu funkcji f(n) dla odpowiednio dużych liczności danych wejściowych. Oprócz tego złożoności algorytmów proporcjonalne do siebie (różniące się stałym czynnikiem) uważane są za takie same, co eliminuje wpływ szybkości działania komputera, oraz wybór operacji podstawowej, która na jednym komputerze może wykonywać się błyskawicznie, a na innym kilka razy dłużej.

Oszacowania asymptotyczne funkcji f(n) dokonywane są przy pomocy znanych funkcji na przykład wielomianowych. Szybkość działania algorytmu może zależeć od rodzaju danych co utrudnia oszacowania. Dlatego są trzy rodzaje oszacowań rzędu funkcji f(n):

- **pesymistyczne** – oznaczane „notacją omikron": $f(n)=O(g(n))$ – określające, że w najgorszym razie, począwszy od pewnej liczby danych n_o, krzywa f(n) wzrostu liczby operacji nie przekroczy wzrostu proporcjonalnego do funkcji g(n) czyli $f(n)<c2*g(n)$ dla $n>n_o$, gdzie c2 jest stałą;
- **dokładne** – oznaczane „notacją theta": $f(n)=\Theta(g(n))$ – szacujące górną i dolną funkcyjną granicę asymptotyczną: $c1*g(n)<f(n)<c2*g(n)$ dla $n>n_o$, gdzie c1, c2 to stałe;
- **optymistyczne** (dotyczące najkorzystniejszych danych dla szybkości algorytmu) – oznaczane „notacją omega": $f(n)=\Omega(g(n))$ – szacujące dolną funkcyjną granicę asymptotyczną: $c1*g(n)<f(n)$ dla $n>n_o$, gdzie c1 jest stałą.

Tabela 3.2. Niektóre rzędy złożoności obliczeniowej

Złożoność obliczeniowa	**Rząd**	**Objaśnienie**
stała	$O(1)$	Stała ilość operacji dominujących bez względu na rozmiar danych wejściowych
liniowa	$O(n)$	Czas wykonania proporcjonalny do liczby danych wejściowych **n**.
kwadratowa	$O(n^2)$	Czas wykonania proporcjonalny do kwadratu liczby danych n.
wielomianowa	$O(n^k)$	Czas wykonania proporcjonalny do określonej potęgi liczby danych
logarytmiczna	$O(\log_2 n)$	Np.: wyszukiwanie binarne w zbiorze uporządkowanym. Algorytm b. szybki.
liniowo logarytmiczna	$O(n \log_2 n)$	
wykładnicza	$O(2^n)$, $O(n!)$	Algorytm, nierealizowalny dla większej liczby danych. ($n!>2^n$ już od n=4)

Najczęściej podaje się przynajmniej oszacowanie pesymistyczne wg notacji omikron – co pokazuje Tabela 3.2 - szacując, że wzrost liczby operacji przy wzroście liczby danych w najgorszym razie będzie proporcjonalny do określonego typu funkcji.

4. WPROWADZENIE DO PROGRAMOWANIA

Proces programowania w szerszym znaczeniu obejmuje zarówno opracowanie algorytmu i zdefiniowanie potrzebnych struktur danych jak i zapisanie tego algorytmu w wybranym języku programowania. W węższym znaczeniu programowanie – zwane też wówczas *kodowaniem* - związane jest wyłącznie z konkretnym językiem programowania.

Mając gotowy algorytm - opisujący kolejne kroki pożądanych działań komputera – można przystąpić do tworzenia programu. Pierwszym krokiem jest wybór języka programowania. Najlepiej zdecydować się na **język programowania** (p.4.3) już w trakcie ustalania wstępnych koncepcji algorytmu, wybierając język z kategorii najwłaściwszej dla tematyki rozwiązywanego programu i umożliwiający programowanie w wybranym przez nas stylu. W ramach wybranego języka, a właściwie łacznie z językiem, wybieramy styl programowania nazywany czasem **paradygmatem programowania** (p.4.4).

Do pisania, testowania i poprawiania programu będą nam potrzebne odpowiednie narzędzia programistyczne (p.4.5) jak **edytor, translator, debugger**, oraz **system pomocy**.

Program będzie tworzony i uruchamiany w konkretnym **systemie operacyjnym,** stanowiącym jego **środowisko uruchomieniowe** (p.4.5), do którego muszą być dostosowane te narzędzia jak i ostateczna, binarna, postać wykonywalna naszego programu.

Innym – coraz bardziej dominującym – rozwiązaniem jest skompilowanie programu do uniwersalnego dla różnych komputerów **kodu pośredniego** (p.4.5.5 i 4.5.6), który następnie będzie (przy pierwszym wykonaniu) tłumaczony na kod konkretnego procesora, za pomocą tak zwanej **maszyny wirtualnej** (p.4.5.5). Do pisania i uruchamiania tego typu programów potrzebna będzie odpowiednia **platforma programistyczna** (a dokładniej programistyczno – uruchomieniowa) na przykład „.NET" (p.4.5.6).

Tworzenie nowoczesnych programów dialogowych z graficznym interfejsem użytkownika, będzie także wymagało wykorzystania bibliotek gotowych obiektów, funkcji i procedur, dołączanych do języków programowania lub zawartych w oddzielnych pakietach zwanych **API** – *Application Programming Interface* (p.4.5.4), albo wchodzących w skład zestawu narzędzi programistycznych **SDK** – *Software Development Kit* (p.4.5.1), czy platformy programistycznej (p.4.5.6).

Jeśli nasz program ma działać „**w chmurze**" czyli na serwerze dostępnym przez Internet to jednym z rozwiązań jest tworzenie go bezpośrednio w chmurze – przy wykorzystaniu odpowiedniego oprogramowania (np.: Windows Azure) a innym zastosowanie **emulatora** – symulującego tą chmurę. Emulator (zwany także maszyną wirtualną) potrzebny będzie także przy tworzeniu na stacjonarnym komputerze **aplikacji dla urządzeń mobilnych** jak smartfony i tablety.

Po tych kilku ogólnych informacjach dotyczących programowania i wprowadzeniu pojęć - objaśnianych w kolejnych podrozdziałach - zaczniemy od najniższego poziomu czyli od tworzenia prostych algorytmów i programów obliczeniowych z wykorzystaniem języka specjalnie opracowanego do tego celu a mianowicie języka BASIC.

4.1. ETAPY

Tworzenie programu realizującego obliczenia – podobnie jak projektowanie urządzenia – trzeba zacząć od **sprecyzowania celów** jakie mają być osiągnięte, opisania niezbędnych do tego danych **danych** oraz **operacji** jakim mają one podlegać.

Zaczynamy więc zawsze od następujących pytań:

- **Co ma być wynikiem** działania programu i jaką ma mieć postać ?
- **Jakie dane wejściowe** są niezbędne do otrzymania takich wyników ?
- Na czym polega **ogólna koncepcja rozwiązania ?**
- **Jakie pomocnicze zmienne** i struktury danych będą do tego potrzebne ?
- **Jakie operacje** muszą być realizowane **i w jakiej kolejności** ?

Odpowiedzi na te pytania prowadzą do utworzenia precyzyjnego zapisu koncepcji działania programu czyli **algorytmu**.

Następny etap to **zapisanie algorytmu w wybranym języku programowania** - zwane czasem **kodowaniem**, dzięki czemu algorytm staje się programem o postaci źródłowej. Wybieramy taki język programowania jaki znamy i którego translator mamy zainstalowany na naszym komputerze.

Tekst programu piszemy w edytorze, a po jego zapisaniu do pliku dyskowego możemy uruchomić procesy translacji i konsolidacji aby otrzymać wykonywalną postać programu – najczęściej jako plik typu *.exe – a następnie spróbować nasz program uruchomić.

Przeważnie będziemy musieli nasz program kilka razy poprawiać, powtarzając cykl:

- **edycja** (pisanie lub poprawianie tekstu programu),
- **translacja i konsolidacja** (tłumaczenie na kod procesora i dołączanie podprogramów bibliotecznych),
- **uruchamianie** (egzekucja – wykonanie, ang.: *execution*)

4.2. PROGRAMOWANIE TO NIE OBLICZENIA

Dość często programowanie kojarzy się początkującym z obliczeniami wykonywanymi na kalkulatorze. Jest to jednak mylne skojarzenie.

Czym więc różni się programowanie od obliczeń na kalkulatorze? Istotne różnice są następujące:

1) Obliczenia na kalkulatorze wykonywane są na konkretnych liczbach, natomiast **program musi operować na zmiennych** reprezentujących dane poddawane przetwarzaniu lub sterujące tym przetwarzaniem. Twórca programu musi ściśle określić **rolę** każdej zmiennej i zastanowić się czy ta zmienna jest niezbędna i nie dubluje roli innej, już istniejącej zmiennej.

Konstruując algorytm działania programu, musimy cały czas mieć na uwadze role poszczególnych zmiennych, przypisując im określone operacje. **Nazwy zmiennych** są wprawdzie mniej istotne niż ich role, ale właściwe dobranie nazw ułatwi nam przypominanie sobie tych ról, a więc i konstruowanie algorytmu.

2) Na kalkulatorze wyznaczamy rozwiązanie pojedynczego zadania, natomiast program - dzięki użyciu zmiennych – stanowi **uogólniony model matematyczny** i może być **używany wielokrotnie dla różnych zestawów danych**, inaczej mówiąc służy do rozwiązywania określonej **klasy zagadnień**.

3) Ponieważ program piszemy dla dowolnych danych (a program dialogowy musi dodatkowo uwzględniać różne akcje użytkownika) więc **musimy przewidzieć wszelkie sytuacje** jakie mogą nastąpić i zaplanować dla nich odpowiednie reakcje komputera – chociażby w tak banalnych sytuacjach jak dzielenie przez zero czy pierwiastkowanie liczby ujemnej.

4) Kolejna różnica wynika z faktu, że w programach występują nie tylko różne **struktury danych** ale także różne **struktury operacji**, które programista musi dostrzec i zapisać w krótki, syntetyczny sposób, na przykład stosując **pętle** programowe.

Szczególnym typem programów są **programy dialogowe**, które oprócz spełniania swych głównych zadań mają także reagować na różne akcje użytkownika czyli **zdarzenia**. Zdarzenia dotyczą **obiektów** – zazwyczaj widzianych na ekranie, ale także sygnalizują stany realizowanych procesów. Zdarzenia mogą być także powodowane działaniem aparatury kontrolno-pomiarowej – jeśli komputer współdziała z oprzyrządowaniem procesów technologicznych lub eksperymentów naukowych.

Obiekty (m.in. widziane na ekranie przyciski, suwaki, okienka, ...) – oprócz **zdarzeń** – mają także przypisane **własności** (kolor, wielkość, położenie, ...), oraz procedury zwane **metodami**. Własności mogą być ustalane wstępnie ale także mogą być uzależnione od akcji użytkownika. Procedury odpowiadające metodom mogą być wywoływane zarówno przez bezpośrednie akcje użytkownika jak i wskutek występowania szczególnych wartości danych.

Zanim dojdziemy do obiektów i programowania obiektowego musimy zacząć od trudnej sztuki budowania najprostszych algorytmów.

Tworzenie programów złożonych z poleceń uporządkowanych w określonej kolejności, pisanych w klasycznych językach programowania (istniejących od lat 70-tych XX wieku) jak Algol, Fortran, BASIC, Pascal, C, określane jest jako programowanie **stacjonarne** (w odróżnieniu od rozproszonego lub współbieżnego) oraz jako **imperatywne** (poleceniowe), o metodyce programowania linearnej (instrukcje jedna po drugiej), strukturalnej (bloki instrukcji), proceduralnej (podprogramy). Istnieją inne metodyki i tzw. paradygmaty programowania określane przymiotnikami: obiektowe, zdarzeniowe, funkcyjne, logiczne, agentowe, komponentowe - scharakteryzowane krótko w rozdziale 4.4.

Pragnę uspokoić czytelnika, że na szczęście nie będziemy się zagłębiać w tą obco brzmiącą i niezbyt przyjazną terminologię, gdyż naszym celem jest poznanie najprostszych zagadnień programowania i to bardziej od strony praktycznej niż teoretycznej.

4.3. JĘZYKI PROGRAMOWANIA

Dla człowieka najwygodniejszy jest język naturalny, który jednak jest złożony i często bywa nieprecyzyjny. Komputer natomiast rozumie tylko rozkazy w binarnym kodzie procesora, praktycznie nieczytelnym dla człowieka. Aby zapewnić komunikację między nimi konieczne stało się więc utworzenie **sformalizowanych języków programowania**, znacznie prostszych niż język naturalny ale za to bardziej wyspecjalizowanych i precyzyjnych.

Jak podaje Encyclopedia Britannica [38] – istnieje ponad 2000 języków programowania ale tylko stosunkowo niewiele z nich jest szeroko rozpowszechnionych.

Języki programowania, różnią się obszarem zastosowań oraz poziomem. Im wyższy poziom języka tym jest on dogodniejszy dla człowieka.

Języki programowania są **językami formalnymi** to znaczy mają ściśle określone:

- **alfabet języka** – czyli zbiór dopuszczalnych, niepodzielnych symboli,
- **gramatykę** (inaczej składnię albo syntaktykę ang.: *syntax*) - czyli reguły tworzenia z tych symboli poprawnych wyrażeń i zdań, oraz
- **semantykę** (*semantics*) definiującą znaczenie wyrażeń i poleceń czyli sposób ich interpretowania i realizowania przez komputer.

W pierwszych latach użytkowania komputerów programowało się je w bardzo żmudny sposób używając języka cyfrowych rozkazów **kodu procesora**. Kolejnym krokiem w rozwoju języków było formułowanie tych rozkazów przy użyciu mnemonicznych kilkuliterowych skrótów słów angielskich czyli kodu assemblera. **Assembler'em** nazywany jest zarówno sam kod jak i program tłumaczący ten kod na rozkazy procesora.

Następnie powstały znacznie dogodniejsze dla programistów **języki wysokiego poziomu** (*high level programming language*) wykorzystujące w zapisie poleceń określone słowa (najczęściej z języka angielskiego) jako tak zwane **słowa kluczowe** (*keywords*).

A więc mamy stąd pierwszy podział języków programowania na:

- **języki niskiego poziomu** zwane też **zorientowanymi maszynowo** – zbliżone do assemblera i dobrze odwzorowujące kod procesora ale niewygodne dla ludzi;
- **języki wysokiego poziomu –** dogodniejsze dla ludzi.

Kolejny podział rozróżnia języki:

- **imperatywne** – wyrażające program jako listę rozkazów do wykonania w określonej kolejności, stad nazwa imperatywne – czyli **rozkazowe**.
- **deklaratywne** – w których **deklaruje** się **zależności** oraz **cele**, które należy osiągnąć, nie podając jednak wprost sposobu ich osiągnięcia.

Większość języków programowania, omawianych w niniejszym podręczniku, to języki imperatywnych. Są to: **PHP** (p.2.5.2.), **BASIC** (p.4.6), **Visual BASIC** (rozdz.7), **MATLAB** (rozdz.8-10), **C** (rozdz.11). Natomiast do deklaratywnych należy **SQL** (p.12.7.1.).

Pod względem łatwości nauczenia się, na pierwszym miejscu jest niewątpliwie BASIC oraz MATLAB na podstawowym poziomie ograniczonym do najprostszych środków. Jednak w MATLAB-ie jest wiele poziomów i kategorii narzędzi i nie wszystkie są łatwe.

Popularne wśród programistów języki: C (1972), oraz C++, C#, PHP należą do trudniejszych.

Do najstarszych, „tradycyjnych" języków, które w swoim czasie uzyskały dużą popularność należą FORTRAN (1957), ALGOL (1958-60), COBOL (1959), LISP (1960).

Specjalnie do celów edukacyjnych opracowano takie języki jak BASIC (1964), Pascal (1970), oraz LOGO (1968). Reprezentują one różne podejścia. BASIC pozwala poczatkującym nie interesować się typami zmiennych i umożliwia szybkie testowanie poszczególnych instrukcji i fragmentów programów w trybie interpretacyjnym. Pascal – odwrotnie – wymusza deklarowanie typów wszystkich zmiennych i jawne dołączanie bibliotek, upomina się o drobne szczegóły jak kończenie poleceń średnikiem, pozwala zagnieżdżać wiele bloków *begin ... end*, słowem „daje solidnie w kość" wymuszając pedantyczny styl programowania. Język LOGO natomiast, w wydaniu szkolnym, to głównie pisanie programów sterujących „żółwiem" rysującym linie w trybie graficznym.

Zaobserwujmy podobieństwa i różnice w poleceniach należących do kilku języków, a realizujących ten sam prosty algorytm obliczania kwadratów dowolnych liczb wpisywanych z klawiatury. Algorytm ten w postaci schematu blokowego pokazano już na Rys. 3.7, natomiast poniżej mamy kolejne postaci zapisu tego algorytmu:

Algorytm obliczania kwadratów liczb – postać 2 – opis słowny:

```
1) Będą użyte zmienne liczbowe: X, Y, oraz zmienna tekstowa: P
2) Wyświetl: „Obliczanie kwadratów. Podaj liczbę:"
3) Wczytaj liczbę do zmiennej X
4) Oblicz: X*X i wynik wstaw do Y
5) Wyświetl tekst „ Kwadrat liczby =" oraz wartość Y
6) Wyświetl pytanie: „Czy nowe obliczenie? (T/N):"
7) Wczytaj znak z klawiatury do zmiennej P
8) Jeśli wartość P = „T" lub P = „t" to skocz do (2)
9) KONIEC
```

W tej postaci algorytmu zastosowano skok warunkowy, który w programie może być zrealizowany przy pomocy instrukcji IF współpracującej z instrukcją skoku bezwarunkowego GOTO:

Algorytm obliczania kwadratów liczb - postać 3 - program w języku PASCAL:

```
{PR 12}
program kwadraty1;
   uses crt, dos; {- przyłącza biblioteki podprogramów}
   VAR x,y:real;  p:char;  label 2; {- to są deklaracje typów}
BEGIN
   2: writeln('Obliczanie kwadratow.');
   write('Podaj liczbe:'); readln(x);
   y:=x*x;  write('Kwadrat liczby='); writeln(y);
   write('Czy nowe obliczenie? (T/N):'); readln(p);
   if (p='T') OR (p='t') then goto 2
END.
```

Skok bezwarunkowy „GoTo ..." wymaga numerowania poleceń lub zastosowania „**etykiety**" (w naszym programie to cyfra 2) oznaczającej miejsce (a ściślej polecenie) do którego program ma wielokrotnie wracać.

Jednakże, w nowoczesnych językach i metodach programowania, albo nie istnieje instrukcja GoTo albo nie jest zalecane jej stosowanie. Zamiast tego lepiej powyższy algorytm dostosować do użycia **pętli typu WHILE** która pozwoli wyeliminować GoTo i etykietę. Pętla ta powtarza swe działanie tak długo dopóki spełniony jest warunek zapisany po słowie WHILE. Zobaczmy programy z użyciem tego typu pętli.

Postać 4 - program w języku PASCAL z użyciem pętli WHILE:

```
{PR 13}
program kwadraty2;
  uses crt, dos;
VAR x,y:real; p:char;
BEGIN
p:='T';
while (p='T') OR (p='t')  do
  begin
    writeln('Obliczanie kwadratow.');
    write('Podaj liczbe:'); readln(x);
    y:=x*x;
    write('Kwadrat liczby='); writeln(y);
    write('Czy nowe obliczenie? (T/N):'); readln(p);
  end
END.
```

Postać 5 – to program w języku MATLAB z użyciem pętli WHILE:

```
% PR 14 - Program oblicza kwadraty liczb
p= 't';
while p=='T' | p=='t'
  disp('Obliczanie kwadratow.');
  x = input('Podaj liczbe :');
  y = x^2;
  disp('Kwadrat liczby = '); disp(y);
  p=input('Czy nowe obliczenie? (T/N):', 's');
end
```

Postać 6 – to program w języku C z użyciem pętli WHILE:

```
main()
{
/* PR 15  */
  float x,y; char p;
  p='t';
  while(p=='t'| p=='T')
    {
    printf("\n OBLICZANIE KWADRATOW. Podaj liczbe:");
    scanf("%f",&x);    y=x*x;
    printf("\n Kwadrat liczby= %f",y);
    printf("\n Czy nowe obliczenia? (T/N):");
    p=getch();
    }
}
```

Programy realizujące te same algorytmy w języku BASIC zamieszczone są w p. 4.6.10.

Tworząc dialogowe programy wykorzystujące interfejs graficzny, należy szczególnie wyróżnić **programowanie obiektowe**. **Obiekty** określane są przez swoje **cechy** i posiadają przypisane im procedury zwane **metodami**, a wśród nich procedury **zdarzeniowe** – określające reakcje na określone zdarzenia, na przykład działania myszką lub naciśnięcia klawiszy. Do języków najpopularniejszych (na początku roku 2013 wg *impagina.org/blog/*) umożliwiających programowanie obiektowe należą m.in.: Java, C, C++, PHP, C#, Visual BASIC (z VB.NET), Python, Objective-C, Java Script, Perl, Ruby.

Oprócz przedstawionych już klasyfikacji, języki bywają dzielone ze względu na stosowane w nich narzędzia i metody programowania, nazywane paradygmatami o czym napisano poniżej.

4.4. PARADYGMATY I METODYKI PROGRAMOWANIA

Wraz z rozwojem i upowszechnianiem komputerów następowała ewolucja metod i narzędzi programowania. Koncepcje i zasady budowania programów - w danym okresie powszechnie przyjęte - nazywane są **paradygmatami programowania** (*programming paradigms*). Słowo „paradygmat" pochodzi od greckiego *paradeigma* – wzorzec, przykład.

Na „paradygmat programowania" składa się przeważnie zestaw środków i metod typowych dla określonej grupy języków programowania oraz sposoby ich realizacji. Przeważnie równocześnie współistnieje kilka paradygmatów programowania.

Najważniejsze, współcześnie istniejące, paradygmaty programowania to:

1. **Programowanie imperatywne**. Istotą wykonywania każdego programu jest wykonywanie poleceń wyrażonych w binarnym kodzie procesora. Większość języków programowania wysokiego poziomu (np.: Fortran, Algol, Cobol, BASIC, Pascal, C) to **języki imperatywne** – w których programy buduje się także z **poleceń**, chociaż wyrażanych w czytelniejszej dla człowieka i bardziej kompleksowej postaci.
2. **Programowanie deklaratywne**. Przeciwieństwem języków imperatywnych są **języki deklaratywne**, w których zamiast opisywania sposobu rozwiązania, deklarowane jest to **co** ma być uzyskane. Inaczej mówiąc **opisywany jest rezultat** a nie jak go uzyskać. Do języków deklaratywnych zaliczany jest m.in. **SQL** (do operowania na bazach danych) oraz **Prolog** (umożliwiający programowanie logiczne i stosowany m.in. w zagadnieniach sztucznej inteligencji).
3. **Programowanie proceduralne.** Idee programowania proceduralnego obowiązują od początku istnienia języków wysokiego rzędu. Istotą jest rozłożenie złożonego zadania na składowe. W algorytmie należy wydzielić jasno określone zadania cząstkowe, które będą realizowane przez **podprogramy** (procedury lub funkcje). W każdym podprogramie **zmienne** potrzebne do jego realizacji powinny być **lokalne**, czyli niedostępne z zewnątrz, a pobieranie danych i zwracanie wyników powinno odbywać się poprzez **parametry** wejściowe i wyjściowe. Nie jest natomiast zalecane korzystanie ze zmiennych **globalnych** (ogólnie dostępnych). Wczesne wersje języka BASIC stanowią przykład języka nieproceduralnego gdyż umożliwiały wprawdzie tworzenie podprogramów ale tylko działających na zmiennych globalnych.

4. **Programowanie strukturalne** – polega na utrzymywaniu przejrzystej struktury programu, tak aby dało się łatwo zaobserwować **bloki** operacji sekwencyjnych, bloki **pętli** (*for .. end, while .. end*) oraz bloki **instrukcji warunkowych** (*if .. then .. else .. end*). Nie należy natomiast używać instrukcji skoku bezwarunkowego (*GoTo*) istniejącej na przykład w BASIC-u a także Pascalu, a szczególnie skoku wstecz.
5. **Programowanie obiektowe** (*Object-Oriented Programming*) – polega na definiowaniu i wykorzystywaniu obiektów komunikujących się pomiędzy sobą w celu wykonywania zadań. Każdy **obiekt** reprezentowany jest przez **rekord danych** składający się z pól których wartości reprezentują **cechy obiektu** oraz może podlegać działaniom określonym przez **procedury zwane metodami**. Tak więc w odróżnieniu od tradycyjnego programowania proceduralnego, **dane i procedury** są ze sobą bezpośrednio związane.
6. **Programowanie zdarzeniowe** - dominuje w graficznych interfejsach użytkownika (np.: w Ms Windows). Zdarzenia to zarówno żądania wykonania różnych akcji, zgłaszane z urządzeń peryferyjnych lub z sieci (np.: działania myszką i klawiszami, żądania transmisji plików) jak i meldunki o wykonaniu operacji (np.: zakończeniu przesyłania pliku czy zwolnieniu bufora drukarki). Ponieważ kolejność zdarzeń może być bardzo różna więc utrudnia to programowanie. Ważne jest też aby nie obsługiwać zbyt długo danego zdarzenia, bo blokuje się w ten sposób obsługę innych. Stosowana jest w tym celu wielowątkowość i inne techniki.
7. **Programowanie agentowe** [65] – to rozwinięcie idei programowania obiektowego. Polega na tworzeniu wielu samodzielnych agentów (*Software Agent*) wykonujących określone zadania. W programowaniu obiektowym zakłada się, że każdy obiekt zwróci poprawne dane, natomiast agent może nie dostarczyć danych lub zwrócić błędne dane, dlatego pożądane jest, aby kilku agentów wykonywało to samo działanie. Właściwości agentów można więc w pełni wykorzystać jeśli zostaną połączone w zespół nazywany systemem wieloagentowym (*Multi Agents System*).W związku z tym, system agentowy może dać wynik przybliżony. Systemy agentowe są przystosowane do uruchamiania w dużych lub niepewnych środowiskach (także heterogenicznych), na przykład sieciach komputerowych, gdzie może zajść awaria lub do śledzenia rezultatów transakcji na giełdach. Mogą one dawać wyniki przybliżone.

4.5. ŚRODOWISKA I INTERFEJSY PROGRAMISTYCZNE

Oprócz symboli i reguł określonych przez język programowania, programista potrzebuje konkretnych narzędzi umożliwiających m.in.:

- wprowadzenie poleceń tworzących program,
- kontrolę poprawności składni poleceń,
- wyświetlanie objaśnień dotyczących języka („help”),
- zapisanie programu do pliku w pamięci masowej komputera,
- translację programu na postać binarną (kod procesora),
- pomoc w wykrywaniu błędów na etapie translacji,

- dołączanie standardowych podprogramów,
- uruchamianie i testowanie programu,
- pomoc w wykrywaniu błędów na etapie działania programu.

Te funkcje wykonują programy narzędziowe tworzące łącznie „środowisko programistyczne". Czasem programista musi sam skompletować sobie odpowiednie programy:

- **edytor** – do pisania proramu,
- **translator** (kompilator lub interpretator) – do tłumaczenia na kod procesora,
- **konsolidator** (linker) do dołączenia standardowych podprogramów,
- **debugger** – ułatwiający wykrywanie i usuwanie błędów,
- oraz **podręcznik** – z opisem języka.

Lepiej jednak gdy wszystko co potrzebne ma skupione w jednym systemie zapewniającym prawidłową współpracę wszystkich elementów. Taki system nazywa się „**zintegrowanym środowiskiem programistycznym"** określanym akronimem IDE.

4.5.1. SDK – ZESTAW DLA PROGRAMISTY

SDK (*Software Development Kit*) – to zestaw elementów ułatwiający tworzenie programów z wykorzystaniem danej biblioteki (np. DirectX SDK) lub dla danego systemu operacyjnego czy sprzętu. Na SDK w tym znaczeniu najczęściej składają się:

- skompilowane biblioteki funkcji,
- kody źródłowe biblioteki (w zależności od licencji i typu SDK),
- kompilatory,
- dokumentacja,
- pliki nagłówkowe dla danego języka programowania,
- przykładowe kody źródłowe.

Ważną cechą SDK jest licencja, na której jest ono udostępniane. Z niej wynikają dodatkowe ograniczenia w korzystaniu (np. zakaz wykorzystania do pewnych zastosowań) oraz ponoszone koszty (np. opłaty uzależnione od sprzedaży wytworzonego rozwiązania).

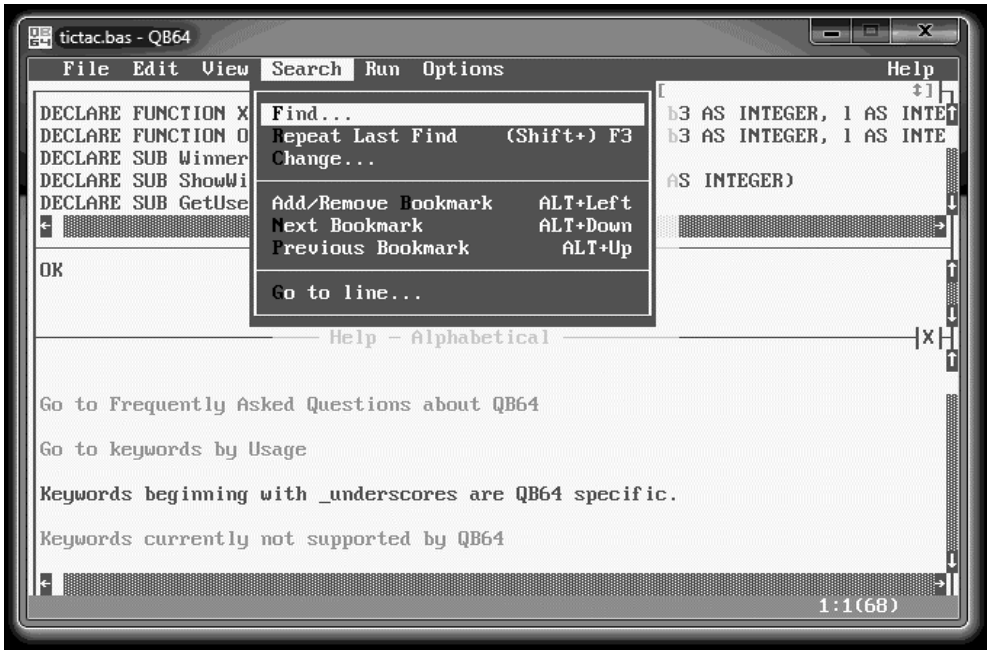
W nazwie lub zamiast pierwszego członu nazwy może być zawarta nazwa biblioteki lub obsługiwanego urządzenia.

Przykłady:

- Driver Development Kits (DDK) – dla sterowników urządzeń komputerowych,
- PalmOS Development Kit – dla palmtopów pracujących pod kontrolą PalmOS,
- Facebook C# SDK – dla pisania aplikacji Facebooka,
- JDK (Java Development Kit) – dla pisania programów w języku Java,
- Android SDK – zestaw narzędzi programistycznych dla systemu Android.

4.5.2. IDE - ZINTEGROWANE ŚRODOWISKO PROGRAMISTYCZNE

Zintegrowane środowisko programistyczne – po angielsku: *Integrated Development Environment* – czyli w skrócie **IDE** - jest to zestaw narzędzi służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania, ale dostępnych z jednej aplikacji (a czasem kilku), stanowiącej centrum sterowania, wyposażonej w menu, edytor do pisania programów i system pomocy. Można więc powiedzieć, że jest to SDK uzupełnione o wygodny interfejs użytkownika, ułatwiający korzystanie z elementów SDK.



Rys. 4.1. Środowisko IDE dla języka BASIC w wersji QB64

Przykładami IDE są systemy: QBASIC i QB64, Turbo Pascal, Turbo C.

Rozwinięciem koncepcji IDE – dla języków obiektowych - jest RAD.

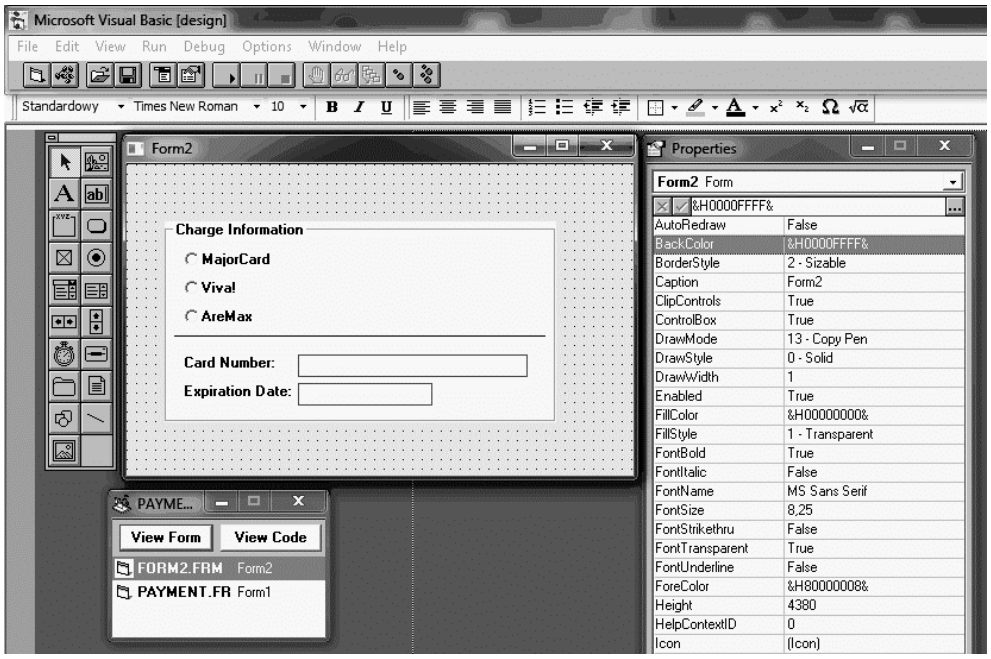
4.5.3. RAD - RAPID APPLICATION DEVELOPMENT

Rapid Application Development czyli RAD - oznacza "szybkie tworzenie aplikacji". Jest to środowisko programowania udostępniające programiście zestawy gotowych komponentów, takich jak: elementy dialogowe, elementy zapewniające dostęp do baz danych, edycję tekstów, edycję grafiki, przeglądanie Internetu i in.

Wygląd aplikacji projektuje się przeciągając tak zwane „kontrolki” z palet elementów do obszaru okna projektowanej aplikacji przy użyciu myszki.

Narzędziami typu RAD są m.in.: Delphi (rozwinięcie obiektowego Pascala), oraz narzędzia zgrupowane w pakiecie **Microsoft Visual Studio**: Visual C#, Visual C++, **Visual BASIC** (Rys. 4.2), Visual J#, Visual Web Developer ASP.NET, Visual F#. Jest również okrojona, przeznaczona głównie dla studentów i hobbystów, wersja darmowa tego pakietu o nazwie Visual Studio Express zawierająca: Visual BASIC Express, Visual C++ Express, Visual C# Express, Visual Web Developer Express. Licencja pozwala na tworzenie programów komercyjnych.

Narzędziami typu RAD dla Linuxa są Lazarus i Kylix a narzędziem RAD niezależnym od platformy jest Qt Designer wspomagający programowanie w C++.



Rys. 4.2. Środowisko typu RAD – Visual BASIC

4.5.4. WINDOWS API

Windows API (*Windows Application Programming Interface*), jest zbiorem funkcji, które można wykorzystywać w programach tworzonych dla systemu Ms Windows. W roku 1993 Microsoft wprowadził wersję 32 bitową (Win32 API) wraz z systemem Windows NT 3.0., a poprzednio istniała wersja 16 bitowa. W bardzo obszernym zbiorze są m.in. funkcje do tworzenia okien dialogowych i ich elementów a także funkcje umożliwiające dostęp do sieci, do innych aplikacji, czy do składników sprzętu komputera. Pliki mają rozszerzenia nazw „.dll" np. kernel32.dll, user32.dll, gdi32.dll czy wsock32.dll.

Większość środowisk programistycznych umożliwia korzystanie z WinAPI, poprzez dołączanie do programów tak zwanych plików nagłówkowych i wywoływanie ich funkcji.

4.5.5. WIRTUALNA MASZYNA JAVY JVM

Ogólnie, pojęcie „wirtualna maszyna" - w domyśle cyfrowa czyli „sztuczny komputer" - oznacza program, który na danym komputerze pozwala symulować (emulować) działanie innego komputera, lub w danym systemie operacyjnym emuluje inny system.

Wraz z upowszechnieniem języków Java oraz Java Script powstała idea „wirtualnej maszyny Java" czyli JVM (*Java Virtual Machine*). Chodziło o to aby nie był potrzebny osobny kompilator języka Java dla poszczególnych typów komputerów i systemów operacyjnych. Wymyślono, że kompilator języka Java będzie taki sam dla wszystkich, bo będzie tłumaczył na **wspólny dla wszystkich język pośredni *Java Byte Code* - JBC czyli „kod bajtowy Javy"**. Najczęściej z Internetu pobiera się programy i funkcje jako pliki „.class", już przetłumaczone na ten kod bajtowy.

Drugi etap translacji – z kodu bajtowego Javy na kod procesora – będą realizowały właśnie wirtualne maszyny Javy, a dokładniej interpretery stanowiące ich część. Darmowe oprogramowanie JVM - dostosowane do danego systemu operacyjnego – pobiera się z Internetu jako dodatek (tzw. wtyczkę) do przeglądarek internetowych. Dzięki temu uzyskano **przenaszalność** i niezależność programów w kodzie bajtowym Javy od systemów operacyjnych i platform sprzętowych. Eksperci szacują liczbę urządzeń wyposażonych w JVM na kilka miliardów.

Maszyna wirtualna Javy ukrywa więc różnice między poszczególnymi platformami tak, że program w kodzie JBC można uruchomić praktycznie na każdym sprzęcie.

Wirtualne maszyny Javy w opisanej postaci to tak zwane **środowiska uruchomieniowe** nazywane **JRE** – *Java Runtime Environment*. W bogatszej postaci mogą także zawierać darmowe środowisko programistyczne **JDK** – *Java Development Kit* - do tworzenia programów w Javie lub innych językach wysokiego poziomu i tłumaczenia ich na JBC.

Język Java powstał w firmie Sun Microsystems a obecnym właścicielem znaku towarowego i technologii jest firma Oracle Corporation, która jednak (tak jak i poprzednia) udostępnia JRE i JDK nieodpłatnie.

4.5.6. PLATFORMA .NET I VISUAL STUDIO

Platforma .NET, a po angielsku *.NET Framework* to środowisko uruchomieniowe rozwijane przez firmę Microsoft w oparciu o opisane powyżej **idee JVM** - wirtualnej maszyny Java. Jest to też w pewnym stopniu platforma programistyczna bo zawiera WinForm (formatki okienkowe) – czyli graficzny interfejs programowania aplikacji.

Zasada działania platformy jest następująca – piszemy program w dowolnym języku programowania (dostosowanym do .NET), następnie jest on kompilowany do kodu pośredniego i w takiej postaci rozpowszechniany, a podczas pierwszego uruchomienia kod pośredni jest tłumaczony na kod procesora i optymalizowany przez kompilator (CLR) wchodzący w skład platformy .NET. W skład platformy wchodzą też definicje obiektów i funkcji zwane klasami. Idea jest więc taka sama jak w JRE tylko inaczej się to nazywa i jest bardziej rozbudowane. Język pośredni nazywa się **CIL** - ***Common Intermediate Language*** (Wspólny Język Pośredni, a nie Java Byte Code).

Środowiskiem programistycznym rozwijanym w Microsoft dla .NET jest **Visual Studio** wyposażone między innymi w kompilatory języków Visual **C#**, Visual **C++**, **Visual BASIC**, Visual **J#**, Visual Web Developer ASP.NET, Visual F#.

Dla studentów i pracowników (także AGH) Visual Studio jest dostępne za darmo w MSDN (http://msdn.imir.agh.edu.pl). W uboższej, darmowej dla wszystkich - lecz pozwalającej na tworzenie programów komercyjnych - wersji o nazwie **Visual Studio Express** są: Visual BASIC Express, Visual C++ Express, Visual C# Express, Visual Web Developer Express.

Jak podaje Wikipedia – istnieje już ponad 40 języków zgodnych z platformą .NET

4.5.7. CASE I DIAGRAMY UML

CASE *Computer-Aided Software Engineering*) - oprogramowanie używane do komputerowego wspomagania projektowania oprogramowania. Oprócz wspomagania samego procesu pisania i uruchamiania programów (jak poprzednie narzędzia), obejmuje więc bardzo ważne i trudne fazy: **analizy problemu** oraz **projektowania systemu** informatycznego i tworzenia **dokumentacji** a także projektowania i dokumentowania niezbędnych zmian.

Jednym z typowych narzędzi stosowanych w CASE są narzędzia do modelowania w języku UML lub podobnych językach.

UML (*Unified Modeling Language*) - czyli **Zunifikowany Język Modelowania** – to metodyka opisywania różnego rodzaju systemów i procesów przy pomocy **diagramów** zawierających określone **symbole**. Zależnie od wersji języka i stopnia skomplikowania modelu stosuje się od kilku do kilkunastu typów diagramów.

W projektowaniu systemów informatycznych, najczęściej wykorzystywane diagramy to:

- klas (*class diagram* - najczęściej spotykane),
- obiektów (*object diagram*),
- czynności (*activity diagram*),
- przypadków użycia (*use case diagram*),
- sekwencji (sequence diagram).

4.5.8. NARZĘDZIA PROGRAMOWANIA URZĄDZEŃ MOBILNYCH

Tworzenie aplikacji dla urządzeń mobilnych jak smartfony, tablety czy palmtopy, ale także współczesne telewizory z dostępem do Internetu (Smart TV), przebiega podobnie jak programowanie komputerów, jednak jest też uzależnione od cech urządzenia i jego urządzeń wejściowych i wejściowych oraz od systemu operacyjnego pod którego kontrolą pracuje dane urządzenie. Tabela 4.1 pokazuje rozpowszechnienie najpopularniejszych systemów operacyjnych urządzeń mobilnych, poprzez ich udziały w rynku tego rodzaju urządzeń. Jak widać dominującą i rosnącą rolę odgrywa system Android.

Android [4] – to system operacyjny dla urządzeń mobilnych takich jak telefony komórkowe, smartfony, tablety PC i netbooki. Android został oparty na jądrze Linuksa oraz oprogramowaniu na licencji GNU.

Początkowo był rozwijany przez niewielką firmę Android Inc. kupioną w roku 2005 przez Google. Od roku 2005 Android jest rozwijany przez konsorcjum **Open Handset Alliance**, w którego skład wchodzą m.in. Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, Sprint Nextel oraz Nvidia.

Konsorcjum OHA udostępnia od roku 2007 zestaw narzędzi programistycznych **Android SDK** [15], przy pomocy którego użytkownicy mogą tworzyć aplikacje dla Androida i udostępniać je lub sprzedawać poprzez portal Google Play (wcześniej Android Market), gdzie jest już dostępnych ponad 450 tys. aplikacji. Tworzenie takich aplikacji mogą też ułatwić wspomniane dalej wersje języka BASIC dla systemu Android. Darmowo dostępnym narzędziem, pozwalającym tworzyć aplikacje dla Androida jest też opisywany dalej MIT App Inventor (Rys. 4.5).

Firma Google zorganizowała w r.2008 konkurs na aplikacje dla systemu Android przeznaczając na ten cel 10 milionów dolarów. Zgłoszono 1788 aplikacji z ponad 70 krajów.

Tabela 4.1. Domiujące systemy operacyjne urządzeń mobilnych

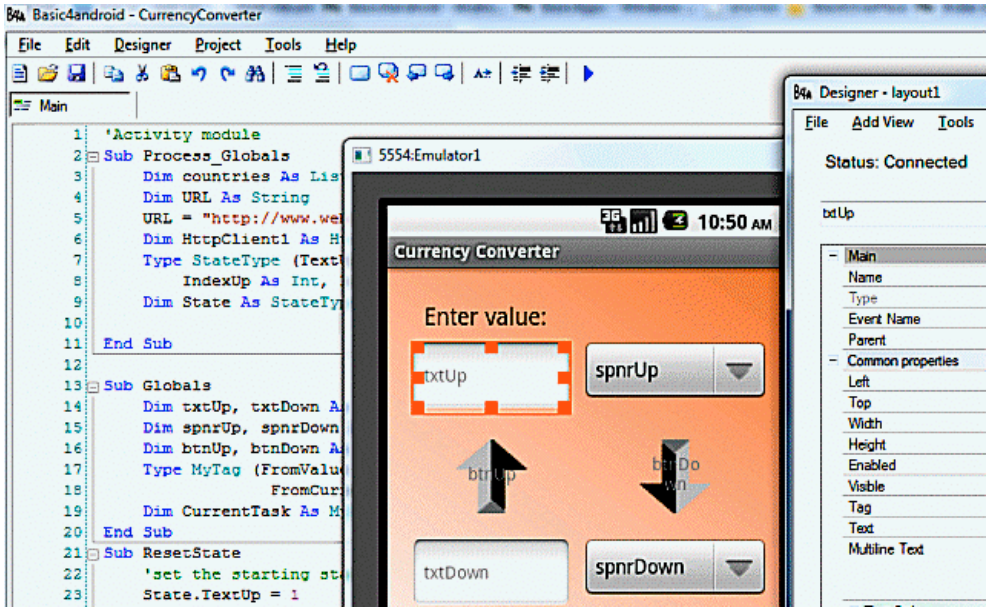
System	**Udział w rynku**	
	Maj 2010	II kwartał 2012
Android	33%	64,1%
Symbian	31%	5,9%
Apple iOS	16%	18,8%
RIM Black Bery	14%	5,2%
Ms Windows Mobile/Phone	3%	2,7%
Źródło:	*mobileos.cba.pl*	*komorkomania.pl*

Inne, mniej popularne systemy operacyjne dla urządzeń mobilnych [4] to m.in.:

- **Bada** - system operacyjny dla telefonów komórkowych firmy Samsung, zaprezentowany w listopadzie 2009.
- **iOS** – system dla telefonów iPhone firmy Apple, dla którego istnieje m.in. ponad 200 tys. gier i aplikacji (głównie rozrywkowych) dostępnych na portalu App Store. Narzędzia dla programistów (developerów) są w witrynie internetowej [16]
- **PalmOS** - obecny od roku 1996 i dość długo wiodący na rynku palmtopów. Posiada wbudowane aplikacje: Date Book (terminarz), Address Book (kontakty), ToDo (zadania) oraz Memo Pad (notatnik). Wprowadzanie tekstu odbywa się przy pomocy rysika dzięki opatentowanemu programowi do rozpoznawania pisma odręcznego Graffiti®. Rozpoznawane są uproszczone znaki alfanumeryczne przypominające litery, co umożliwia dość szybkie wpisywanie tekstów. Do palmtopa mogą być podłączane zewnętrzne klawiatury. Palm OS konkuruje z nowocześniejszym systemem Pocket PC przewyższając go jednak szybkością działania i stabilnością. Powstała dla niego duża liczba aplikacji począwszy od gier poprzez pakiety biurowe, na specjalizowanych aplikacjach medycznych kończąc.

- **Symbian** – tracący już na popularności system operacyjny typu open source (stworzony w oparciu o system EPOC komputerka naręcznego firmy Psion PLC) rozpowszechniany przez konsorcjum Symbian, w skład którego wchodzą m.in.: Nokia, Motorola, Siemens, Sony Ericsson. Jest systemem wielozadaniowym. Aplikacje dla telefonów i palmtopów na platformę Symbian można pisać m.in. w językach C++ i Java, dzięki udostępnianym na stronie producenta narzędziom SDK.
- **Windows Phone** – system operacyjny firmy Microsoft, następca Windows Mobile zaprezentowany w r.2010. Zawiera oprogramowanie biurowe, umożliwia obsługę Internetu oraz multimediów. Aplikacje na system Windows Phone 7 mogą być tworzone m.in. z użyciem **Microsoft Visual Studio 2010**. Ze stron WWW firmy Microsoft można też pobrać bezpłatny pakiet narzędzi - **Windows Phone Developer Tools**[4].

Do programowania smartfonów i tabletów [14] używane są różne języki, m.in. Java, Java Script, Objective C i Visual BASIC. Rys. 4.3 pokazuje komercyjny BASIC4android czyli wersję języka Visual BASIC przeznaczoną specjalnie do tworzenia programów dla telefonów komórkowych (dokładniej smartfonów) pracujących pod kontrolą systemu Android.



Rys. 4.3. BASIC4android – język BASIC dla systemu Android (źródło: www.freevistafiles.com)

Oprócz tej komercyjnej wersji BASIC-a działającej na komputerach, można pobrać, bezpośrednio na tablet czy smartfon, jedną z darmowych wersji języka BASIC, pracującą pod kontrolą systemu Android i dostosowaną do możliwości urządzenia.

Jedną z takich **darmowych wersji** jest **RFO BASIC!** opracowany przez Paula Laughton-a z „**R**ichard **F**eynman **O**bservatory".

W języku **RFO BASIC!,** oprócz tradycyjnych możliwości BASIC-a, programista ma do dyspozycji [18] polecenia umożliwiające m.in. obsługę:

- struktur danych jak: tablice, **listy, kolejki, stosy** i in.
- grafiki i **ekranu dotykowego** łacznie z klawiaturą ekranową,
- interfejsów użytkownika w **HTML** i **JavaScript**,
- komunikowania się z bazami danych **SQL**,
- korzystania z **GPS**,
- **sensorów** urządzenia (jak żyroskop czy akcelerometr),
- nagrywania, generowania dźwięków oraz generowania i **rozpoznawania mowy**,
- aparatu fotograficznego,
- przegladarki internetowej, klienta **FTP** oraz **TCP/IP**
- szyfrowania,
- **Bluetooth**,
- wysyłania **SMS** i **e-mail**,
- prowadzenia rozmów telefonicznych.

4.5.9. PROGRAMOWANIE WIZUALNE CZYLI PROGRAM Z KLOCKÓW

Programowanie wizualne polega na budowaniu aplikacji z gotowych "klocków" wybieranych z magazynu elementów. Systemy typu RAD są zaliczane do tej kategorii bo tam także „kontrolki” (przyciski, suwaki, listy rozwijalne, ...) ustawia się myszką wybierając z wykazu, jednak kod funkcji i procedur trzeba wpisać.

Od niedawna rozwija się inny typ środowisk programistycznych w którym wszystkie polecenia są w postaci „klocków” czy jak kto woli „puzli”.

Jednym z nich jest zalecany dla szkół podstawowych i średnich system **SCRATCH**. Jest to obiektowy język i środowisko programowania, utworzone w roku 2006 w Massachusetts Institute of Technology (MIT) pod kierunkiem Mitchela Resnicka - pomysłodawcy zabawek Lego MindStorms i twórcy języka StarLogo.

Scratch można pobrać bezpłatnie ze stron http://scratch.mit.edu/.

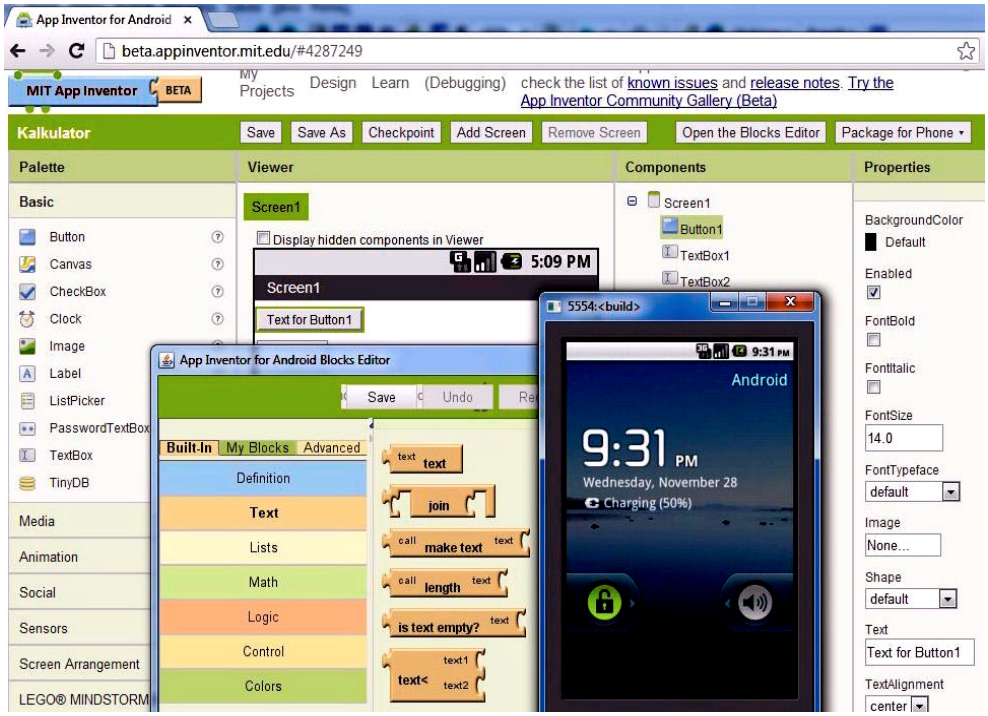
Programowanie sprowadza się do wyboru puzzli symbolizujących elementy języka i układanie ich w określonym porządku. Możliwe jest m.in.:

- projektowanie animowanych postaci (tzw. duszków),
- tworzenie interaktywnych animacji, historyjek i gier z dźwiękiem i kolorami,
- testowanie stworzonych animacji o oknie programu,
- udostępnianie gotowych aplikacji w internecie.

Innym podobnym systemem jest App Inventor opracowany w roku 2010 w firmie Google i przejęty w roku 2011 przez MIT – stąd obecna nazwa: **MIT App Inventor**.

App Inventor umożliwia łatwe tworzenie aplikacji dla mobilnego systemu operacyjnego Android firmy Google. Przed pracą powinniśmy zainstalować pliki pomocnicze opisane w [17]. Po wejściu na stronę appinventor.mit.edu możemy zapoznawać się z podręcznikami lub kliknąć przycisk „Invent - Create Mobile Apps” aby przejść do stron budowy aplikacji.

Rys. 4.4. Programowanie wizualne w systemie SCRATCH



Rys. 4.5. MIT App Inventor – programowanie wizualne dla systemu Android

App Inventor posiada trzy główne komponenty.

Pierwszym jest *App Inventor Designer*. Pozwala on wybierać elementy interfejsu aplikacji (przyciski okienka edycyjne, ...).

Drugi komponent, to edytor bloków przetwarzających dane (*App Inventor Blocks Editor*) z których tworzymy procedury i funkcje, a trzecim jest emulator czyli wirtualny telefon, na którym łatwo można przetestować program. Pliki aplikacji przechowujemy w „chmurze" czyli na serwerach firmy Google. Efekty pracy mogą być natychmiast skompilowane i przetestowane w telefonie lub w emulatorze telefonu z Androidem.

4.6. CHARAKTERYSTYKA JĘZYKA BASIC

4.6.1. OD HISTORII DO TERAŹNIEJSZOŚCI BASIC-A

BASIC jest językiem programowania przeznaczonym szczególnie dla początkujących użytkowników komputerów i do wstępnej nauki programowania. BASIC oznacza w języku angielskim „podstawowy" ale nazwa języka jest także wywodzona od słów: *Beginner's All-purpose Symbolic Instruction Code*.

Pierwszą wersję języka **BASIC** - opracowano w roku 1964 na uniwersytecie w Dartsmouth. Od początku ery mikrokomputerów i komputerów personalnych, praktycznie wszystkie te urządzenia dysponowały różnymi wersjami BASIC-a, najczęściej wbudowanymi w system i zgłaszającymi się natychmiast po włączeniu komputerka.

Firma **Microsoft** standardowo dołączała, kolejne coraz bogatsze wersje języka BASIC, do systemu operacyjnego Ms DOS. Od wersji 5.0 był to **QBASIC**, który pozwala programować strukturalnie, budować własne funkcje i procedury, generować rysunki, animacje, dźwięki i melodie, wykorzystywać okienka i zdarzenia; obsługiwać interfejs szeregowy i t d.

System DOS odszedł do historii ale BASIC jest żywy i popularny chociaż nie zawsze w szkołach i na uczelniach, gdzie preferowane są (czasem zbyt ambitnie) o wiele trudniejsze języki C i C++, nauczane z miernym skutkiem. O popularności BASIC-a może świadczyć choćby liczba zajmujących się nim stron internetowych oraz wciąż rozwijanych wersji, działających we współczesnych systemach operacyjnych. Przykładowo witryna internetowa ***basic.mindteq.com*** podaje listę ponad **100** (przeważnie darmowych) wersji języka BASIC, w większości działających w systemach Windows.

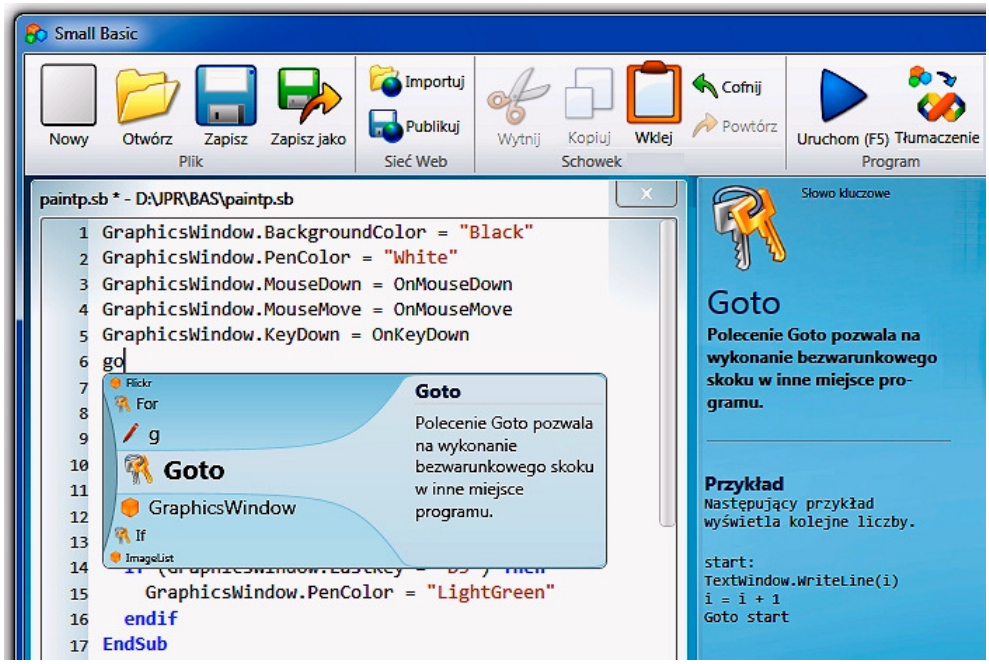
Język QBASIC działa też we współczesnych wersjach Ms Windows, jako język **QB64**, który można w postaci jednego pliku pobrać darmowo ze stron **www.qb64.net**.

QB64 to środowisko łudząco podobne do QBASIC-a, zawierajace taki sam edytor i menu ale wyposażone w nowoczesny kompilator który wytwarza pliki „.exe" także dla 64-bitowych wersji Ms Windows. QB64 umożliwia pisanie nowych programów jak i uruchamianie w Ms Windows starych, napisanych kiedyś dla systemu DOS.

Wiele cech BASIC-a przejął nowoczesny, „okienkowy”, obiektowy i zdarzeniowy język Ms Visual BASIC projektowany już dla środowiska Ms Windows. Powstało szereg wersji Visual BASIC-a a ostatnie wchodzą w skład pakietu Microsoft Visual Studio oraz platformy „.NET” (język VB.NET).

Od roku 2010 dostępny jest także darmowy edukacyjny **Ms Small BASIC** (Rys. 4.6), w wielu wersjach językowych w tym także polskiej.

Ms Small BASIC a także języki w pakiecie Ms Visual Studio wyposażone są **w system pomocy** zwany ***IntelliSense***. Przykład działania Intellisense mamy na Rys. 4.6 **–** gdy wpiszemy fragment słowa to pokazują się możliwe podpowiedzi dalszego ciągu, a także objaśnienia dotyczące instrukcji zaczynającej się od wpisanych liter



Rys. 4.6. Przykładowy ekran Ms Small BASIC

Ewolucję języka BASIC można prześledzić m.in. na podstawie kalendarium wprowadzania ważniejszych wersji BASIC-a przez firmę Microsoft lub społeczność internetową:

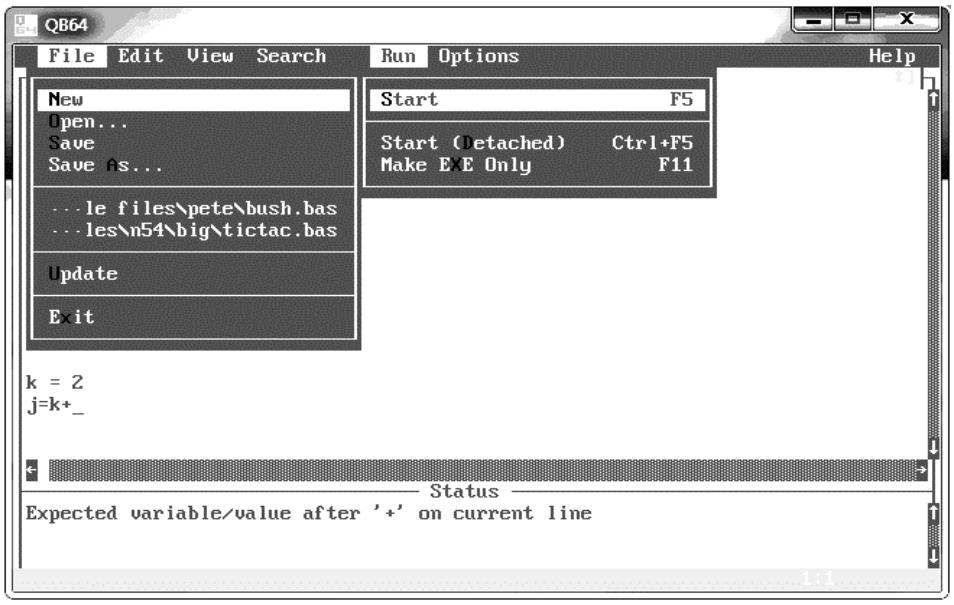
1981 – BASICA,
1982 – Ms BASIC Compiler,
1983 – GW-BASIC,
1991 – Qbasic, Visual BASIC,
2001 – Visual BASIC.NET (VB.NET) unowocześniona wersja Visual BASIC,
2004 – FreeBASIC,
2007 – darmowy QB64 - wstecznie kompatybilny z Qbasic, z kompilatorem,

2008 – Microsoft Small BASIC (zaprojektowany do celów edukacyjnych),
2010 – spolszczone: Visual Studio (wraz z Visual BASIC) oraz Ms Small BASIC,
2012 - 2013 – BASIC4Android oraz darmowy RFO BASIC! dla Androida.

W niniejszym podręczniku, wprowadzone niżej, podstawowe elementy języka BASIC a konkretnie jego wersji QB64, mają w sposób możliwie łagodny wprowadzić nas w istotę programowania komputerów. Każdy, kto pozna podstawowe idee i zasady programowania w tym dość prostym języku, będzie bez większych problemów mógł - w razie potrzeby - opanować programowanie w dowolnym innym języku programowania.

4.6.2. PROGRAMOWANIE W ŚRODOWISKU QB64

Aby próbować pisać i uruchamiać swe pierwsze programy w języku BASIC wystarczy pobrać ze strony www.qb64.net plik wersji instalacyjnej dla Ms Windows i po pobraniu uruchomić – instalując w ten sposób „zintegrowane środowisko programistyczne" czyli IDE (*Integrated Development Environment*) QB64.



Rys. 4.7. Środowisko programistyczne języka BASIC w wersji QB64

Aby móc **używać polskich liter** w komentarzach i drukowanych tekstach należy:

1. po wybraniu z menu: ***Options – Display*** zaznaczyć opcję „***Custom font***" i wpisać ścieżkę do pliku czcionki, zawierającego polskie litery, na przykład w Ms Windows 7 najodpowiedniejszą będzie czcionka pogrubiona Courier New zawarta w pliku: **c:\windows\fonts\courbd.ttf** (nazwę pliku widać we „właściwościach" tej czcionki).
2. wybrać z menu: ***Options - Language - MICSFT_PC_CP852***

Jeśli jednak chcielibyśmy kopiować do QB64 polskie teksty poprzez schowek z Notatnika czy innego edytora, to musimy wówczas ustawić: ***Options - Language - MICSFT_WINDOWS_CP1250***.

Pisząc polecenia składające się na program, możemy czytać podpowiedzi w linii u stanu (*Status*) dołu. Na przykład na Rys. 4.7. jest po angielsku podpowiedź oznaczająca: „oczekiwana jest zmienna lub wartość po znaku '+' w bieżącej linii".

Po napisaniu programu - lub jego fragmentu, którego działanie chcemy przetestować – powinniśmy zapisać nasz program do pliku wykorzystując menu „File – Save as" (choć nie jest to obowiązkowe), a następnie, aby uruchomić program, naciskamy klawisz [F5] lub wykorzystujemy menu „Run – Start" (Rys. 4.7.).

Po uruchomieniu w ten sposób programu następuje etap kompilacji i wytwarzania postaci binarnej wykonywalnej - jako pliku z rozszerzeniem „.exe". Następnie otwiera się okno wyników wyświetlające rezultaty działań instrukcji PRINT, a w przypadku błędu na etapie wykonywania pojawi się dodatkowe okienko z komunikatem o błędzie.

Uwaga: dla zwiększenia czytelności rysunków – kopie ekranów QB64 zamieszczono w negatywach, przekonwertowane na obrazy szare.

4.6.3. ZBIÓR ZNAKÓW JĘZYKA QBASIC

Zbiór znaków, języka QBASIC (lub QB64) pokazuje Tabela 4.2.

Tabela 4.2. Zbiór znaków języka QBASIC

Znaki	Rola
ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz	litery alfabetu angielskiego używane są do tworzenia nazw: zmiennych, tablic, funkcji, podprogramów; duże i małe nie są rozróżniane
0123456789	cyfry dziesiętne
+, -, ***(mnożenie), **/** (dzielenie), **** (dzielenie całkowite), **^** (potegowanie)	znaki działań arytmetycznych:
.	kropka oddziela część ułamkową liczby
( )	nawiasy okrągłe używane są: (a) w wyrażeniach, (b) dla argumentów funkcji, (c) dla indeksów elementów tablic
>, **<**, **>=**, **<=**, **=**, **<>**	operatory relacji
' (apostrof)	rozpoczyna komentarz
, (przecinek), **;** (średnik)	sterują postacią wydruku, a dodatkowo przecinek separuje indeksy i elementy list
: (dwukropek)	oddziela instrukcje zapisane w jednej linii
!(single), **#**(double), **$**(string), **%**(integer), **&**(long)	przyrostki - znaki używane na końcu nazw zmiennych - sygnalizują typ zmiennej

Inne znaki drukarskie, nie uwzględnione w tej tabeli, a w szczególności także polskie małe i duże litery (ą, Ą, ć, Ć, ę, Ę, ł, Ł, ń, Ń, ó, Ó, ...), mogą być używane tylko w łańcuchach tekstowych stanowiących wartości zmiennych i wyrażeń typu *string*.

Nazwy zmiennych i podprogramów można tworzyć tylko **z liter angielskich** i cyfr oraz znaku podkreślnika, przy czym pierwszym znakiem musi być litera, a na końcu nazwy może dodtkowo znajdować się symbol określający typ zmiennej.

4.6.4. ZMIENNE I TYPY WARTOŚCI. WYŚWIETLANIE OBJAŚNIEŃ

O zmiennych napisano już w p.3.5. . Przypomnijmy, że **zmienna** w programie komputerowym to pojemnik, identyfikowany przez swoją **nazwę** i przechowujący konkretną **wartość –** inaczej mówiąc stałą **-** określonego **typu**. „Zmienna przechowuje stałą" (?) - to nie paradoks - wprawdzie zmienna przechowuje w danym momencie wartość stałą, ale w następnym momencie może to być inna wartość stała i stąd określenie „zmienna".

Typ zmiennej można zadeklarować albo przez umieszczenie na końcu jej nazwy odpowiedniego symbolu (patrz Tabela 4.2) albo przy pomocy polecenia DIM.

Objaśnienie tego **polecenia** – tak jak i innych – pokaże się gdy ustawimy kursor w słowie DIM (czy innym słowie kluczowym) i naciśniemy **klawisz [F1]**.

Jeśli nie użyjemy żadnego z tych dwu sposobów deklarowania typu to BASIC domyślnie przyjmie, że zmienna jest typu SINGLE (liczby rzeczywiste pojedynczej precyzji).

Nazwa zmiennej w BASIC-u (podobnie jak w innych językach) może się składać **tylko z liter angielskich, cyfr i znaku „_", ale pierwszym znakiem musi być litera**.

BASIC nie rozróżnia dużych i małych liter. Zalecane jest pisanie małymi literami bo po rozpoznaniu prawidłowo napisanych słów kluczowych BASIC automatycznie zamieni w nich litery na duże.

Wartościami zmiennych liczbowych (typów: *integer, single, double, long*) są **stałe liczbowe**. Przykładowo przypiszmy zmiennym A i B takie stałe:

```
A = -3.234:     B = 7.1e7
```

Pamiętajmy, że w zapisie stałej liczbowej **kropka oddziela część ułamkową liczby** oraz, że można stosować symbol „e" oznaczający „razy 10 do potęgi".

Stałe typu *string* czyli ujęte w cudzysłowy **łańcuchy znaków** drukarskich można przypisywać zmiennym typu *string* - których nazwy zakończone są znakiem dolara:

```
Nap$ = "Naprężenie zginające"
```

4.6.5. WPROWADZANIE DANYCH Z KLAWIATURY

W najprostszych programach będziemy **wprowadzać wartości danych z klawiatury** do określonych zmiennych przy pomocy polecenia INPUT o następującej postaci:

```
INPUT "Tekst żądania danych"; zmienna
```

Na przykład aby wprowadzić z klawiatury wartości (do) zmiennych **Fobc** oraz **Dlug**, trzeba napisać polecenia:

```
INPUT "Obciążenie dźwigni="; Fobc
INPUT "Długość dźwigni="; Dlug
```

Słowo **INPUT** oznacza „wprowadź". Gdy komputer będzie wykonywał polecenie INPUT, to wyświetli na ekranie ***tekst żądania danych*** (zawarty w cudzysłowach) i dzięki temu użytkownik dowie się, nie tylko, że ma przy pomocy klawiatury wpisać wartość danej, ale także jaki sens ma ta dana. Po cudzysłowie zamykającym musi być **średnik** i po nim ***nazwa zmiennej*****,** do której ma być przesłana wprowadzona wartość. Nazwy zmienych nie będą widoczne na ekranie jeśli nie umieścimy ich wewnątrz tekstu żądania danych.

4.6.6. INSTRUKCJA PRZYPISANIA

Instrukcja przypisania, zwana też instrukcją (poleceniem) podstawiania, **pozwala obliczyć wartość wyrażenia i wynik przesłać do określonej zmiennej**. Instrukcja ta ma postać:

zmienna = wyrażenie

Znak „**=**" należy rozumieć jako „przypisz" albo „podstaw". „Wyrażenie" może być wyrażeniem arytmetycznym (algebraicznym), logicznym, tekstowym.

Przykładowo, na podstawie wprowadzonych poprzednio zmiennych **Fobc** oraz **Dlug** można wyznaczyć moment zginający dźwignię:

```
Mzg = Fobc * Dlug
```

Wyrażenie arytmetyczne może składać się ze zmiennych, stałych oraz funkcji, połączonych operatorami arytmetycznymi i ewentualnie pogrupowanych przy pomocy nawiasów okrągłych. Niektóre funkcje przedstawiono w p.4.6.12. . W najprostszych przypadkach wyrażenie może być zredukowane do pojedynczej zmiennej lub stałej:

```
sila5 = 13.7:    sila5 = sila4:    naz_masz$ = "tokarka"
```

Ponieważ najpierw obliczana jest wartość prawej strony (wyrażenia) a potem wynik podstawiany jest do zmiennej zapisanej po lewej, więc mają sens polecenia:

```
x = x+1:   zk = zk*zk+1:   p = p-1:   w = 2*w
```

- bo po prawej zmienna ma dotychczasową wartość a po lewej już utworzy się nowa wartość, jako wynik działania tego polecenia.

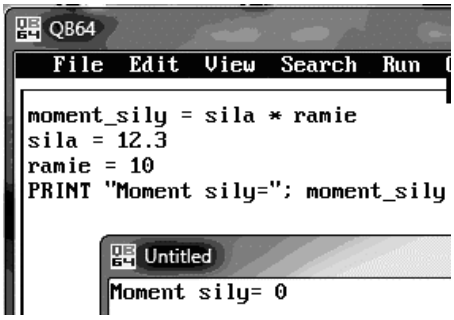
4.6.7. WYPROWADZANIE WYNIKÓW. KOLEJNOŚĆ POLECEŃ

Trzecim z najbardziej podstawowych poleceń (po instrukcjach INPUT oraz przypisania) jest **wyprowadzanie wyników** przy pomocy polecenia **PRINT** wyświetlającego teksty i wartości zmiennych lub wyrażeń. Przed pierwszą instrukcją PRINT dobrze jest wyczyścić ekran poleceniem **CLS** pochodzącym od słów: *clear screen*. Zamiast słowa PRINT można wpisywać pytajnik.

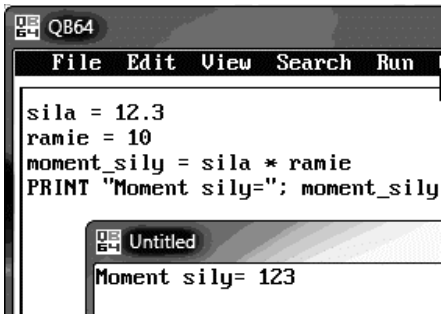
Aby wyświetlić **tekst** – trzeba ten tekst ująć w **cudzysłowy** i umieścić po słowie PRINT. Zamiast tekstu w cudzysłowach może wystąpić zmienna typu tekstowego (np.: A$). Aby wyświetlić **wartość** zmiennej – nie wolno jej ujmować w cudzysłowy bo nie chodzi o wyświetlenie jej nazwy tylko wartości. Po słowie PRINT może być wymienionych wiele elementów do wyświetlenia, przy czym odseparowujemy je od siebie **średnikami** – gdy mają być sklejone na ekranie, albo **przecinkami** – gdy mają być w pewnej odległości od siebie. Przykład:

```
PRINT "Wyniki:  X=";x, " Y=";y, " Z=";z
```

a)

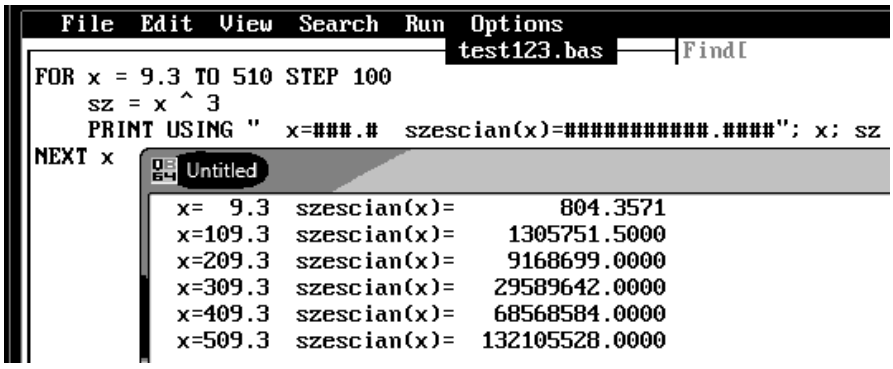


b)



Rys. 4.8. PR 16. Kolejność poleceń: a) zła, b) dobra

Jak już powiedziano: **zasadnicze znaczenie** w algorytmach i programach **ma KOLEJNOŚĆ POLECEŃ !!** Pokazuje to Rys. 4.8 na przykładzie programu o złej kolejności poleceń (Rys. 4.8a) – obliczany moment siły wykorzystuje **nie istniejące jeszcze wartości** bo siła i ramie definiowane są dopiero **poniżej** czyli **w następnych chwilach**. Dlatego **wynik jest równy zero**. Poprawny jest program drugi (Rys. 4.8b) gdzie najpierw wprowadzono wartości danych do zmiennych wejściowych a dopiero potem wykorzystano te zmienne w obliczeniu momentu siły i dlatego uzyskano poprawny wynik.



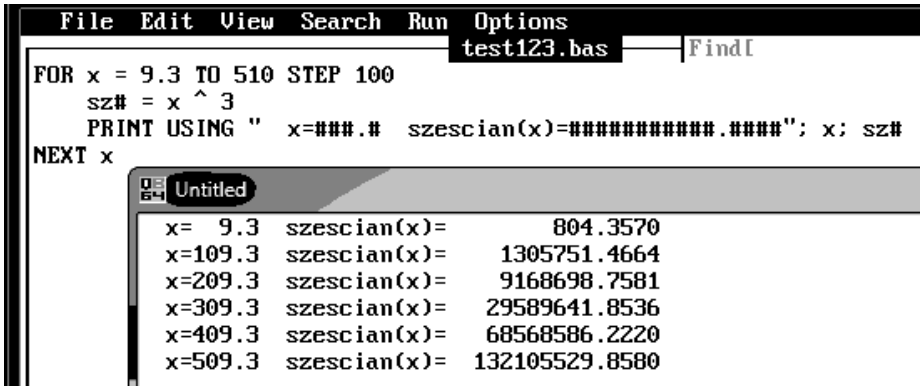
Rys. 4.9. Program PR 17. Przykład instrukcji PRINT USING. Wyniki pojedynczej precyzji.

Do sformatowanego wyprowadzania wartości zmiennych służy instrukcja:

PRINT USING *format* **;** *zmienna* **; ... ;** *zmienna*

gdzie *format* w najprostszym przypadku podaje liczbę cyfr przed i po kropce dziesiętnej, przy pomocy odpowiedniej liczby znaków „#", ale może zawierać też dowolne teksty i spacje które będą przekopiowywane na wydruk, tak jak pokazuje przykład na Rys. 4.9

Zera po kropce dziesiętnej wynikają z faktu, że BASIC **domyślnie** przyjmuje typ ***single*** – **pojedynczej precyzji** – dla zmiennych. Dokładniejszy wynik uzyskamy deklarując przynajmniej typ zmiennej wynikowej „sz" jako ***double*** – **podwójnej precyzji**. Najłatwiej to zrobić dopisując znak „#" na końcu jej nazwy, zgodnie z tym co podaje Tabela 4.2. Po zmianie nazwy „sz" na „sz#" mamy dokładniejsze wyniki jak na Rys. 4.10.



Rys. 4.10. Program PR 18. Przykład PRINT USING. Wyniki podwójnej precyzji.

Polecenie LOCATE ustawia kursor w określonym miejscu ekranu gdzie chcemy wyświetlić napis lub liczbę. Przykładowy program *' PR 19* – symuluje odbijanie się kuli bilardowej od krawędzi (band) stołu:

```
' PR 19 - Symulacja ruchu kuli bilardowej:
CLS
' ---- granice obszaru ruchu kuli:
xmin = 1: ymin = 1: xmax = 80: ymax = 20
FOR y = 1 TO 20
    PRINT STRING$(80, "."):' – obszar wypełniamy kropkami
NEXT y
LOCATE 21, 1, 0: PRINT STRING$(80, "^"):' – to granica dolna
' ---- x,y - to współrzędne położenia kuli:
x = xmin + 1
y = ymin + 1
dx = 1: dy = 1: ' ---- to początkowe przyrosty współrzędnych
' ---- Pętla FOR ... NEXT powtarza przemieszczanie kuli:
FOR j = 1 TO 600
    LOCATE y, x: PRINT "o": ' -- wyświetla kulę
    _DELAY 0.1: ' – przez 0.1 sekundy
     ' --- Przy odbiciu kuli przyrost zmieni znak:
```

```
    IF x = xmax OR x = xmin THEN dx = -dx: SOUND 800, 1
    IF y = ymax OR y = ymin THEN dy = -dy: SOUND 800, 1
    ' ---- SOUND daje dźwięk przy odbijaniu kuli.
    ' ---- aby wymazać obraz kuli, wyświetlimy spację:
    LOCATE y, x: PRINT " ";
    ' ---- no i wyznaczymy współrzędne nowego położenia kuli:
    x = x + dx: y = y + dy:
NEXT j
```

Zarówno powyższe jak i inne instrukcje QB64 opisano na stronach: **http://qb64.net/wiki/**.

4.6.8. INSTRUKCJA WARUNKOWA „*IF* ...”

Instrukcja ta należy do objaśnionych w p.3.5. . podstawowych poleceń i występuje we wszystkich językach programowania chociaż w nieco różniących się od siebie postaciach.

W języku BASIC instrukcja warunkowa IF (a inaczej: „rozgałęzienie warunkowe”) ma następującą budowę:

IF *warunek* **THEN**
blok instrukcji 1
ELSE
blok instrukcji 2
END IF

Warunek to wyrażenie logiczne.

Najprostszym wyrażeniem logicznym jest **relacja** - składająca się z dwu wyrażeń matematycznych przedzielonych **operatorem relacji**:

>	większy
>=	większy lub równy
<	mniejszy
<=	mniejszy lub równy
=	równy
<>	nie równy

Przykłady relacji:

```
x > 3,    a >= b,   (m*x^2-12.342) <> (b-x)/(b^3-x^2)
```

Załóżmy, że program w języku BASIC ma obliczać jedną z przyprostokątnych trójkąta prostokątnego (np.: **a**) gdy dane są pozostałe boki (**b**, **c**).

Rys. 4.11 pokazuje program oraz dwa warianty wyświetlanych wyników - dla złych oraz dobrych danych. Przy pomocy instrukcji IF sprawdzono poprawność danych to znaczy zapobiegnięto sytuacji gdy wczytana długość przeciwprostokątnej **c** byłaby mniejsza od długości przyprostokątnej **b**.

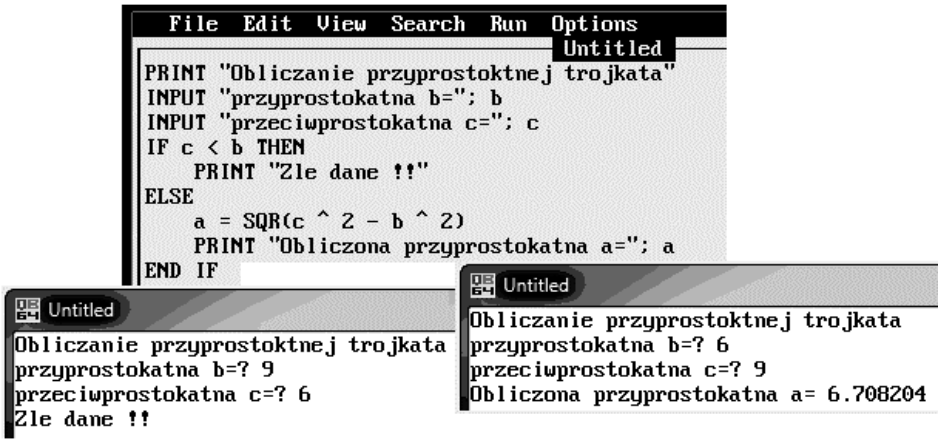
Dodatkowe zastosowanie operatorów logicznych pozwala tworzyć złożone wyrażenia logiczne.

Operatory logiczne w języku BASIC to:

NOT - nie
AND - i
OR - lub

Istnieją również operatory:

XOR - *Exclusive OR*, **EQU** - *Equivalence*, **IMP** - *Implication*



Rys. 4.11. Program PR 20. Przykład zastosowania instrukcji IF

4.6.9. PĘTLA „*FOR* ...”

Jedną z podstawowych zalet programowania jest możliwość krótkiego zapisania wielu powtarzających się operacji – przy użyciu instrukcji tworzących tak zwaną „**pętlę programową**” objaśnioną już w p.3.5. . Jedną z tego typu instrukcji jest instrukcja **FOR** ... **NEXT.**

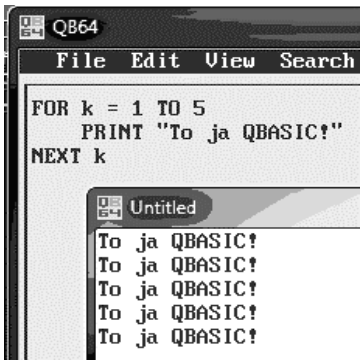
Pętla FOR ma następującą budowę:

FOR *zmienna_kontrolna* **=** *wart_pocz* **TO** *wart_konc* **STEP** *przyrost*
instrukcje
NEXT *zmienna_kontrolna*

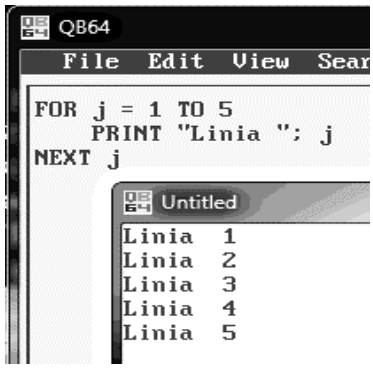
Polski sens jest nastepujący: „Dla wartości, które przyjmuje *zmienna_kontrolna* w zakresie od *wart_pocz* do *wart_konc* z określonym *przyrostem*, powtarzaj wykonywanie *instrukcji* zawartych między FOR a NEXT”.

Jak widać NEXT musi zamykać pętlę, którą rozpoczęło słowo FOR, a po obu tych słowach musi wystąpić ta sama nazwa zmiennej kontrolnej. Jeśli wewnątrz pętli wystąpi druga pętla to musi mieć ona inną zmienną kontrolną. Między FOR a NEXT może być jedna lub wiele instrukcji i wszystkie one będą wykonane tyle razy ile wygenerowanych będzie wartości zmiennej kontrolnej.

a)



b)

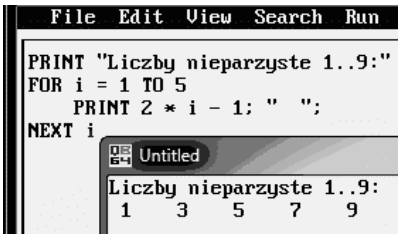


Rys. 4.12. Przykłady pętli FOR. (PR 21 a i b)

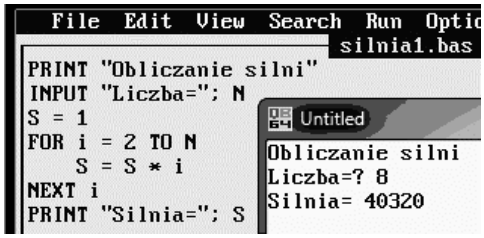
Najlepiej zobaczyć to na konkretnych przykładach napisanych w QB64 i zaprezentowanych na Rys. 4.12. Rys. 4.12a pokazuje jak można powtarzać wyświetlanie stałego tekstu a Rys. 4.12b pokazuje jak dodatkowo można wyświetlić zmieniającą się wartość zmiennej kontrolnej j.

Oczywiście zmienną kontrolną można wykorzystać na wiele sposobów - na przykład budując na jej podstawie wyrazy jakiegoś ciągu liczb (np.: nieparzystych - Rys. 4.13a) czy wyznaczając silnię danej liczby N (Rys. 4.13b).

a)



b)



Rys. 4.13. Przykłady pętli FOR c.d. (PR 22, PR 23)

Dalsze przykłady pętli FOR zamieszczono w p.5.2 oraz 5.3.

4.6.10. SKOK BEZWARUNKOWY I PĘTLA „*WHILE* ...”

Skok bezwarunkowy, czyli instrukcję „**GoTo** *etykieta*”, pokazano już w przykładowych programach w p. 4.3. . Instrukcja ta poleca: „**Skocz do** instrukcji oznaczonej *etykietą*”

Etykietą jest w języku BASIC liczba całkowita umieszczona przed poleceniem do którego ma nastąpić skok. Przykładowo – prosty algorytm obliczania kwadratów liczb zamieszczony jako schemat blokowy w p. 3.6.6. oraz jako opis słowny w p. 4.3. . można zapisać w języku BASIC z użyciem skoku GOTO następująco:

```
REM program PR 24
2 PRINT "Obliczanie kwadratów"
  INPUT "Podaj liczbe:"; X
  Y = X * X
```

```
  PRINT "Kwadrat liczby ="; Y
  INPUT "Czy nowe obliczenie? (T/N):"; P$
  IF P$ = "T" OR P$ = "t" GOTO 2
  STOP
```

Jak już wspomniano stosowanie instrukcji GoTo nie jest zalecane, więc zamiast niej lepiej powyższy algorytm dostosować do użycia **pętli typu WHILE** która pozwoli wyeliminować GoTo i etykietę. Pętla ta powtarza swe działanie tak długo dopóki spełniony jest warunek zapisany po słowie WHILE. Powtarzane instrukcje muszą być umieszczone między WHILE i WEND. Zmodyfikowany program będzie miał postać:

```
REM program PR 25
PRINT "Obliczanie kwadratów"
P$ = "T"
WHILE UCASE$(P$) = "T"
    INPUT "Podaj liczbe:"; X
    Y = X * X
    PRINT "Kwadrat liczby ="; Y
    INPUT "Czy nowe obliczenie? (T/N):"; P$
WEND
STOP
```

Zauważmy, że konieczne było polecenie P$ = "T" aby pętla była wykonana po raz pierwszy. W programie zastosowano także funkcję UCASE$(...), która zamienia małe litery na duże (*upper case*), bo inaczej po naciśnięciu litery "t" warunek nie był spełniony.

4.6.11. PĘTLA „*DO ... LOOP UNTIL ...*”

W języku BASIC (a także w języku Pascal) jest jeszcze trzeci typ pętli, także o nieokreślonej z góry liczbie cykli, lecz różniący się od pętli WHILE dwoma cechami:

1. sprawdzanie warunku następuje na końcu a nie na początku pętli,
2. spełnienie warunku kończy działania (odwrotnie niż w pętli WHILE)

Pętla „DO ... LOOP UNTIL ...” pokazana została poniżej w algorytmie wielokrotnego obliczania pierwiastków równań kwadratowych.

Mamy utworzyć algorytm (i program) rozwiązywania **dowolnego równania kwadratowego** dla dowolnych wartości danych (współczynników równania) i dowolną liczbę razy.

Zaczynamy od określenia potrzebnych zmiennych:

A, B, C – dla przechowania współczynników równania (liczb rzeczywistych)

Delta – dla wartości wyróżnika,

X1, X2 – dla obliczonych wartości rozwiązań równania,

ZN$ – dla przechowania litery "t" lub "n" wprowadzonej przez nas z klawiatury jako odpowiedź "tak" lub "nie" na pytanie czy chcemy dalej obliczać.

Konstruowanie algorytmu opieramy na prześledzeniu czynności jakie wykonywalibyśmy bez komputera. Załóżmy, że mając konkretne równanie np.: $-3.5x^2 + 6x + 10 = 0$ zaczęlibyśmy od przypisania danych zmiennym A, B, C reprezentujacym współczynniki równania:

wstaw –3.5 do A, wstaw 6 do B, wstaw 10 do C

Jednak użycie instrukcji przypisania do wprowadzenia danych nie jest dobrym sposobem bo powodowałoby konieczność zmiany tego fragmentu programu przy zmianie danych. Zamiast tego konieczne jest więc użycie polecenia **wczytania danych** z urządzenia wejściowego do zmiennych A, B, C.

Ponieważ tok działań musi zależeć od znaku wyróżnika równania kwadratowego DELTA więc w algorytmie musi być **rozgałęzienie**, zrealizowane instrukcją IF.

Dodatkowo, jeśli chcemy aby po jednym uruchomieniu programu można było rozwiązać wiele równań to potrzebna jest **pętla** powtarzającą operacje dla kolejnych równań kwadratowych.

Ostatecznie nasz algorytm może wyglądać tak jak pokazano na Rys. 4.14, gdzie zamieszczono schemat blokowy algorytmu, ale może też być zapisany w innych postaciach, jak to pokazano poniżej, na przykład w postaci opisu słownego (po polsku) albo w postaci programu w języku BASIC.

Postać 1 - opis słowny:

1. *wyczyść ekran*
2. *do zmiennej **Zn$** wstaw znak "T"*
3. ***Pętla: powtarzaj wykonywanie tego co poniżej aż do 11***
4. *wyświetl napis "Rozwiązywanie równania kwadratowego:"*
5. *wyświetl napis "A=" i czekaj na wprowadzenie wartości dla **A***
6. *wyświetl napis "B=" i czekaj na wprowadzenie wartości dla **B***
7. *wyświetl napis "C=" i czekaj na wprowadzenie wartości dla **C***
8. *oblicz vartość **B * B - 4 * A * C** i wstaw do zmiennej **Delta***
9. *JEŚLI **Delta < 0** TO*

   wyświetl napis "Brak pierwiastkow rzeczywistych"

   9a) W PRZECIWNYM PRZYPADKU wykonaj instrukcje aż do 9b:

   oblicz $(-B - \sqrt{Delta})/(2 \cdot A)$ *i wstaw do zmiennej **x1;***

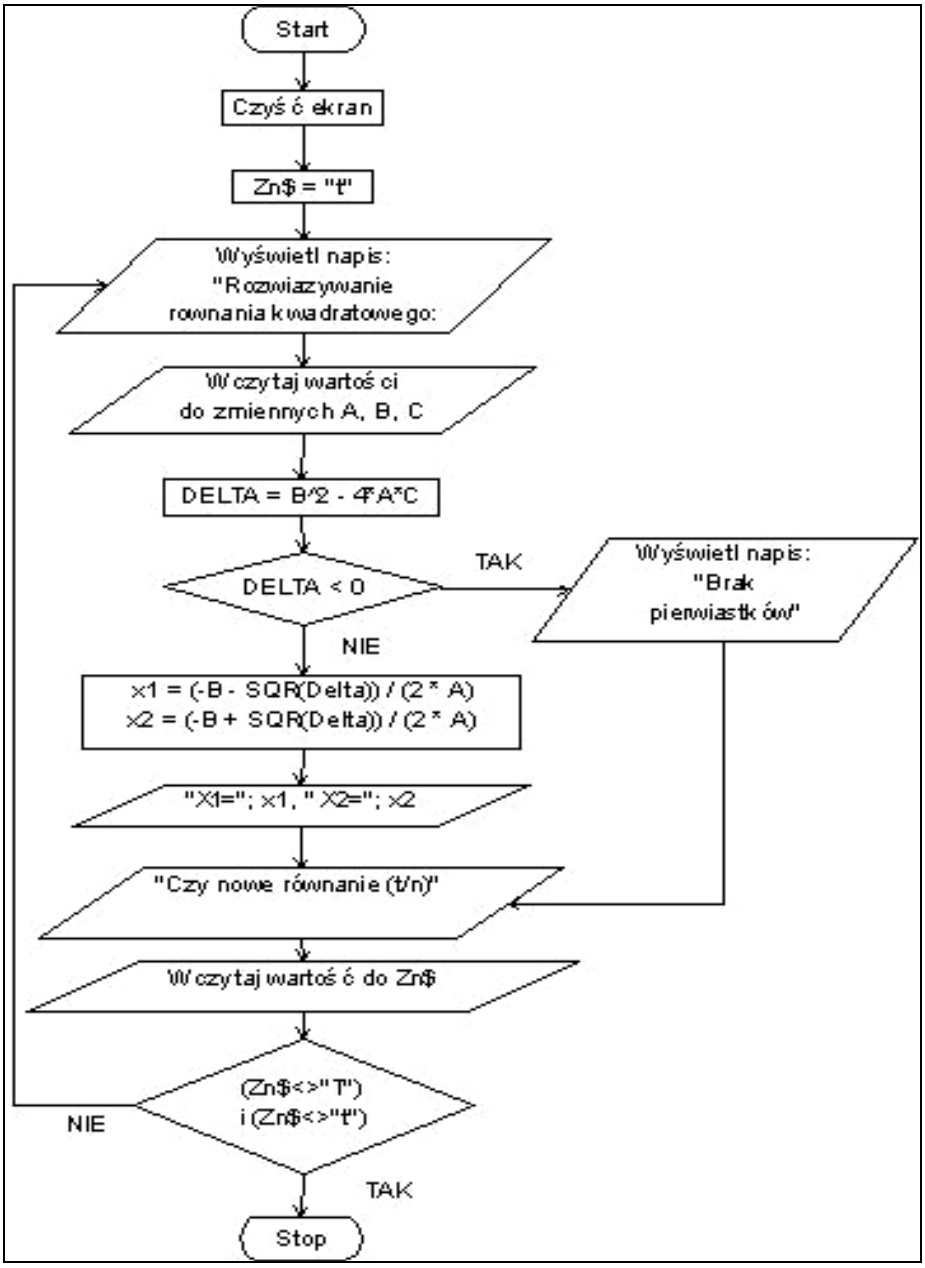
   oblicz $(-B + \sqrt{Delta})/(2 \cdot A)$ *i wstaw do zmiennej **x2;***

   *wyświetl napis "x1=" oraz wartość zmiennej **x1;***

   *wyświetl napis " x2=" oraz wartość zmiennej **x2***

   9b) KONIEC instrukcji "JEŚLI"
10. *wyświetl napis "Czy nowe rownanie ?(t/n):" i wprowadź znak z klawiatury do **Zn$***
11. ***Koniec pętli**: jeśli (**Zn$** <> "T") i (**Zn$** <> "t") idź do 12 a jesli nie to wróć do 3*
12. Wyświetl napis: " *** Do Widzenia.***"

Postać 2 - schemat blokowy



Rys. 4.14. Schemat blokowy algorytmu rozwiązywania równania kwadratowego

Postać 3 - program w języku BASIC. Opisy są w komentarzach poprzedzonych apostrofem

```
' Program PR 26 – rozwiązywanie równania kwadratowego
 CLS :' - to jest instrukcja "wyczyść ekran"
 Zn$ = "T" :' - do zmiennej Zn$ wstawia znak "T"
 ‘ Poniżej zaczyna się pętla DO ... LOOP UNTIL .. :
DO
   PRINT "Rozwiazywanie rownania kwadratowego:"
   INPUT "A=", a : '- wyświetla "a=" i czeka na wartość dla a
   INPUT "B=", b : INPUT "C=", c
   Delta = B * B – 4 * A * C :' - oblicza wartość zmiennej Delta
  ' IF - znaczy jeśli; THEN - to; ELSE - w przeciwnym przypadku:
   IF Delta < 0 THEN
     PRINT "Brak pierwiastkow rzeczywistych"
   ELSE
      x1 = (–B – SQR(Delta)) / (2 * A)
      x2 = (–B + SQR(Delta)) / (2 * A)
      PRINT "x1="; x1, " x2="; x2
   END IF :'- koniec instrukcji IF ... THEN ... ELSE ... END IF
   INPUT "Czy nowe rownanie ?(t/n):"; Zn$
LOOP UNTIL (Zn$ <> "T") AND (Zn$ <> "t"): ' - koniec pętli
PRINT " *** Do Widzenia.***"
STOP
```

4.6.12. FUNKCJE STANDARDOWE I FUNKCJE UŻYTKOWNIKA

BASIC oferuje wiele gotowych funkcji zwanych też standardowymi. Ważniejsze standardowe funkcje matematyczne to:

ABS(x)	- wartość bezwzględna
ATN(x)	- arcus tangens
COS(x)	- cosinus
EXP(x)	- e do potęgi x
CINT(x)	- zaokrągla do najbliższej całkowitej
INT(x)	- zaokrągla do całkowitej w dół
FIX(x)	- zaokrągla w stronę zera
LOG(x)	- logarytm naturalny
RND(x)	- liczba pseudolosowa
SGN(x)	- znak (+1 lub -1)
SIN(x)	- sinus
SQR(x)	- pierwiastek kwadratowy
TAN(x)	- tangens

Zauważmy, że po nazwie funkcji zawsze musi wystąpić nawias okrągły a w nim argumenty dla których wartość funkcji ma być wyznaczona. Nawiasy okrągłe są też stosowane dla indeksów tablic, ale BASIC wymaga uprzedniego zadeklarowania tablic poleceniem DIM. Tablica niezadeklarowana zostanie potraktowana jak nieznana funkcja.

BASIC posiada też funkcje standardowe stosowane w operacjach na **tekstach**:

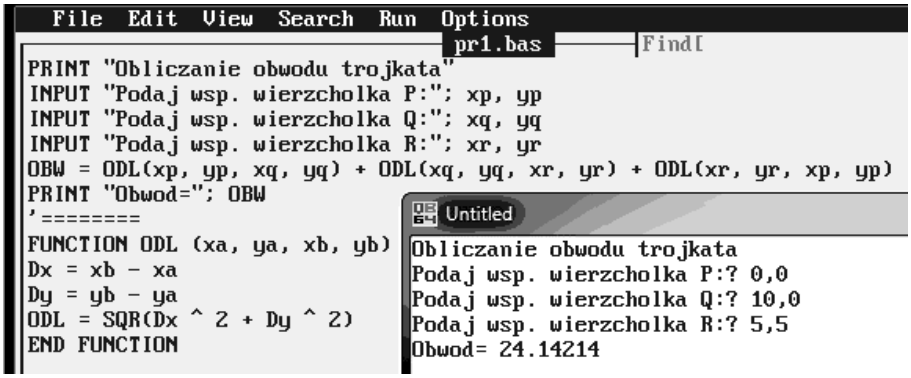
ASC(Z$)	- kod podanego znaku Z$
CHR$(N)	- znak o podanym kodzie N
HEX$(N)	- zamiana liczby N na postać liczby szesnastkowej
INSTR(N,T$,B$)	- szuka fragmentu B$ w tekście T$ począwszy od N-tego znaku
LCASE$(T$)	- zamienia T$ na małe litery
LEFT$(T$,N)	- pobiera z T$ N znaków od lewej
LEN(T$)	- wyznacza długość tekstu T$
LTRIM$(T$)	- usuwa spacje przed tekstem
MID$(T$,P,N)	- pobiera z T$ N znaków począwszy od P-tego
OCT$(N)	- zamiana na postać liczby ósemkowej
RIGHT$(T$,N)	- pobiera z T$ N znaków z prawej
RTRIM$(T$)	- usuwa spacje za tekstem
SPACE$(N)	- wytwarza ciąg spacji
STR$(N)	- zamienia liczbę na łańcuch tekstowy
STRING$(N,Z$)	- tworzy łańcuch powtarzając N razy dany znak Z$
UCASE$(T$)	- zamienia na duże litery
VAL(T$)	- zmienia tekst zawierający liczbę na liczbę

Dokładniejsze objaśnienia tych funkcji można uzyskać w pomocy (HELP) na przykład wpisując nazwę, umieszczając w niej kursor i naciskając [Ctrl][F1].

Oprócz używania gotowych funkcji, **użytkownik może zdefiniować i używać dowolne własne funkcje**. Jest to opłacalne wtedy gdy przynajmniej kilkukrotnie będziemy zdefiniowaną funkcję wykorzystywać.

Przy korzystaniu z funkcji – czyli jej **wywoływaniu** – trzeba znać nie tylko **nazwę funkcji** ale także: **liczbę jej argumentów** (parametrów), ich **znaczenie** oraz **kolejność** w jakiej muszą być podawane. Nie są natomiast wtedy istotne nazwy argumentów.

Definiowanie własnej funkcji objaśnię na przykładzie obliczania na płaszczyźnie odległości między dwoma dowolnymi punktami A i B o danych współrzędnych: (xa,ya) oraz (xb,yb). Zadanie sprowadza się do obliczenia przeciwprostokątnej trójkąta, którego przyprostokątnymi są Dx=(xb-xa) oraz Dy=(yb-ya).



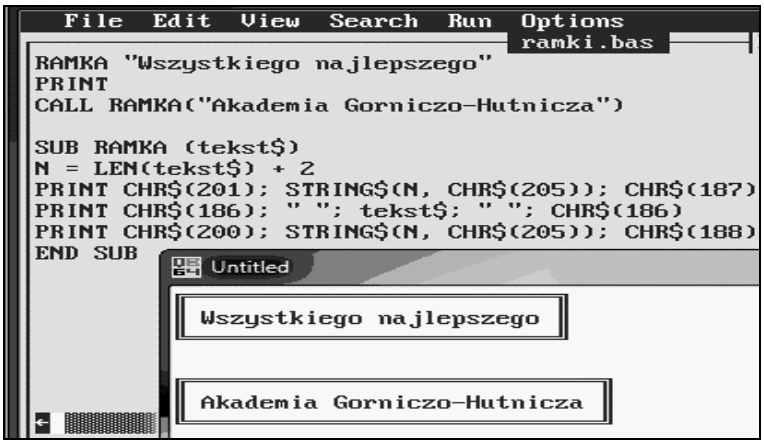
Rys. 4.15. Przykład (PR 27) definiowania i wykorzystania funkcji użytkownika

4.6.13. PODPROGRAMY PROCEDURALNE

Jak wspomniano już w p.3.6.8. , podprogramy proceduralne mogą mieć wiele parametrów wejściowych i wyjściowych a mogą także nie mieć ich wcale. W języku BASIC (QB64) szkielet podprogramu (*subroutine*) ma następującą budowę:

```
SUB nazwa_podprogramu (parametry)
..... instrukcje
END SUB
```

Rys. 4.16 pokazuje przykład dwukrotnie wywołanego podprogramu RAMKA(tekst$), który wyświetla tekst zadany jako parametr ale dodatkowo otacza go podwójną ramką.



Rys. 4.16. Podprogram (PR 28) rysujący ramkę w trybie tekstowym

Jak widać w QB64 są dwa sposoby wywołania podprogramu. Sposób tradycyjny:

```
CALL nazwa_podprogramu (parametry_aktualne)
```

alternatywny sposób to:

```
nazwa_podprogramu parametry_aktualne
```

4.6.14. KOMUNIKACJA Z PLIKAMI TEKSTOWYMI

Przed rozpoczęciem pierwszego odczytywania z pliku lub zapisywania do niego trzeba plik utworzyć – jeśli nie istniał - oraz otworzyć instrukcją:

OPEN *plik$* **[FOR** *tryb***] AS [#]***id%*

gdzie: *plik$* - wyrażenie tekstowe określające nazwę pliku
tryb - to jedno ze słów kluczowych:
APPEND, BINARY, INPUT, OUTPUT, lub **RANDOM**
id% - to identyfikator czyli numer z zakresu 1 do 255 którym muszą się posługiwać instrukcje odwołujące się do tego pliku

Aby zakończyć operacje z danym plikiem trzeba go zamknąć instrukcją:

CLOSE #*id%*

a jeśli chcemy zamknąć wszystkie otwarte pliki to wystarczy:

CLOSE

Przykład zapisu tekstu do pliku:

```
CLS: ' PR 29
    INPUT "Podaj nazwe pliku: "; n$
    OPEN n$ FOR OUTPUT AS #1
    PRINT #1, "Ten tekst będzie zapisany do pliku"
    CLOSE
```

Przykład wczytywania linii tekstu aż do końca pliku:

```
CLS: ' PR 30
OPEN "DANE.TXT" FOR INPUT AS #1:   'Otwarcie do odczytu
PRINT "Tekst pliku:"
WHILE NOT EOF(1)
    LINE INPUT #1, LINIA$:  'Wczytuje linię tekstu z pliku
    PRINT LINIA$:      'Wyświetla wczytaną linię tekstu
WEND:                   'Koniec pętli
CLOSE #1                'Zamknięcie pliku
```

Przykład programu zapisującego wyniki obliczeń do pliku D:\WYNIKI.TXT:

```
CLS: ' PR 31
PRINT "Drukowanie do pliku WYNIKI.TXT"
OPEN "D:\WYNIKI.TXT" FOR OUTPUT AS #1
    PRINT #1, "Obliczanie kwadratow liczb"
    PRINT #1, "liczba", "kwadrat"
    FOR y = 0 TO 100 STEP 9.05
          PRINT #1, USING "######.####"; y; y ^ 2
    NEXT y
CLOSE #1
```

Sprawdź działanie programów. Następnie otwórz plik wyników w NOTATNIKU i przeglądnij jego treść. Kolejny przykład pokazuje jak wczytać dane liczbowe z pliku o nazwie POM24.TXT zlokalizowanego na dysku D:

```
CLS: DIM A(1 TO 30) AS SINGLE: 'PR 32
PRINT "Wczytywanie z pliku "; : '- liczby oddzielane spacjami
OPEN "D:\POM24.TXT" FOR INPUT AS #1
i = 1
WHILE NOT EOF(1)
    INPUT #1, A(i)
    PRINT A(i)
    i = i+1
WEND
CLOSE #1
PRINT "Wczytano liczb: "; i
```

4.6.15. GRAFIKA W QB64

Domyślnie BASIC pracuje w trybie tekstowym, więc przed użyciem poleceń graficznych trzeba użyć polecenia ***SCREEN nr_trybu*** – aby włączyć jeden z dostępnych trybów graficznych w oknie wyników. Najlepiej użyć trybu 12 o najwyższej rozdzielczości 640x480 (współrzędne 0 - 639 i 0 - 479) i 16 kolorach pokazanych w tabeli 7.

Tabela 4.3. Kody kolorów w trybie SCREEN 12

Kod koloru	**Kolor**	**Kod koloru**	**Kolor**
0	czarny	8	ciemno-szary
1	granatowy	9	niebieski
2	zielony	10	jasno zielony
3	ciemny błękit	11	jasny błękit
4	czerwony	12	jasna czerwień
5	fiolet	13	jasny fiolet
6	pomarańczowy	14	żółty
7	jasno-szary	15	biały

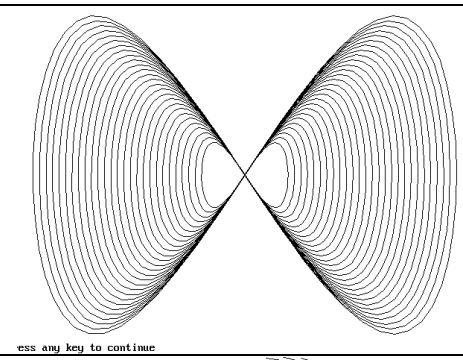
Po wykonaniu polecenia **SCREEN 12** możemy używać różnych poleceń rysujących. Mamy między innymi możliwość wyświetlania punktów poleceniem **PSET**, rysowania odcinków lub prostokątów (także wypełnionych kolorem) przy pomocy polecenia **LINE**, oraz rysowania łuków, okregów i elips przy pomocy polecenia **CIRCLE**. Przykłady użycia tych poleceń, w prostych programach graficznych, zamieszczono w tabeli 8 oraz na Rys. 4.17.

Polecenie: **VIEW (*xp, yp*)-(*xk, yk*)[, *tło*][, *ramka*]** - definiuje okno graficzne jako część ekranu podając współrzędne lewego **górnego** rogu tego okna: ***xp, yp***, oraz prawego dolnego: ***xk, yk***, a dodatkowo można podać numery kolorów ***tła*** oraz ***ramki*** (Rys. 4.18).

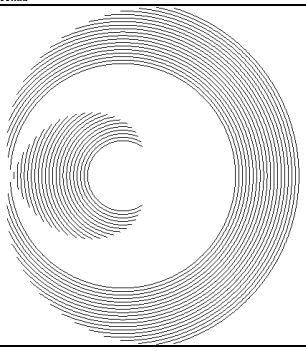
Tabela 4.4. Przykłady poleceń graficznych

Polecenie	**Objaśnienie**
SCREEN 12	włącza tryb graficzny 12 (640x480)
PSET (**x**, **y**), **c**	wyświetla punkt koloru **c** w kolumnie **x**, wierszu **y**
LINE (**x1,y1**)-(**x2,y2**), **c**	rysuje odcinek od (**x1,y1**) do (**x2,y2**) kolorem **c**
LINE -STEP(**x2,y2**), **c**	odcinek od ostatnio wyprowadzonego punktu do (**x2,y2**)
LINE (**x1,y1**)-(**x2,y2**), **c**, **B**	rysuje prostokąt (*box*) od (**x1,y1**) do (**x2,y2**) kolorem **c**
LINE (**x1,y1**)-(**x2,y2**), **c**, **BF**	prostokąt (*box*) j.w. ale wypełniony (*filled*) kolorem **c**
CIRCLE (**x, y**), **r,c**	okrąg o środku (**x,y**) promieniu **r** i kolorze **c**
CIRCLE (**x, y**), **r, c, kp, kk**	łuk jak wyżej - od kąta **kp** do kąta **kk** względem osi x
CIRCLE (**x, y**), **r,c, , ,s**	elipsa czyli okrąg spłaszczony w pionie o współcz. **s**<1

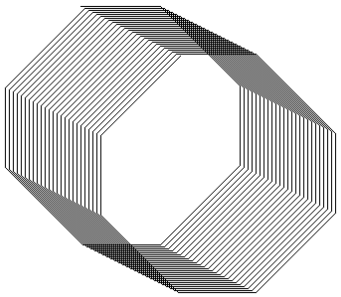
```
' PR 33. Figury Lissajous:
SCREEN 12
WINDOW (-11, -11)-(11, 11)
FOR w = 2 TO 10 STEP 0.3
    xp = w: yp = 0
    FOR a = 0 TO 6.3 STEP 0.1
       x = w * COS(a)
       y = w * SIN(2 * a)
       LINE(xp, yp)-(x, y), 15
       xp = x: yp = y
    NEXT a
NEXT w
```



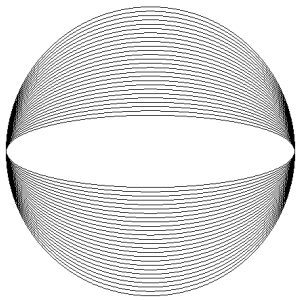

```
' PR 34. Łuki:
SCREEN 12
WINDOW (-13, -13)-(13, 13)
p = 6.28
FOR r = 2 TO 10 STEP 0.2
    kp = r / 2: kk = p - kp
    CIRCLE(0,0), r, 15, kp, kk
NEXT r
SLEEP
```



```
' PR 35. Sprężyna ośmiokątna:
SCREEN 12
A$="C15 R80 F80 D80 G80 L78 H78 U80 E80"
PSET (100, 10), 15
FOR i = 1 TO 25
    DRAW A$
NEXT i
```



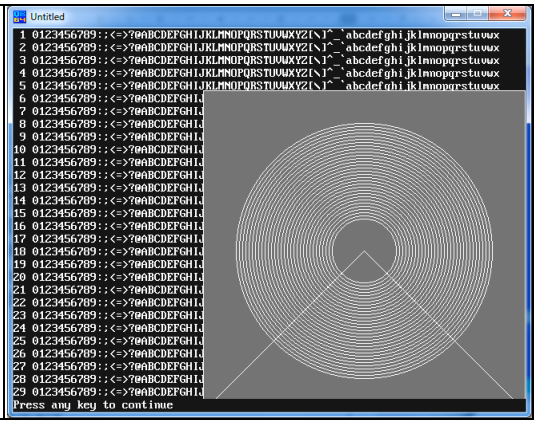
```
' PR 36. Elipsy:
SCREEN 12
WINDOW (-12, -10)-(12, 10)
FOR s = 0.25 TO 1 STEP 0.03
    CIRCLE(0,0), 8, 15, , , s
NEXT s
SLEEP
```



Rys. 4.17. Przykłady programów graficznych w języku BASIC

Bardzo przydatnym jest polecenie **WINDOW** (***xp,yp***)-(***xk,yk***) pozwalające przypisać obszarowi całego ekranu lub okna graficznego określonego przez **VIEW**, dowolny zakres współrzędnych (Rys. 4.18), przez podanie współrzędnych lewego dolnego rogu ekranu (***xp, yp***) oraz dla prawego górnego (***xk,yk***). Bez tego polecenia początek układu współrzędnych ekranowych jest w lewym, **górnym** rogu ekranu lub okna graficznego.

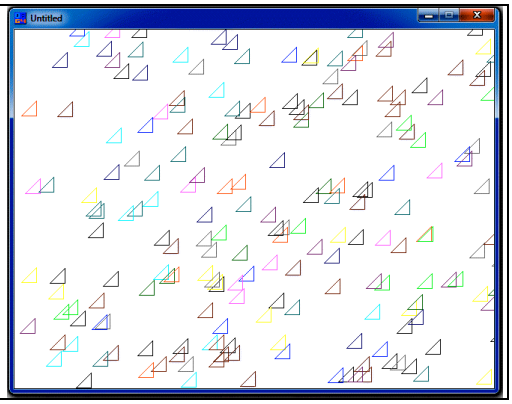
```
CLS: SCREEN 12:' PR 37.
FOR i = 1 TO 29
    PRINT
    PRINT USING "## "; i;
    FOR k = 48 TO 120
        PRINT CHR$(k);
    NEXT k
NEXT i
VIEW (238, 78)-(638, 478),8,15
WINDOW (0, 0)-(100, 100)
LINE (0, 0)-(50, 50)
LINE (50, 50)-(100, 0)
FOR R = 10 TO 40
    CIRCLE (50, 50), R
NEXT R : SLEEP
```



Rys. 4.18. Przykład definiowania okna graficznego w QB64

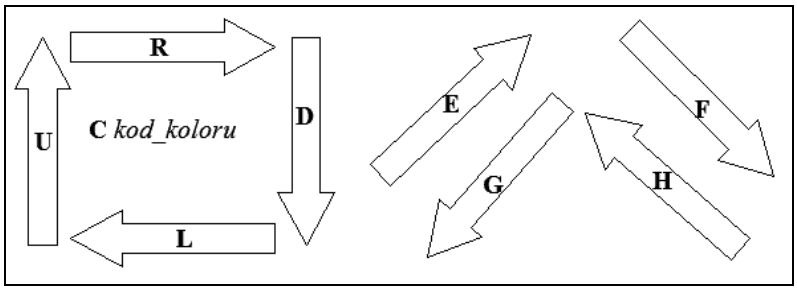
Przy rysowaniu odcinków linii prostych można zastosować współrzędne przyrostowe, tak jak to pokazano na Rys. 4.19, w programie rysującym wiele jednakowych trójkątów, w losowo wybranych położeniach i barwach. Zastosowano m.in. funkcję bezargumentową RND, która zwraca liczbę pseudolosową z zakresu 0 do 1.

```
CLS: SCREEN 12: ' PR 38
XM = 639: YM = 479: a = 20
VIEW (0, 0)-(XM, YM), 15
WINDOW (0, 0)-(XM, YM)
FOR i = 1 TO 200
    X = XM * RND: Y = YM * RND
    C = INT(15 * RND)
    PSET (X, Y)
    LINE -STEP(a, a), C
    LINE -STEP(0, -a), C
    LINE -STEP(-a, 0), C
NEXT i: SLEEP
```



Rys. 4.19. Przykład zastosowania współrzędnych przyrostowych - instrukcja LINE –STEP

Interesujące jest także polecenie **DRAW "***łańcuch rozkazów***"** – które pozwala rysować łamaną (na przykład tak jak na Rys. 4.17c), na podstawie ciągu rozkazów, zapisanego jako łańcuch tekstowy. Rozkazy składają się z litery określającej kierunek – tak jak podano to na Rys. 4.20 - oraz liczby określającej długość tego odcinka.



Rys. 4.20. Znaczenie liter w "łańcuchu rozkazów" polecenia DRAW

4.6.16. DŹWIĘK I MUZYKA W QB64

W QB64 funkcjonują takie same jak w Qbasic polecenia dotyczące dźwięków i muzyki. Najprymitywniejsza komenda BEEP generuje dźwięk używany jako ostrzeżenie.

Generowanie dowolnych dźwięków komendą SOUND wymaga określeniawartości jej dwu parametrów: częstotliwości (37 do 32767) oraz czasu trwania (0 do 65535).

Do zapisywania i generowania muzyki najlepiej jednak nadaje się polecenie **PLAY**, którego parametrem jest łańcuch znaków, w którym mogą wystąpić:

- litera **O** i następująca po niej cyfra od 0 do 6 – określają oktawę,
- znak większości lub mniejszości (**>**,**<**) – przejście do wyższej lub niższej oktawy,
- litera **P** i następująca po niej liczba (1 do 64) – pauza (P1 – pauza całonutowa),
- litera **T** i następująca po niej liczba (32 - 255) – tempo w ćwierćnutach na minutę,
- **A,B,C,D,E,F,G** – nuty a po każdej liczba (1 - 64) gdzie 1 to cała nuta, 2 to półnuta, ...
- + lub **#** to krzyżyk, a minus to bemol.

Oto przykładowy programik generujący melodię „Ody do wolności":

```
' Oda do wolnosci:
PLAY "O2T60e8e8f8g8g8f8e8d8c8c8d8e8e8d12d4"
PLAY "e8e8f8g8g8f8e8d8c8c8d8e8d8c12c4"
PLAY "d8d8e8c8d8e12f12e8c8d8e12f12e8d8c8d8p8"
PLAY "e8e8f8g8g8f8e8d8c8c8d8e8d8c12c4"
```

W odróżnieniu od QBASIC-a, który używa tylko głośniczka systemowego, QB64 posługuje się się kartą dźwiękową. QB64 umożliwia także odsłuchiwanie różnego typu plików muzycznych.

5. PRZYKŁADY ALGORYTMÓW I PROGRAMÓW W JĘZYKU BASIC

W poprzednim rozdziale zamieszczono szereg przykładów obrazujących budowę i możliwości zastosowań poszczególnych instrukcji języka BASIC, natomiast niniejszy rozdział skupia się już na przykładach konkretnych algorytmów obliczeniowych i realizacji wybranych metod numerycznych, pokazując je czasem w postaci programów w języku BASIC. Przykłady te mogą być podstawą do opracowywania mniej lub bardziej rozbudowanych programów w różnych językach programowania.

Niektóre z przykładów przedstawiono w Mathcadzie – co może stanowić bodziec do odświeżenia wiedzy o tym narzędziu, nabytej w poprzednim semestrze i skumulowanej w podręczniku [2].

Algorytmy bywają często przedstawiane w postaci schematów blokowych ale można uniknąć dość pracochłonnego rysowania i zapisać algorytm od razu w języku programowania, co przynosi dodatkowo poważną korzyść, bo umożliwia testowanie poprawności działania tego algorytmu.

Najprostszym do nauczenia się jest język BASIC, dlatego posłużył do przedstawienia niektórych z prezentowanych algorytmów.

5.1. ELEMENTARNE PROGRAMY OBLICZENIOWE

Do elementarnych zagadnień obliczeniowych będziemy zaliczać zadania, w których, na podstawie jednej lub kilku danych, wyznaczane są – przy pomocy niewielu wzorów obliczeniowych - wartości jednej lub kilku zmiennych wynikowych.

Dla każdego problemu obliczeniowego można utworzyć wiele algorytmów i programów, w szczególności różniących się:

- sposobem wprowadzania i kontroli poprawności danych,
- bogactwem objaśnień i systemem pomocy,
- możliwością wykonania obliczeń jednorazowo lub wielokrotnie (wektoryzacja),
- sposobem wyprowadzania wyników (na ekran, na drukarkę, do pliku, jako wykres)
- przechowywaniem danych i wyników tylko do momentu wyprowadzenia lub z możliwością zmagazynowania i ewentualnego dalszego przetwarzania

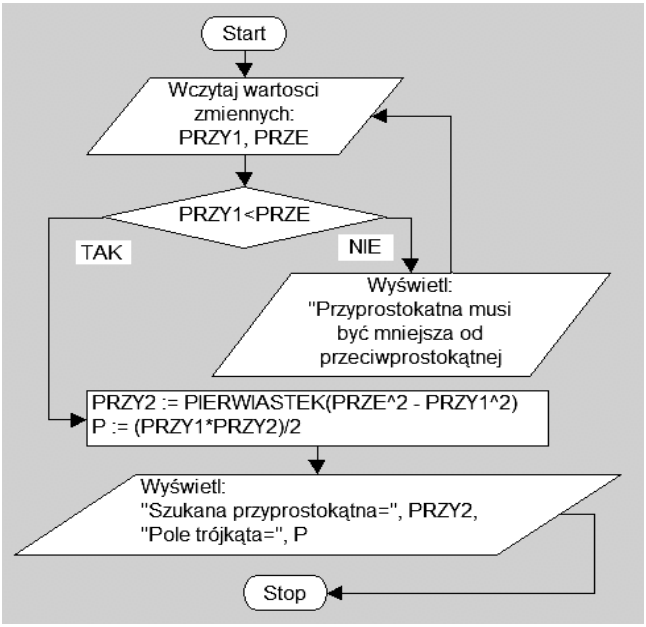
Prześledźmy kilka możliwych wariantów dla jednego z najprostszych zadań jakie można wymyślić, a mianowicie:

Napisać algorytm i program, który mając przyprostokątną i przeciwprostokątną trójkąta prostokątnego wyznacza trzeci bok, oraz pole trójkąta.

Warianty algorytmów i programów dla tego zadania pokazano w kolejnych podrozdziałach.

5.1.1. JEDNORAZOWE OBLICZENIA Z KONTROLĄ DANYCH

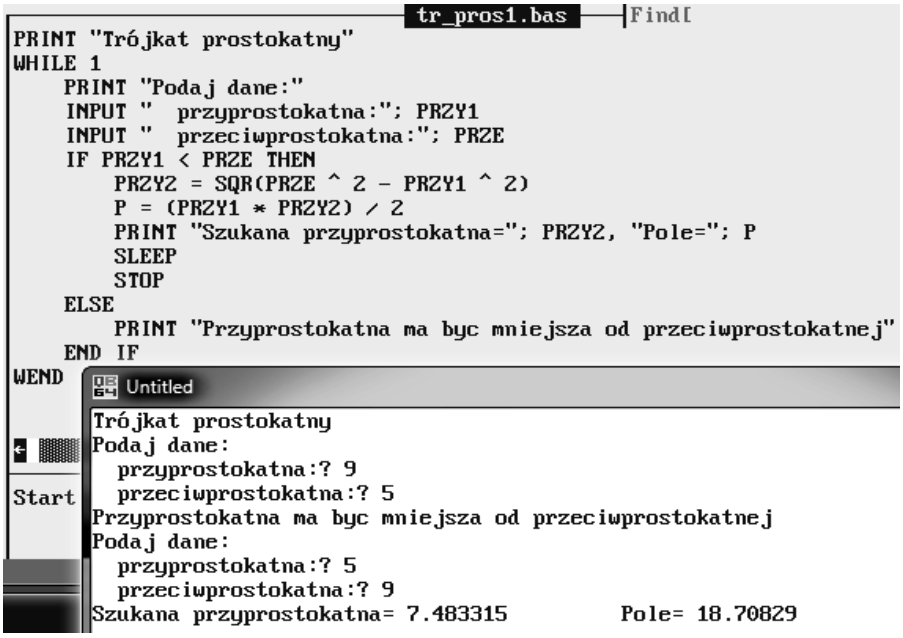
1. **Dane**: przyprostokątna i przeciwprostokątna
2. **Szukane**: przyprostokątna i pole trójkąta prostokątnego
3. Definiujemy potrzebne **zmienne**:
   PRZY1 – przyprostokątna dana, PRZE – przeciwprostokątna dana,
   PRZY2 – przyprostokątna szukana, P – pole trójkąta
4. **Warunek poprawności danych**: PRZY1<PRZE
5. **Algorytm** obliczeń (Rys. 5.1) – według twierdzenia Pitagorasa
   (^ oznacza potęgowanie)



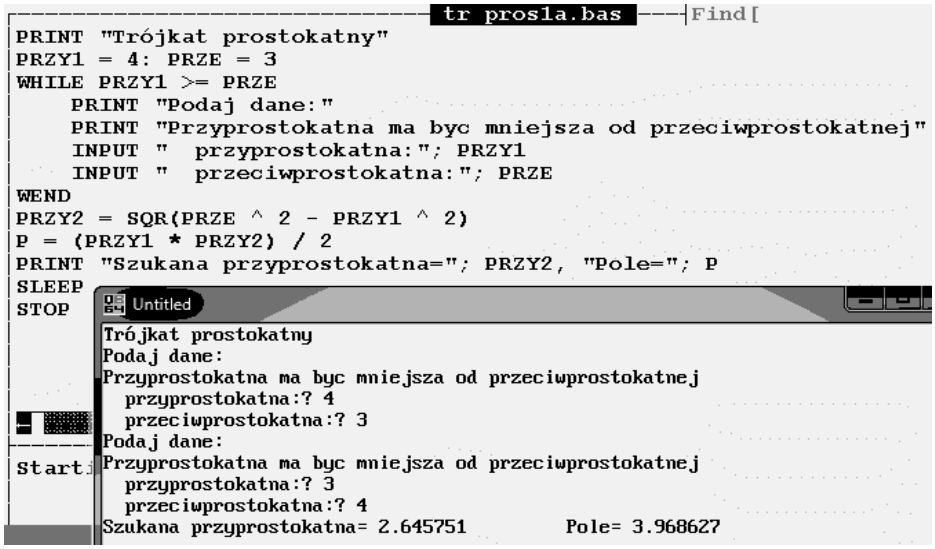
Rys. 5.1. Przykład algorytmu obliczeń skalarnych jednorazowych – wariant 1

W przypadku nie spełnienia warunku wykonalności obliczeń wyświetlony musi być komunikat: „Przyprostokątna ma być mniejsza od przeciwprostokątnej” (Rys. 5.1) a następnie powrót do wprowadzania danych. W BASIC-u powrót można uzyskać m.in. przez zastosowanie polecenia „**skoku bezwarunkowego**” o postaci ***GoTo*** *etykieta*, gdzie *etykieta* jest numerem, który musimy postawić przed linią do której ma nastąpić powrót.

Jeśli jednak chcemy programować strukturalnie to powinniśmy zastąpić instrukcję skoku bezwarunkowego innym sposobem. Jednym z nich jest pokazana w programie na Rys. 5.2 pętla ***WHILE*** (p.4.6.10.), która działałaby bez końca - bo zamiast warunku jest jedynka czyli „prawda” - ale zapobiega temu instrukcja ***IF*** kontrolująca poprawność danych oraz polecenie ***STOP*** kończące działanie programu. Polecenie ***SLEEP*** realizuje pauzę zapobiegającą zamknięciu okna wyników zaraz po ich wyświetleniu.



Rys. 5.2. Program (PR 39) obliczeń jednorazowych z kontrolą danych – wariant 2



Rys. 5.3. Program (PR 40) obliczeń jednorazowych z kontrolą danych – wariant 3

Innym, zgrabniejszym rozwiązaniem pokazanym na Rys. 5.3 jest zrezygnowanie z instrukcji warunkowej ***IF***, a zastosowanie pętli ***WHILE*** z warunkiem powodującym powtarzanie wprowadzania danych tak długo dopóki są one błędne, czyli **PRZY1>=PRZE**.

Aby nastąpiło wejście do tej pętli (Rys. 5.3), przed nią, przypisano zmiennym **PRZY1** i **PRZE** celowo błędne wartości.

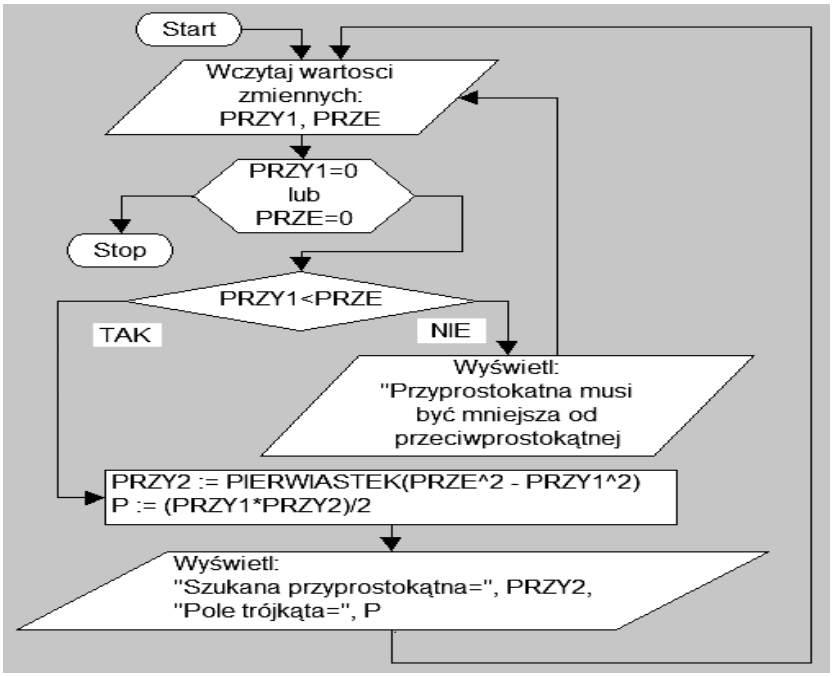
Sporządź schemat blokowy algorytmu dla tego programu.

5.1.2. WIELOKROTNE OBLICZENIA SKALARNE

W praktyce, pisanie programu dla obliczeń jednorazowych nie jest opłacalne, gdyż wystarcza do tego w zupełności kalkulator. Dlatego nieco sensowniejszym będzie program, który po jednorazowym uruchomieniu, potrafi powtarzać obliczenia dla dowolnej liczby zestawów danych .

Zastosowano te same zmienne i warunek poprawności danych. Po wykonaniu obliczeń i wyprowadzeniu wyników program ma żądać następnego zestawu danych. Obliczenia mają być zakończone gdy jedna z wprowadzonych danych będzie równa zero – co stanowi tak zwany „warunek stopu".

Jednym z możliwych wariantów jest napisanie programu według algorytmu z Rys. 5.4, z zastosowaniem instrukcji ***IF*** oraz skoku bezwarunkowego ***GoTo***.



Rys. 5.4. Przykład algorytmu obliczeń skalarnych wielokrotnych

Ponieważ jednak, ma nastąpić cykliczne powtarzanie operacji aż do spełnienia warunku stopu (lub inaczej – tak długo dopóki warunek ten nie jest spełniony), więc jest to przypadek nadający się do zastosowania pętli programowej typu ***WHILE ... WEND***, a jeszcze lepiej pętli ***DO ... LOOP UNTIL ...***

W pętli ***WHILE*** warunek będzie sprawdzany na początku pętli, więc przed nią, musimy nadać zmiennym występującym w warunku takie wartości aby warunek stopu nie był spełniony i nastąpiło wejście do wnętrza pętli. Można to uzyskać przez podstawienie lub wprowadzenie z klawiatury wartości tych zmiennych przed pętlą, jednakże wewnątrz pętli – po obliczeniach i wydruku – trzeba ponownie umieścić polecenie wprowadzenia nowych wartości tych zmiennych, bo inaczej mielibyśmy ich niezmienne wartości – co spowodowałoby tak zwane „zapętlenie" programu – czyli działanie petli bez końca. W przypadku zapętlenia należy próbować zatrzymać program klawiszami [CTRL] [C] lub [CTRL] [Break]. Czasem trzeba to zrobić wielokrotnie.

```
PRINT "Trojkat prostokatny": ' PR 41
PRINT: PRINT "Podaj dane lub zero aby zakonczyc:"
INPUT "  przyprostokatna A="; PRZY1
INPUT "  przeciwprostokatna C="; PRZE
WHILE PRZY1 <> 0 AND PRZE <> 0
    IF PRZY1 >= PRZE THEN
        PRINT "Musi byc A<C"
    ELSE
        PRZY2 = SQR(PRZE ^ 2 - PRZY1 ^ 2)
        P = (PRZY1 * PRZY2) / 2
        PRINT "Przyprostokatna B="; PRZY2, "Pole="; P
    END IF
    PRINT: PRINT "Podaj nastepne dane lub zero aby zakonczyc:"
    INPUT "  przyprostokatna A="; PRZY1
    INPUT "  przeciwprostokatna C="; PRZE
WEND
PRINT "KONIEC"
SLEEP
```

5.1.3. OBLICZENIA ITERACYJNE BEZ UŻYCIA TABLIC

Podstawową niedogodnością poprzedniego algorytmu była konieczność wielokrotnego wpisywania danych z klawiatury. Jeśli jednak dane mogłyby zmieniać się w sposób regularny – na przykład tworząc ciąg arytmetyczny, geometryczny czy inny o określonym sposobie generowania kolejnych wartości, to taki ciąg może być wygenerowany automatycznie przez zastosowanie algorytmu iteracyjnego.

Algorytmy iteracyjne - powtarzające w pętli programowej określone operacje – stosowane są nie tylko do generowania ciągów, ale i do wielu innych procedur numerycznych, jak wyznaczanie sum, iloczynów, całkowanie, różniczkowanie, porządkowanie, sortowanie, wyznaczanie pierwiastków, minimów, maksimów i t.d.

Sporządźmy i zapiszmy w BASIC-u algorytm iteracyjny dla następującego zadania:

1. Szukane są: przyprostokątna i pole trójkąta prostokątnego
2. Oznaczenia zmiennych (inne niż poprzednio):
   A – przyprostokątna dana jako **ciąg** w zakresie od **Ap** do **Ak**

N = 100 – liczba elementów ciągu,
i – numer bieżącego elementu ciągu,
dA = (Ak-Ap)/(N-1) – przyrost zmiennej A,
C – dana przeciwprostokątna,
B – przyprostokątna szukana,
P – pole trójkąta

3. Warunek poprawności danych i wykonalności obliczeń:
   Ap<C i Ak<C

4. Program w języku BASIC:

```
PRINT "Trójkat prostokątny": ' PR 42
PRINT "Zmiany boku B i pola P w funkcji zmian boku A"
PRINT "Podaj dane:"
INPUT "  przeciwprostokatna C="; C
PRINT "  zakres zmian przyprostokątnej A: ";
INPUT ; " Ap="; Ap: INPUT " Ak="; Ak
INPUT "  liczba wierszy tabeli N="; N
dA = (Ak - Ap) / (N - 1)
PRINT "Bok A", "Bok B", "Pole"
FOR A = Ap TO Ak STEP dA
    PRINT USING "####.##"; A,
    IF A > C THEN
        PRINT "   *******"
    ELSE
        B = SQR(C ^ 2 - A ^ 2)
        P = (A * B) / 2
        PRINT USING "        ####.##"; B, P
    END IF
NEXT A
```

Zauważmy, że program **oblicza i drukuje tabelę** wartości zmiennych a mimo tego w programie tym **nie zastosowano tabel** jako struktur danych a jedynie zmienne skalarne.

Było to możliwe dzięki temu, że w algorytmie każda, pojedyncza, wygenerowana w pętli FOR, wartość zmiennej A zostaje użyta do obliczeń zaraz po wygenerowaniu a później nie jest do niczego potrzebna. W ramach tej samej pętli następuje także drukowanie pojedynczych wartości wyników, które również po wydrukowaniu nie są już do niczego potrzebne. Tak więc – w takim przypadku - nie zachodzi konieczność zapamiętywa ani ciągu wartości zmiennej ***A*** ani ciągów wyników.

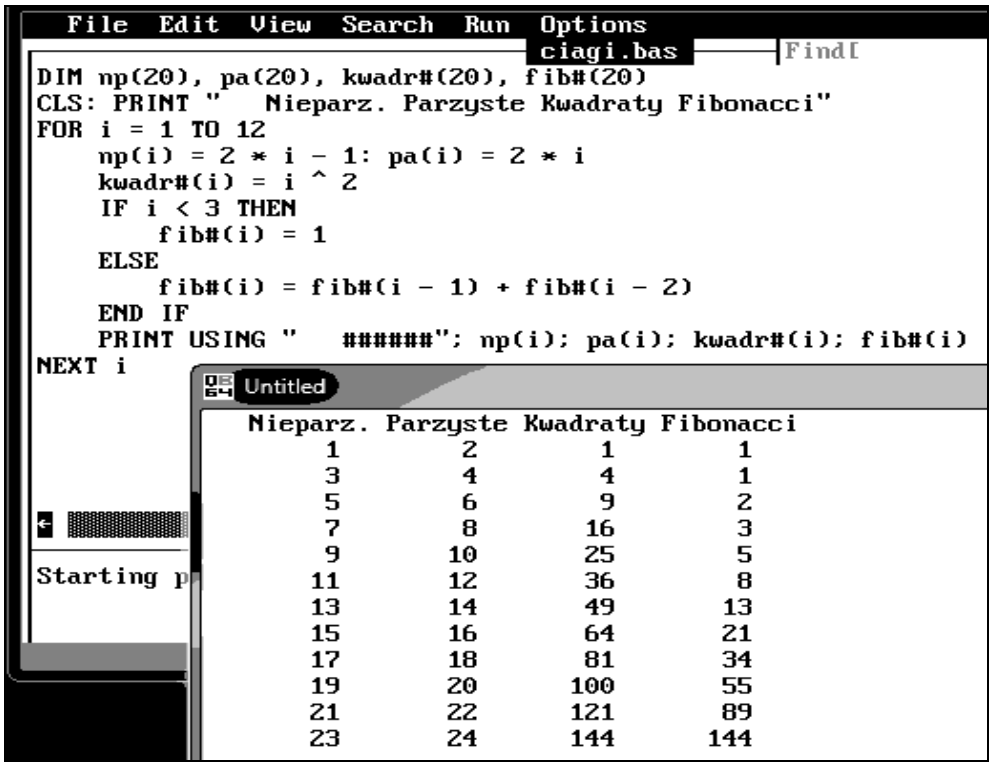
Innym przykładem algorytmu iteracyjnego nie używającego tablic jest przedstawiony dalej algorytm sumowania elementów ciągu.

Zastosowanie tablic (jedno lub wielo-wymiarowych) może natomiast być konieczne wtedy, gdy po zakończeniu obliczeń w pętli, zbiór wyników będzie jeszcze potrzebny do kolejnych operacji. Przykładem algorytmu w którym zaszła potrzeba zastosowania tablic jest algorytm wyznaczania wariancji (p.5.5.) – gdzie ciąg jest najpierw potrzebny do wyznaczenia średniej a potem w kolejnej pętli wyznaczane są odchylenia elementów tego ciągu od wyznaczonej średniej.

Inny przykład konieczności zapamiętania ciągu wyników jako wektora napotkamy przy pisaniu w MATLAB-ie programu rysowania wykresu funkcji z użyciem funkcji PLOT, której argumentami muszą być **wektory** wartości X oraz Y czyli tablice jednowymiarowe.

5.2. GENEROWANIE CIĄGÓW

Ciągi liczbowe możemy generować w pętli, według wzoru określającego wartość i-tego elementu ciągu na podstawie indeksu „i" a czasem także na podstawie wyrazów poprzedzających ten i-ty wyraz.



Rys. 5.5. Program (PR 43) generujący wybrane ciągi liczbowe

Jeśli ciąg ma być tylko wyświetlony czy wydrukowany to nie jest potrzebna tablica (wektor) do jego zapamiętania, jeśli jednak będzie jeszcze wykorzystywany do innych celów to jego wyrazy muszą być zapamietane w kolejnych elementach wektora, do czego potrzebna będzie zmienne pełniąca rolę indeksu i przyjmująca kolejne wartości naturalne: 1, 2, 3, ...

Przykładowy program generowania kilku ciągów pokazano na Rys. 5.5.

5.3. TABELARYZACJA I WYKRES FUNKCJI Y(X)

Generowanie tabeli wartości funkcji w zasadzie nie różni się od generowania ciągów, ale tym razem chcemy wyprowadzić kolumny wartości X oraz Y(X) do pliku tekstowego. Operacje te realizuje poniższy program tabelaryzacji funkcji do pliku tekstowego, którego wadą jest fakt, że zarówno funkcję:

$$y(x) = \frac{\sin(2 \cdot x)}{e^{0.2*x}}$$

jak i granice Xp, Xk przedziału zmienności zmiennej X, zdefiniowano wewnątrz programu i tam też musi się je ewentualnie modyfikować. W niektórych językach – na przykład w MATLAB-ie a także języku Clipper – istnieje możliwość wczytania danych tekstowych w których zapisane są wzory a następnie realizacji obliczeń według tych wzorów. Niestety brak takich środków w BASIC-u.

```
' PR 44
CLS: PRINT "Program tabelaryzuje funkcję do pliku WYNIKI.TXT
'==== Tworzymy plik "WYNIKI.TXT" do zapisu jako kanał  nr.#1:
OPEN "wyniki.txt" FOR OUTPUT AS #1
'==== Do tego pliku generujemy wartości x oraz funkcji:
Xp = -1: Xk = 9: Dx = (Xk - Xp) / 200
FOR X = Xp TO Xk STEP Dx
    PRINT X, SIN(2 * X) / EXP(.2 * X)
    PRINT #1, X, SIN(2 * X) / EXP(.2 * X)
NEXT X
'==== Zamykamy plik
CLOSE #1
```

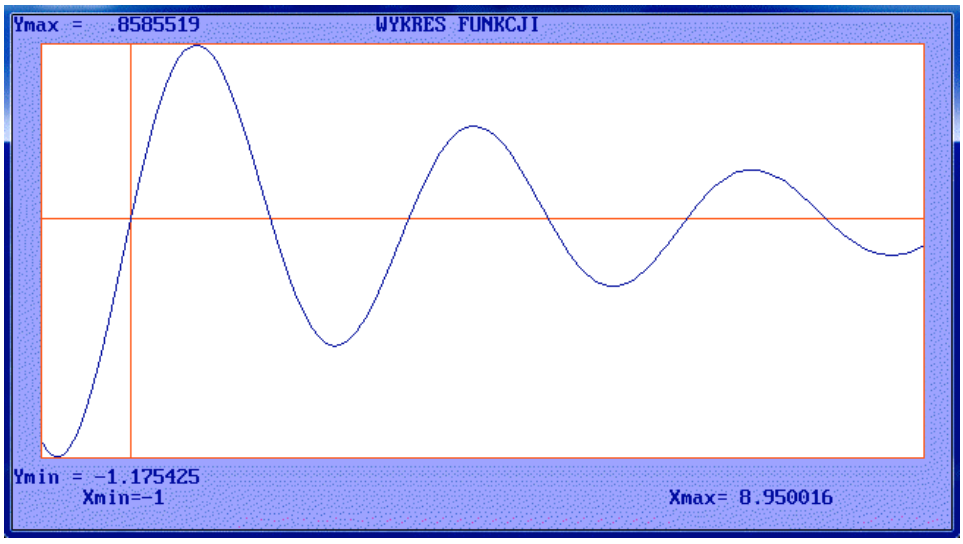
Polecenie OPEN utworzyło w bieżącym folderze plik o nazwie "wyniki.txt" w trybie „do wyprowadzania" (*FOR OUTPUT*) i przypisało mu numer kanału **#1**. Na ten numer muszą powoływać się wszystkie polecenia (np.: PRINT) wyprowadzające wyniki i teksty do pliku "wyniki.txt". Polecenie PRINT nie zawierające identyfikatora #1 wyswietli dodatkowo wyniki na ekranie (aby użytkownik nie poczuł się zbyt zagubiony).

A teraz następny program, który ma wczytać dane z pliku i na ich podstawie wygenerować wykres, automatycznie dostosowując do niego rozmiary okna graficznego:

```
' PR 45
'==== Deklarujemy dwie tablice jednowymiarowe:
CLS: DIM xx(1000), yy(1000)
'==== Otwieramy plik do odczytu (nr 2)
OPEN "wyniki.txt" FOR INPUT AS #2
w = 1: INPUT #2, xx(w), yy(w)
'==== Wyznaczamy granice wykresu:
xmin = xx(w): xmax = xx(w)
ymin = yy(w): ymax = yy(w)
WHILE NOT EOF(2)
    w = w + 1
    INPUT #2, xx(w), yy(w)
    IF xx(w) > xmax THEN xmax = xx(w)
```

```
    IF xx(w) < xmin THEN xmin = xx(w)
    IF yy(w) > ymax THEN ymax = yy(w)
    IF yy(w) < ymin THEN ymin = yy(w)
WEND
'====== Dalej już w trybie graficznym
SCREEN 9: COLOR 1, 15
'== Definiujemy okno graficzne
VIEW (20, 20)-(620, 300), 15, 12
'== Przypisujemy rogom okna granice wykresu:
WINDOW (xmin, ymin)-(xmax, ymax)
'== Wyprowadzamy napisy:
LOCATE 1, 1: PRINT "Ymax = "; ymax;
LOCATE 1, 32: PRINT "WYKRES FUNKCJI";
LOCATE 23, 1: PRINT "Ymin = "; ymin;
LOCATE 24, 7: PRINT "Xmin="; xmin;
LOCATE 24, 57: PRINT "Xmax="; xmax;
'===== rysujemy osie:
LINE (xmin, 0)-(xmax, 0), 12
LINE (0, ymin)-(0, ymax), 12
'===== i rysujemy wykres:
FOR i = 1 TO w - 1
    LINE (xx(i), yy(i))-(xx(i + 1), yy(i + 1))
NEXT i
SLEEP
```

Efekt działania tego programu pokazano na Rys. 5.6. Jak widać, znacznie łatwiej i lepiej robić wykresy w arkuszu kalkulacyjnym lub uniwersalnym programie matematycznym, chociaż, przy programowaniu mamy (teoretycznie) nieograniczone możliwości.



Rys. 5.6. Wynik działania programu PR 45

Problemy skalowania wykresu – w przypadku gdyby nie było polecenia WINDOW – oraz zadanie automatycznego wygenerowania sensownej siatki – pozostawiam do rozważań czytelnikowi.

5.4. ALGORYTM SUMOWANIA

Jako prosty przykład iteracji spróbujmy skonstruować algorytm sumowania elementów dowolnego ciągu ***N*** liczb:

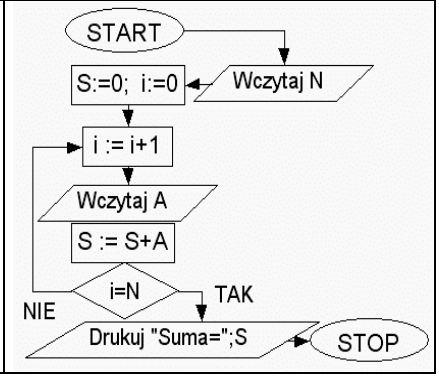
$$A_1, A_2, A_3, ..., A_N$$

Zaczniemy od nieudolnych prób i ich krytyki. Rys. 5.7a pokazuje pierwszą próbę zapisania sekwencji wyznaczania sum częściowych ($S_1, S_2,$ ...). Widać, że pierwsza linijka nie jest podobna do pozostałych a zmiany indeksów mogłyby być uproszczone.

Lepszy sposób pokazuje kolumna (b), gdzie dzięki wstawieniu zera jako początkowej wartości sumy udało się uzyskać prostszą i regularną postać pozostałych wierszy. Czy jednak potrzebne nam jest zapamiętywanie całego ciągu sum częściowych – NIE! – wystarczy więc pojedyncza zmienna skalarna ***S***, która będzie przechowywała kolejne wartości sumy, co pokazuje kolumna (c) na Rys. 5.7.

Kolejne pytanie – czy konieczna jest tablica dla przechowywania całego ciągu A_i ?

W algorytmach wyznaczających sumę (lub iloczyn) elementów ciągu, można uniknąć zastosowania tablicy, ponieważ po wczytaniu elementu ciągu i wykorzystaniu go do uaktualnienia bieżącej wartości sumy (lub iloczynu), kolejny element ciągu można znowu wczytać do tej samej zmiennej skalarnej, co ostatecznie pokazuje algorytm Rys. 5.7d.

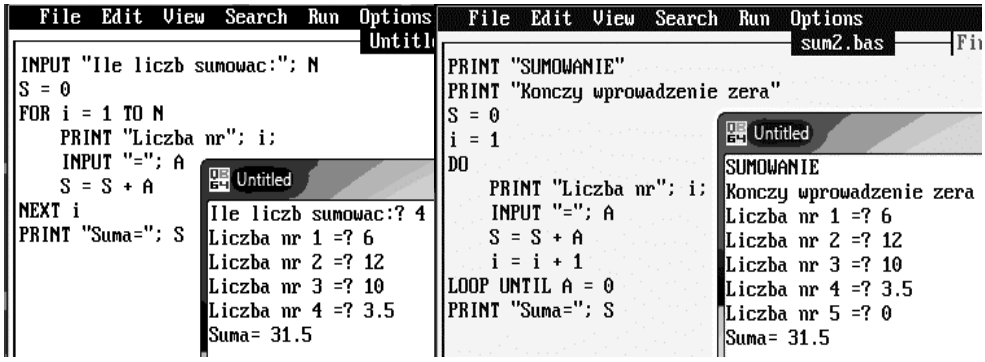
a)	b)	c)	d)
$S_1 = A_1 + A_2$ $S_2 = S_1 + A_3$ $S_3 = S_2 + A_4$ . . . . . . $S_i = S_{i-1} + A_{i+1}$	$S_0 = 0$ $S_1 = S_0 + A_1$ $S_2 = S_1 + A_2$ . . . . . . $S_i = S_{i-1} + A_i$	$S := 0$ $S := S + A_1$ $S := S + A_2$ . . . . . . $S := S + A_i$	

Rys. 5.7. Konstruowanie algorytmu sumowania

Jak widać, zmienna ***„i”*** pozostała jako licznik elementów ciągu ale udało się uniknąć zastosowania tablic i indeksów, co oczywiście nie zawsze będzie możliwe. Zawsze jednak – przed zastosowaniem pętli – musimy uzyskać identyczną postać powtarzanych instrukcji.

Okazało się więc, że dla algorytmu sumowania ciągu liczb niezbędne są tylko cztery zmienne, które bedą reprezentować (oraz przechowywać):

i - numer elementu ciągu,
N - liczbę elementów ciągu,
A - wartość i-tego elementu ciągu,
S - wartość sumy elementów (o numerach od 1 do i).



Rys. 5.8. Dwie wersje programu sumowania ciągu liczb (PR 46 i PR 47)

Zasadnicze znaczenie dla iteracji mają objaśnione już wczesniej instrukcje przypisania – takie jak (i=i+1) lub (S=S+A) - w których po obu stronach znaku przypisania występuje ta sama zmienna. Należy jednak pamiętać, że: **po prawej** stronie do obliczeń brane są **dotychczasowe** wartości zmiennych, a wynik obliczeń zostanie przypisany zmiennej **po lewej** stronie jako **nowa** wartość.

Dwie wersje programów sumujących dowolne liczby podawane z klawiatury pokazuje Rys. 5.8. W pierwszej trzeba na początku podać ile liczb będzie sumowanych a w drugiej nie potrzeba i wystarczy ciąg podawanych liczb zakończyć zerem.

5.5. ŚREDNIA, WARIANCJA I ODCHYLENIE STANDARDOWE

Mając ciąg wyników pomiarów najczęściej wyznaczamy dla niego wartość średnią oraz wariancję i odchylenie standardowe jako miary rozrzutu wyników wokół średniej.

Wariancja to „średnie kwadratowe odchylenie od średniej" natomiast odchylenie standardowe to „pierwiastek kwadratowy z wariancji".

Jeśli wyniki pomiarów zawarte są w ciągu:

$$A_1, A_2, A_3, \ldots, A_N$$

to wariancja V oraz odchylenie standardowe σ dane są wzorami:

$$V = \frac{1}{N} \cdot \sum_{i=1}^{N} (A_i - \mu)^2 \qquad \sigma = \sqrt{V} \qquad \text{gdzie: } \mu = \frac{1}{N} \cdot \sum_{i=1}^{N} A_i \text{ to średnia}$$

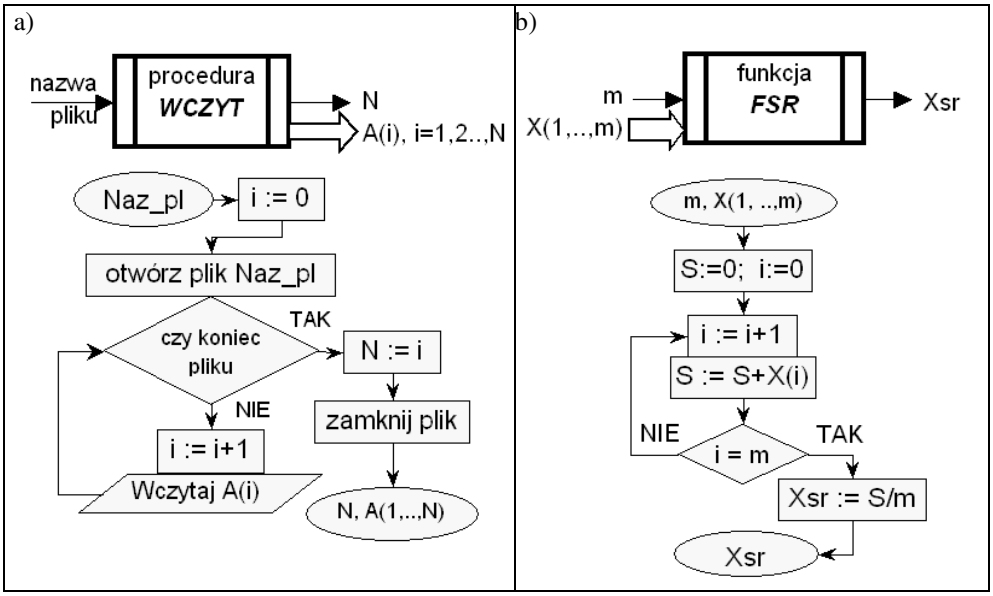
Jak widać, operacje na ciągu liczb będą dokonywane w algorytmie więcej niż jeden raz więc ciąg ten należy wczytać. Oprócz tego widać, że wyznaczanie średniej występuje dwukrotnie. Opłaca się więc skonstruować podprogram funkcyjny czyli osobny algorytm wyznaczania średniej ciągu liczb (Rys. 5.9b), a następnie dwukrotnie ten algorytm wykorzystać. Przydatne może być też jednorazowe zdefiniowanie algorytmu wczytywania ciągu liczb z określonego pliku dyskowego (Rys. 5.9a).

Wczytywanie ciągu liczb z pliku dyskowego (Rys. 5.9a) polega na otwarciu pliku (o danej nazwie ***naz_pl***) a następnie - sprawdzaniu czy pojawił się znacznik końca pliku. Jeśli nie pojawił się, to następuje zwiększenie indeksu „***i***" oraz wczytanie kolejnej liczby z pliku do elementu „***A(i)***". Jeśli natomiast znacznik ten pojawił się, to aktualna wartość indeksu zostaje uznana za liczbę „***N***" wczytanych elementów, plik zostaje zamknięty a działanie podprogramu zakończone. Jak widać, w podprogramie tym zastosowano pętlę programową typu „dopóki ...", opisaną w 3.5.

Funkcja FSR (Rys. 5.9b) posłuży do wyznaczania średniej arytmetycznej. Zastosowano w niej sumowanie zrealizowane tak samo jak w p. 5.2. , przy pomocy pętli programowej typu „dla ...".

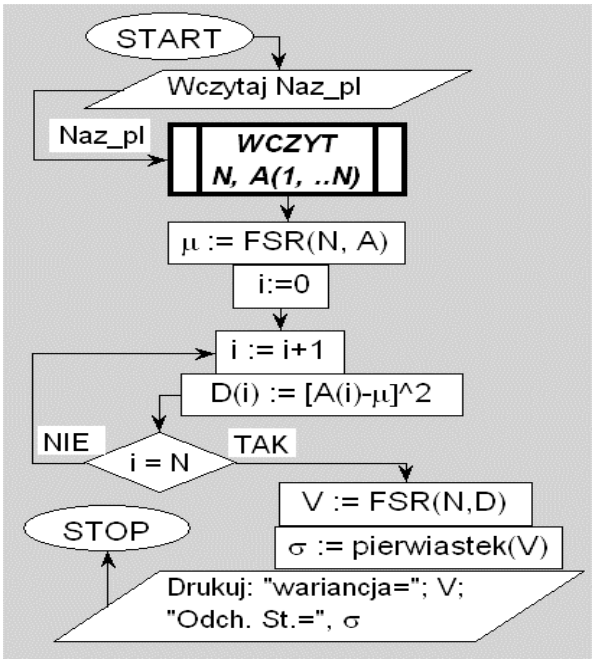
Ostatecznie, algorytm wyznaczania średniej arytmetycznej μ, wariancji **V** i odchylenia standardowego σ, dla ciągu wyników pomiarów wczytanych z dysku, może mieć postać taką jak na Rys. 5.10.

Jak widać, do wczytania ciągu zastosowano w nim zdefiniowaną uprzednio procedurę WCZYT, natomiast do dwukrotnego wyznaczania średniej - funkcję FSR.



Rys. 5.9. Podprogramy: a) procedura wczytywania ciagu liczb z pliku;
b) funkcja obliczania średniej arytmetycznej ciagu liczb

Przykład ten najlepiej zaprogramować w MATLAB-ie gdyż w BASIC-u tablice nie mogą być parametrami procedur. Zamiast tego mogą być one wspólne – dostępne dla programu głównego i procedur - jeśli zadeklarujemy je deklaracją DIM SHARED.



Rys. 5.10. Algorytm wyznaczania wariancji i odchylenia standardowego (PR 48)

5.6. ALGORYTM SELEKCJI ELEMENTÓW CIĄGU

Elementy danego ciągu A, które są większe od danej wartości G mają być przeniesione do ciągu B a pozostałe elementy przeniesione do ciągu C.

Zauważmy, że numery (indeksy) elementów tych trzech ciągów muszą być niezależne od siebie a więc potrzeba dla nich trzech osobnych zmiennych ***i, j, k***:

A(i), i=1, 2, ..., Li; ***B(j), j=1, 2, ..., Lj;*** ***C(k), k=1, 2 ...,Lk***

Wprowadziliśmy więc następujące oznaczenia zmiennych :

A – wektor danych,

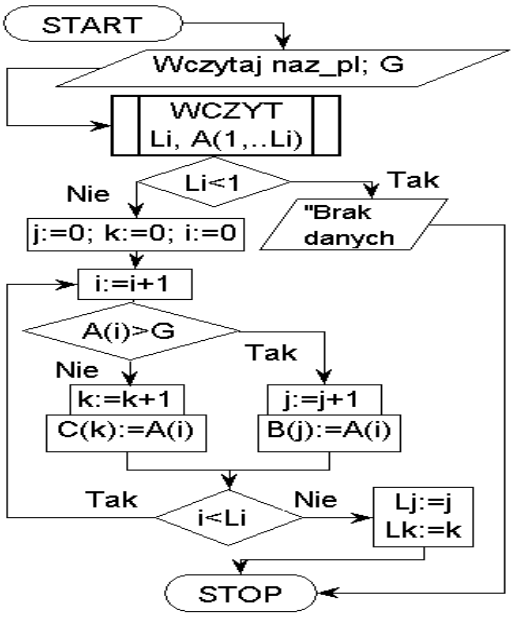
B, C – wektory wynikowe,

i, j, k – indeksy tych trzech wektorów,

Li, Lj, Lk – liczby elementów tych wektorów

G – wartość progowa.

Nasz algorytm ma działać dla dowolnych danych, a w szczególności także dla braku elementów w ciągu danych ***A***. Również, jeden z wektorów ***B, C*** może nie uzyskać ani jednego elementu, a wówczas do drugiego wektora przejdą wszystkie elementy z wektora ***A***.



Rys. 5.11. Algorytm selekcji elementów ciągu (PR 49)

5.7. OBLICZANIE WARTOŚCI WIELOMIANU

Algorytmy wielu obliczeń, przeznaczone dla komputera, staramy się tak sformułować aby – jesli to możliwe – występowało cykliczne powtarzanie możliwie prostych operacji.

Obliczanie wartości wielomianu według wzoru:

$$W(x) := a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + ... + a_1 \cdot x + a_0$$

wymaga zastosowania **n** dodawań oraz **1+2+...+n** mnożeń i nie jest najlepszą dla komputera metodą.

Lepiej zastosować w tym celu tak zwany **schemat Hornera** (PR 50):

$$W(x) := (...((a_n \cdot x + a_{n-1}) \cdot x + a_{n-2}) \cdot x + ... + a_1) \cdot x + a_0 ,$$

w którym mnożeń jest tyle samo co dodawań, a co najważniejsze osiągnięto cykliczną powtarzalność tych dwu operacji.

5.8. ROZWINIĘCIA FUNKCJI W SZEREG MACLAURINA

Rozwinięcie funkcji *f(x)* w szereg potęgowy pozwala obliczać przybliżone wartości tej funkcji przy pomocy odpowiedniego wielomianu. Jednym z najczęściej stosowanych jest szereg Maclaurina. Jest to szereg potęgowy o i-tym wyrazie równym:

$$a_i = \frac{f^{(i)}(0)}{i\,!}x^i$$

gdzie: $f^{(i)}(0)$ to wartość i-ej pochodnej funkcji *f(x)* dla x=0

Jeśli funkcja *f(x)* jest różniczkowalna *n* razy to jej wartość *x* w otoczeniu zera można wyznaczyć wzorem Maclaurina:

$$f(x) = f(0) + \sum_{i=1}^{n-1} a_i + R_n$$

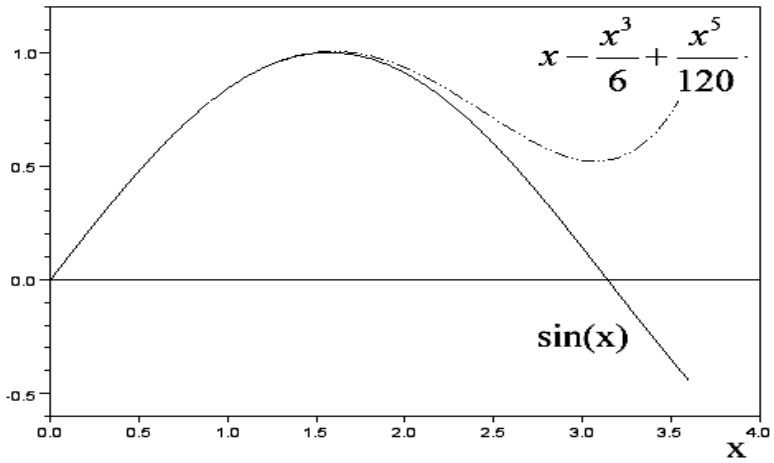
(tak zwana „reszta" R_n dąży do zera gdy *n* dąży do nieskończoności)

Podstawiając poprzedni wzór, otrzymujemy praktyczne rozwinięcie wzoru Maclaurina:

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \frac{f^{(4)}(0)}{4!}x^4 + \ldots$$

Jako przykład, wyznaczmy kilka wyrazów rozwinięcia dla funkcji sinus. Ponieważ ***sin(0)=0 a cos(0)=1*** oraz pochodne: ***(sin(x))' = cos(x), (cos(x))'=-sin(x)*** więc wzory na kolejne pochodne funkcji sinus i ich wartości w zerze są następujące:

`f'(x)=cos(x),`	`f''(x)=-sin(x),`	`f'''(x)=-cos(x),`	$f^{(4)}(x)=\sin(x)$
`f'(0)=cos(0),`	`f''(0)=-sin(0),`	`f'''(0)=-cos(0),`	$f^{(4)}(0)=\sin(0)$
`f'(0)=1,`	`f''(0)=0,`	`f'''(0)=-1,`	$f^{(4)}(0)=0$



Rys. 5.12. Przybliżenie funkcji sinus wielomianem Maclaurina

Stąd dla *f(x)=sin(x)* otrzymujemy: $\sin(x) \cong 0 + x + 0 - \frac{x^3}{3!} + 0 + \frac{x^5}{5!} + 0 + \ldots$

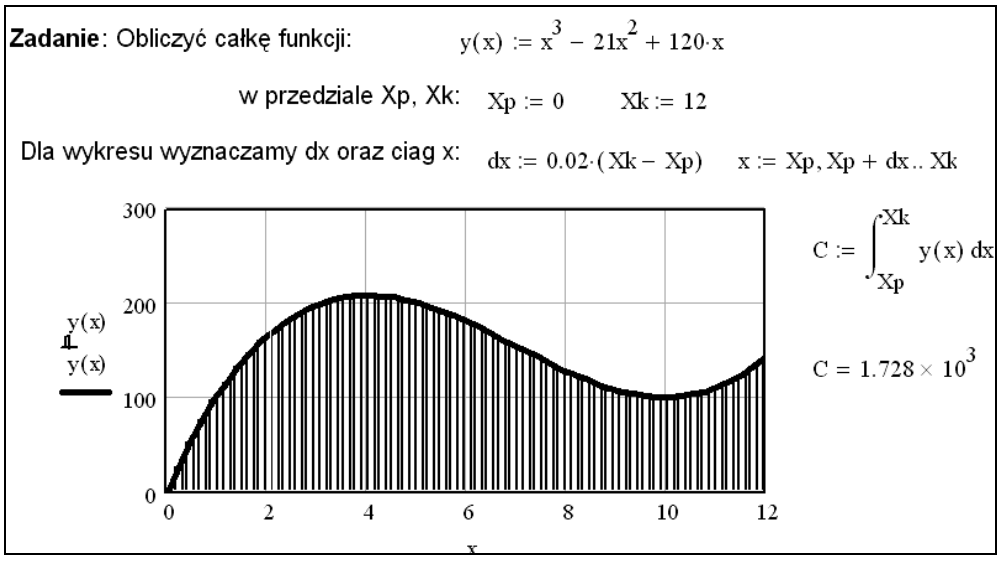
Należy jednak pamiętać, że przybliżenie jest słuszne w otoczeniu zera, co widać na wykresie (Rys. 5.12) porównującym funkcję sinus z odpowiadającym jej wielomianem Maclaurina.

5.9. CAŁKOWANIE NUMERYCZNE

Wyznaczanie wartości całki oznaczonej dla danej funkcji ***y(x)***, w danym przedziale ***[Xp, Xk]***, sprowadza się do wyznaczenia w tym przedziale pola między osią ***x*** a krzywą wykresu funkcji (Rys. 5.13). Większość metod numerycznych realizuje to zadanie przez podział tego pola na pionowe paski (np.: o kształcie prostokątów lub trapezów) i wyznaczenie sumy ich pól.

Inny sposób – oparty o równomierne pseudolosowe rozsiewanie punktów i założenie, że ich liczba jest proporcjonalna do pola danego obszaru – wykorzystywany jest w metodzie Monte Carlo.

Rys. 5.13 (dokument uzyskany w Mathcadzie) przedstawia wykres funkcji i zaznaczone pole pod krzywą - które mamy wyznaczyć jako wartość całki, oraz podaje tą wartość (wyznaczoną operatorem całkowania) dla możliwości porównania jej z wynikami uzyskiwanymi dla tych samych danych kilkoma metodami całkowania numerycznego.

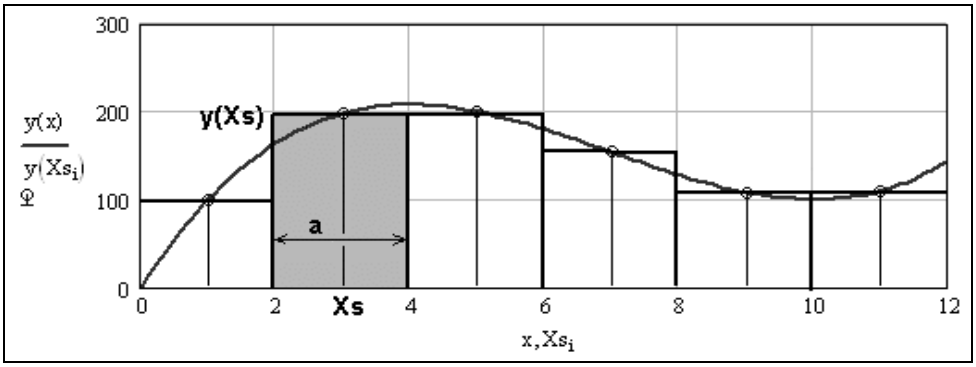


Rys. 5.13. Wykres funkcji w Mathcadzie i obliczenie całki

Omówimy tylko trzy z wielu metod całkowania numerycznego, a mianowicie „metodę prostokątów", „metodę trapezów" i „metodę Monte Carlo".

5.9.1. METODA PROSTOKĄTÓW

Ideę obliczania całki metodą prostokątów prezentuje wykres na Rys. 5.14 a przebieg obliczeń w Mathcadzie (których rezultatem jest ten wykres) pokazuje Rys. 5.15. Dla przejrzystości rysunku zastosowano bardzo małą liczbę prostokątów, czego oczywiście nie należy stosować w praktyce gdyż pogorszyłoby dokładność obliczeń.



Rys. 5.14. Idea obliczania całki metodą prostokątów

Liczba prostokątów: `LP:=6` `i:= 1..LP` `ORIGIN ≡ 1`

Podstawa prostokąta: $a := \frac{Xk - Xp}{LP}$ $a = 2$

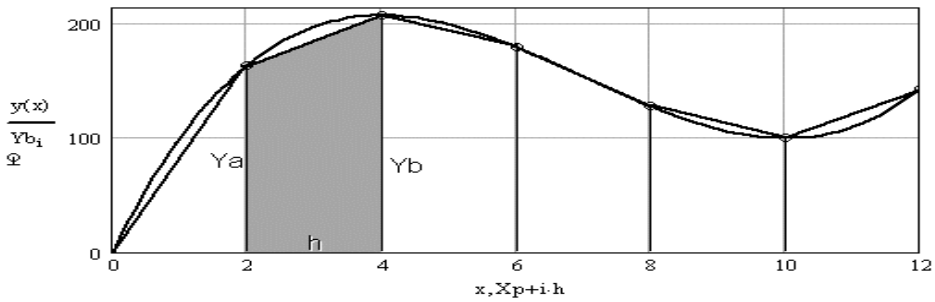
Odcięte środków prostokątów: $Xs_i := Xp + i \cdot a - 0.5 \cdot a$

Wysokości prostokątów: $y(Xs_i)$

Suma pól trapezów: $SP := \sum_i a \cdot y(Xs_i)$ $SP = 1.74 \times 10^3$

Rys. 5.15. Obliczanie w Mathcadzie całki metodą prostokątów (PR 51)

5.9.2. METODA TRAPEZÓW



Rys. 5.16. Idea obliczania całki metodą trapezów

Liczba trapezów: $LP := 6$ $i := 1..LP$ $ORIGIN \equiv 1$

Wysokość trapezu: $h := \frac{Xk - Xp}{LP}$ $h = 2$

Podstawa Ya trapezu: $Ya_i := y[Xp + (i-1)\cdot h]$

Podstawa Yb trapezu: $Yb_i := y[Xp + i \cdot h]$

Suma pól trapezów: $SPT := \sum_i h \cdot 0.5 \cdot (Ya_i + YB_i)$ $SPT = 1.704 \times 10^3$

Rys. 5.17 Obliczanie w Mathcadzie całki metodą trapezów (PR 52)

Ideę obliczania całki metodą trapezów prezentuje wykres na Rys. 5.16 a przebieg obliczeń w Mathcadzie (których rezultatem jest ten wykres) pokazuje Rys. 5.17. Dla przejrzystości rysunku zastosowano bardzo małą liczbę trapezów co oczywiście pogarsza dokładność obliczeń.

Podobną do metody trapezów, lecz bardziej złożoną, jest metoda parabol (Simpsona), gdzie ukośny bok trapezu jest zastępowany odcinkiem paraboli.

5.9.3. METODA MONTE CARLO

Metodami Monte Carlo nazywana jest grupa metod symulacyjnego rozwiązywania zagadnień poprzez generowanie wielu wartości pseudolosowych poszczególnych zmiennych, (według określonych rozkładów prawdopodobieństwa), a następnie zliczanie zdarzeń polegających na spełnieniu określonych warunków.

Wyznaczamy prostokąt dla losowego rozmieszczania punktów:

granicami boku poziomego są: Xp = 0 Xk = 12

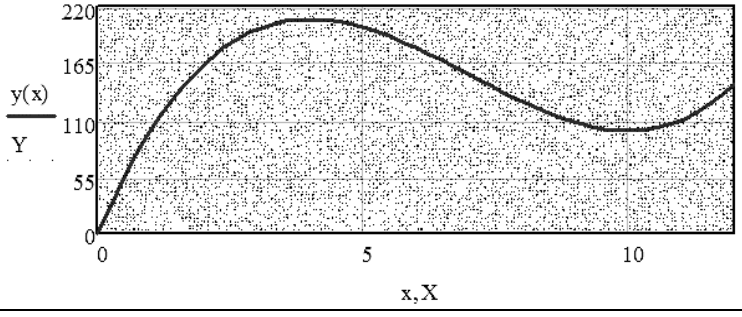
a wysokość boku pionowego przyjmujemy (w/g rysunku): H := 220

Liczba losowanych punktów: N := 10000

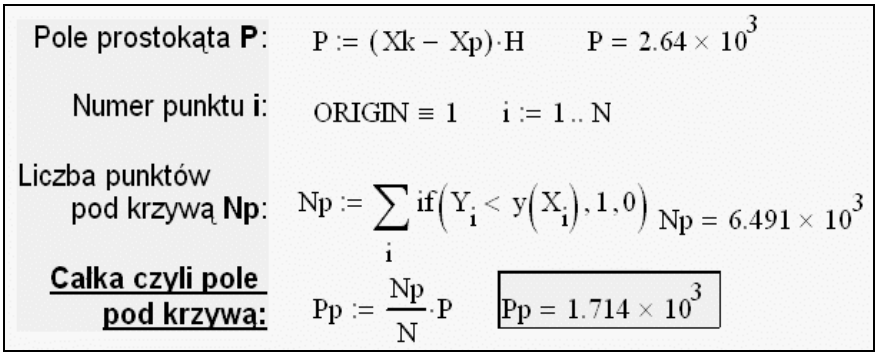
do losowania wektorów współrzędnych **X, Y**

wykorzystamy funkcję ***runif***: X := runif(N, Xp, Xk)

Zauważ, że **X, Y** to inne zmienne niż **x, y** Y := runif(N, 0, H)



Rys. 5.18. Obliczanie całki metodą Monte Carlo (AL6) - wykres



Rys. 5.19. Obliczanie całki metodą Monte Carlo – obliczenia (PR 53)

Obliczanie całek metodą Monte Carlo polega na określeniu prostokąta (Rys. 5.18), którego podstawą jest przedział całkowania **[Xp; Xk]** a wysokość **H** jest dowolna ale nie mniejsza od maksymalnej wartości funkcji **y(x)** w przedziale całkowania.

W tym prostokącie rozmieszczanych jest N punktów o współrzędnych **Xi, Yi** generowanych pseudolosowo według rozkładów równomiernych (Rys. 5.18).

Przy założeniu, że liczba punktów w danym obszarze jest w przybliżeniu proporcjonalna do pola tego obszaru, przybliżoną wartość całki, czyli pola **Pp** pod krzywą, można otrzymać z proporcji: **Pp : P = Np : N**, gdzie: **P** to pole prostokąta a **Np** liczba punktów pod krzywą (Rys. 5.19).

5.10. SORTOWANIE ZBIORÓW LICZBOWYCH

Sortowanie danych polega na ustawieniu elementów zbioru danych w określonym porządku, na przykład alfabetycznym, chronologicznym, a w najprostszym przypadku, dla ciągu liczbowego – monotonicznym. Ponieważ wszelkie dane są w komputerze reprezentowane przez liczby więc ich sortowanie zawsze sprowadza się do sortowania zbioru liczb. Niniejszy podrozdział przedstawia kilka **metod sortowania zbiorów liczbowych**.

Zakładamy, że dane wejściowe stanowi wektor (zbiór) $\boldsymbol{n}$ dowolnych liczb $\boldsymbol{V_i}$
gdzie $\boldsymbol{i}$ = 1, 2, 3, ... $\boldsymbol{n}$ - jest numerem elementu ciągu.

Przyjmujemy, że ciąg ten ma być **uporządkowany rosnąco** to znaczy dla każdej pary sąsiadujących elementów ma zachodzić relacja: $\boldsymbol{V_i \geq V_{i-1}}$, dla $\boldsymbol{i}$=2,3,4,...$\boldsymbol{n}$.

5.10.1. SORTOWANIE BĄBELKOWE

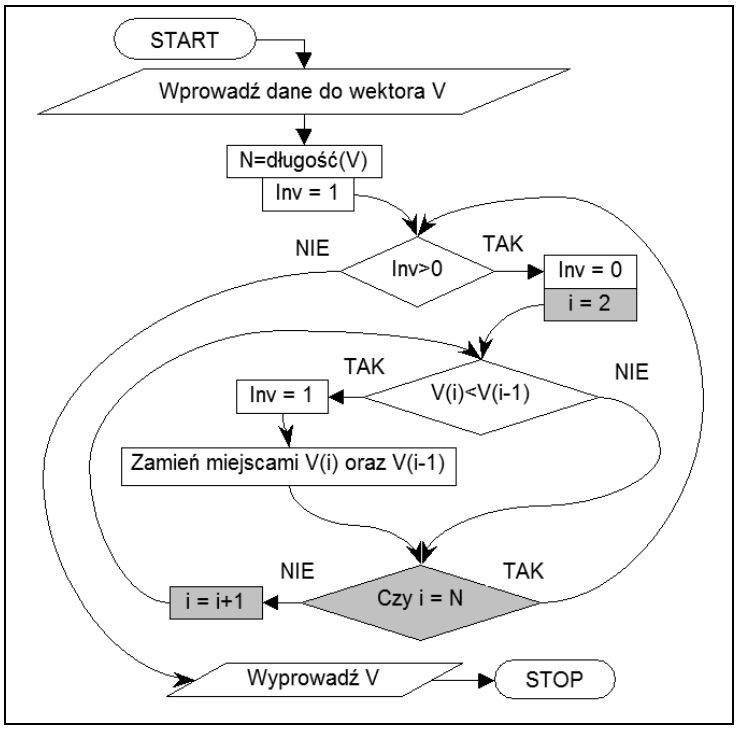
Algorytm **sortowania bąbelkowego** (AL7) polega na sprawdzaniu czy każda para sąsiadujących elementów ciągu zachowuje żądany porządek, a więc czy $\boldsymbol{V_j \geq V_{j-1}}$, a jeśli warunek ten nie jest spełniony (czyli zachodzi inwersja ustawienia) to algorytm zamienia kolejność tych elementów.

Rozpatrzmy przykładową sytuację, że V_4=12,5 a V_5=11. Gdybyśmy wpisali polecenie $V_5 := V_4$ to bezpowrotnie tracimy dotychczasową wartość V_5 czyli 11. Do operacji **zamiany** potrzebna jest więc pomocnicza zmienna (zwana buforową) np.: ***B***, która będzie pełnić podobną rolę jak stolik w sytuacji gdy trzymamy w lewej ręce kieliszek a w prawej filiżankę herbaty i chcemy zamienić je miejscami:

$$B:=V_i ; \qquad V_i := V_{i-1}; \qquad V_{i-1} :=B$$

Po wykonaniu takich trzech operacji podstawiania, wartości pary elementów zamienią się miejscami.

Na ogół po jednym przebiegu (sprawdzeniu wszystkich par elementów) ciąg nie jest jeszcze posortowany i wymagane są kolejne przebiegi, tak długo, aż przy którymś z przebiegów nie zostanie wykryta ani jedna inwersja i nie nastąpi ani jedna zamiana. Jest więc potrzebna następna pomocnicza zmienna (zwana czasem flagą – po angielsku *flag*), która musi zmienić wartość („wystawić flagę") przy operacji zamiany elementów. Nazwijmy tą zmienną ***Inw*** gdyż sygnalizuje, że były inwersje (***Inw***=1) lub, że ich nie było (***Inw***=0). Zgodny z tą analizą algorytm pokazuje Rys. 5.20. Indeksy elementów ciągu zapisano w nawiasach – tak jak się to robi w większości języków programowania.



Rys. 5.20. Sortowanie bąbelkowe (PR 54)

Algorytm ten nie jest jedynym możliwym a co gorsze należy do najwolniejszych. Wiele innych, lepszych, algorytmów sortowania przedstawiono w [37].

5.10.2. SORTOWANIE PRZEZ WYBÓR

Nieco inną – z wielu możliwych metod - jest „sortowanie przez wybór". Załóżmy, że mamy nieposortowany zbiór ***A(1), A(2), ..., A(N)***.
Metoda realizowana jest w następujących krokach:

- wyszukujemy najmniejszy element w zbiorze, zapamiętujemy jego indeks w zmiennej ***Nmin*** a następnie zamieniamy z pierwszym elementem zbioru
- wśród pozostałych elementów wyszukujemy znowu element najmniejszy i zamieniamy z drugim elementem zbioru a ogólnie z elementem ***A(j)*** gdzie ***j=1, 2, ..., (N-1)***

Oto program w języku BASIC realizujący sortowanie zbioru 20 elementowego:

```
' PR 55
N = 20: '--- liczba el. zbioru.
DIM A(1 TO N) AS INTEGER
PRINT " SORTOWANIE PRZEZ WYBOR "
PRINT "------------------------"
' ---- Wypelniamy tablice A liczbami pseudolosowymi:
RANDOMIZE TIMER
FOR i = 1 TO N: A(i) = INT(RND * 100): NEXT i
' ---- Wyswietlamy tablice A przed sortowaniem:
PRINT "Przed sortowaniem:"
FOR i = 1 TO N: PRINT USING "####"; A(i);: NEXT i
PRINT
' ---- Sortujemy
FOR j = 1 TO N - 1
    Nmin = j
    FOR i = j + 1 TO N
        IF A(i) < A(Nmin) THEN Nmin = i
    NEXT i
    BB = A(j): A(j) = A(Nmin): A(Nmin) = BB
NEXT j
' ---- Wyswietlamy tablice A po sortowaniu:
PRINT: PRINT "Po sortowaniu:"
FOR i = 1 TO N
    PRINT USING "####"; A(i);
NEXT i
```

Przykładowe wyniki działania tego programu mogą być następujące:

```
 SORTOWANIE PRZEZ WYBOR
-----------------------
Przed sortowaniem:
  90  97  11  11  55  80  53  36  48  25  75  20  73  48  41  52  17  19  38  47

Po sortowaniu:
  11  11  17  19  20  25  36  38  41  47  48  48  52  53  55  73  75  80  90  97
```

Instrukcja RANDOMIZE TIMER inicjuje generator liczb pseudolosowych stanem zegara czasu rzeczywistego, co zapobiega powtarzaniu się sekwencji pseudolosowych.

Funkcja RND losuje liczbę pseudolosową z zakresu (0;1) a funkcja INT zamienia na wartość całkowitą.

5.11. ROZWIĄZYWANIE RÓWNAŃ NIELINIOWYCH

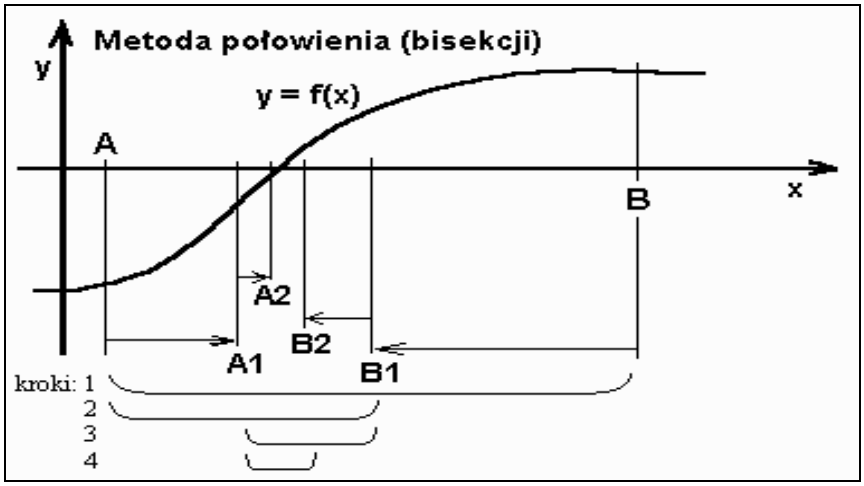
Numeryczne rozwiązywanie równania **f(x) = 0**, zwane inaczej znajdowaniem pierwiastków albo znajdowaniem miejsc zerowych funkcji **f(x)**, wymaga najczęściej najpierw określenia przedziałów w których takie pierwiastki się znajdują. W każdym takim przedziale **(A; B)** funkcja przechodząc przez zero zmienia znak, a więc wartości **f(A), f(B)** mają różne znaki: **f(A) · f(B) < 0**.

Dla dokładniejszego wyznaczenia pierwiastka **xp** w takim przedziale stosowane są różne metody, z których trzy opisano poniżej. We wszystkich trzech metodach muszą być spełnione warunki:

1) funkcja **f(x)** jest ciągła w przedziale **[A; B]**,
2) na końcach przedziału **[A; B]** funkcja ma różne znaki,
3) w przedziale tym istnieje dokładnie jeden pierwiastek
   a dodatkowe warunki zależą od przyjętej metody.

Metody te są metodami iteracyjnymi w których powtarza się pewien cykl operacji a dokładność zależy od liczby powtórzeń. Sygnałem do zakończenia iteracji jest spełnienie pewnego „warunku stopu", na przykład stwierdzenie, że wartość funkcji w znalezionym punkcie **xp** dostatecznie mało różni się od zera albo, że kolejna wartość **xp** dostatecznie mało różni się od poprzedniej.

5.11.1. METODA BISEKCJI



Rys. 5.21. Metoda bisekcji (PR 56)

Metoda bisekcji zwana też „metodą połowienia" albo „równego podziału" polega na wyznaczeniu pierwszego przybliżenia pierwiastka **xp** jako środka przedziału [A; B], a następnie przyjęcia jako nowego przedziału [A; B] tej z dwu połówek w której funkcja zmienia znak. Inaczej mówiąc końce przedziału A i B są przesuwane coraz bliżej pierwiastka, co pokazano na Rys. 5.21 jako A1, A2, ...; B1, B2, ...

Algorytm można wyrazić w następujących krokach:

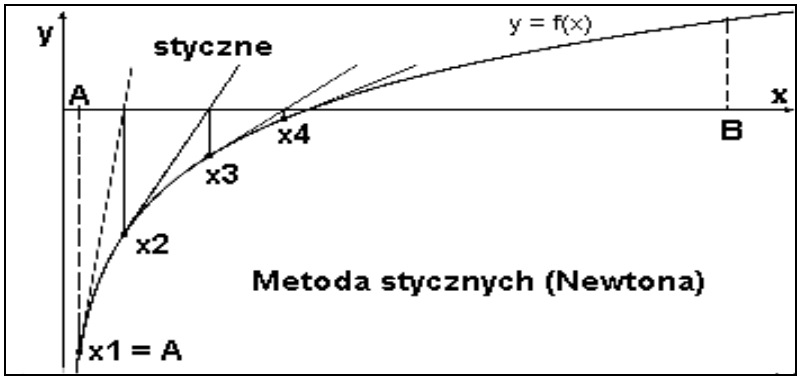
1. wczytaj wartości **A** i **B** oraz **e**;
2. oblicz **Xs=(B-A)/2**;
3. jeśli **|f(Xs)|<e** to jako **xp** przyjmij wartość **Xs** i zakończ obliczenia a jeśli nie to:
4. jeśli **f(A)·f(Xs)<0** to jako nową wartość **B** przyjmij **Xs** a w przeciwnym przypadku jako nową wartość **A** przyjmij **Xs** i skocz do punktu 2.

5.11.2. METODA STYCZNYCH

Metoda stycznych (zwana również metodą Newtona-Raphsona lub metodą Newtona) to iteracyjny algorytm wyznaczania przybliżonej wartości pierwiastka funkcji.

Oprócz warunków (1) i (2) pierwsza pochodna f′(x) i druga pochodna f″(x) funkcji f(x) musi mieć w przedziale (A; B) stały znak.

Za wartość początkową **x1** szukanego pierwiastka **xp** przyjmuje się ten kraniec przedziału, dla którego znak funkcji i znak drugiej pochodnej są jednakowe (Rys. 5.22). Następnie wyznaczana jest styczna do krzywej w tym punkcie. Odcięta **x2** punktu przecięcia stycznej z osią **x** jest przyjęta jest jako kolejne przybliżenie rozwiązania. Z punktu **x2; f(x2)** wystawiana jest kolejna styczna, znajdowany kolejny punkt **x3** jej przecięcia z osią **x** i cykl ten może być powtarzany aż do spełnienia określonego „warunku stopu".



Rys. 5.22. Metoda stycznych (PR 57)

Równanie prostej o współczynniku kierunkowym **m** przez punkt o współrzędnych **x1, f(x1)** to: **y - f(x1) = m·(x-x1)**. Ponieważ współczynnik kierunkowy jest równy wartości pochodnej (tangensowi kąta nachylenia stycznej) więc: **y – f(x1) = f′ (x1) · (x-x1)**. Z tego, dla **y=0**, wyznaczamy odciętą **x** punktu gdzie styczna przecina oś odciętych, oznaczoną x2:

x2 = x1 – f(x1) / f′ (x1)

Tak więc algorytm można zapisać w następujących punktach:

1. jeśli **znak(f(A)) = znak(f″(A)** to **xp:=A**, w przeciwnym przypadku **xp:=B**,
2. wyznacz nową wartość xp ze wzoru: **xp := xp – f(xp) / f′ (xp)**,
3. jeśli **|f(xp)|>e** to wróć do punktu 2 a jeśli nie to zakończ,

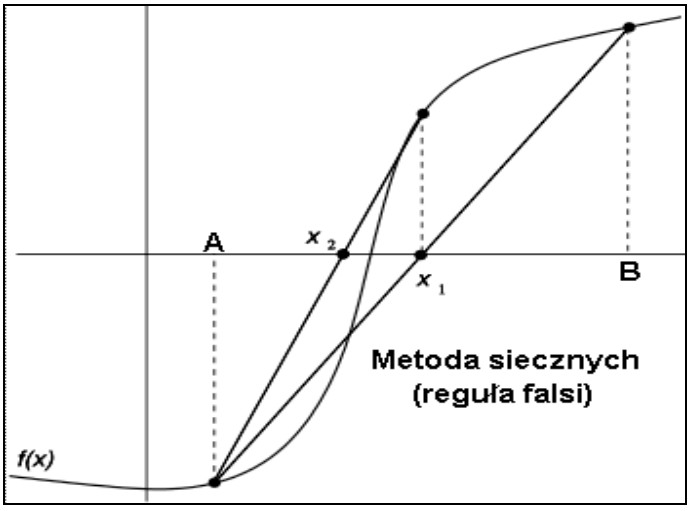
przy czym wartość parametru **e** określa dokładność obliczeń, natomiast znak „**:=**" jest znakiem podstawiania.

5.11.3. METODA SIECZNYCH

Metoda siecznych, zwana też „metodą interpolacji liniowej", „metodą Eulera" lub „regułą falsi" polega na zastępowaniu przebiegu funkcji nieliniowej w przedziale [A; B] linią prostą „sieczną" (Rys. 5.23), to znaczy przechodzącą przez punkty **(A, f(A))** oraz **(B, f(B))**.

Za przybliżoną wartość pierwiastka **xp** przyjmowana jest odcięta punktu przecięcia siecznej z osią **x**, a następnie koniec przedziału A lub B zostaje przemieszczony do **xp,** tak aby w nowym węższym przedziale funkcja nadal zmieniała znak. Procedura jest więc podobna jak w metodzie bisekcji tylko inaczej wyznaczany jest punkt wewnątrz przedziału, a mianowicie:

1. wczytaj wartości **A** i **B** oraz **e**;
2. oblicz **Xs = A – f(A)*(B-A)/(f(B)-f(A))**;
3. jeśli **|f(Xs)|<e** to jako **xp** przyjmij wartość **Xs** i zakończ obliczenia a jeśli nie to:
4. jeśli **f(A)·f(Xs)<0** to jako nową wartość **B** przyjmij **Xs** a w przeciwnym przypadku jako nową wartość **A** przyjmij **Xs** i skocz do punktu 2.

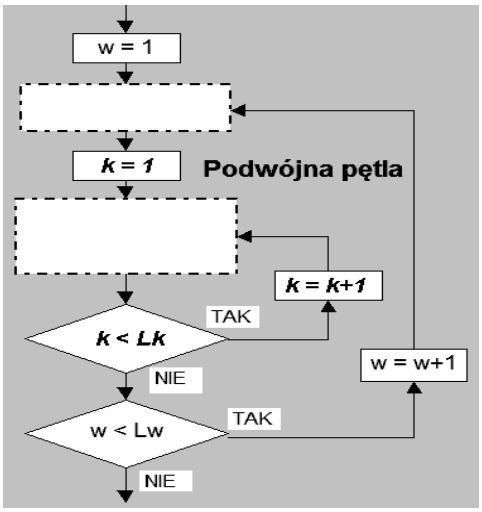


Rys. 5.23. Metoda siecznych (PR 58)

5.12. OPERACJE NA MACIERZACH

Macierz (p.3.1) to **prostokątna tablica** o ponumerowanych kolumnach i wierszach.

Każdy element macierzy pełni rolę zmiennej a identyfikowany jest przez nazwę macierzy oraz indeksy (numery) wiersza i kolumny, na przykład A(2,5) to element położony w drugim wierszu i piątej kolumnie. W języku BASIC, Pascal, Fortran, C i wielu innych – wszystkie elementy danej tablicy (wektora, macierzy) muszą być jednakowego typu. Na ogół dopuszczalne są również tablice o większej liczbie wymiarów - co objawia się większą niż dwa liczbą indeksów – na przykład ELB(1,3,4) to element tablicy trójwymiarowej.




```
'Podwójna pętla w BASIC-u:
. . . . . .
. . . . . .
FOR w=1 TO Lw
 . . . . . .
    FOR k=1 TO Lk
      . . . . . .
      . . . . . .
    NEXT k
. . . . . .
NEXT w
```

Rys. 5.24. Schemat podwójnej pętli dla operowania na elementach macierzy

Operowanie na macierzach i tablicach o większej liczbie wymiarów wymaga zdefiniowania osobnej zmiennej dla każdego indeksu i zazwyczaj osobnej pętli w której ten indeks będzie zmieniany. W przypadku działań na poszczególnych elementach macierzy, najczęściej wystąpi podwójna pętla o ogólnej budowie pokazanej na Rys. 5.24.

5.12.1. GENEROWANIE MACIERZY

Generowanie macierzy jest zagadnieniem podobnym do generowania ciągów liczbowych

Wygenerować macierz:

```
1   5   9  13
3   7  11  15
5   9  13  17
7  11  15  19
```

Analiza problemu - widać, że w każdym wierszu macierzy:
– pierwszy element jest kolejną liczbą nieparzystą
– każdy następny element jest większy o 4 od poprzedniego
Indeksy wierszy i kolumn muszą - jak zawsze - przyjmować kolejne wartości w zakresie liczb naturalnych (w naszym przypadku: 1, 2, 3, 4).

Przy generowaniu macierzy w których elementy wierszy i kolumn stanowią ciągi liczbowe o określonej regularności, możliwe są dwa podejścia:

1. generować w pętlach indeksy wierszy i kolumn (jako zmienne kontrolne) a następnie wykorzystywać je do wyznaczania wartości elementów macierzy,
2. generować w pętlach bezpośrednio wartości elementów jako zmienne kontrolne pętli a dodatkowo generować indeksy wierszy i kolumn

Zadanie można oczywiście rozwiązać na wiele sposobów, z których pokazano dwa:

W programie PR59 pętla zewnętrzna generuje numery (indeksy) wierszy ***w***=1,2,3,4, a pętla wewnętrzna dla każdego numeru wiersza generuje numery kolumn ***k***. Wartości elementów macierzy ***a(w,k)*** są tworzone z wyrażeń wykorzystujących indeksy ***w*** i ***k***:

```
' PROGRAM PR 59
CLS
FOR w = 1 TO 4
    el = 2 * w - 1
    FOR k = 1 TO 4
        a(w, k) = el
        PRINT USING "####"; a(w, k);
        el = el + 4
    NEXT k
    PRINT
NEXT w
```

' PROGRAM PR 60 pokazuje inną koncepcję generowania tej samej macierzy. Tutaj w pętli zewnętrznej generowane są wartości zmiennej ***i*** reprezentującej pierwszy element każdego wiersza, natomiast w pętli wewnętrznej - jako ***j*** - generowane są kolejne elementy wiersza zapoczątkowanego przez wartość ***i***. Osobno - przy pomocy poleceń ***w=0, w=w+1*** oraz ***k=0, k=k+1*** umieszczonych w odpowiednich miejscach, generowane są indeksy elementów macierzy, które - jak zawsze - muszą przyjmować wartości kolejnych liczb naturalnych.

```
' PROGRAM PR 60
CLS: w = 0
FOR i=1 TO 7 STEP 2
    w = w+1: k = 0
    FOR j=i TO i+3*4 STEP 4
        k=k+1
        a(w,k)=j
        PRINT USING "####"; a(w, k);
    NEXT j
    PRINT
NEXT i
```

5.12.2. MNOŻENIE MACIERZY

Mnożenie macierzy jest istotną operacją, przydatną chociażby przy rozwiązywaniu układów równań liniowych czy transformacjach geometrycznych.

Dokładniej zaznaczmy o jakie działanie nam chodzi, gdyż w przypadku dwu macierzy możliwe są przynajmniej dwa różne sposoby mnożenia, dlatego na przykład MATLAB posiada w tym celu dwa różne operatory (*) oraz (.*).
Gdy dwie macierze ***A, B*** mają identyczne rozmiary, to możliwe jest **mnożenie tablicowe** tych macierzy:

C=A.*B

polegające na wyznaczeniu macierzy **C** o tych samych rozmiarach, w której każdy element powstaje jako iloczyn pary odpowiednich elementów:

C(w,k)=A(w,k)*B(w,k)

Najczęściej jednak mówiąc o mnożeniu macierzy i nie dodając do niego żadnych przymiotników, mamy na myśli operację **mnożenia macierzowego** macierzy, opisanego poniżej.

Załóżmy, że mamy wyznaczyć iloczyn macierzowy macierzy ***A, B:***

C = A*B

przy czym:

- macierz ***A*** ma ***Lwa*** wierszy i ***Lka*** kolumn,
- macierz ***B*** ma ***Lwb*** wierszy i ***Lkb*** kolumn,

Elementy wiersza macierzy ***A*** są mnożone przez elementy kolumny macierzy ***B*** a suma otrzymanych iloczynów tworzy element wynikowy, tak jak to pokazuje Rys. 5.25.

Wynika stąd warunek wykonalności mnożenia macierzy: **liczba kolumn macierzy pierwszej musi być równa liczbie wierszy macierzy drugiej** czyli ***Lka=Lwb.***

Elementy macierzy C będą powstawać niejako na przecięciach wierszy ***A*** z kolumnami ***B*** skąd wymiary macierzy wynikowej ***C(Lwa,Lkb)***

Konkretnie dla macierzy ***4x3 i 3x2*** (Rys. 5.25) mnożenie jest możliwe bo ***Lka=Lwb=3*** a macierz wynikowa ***C*** będzie miała rozmiary ***4x2.***

$$A=\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \\ a41 & a42 & a43 \end{bmatrix} \qquad B=\begin{bmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b32 \end{bmatrix}$$

$$C=A\cdot B=\begin{bmatrix} a11\cdot b11+a12\cdot b21+a13\cdot b31 & a11\cdot b12+a12\cdot b22+a13\cdot b32 \\ a21\cdot b11+a22\cdot b21+a23\cdot b31 & a21\cdot b12+a22\cdot b22+a23\cdot b32 \\ a31\cdot b11+a32\cdot b21+a33\cdot b31 & a31\cdot b12+a32\cdot b22+a33\cdot b32 \\ a41\cdot b11+a42\cdot b21+a43\cdot b31 & a41\cdot b12+a42\cdot b22+a43\cdot b32 \end{bmatrix}=\begin{bmatrix} c11 & c12 \\ c21 & c22 \\ c31 & c32 \\ c41 & c42 \end{bmatrix}$$

Rys. 5.25. Mnożenie macierzowe macierzy

Popatrzmy (Rys. 5.25) jak obliczono dowolny element macierzy ***C*** na przykład ***c32*** a dokładniej ***C(3,2)*** bo w językach programowania zapiszemy wskaźniki w nawiasach i oddzielone przecinkiem:

$$C(3,2) = \sum_{i=1}^{Lka} A(3,i) \cdot B(i,2)$$

Uogólniając, otrzymujemy wzór na obliczanie dowolnego elementu macierzy wynikowej:

$$C(w,k) = \sum_{i=1}^{Lka} A(w,i) \cdot B(i,k)$$

Ze wzoru tego widać, że wystarczą trzy zmienne do oznaczenia indeksów macierzy a nie 6 (po 2 niezależne dla A, B, C), bo macierze A, B, C są ściśle powiązane zależnościami. Jednak taki syntetyczny wzór, byłby przydatny w Mathcadzie, a my mamy rozpisać mnożenie wykorzystując pętle. Spróbujmy najpierw z grubsza określić kolejność operacji:

1. Najpierw trzeba wczytać rozmiary macierzy ***A*** i ***B*** czyli ***Lwa, Lka, Lwb, Lkb***, oraz sprawdzić czy mnożenie jest wykonalne to znaczy czy ***Lka=Lwb***, jeśli tak to warto – zgodnie ze wzorem na ***C(w,k)*** - wprowadzić zmienną „***i***" zamiast zmiennych ***ka*** i ***wb***, oraz ***Li*** zamiast ***Lka=Lwb***
2. Następnie można wczytać macierze A i B.
3. Podwójna pętla musi wyznaczyć wszystkie elementy macierzy wynikowej ***C(w,k)*** czyli macierz o rozmiarach ***Lwa*** x ***Lkb***
4. Wewnątrz tej podwójnej pętli będzie wyznaczanie sum iloczynów, czyli jeszcze jedna pętla ze zmienną kontrolną „***i***" ; nie należy zapomnieć, że przed tą pętlą każdą z wyznaczanych sum ***C(w,k)*** trzeba wyzerować.

A oto program mnożenia macierzy (' PR 61), napisany w języku BASIC i przetestowany w QB64 działającym w środowisku Ms Windows 7:

```
' PR 61
CLS: PRINT "Mnozenie macierzy"
PRINT "Macierz A:"
INPUT "Liczba wierszy Lwa="; Lwa
INPUT "Liczba kolumn  Lka="; Lka
PRINT "---------------"
PRINT "Macierz B:"
INPUT "Liczba wierszy Lwb="; Lwb
INPUT "Liczba kolumn  Lkb="; Lkb
IF Lka <> Lwb THEN
    PRINT "!! Mnozenie NIEWYKONALNE: Lka <> Lwb"
    INPUT x
    STOP
END IF
PRINT "--- Wczytywanie macierzy A:"
FOR w = 1 TO Lwa
    FOR k = 1 TO Lka
```

```
        PRINT "  A("; w; ","; k; ")";: INPUT ; "=", A(w, k)
    NEXT k
    PRINT
NEXT w
PRINT "---- Wczytywanie macierzy B:"
FOR w = 1 TO Lwb
    FOR k = 1 TO Lkb
        PRINT "  B("; w; ","; k; ")";: INPUT ; "=", B(w, k)
    NEXT k
    PRINT
NEXT w
PRINT "==== Macierz wynikowa C: ===="
Li = Lka
FOR w = 1 TO Lwa
    FOR k = 1 TO Lkb
        C(w, k) = 0
        FOR i = 1 TO Li
            C(w, k) = C(w, k) + A(w, i) * B(i, k)
        NEXT i
        PRINT "  "; C(w, k);
    NEXT k
    PRINT
NEXT w
```

Program ten wyświetla dane i wyniki w następującej postaci:

```
Mnozenie macierzy
----- Macierz A:
Liczba wierszy Lwa=? 2
Liczba kolumn  Lka=? 3
----- Macierz B:
Liczba wierszy Lwb=? 3
Liczba kolumn  Lkb=? 1
--- Wczytywanie macierzy A:
  A( 1 , 1 )=3  A( 1 , 2 )=2  A( 1 , 3 )=0.5
  A( 2 , 1 )=1  A( 2 , 2 )=3  A( 2 , 3 )=2
--- Wczytywanie macierzy B:
  B( 1 , 1 )=2
  B( 2 , 1 )=4
  B( 3 , 1 )=6
==== Macierz wynikowa C: ====
   17
   26
```

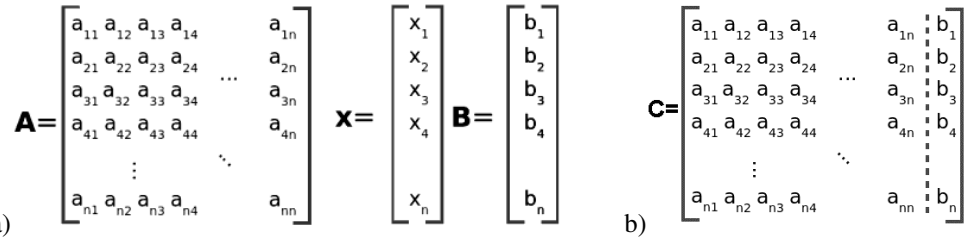
5.12.3. ROZWIĄZYWANIE UKŁADÓW RÓWNAŃ LINIOWYCH

Układ n równań liniowych z n niewiadomymi $x_1, x_2, \ldots x_n$, zapisany macierzowo, jest najczęściej rozwiązywany **metodą eliminacji Gaussa** [43].

Macierzowy zapis takiego układu to **A · x = B**, gdzie: **A** - to macierz współczynników przy niewiadomych (Rys. 5.26), **x** – to wektor niewiadomych, a **B** – to wektor wyrazów wolnych występujących po prawej stronie. Jak wiadomo: obie strony każdego równania można pomnożyć lub podzielić przez dowolną liczbę różną od zera, można także dodać stronami równanie do innego równania.

Metoda eliminacji Gaussa (PR 62) polega na wykonywaniu tego rodzaju operacji na macierzy **C** dla otrzymania trójkątnej macierzy współczynników takiej, która ma jedynki na przekątnej głównej oraz zera poniżej tej przekątnej.

Dla zastosowania metody Gaussa konieczne jest aby elementy przekątnej głównej były różne od zera. Jeśli tak nie jest to można zmienić kolejność wierszy w macierzy **C**.



Rys. 5.26. Metoda eliminacji Gaussa
a) macierze liniowego układu równań, b) macierz rozszerzona współczynników

W pierwszym kroku elementy pierwszego wiersza macierzy **C** zostają podzielone przez współczynnik a_{11} dzięki czemu uzyskuje się pierwszą jedynkę na przekątnej.

Następnie, aby pod tą jedynką uzyskać zera, dla $i = 2, 3,...n$, od każdego i-tego wiersza odejmowany jest pierwszy wiersz pomnożony przez a_{i1}.

Analogiczne operacje powtarzane są teraz dla drugiego i następnych wierszy: drugi wiersz zastaje najpierw podzielony przez a_{22}, a następnie dla $i=3, 4,...n$, od każdego i-tego wiersza odejmowany jest drugi wiersz pomnożony przez a_{i2}.

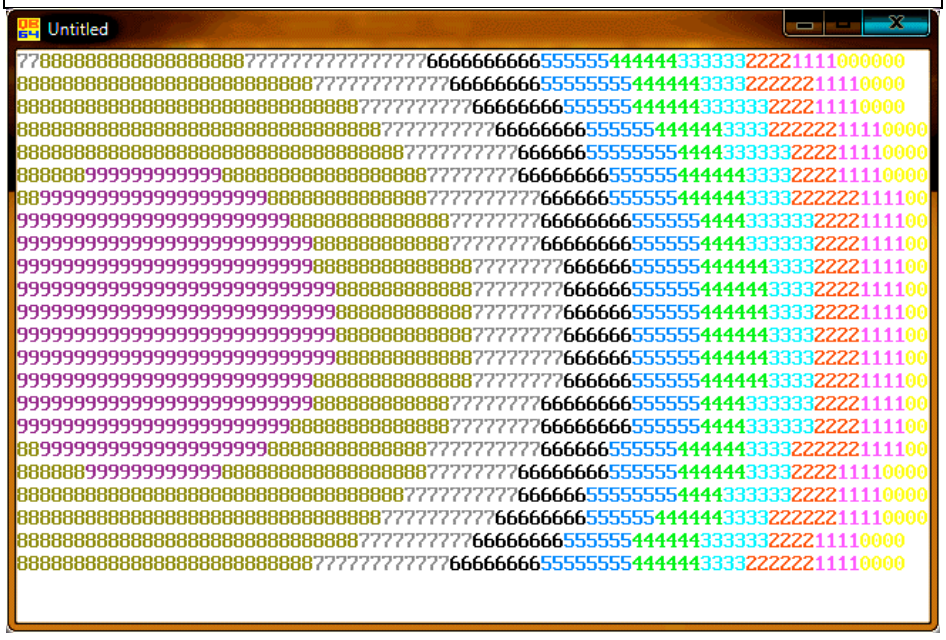
Kontynuując takie operacje uzyskuje się macierz trójkątną, której ostatni wiersz podaje wartość niewiadomej x_n. Wstawiając tą wartość do przedostatniego równania uzyskujemy wartość niewiadomej x_{n-1} i tak od końca aż do niewiadomej x_1 uzyskujemy rozwiązanie układu.

5.12.4. WARSTWICE FUNKCJI DWU ZMIENNYCH

Jako kolejny przykład zamieszczono poniżej program w języku BASIC, który ma wyświetlić przybliżoną mapę warstwicową dla funkcji a(x,y) zdefiniowanej jako podprogram.

```
' PR 63 – Program WARSTWICE1
DECLARE FUNCTION a! (x!, y!)
lw = 23: lk = 40: ' – rozmiary mapy w znakach
lpoz = 10: ' – liczba poziomow
DIM tz(1 TO lw, 1 TO lk) AS SINGLE
'-- Tabelaryzacja i wyznaczanie ekstremow:
' zakresy x i y:
xp = –2: xk = 13: yp = –5: yk = 5
dx = (xk – xp) / lk: dy = (yk – yp) / lw
```

```
CLS: zmin = 1E+36: zmax = -1E+36
y = yp
FOR w = 1 TO lw
    x = xp
    FOR k = 1 TO lk
        z = a(x, y): tz(w, k) = a(x, y)
        IF z < zmin THEN zmin = z
        IF z > zmax THEN zmax = z
        x = x + dx
    NEXT k
    y = y + dy
NEXT w
'--- warstwice:
dz = (zmax - zmin) / lpoz
FOR w = 1 TO lw
    FOR k = 1 TO lk
        kolor = ((tz(w, k) - zmin) / dz)
        kod = 47 + kolor
        COLOR kolor: PRINT CHR$(kod); CHR$(kod);
    NEXT k
    PRINT
NEXT w
'----
FUNCTION a (x, y)
a = -x * x - y * y
END FUNCTION
```



Rys. 5.27. Warstwice funkcji z(x,y) = -(x*x+y*y) wygenerowane programem WARSTWICE1

Program ten przypisuje odpowiednią cyfrę oraz kolor każdemu obszarowi między dwoma warstwicami. Wynikowy ekran (a dokładniej jego negatyw) pokazuje Rys. 5.27.

5.13. OPTYMALIZACJA

Optymalizacja to procedura **znajdowania najlepszego czyli optymalnego rozwiązania** spośród rozwiązań dopuszczalnych. Metody optymalizacji - nazywane też (m.in. w ekonomii) programowaniem matematycznym lub badaniami operacyjnymi - to bardzo rozbudowana dziedzina matematyki znajdująca zastosowania w bardzo wielu działach techniki i gospodarki. Stanowią też istotny obszar działalności inżyniera mechanika, szczególnie przy projektowaniu.

Według M. Komosińskiego [49], przystępując do optymalizacji należy określić odpowiedzi na następujące cztery pytania:

1. Jakie są moje **cele** czyli co chcę osiągnąć?
   - na przykład maksymalizację zysku lub minimalizację strat - i jakie zmienne (wyjściowe, docelowe) reprezentują te cele.
2. Jakie **decyzje** mogę podjąć?
   – a w sensie matematycznym – **które zmienne są wejściowymi, decyzyjnymi**, które będzie można zmieniać aby uzyskiwać rózne efekty?
3. Jakie **ograniczenia** muszę uwzględnić?
   – te ograniczenia określają **dopuszczalne zakresy zmiennych decyzyjnych**.
4. Jak ocenić wpływ decyzji na cele?
   – tu trzeba określić „**funkcję celu**” czyli funkcyjną zależność zmiennych docelowych (wyjściowych) od zmiennych decyzyjnych.

Tak więc zawsze wstępem dla prowadzenia optymalizacji jest **określenie celów** i **zmiennych decyzyjnych** oraz **ograniczeń** dla tych zmiennych określających tak zwaną **przestrzeń decyzyjną**, a następnie sformułowanie **kryterium optymalizacji** w postaci **funkcji celu**.

Optymalizacja polega na **znajdowaniu ekstremum funkcji celu** na przykład minimum kosztów czy maksimum wydajności, przy uwzględnieniu ograniczeń. Czasem może polegać na osiągnięciu określonej wartości funkcji celu. **Rozwiązaniem optymalnym** jest taki zestaw wartości zmiennych decyzyjnych – z przestrzeni decyzyjnej - dla którego uzyskano ekstremum (lub określoną wartość) funkcji celu.

Stosuje się **optymalizacje jedno i wielokryterialne**. Optymalizacja wielokryterialna nazywana jest też **poli-optymalizacją**. Przykładami optymalizacji wielokryterialnej mogą być:

- maksymalizacja zysków i minimalizacji kosztów produktu,
- maksymalizacja wydajności przy ograniczaniu zużycia paliwa pojazdu,
- obniżenie masy urządzenia a maksymalizacja wytrzymałości jego komponentów.

Przykładem narzędzia służącego do optymalizacji jest **Solver** (Rys. 5.28) stanowiący dodatek do arkusza kalkulacyjnego **Ms Excel**.

Solver w Excelu może nie być standardowo zainstalowany i wówczas należy go w Ms Office doinstalować.

Rys. 5.28. Solver w Ms Excel – narzędzie do optymalizacji

W analogiczny Solver bywają wyposażone parametryczne programy do modelowania geometrycznego, na przykład Pro/Desktop [50]

Pakiety oprogramowania matematycznego także są zazwyczaj wyposażone w obszerne biblioteki procedur optymalizacyjnych. W MATLAB-ie jest to „Optimization Toolbox" a w Mathcadzie osobna biblioteka „Solving and Optimization".

Wiadomości teoretyczne oraz przykłady dotyczące optymalizacji konstrukcji mechanicznych można znaleźć w podręczniku [48].

5.13.1. PROSTY PRZYKŁAD PROGRAMOWANIA LINIOWEGO

Do najprostszych metod optymalizacji zaliczane jest **programowanie liniowe**, w którym zarówno funkcja celu jak przestrzeń decyzyjna są liniowe.

Przy dwu zmiennych decyzyjnych - przestrzeń decyzyjna jest wielobokiem, a funkcja celu jest płaszczyzną o określonym nachyleniu.

Przykład: Załóżmy, że chcemy się odżywiać tylko chlebem i serem o następujących parametrach:

Produkt	***kcal/100g***	***Cena/100g***	***Cena/100kcal***	***Min.spożycie***	***Maks.spoż.***
x = Chleb	250	0,3 zł	0,12 zł	x1=500 kcal	x2=1000 kcal
y = Ser	150	1,2 zł	0,80 zł	y1=200 kcal	y2=1300 kcal

1. **Cel optymalizacji**: Ile chleba i sera dziennie mamy kupować aby wydać jak najmniej pieniędzy i albo: (a) nie przekroczyć normy dziennej 2000 kilokalorii (kcal), albo (b) utrzymać normę dzienną 2000 kcal, mieszcząc się zarazem w określonych granicach dawek żywieniowych?

2. **Zmiennymi decyzyjnymi** są:
   x = dzienna porcja chleba – przeliczona na kilokalorie,
   y = dzienna porcja sera – przeliczona na kilokalorie.

3. **Ograniczenia** dotyczące zmiennych decyzyjnych – uwzględnimy dwa warianty.

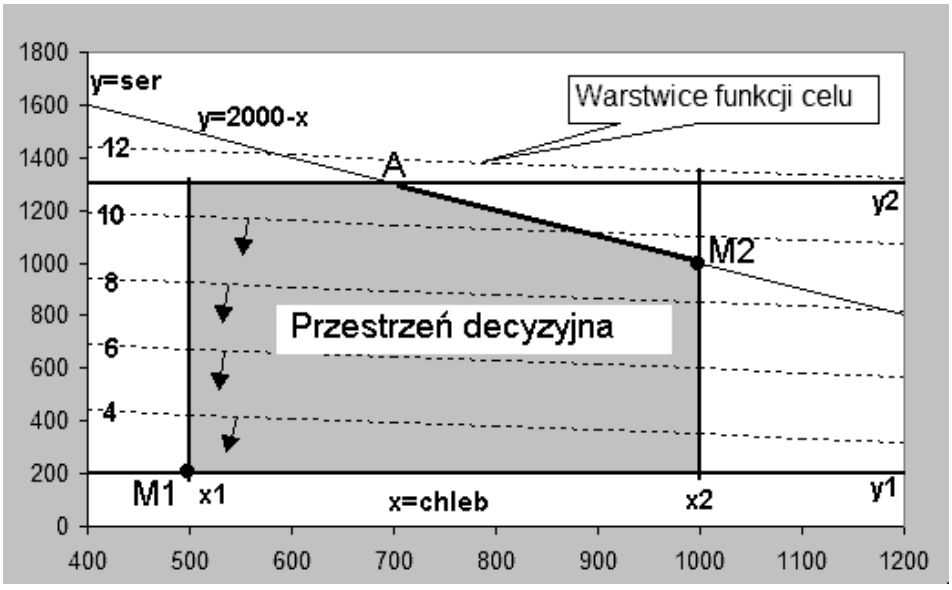
   a) wariant dla odchudzających się – nie przekroczyć 2000 kcal na dzień
   $x1 < x < x2;\quad y1 < y < y2;\quad x+y < 2000;$
   czyli: $500 < x < 1000;\quad 15 < y < 1300;\quad y < 2000-x$

   b) wariant stabilny – mamy spożywać 2000 kcal na dzień:
   $x1 < x < x2;\quad y1 < y < y2;\quad x+y = 2000;$
   czyli: $500 < x < 1000;\quad 15 < y < 1300;\quad y = 2000-x$

4. **Funkcja celu**: to koszt zakupionych dziennych porcji chleba i sera czyli:
   $z(x,y) = (0{,}12*x + 0{,}8*y)/100$

Rys. 5.29 pokazuje graficzne rozwiązanie powyższego przykładu programowania liniowego. Liniami ciągłymi zaznaczono wykresy warunków ograniczających przestrzeń decyzyjną, a liniami przerywanymi warstwice funkcji celu. Strzałki przy warstwicach pokazują kierunek opadania płaszczyzny funkcji celu.

Dla wariantu (a) z dziennym wyżywieniem x+y < 2000 kcal, przestrzenią decyzyjną jest zacieniowany na Rys. 5.29 **wielobok**. Najmniejsza wartość fukcji celu – czyli najmniejszy koszt dziennego wyżywienia wypada w punkcie M1 co odpowiada porcji chleba o wartości 500 kcal plus plaster sera o wartości 200 kcal.



Rys. 5.29. Graficzne rozwiązanie przykładu programowania liniowego

Dla wariantu (b), w którym mamy nie być głodni bo utrzymamy normę x+y = 2000 kcal, przestrzenią decyzyjną jest **odcinek** prostej AM2. Na tym odcinku najmniejsza wartość funkcji celu odpowiada punktowi M2 czyli: chleb 1000 kcal + ser 1000 kcal.

Jak widać, rozwiązaniem optymalnym dla przypadku programowania liniowego jest ten wierzchołek przestrzeni decyzyjnej, w którym wystąpi minimum funkcji celu (kosztu).

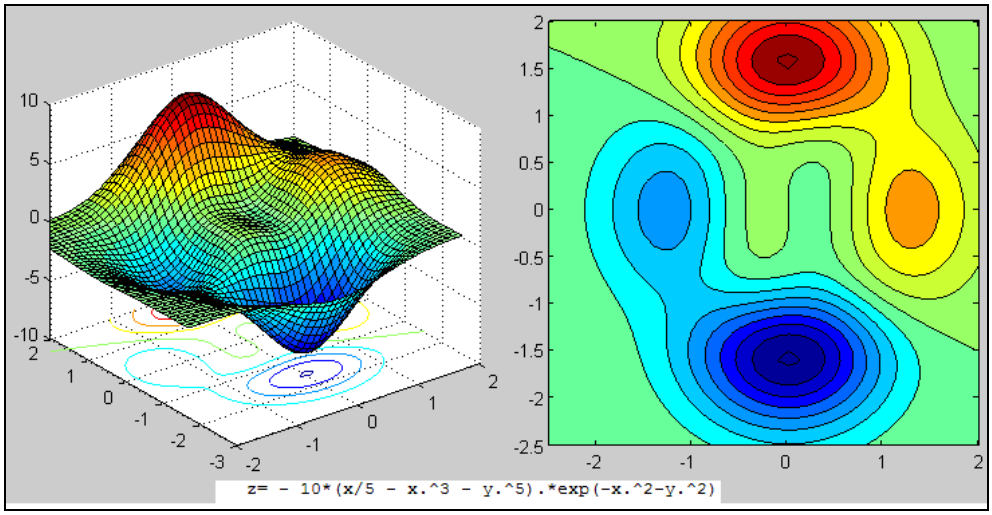
Uogólniając, algorytm powinien wyznaczyć wierzchołki wieloboku tworzącego przestrzeń decyzyjną – czyli dla każdej pary warunków wyznaczyć punkt wspólny linii granicznych, a następnie sprawdzić w którym z nich wystąpi minimum funkcji celu.

5.13.2. EKSTREMUM NIELINIOWEJ FUNKCJI CELU

W bardziej złożonych przypadkach zagadnień optymalizacyjnych, zarówno funkcja celu może być nieliniowa, jak i przestrzeń decyzyjna może mieć różne postacie: wieloboków wklęsłych, obszarów ograniczonych krzywymi, a także oddzielonych od siebie obszarów (wielospójnych).

Oczywiście – w przypadkach gdy funkcja celu jest nieliniowa – to minimum nie musi wypadać na brzegu przestrzeni decyzyjnej. Wówczas możemy mieć do czynienia z klasycznym wyznaczaniem minimum dla funkcji ciągłej wielu zmiennych. Czasem zadanie może komplikować występowanie wielu minimów lokalnych jak na Rys. 5.30

Jak widać – problemów może być wiele, dlatego dziedzina optymalizacji jest bardzo szeroka i obejmująca wiele dość skomplikowanych metod.



Rys. 5.30. Nieliniowa funkcja celu z minimami lokalnymi

Do najprostszych metod znajdowania minimum funkcji nieliniowej, należy metoda gradientu prostego [4]. Uproszczoną adaptację tej metody przedstawiono poniżej w postaci programu w języku BASIC.

Danymi dla tej metody – oprócz wzoru funkcji – są:

- współrzędne punktu początkowego: **x1, y1**;
- długość kroku: **d;**
- minimalna różnica między kolejnymi przybliżeniami, czyli „warunek stopu": **dzmin**.

Oto treść (t.zw.: listing) programu PR64:

```
' PR 64
PRINT "Minimum funkcji z=f(x,y)"
PRINT "z= - 10*(x/5 - x^3 - y^5) * exp(-x^2-y^2)"
INPUT ; " Xp="; x1#: INPUT ; "  Yp="; y1#
INPUT; " przyrost d="; d#
dzmin# = 0.01
PRINT: PRINT " Xp="; x1#; "   Yp="; y1#; "   d="; d#
PRINT
DATA 1,0,1,1,0,1,-1,1,-1,0,-1,-1,0,-1,1,-1
FOR i = 1 TO 8
    READ zx%(i), zy%(i)
NEXT i
dz# = 1E100
PRINT "      x           y           z"
WHILE dz# > dzmin#
    z1# = Fz(x1#, y1#)
    Zmin# = z1#
    PRINT USING "   ##.######"; x1#; y1#; z1#
    FOR i = 1 TO 8
        z2# = Fz(x1# + zx%(i) * d#, y1# + zy%(i) * d#)
        IF z2# < Zmin# THEN
            Zmin# = z2#
            x2# = x1# + zx%(i) * d#
            y2# = y1# + zy%(i) * d#
        END IF
    NEXT i
    x1# = x2#: y1# = y2#: dz# = z1# - Zmin#
WEND
'=======
FUNCTION Fz (x, y)
Fz = -10 * (x / 5 - x ^ 3 - y ^ 5) * EXP(-x ^ 2 - y ^ 2)
END FUNCTION
```

Koncepcja programu PR64 jest następująca:

- dla punktu **x1, y1** wyznaczyć wartość funkcji celu **z1 = Fz(x1,y1)**
- następnie sprawdzić w którym z 8-miu kierunków w otoczeniu **d** względem **(x1, y1)** jest najmniejsza wartość **z2=Fz(x1±d, y1±d)** i ustanowić tam nowe położenie **(x1, y1),**
- powtarzać tą czynność tak długo, aż różnica **dz=z2-z1** stanie się mniejsza niż **dzmin**.

Wyniki działania tego programu - dla kilku zestawów danych – pokazano na Rys. 5.31.
Dla funkcji posiadającej minima lokalne, program nie zawsze znajdzie minimum globalne.

```
Minimum funkcji z=f(x,y)
z= - 10*(x/5 - x^3 - y^5) * exp(-x^2-y^2)

 Xp=-1    Yp=-2    d= .1

      x          y          z
   -1.000000  -2.000000  -2.210047
   -0.900000  -1.900000  -3.045863
   -0.800000  -1.800000  -3.974805
   -0.700000  -1.700000  -4.903368
   -0.600000  -1.600000  -5.707133
   -0.500000  -1.600000  -6.328002
   -0.400000  -1.600000  -6.896929
   -0.300000  -1.600000  -7.385002
   -0.200000  -1.600000  -7.764381
   -0.100000  -1.600000  -8.010792
   -0.000000  -1.600000  -8.105989
```

```
Minimum funkcji z=f(x,y)
z= - 10*(x/5 - x^3 - y^5) * exp(-x^2-y^2)

 Xp= 0    Yp= 0    d= .5

      x          y          z
    0.000000   0.000000  -0.000000
   -0.500000  -0.500000  -0.341173
    0.000000  -1.000000  -3.678794
    0.000000  -1.500000  -8.003754
```

```
Minimum funkcji z=f(x,y)
z= - 10*(x/5 - x^3 - y^5) * exp(-x^2-y^2)

 Xp=-1    Yp=-2    d= 1

      x          y          z
   -1.000000  -2.000000  -2.210047
    0.000000  -2.000000  -5.861004
```

```
Minimum funkcji z=f(x,y)
z= - 10*(x/5 - x^3 - y^5) * exp(-x^2-y^2)

 Xp= 0    Yp= 0    d= .1

      x          y          z
    0.000000   0.000000  -0.000000
    0.100000   0.000000  -0.188109
    0.200000   0.000000  -0.307453
```

Rys. 5.31. Wyniki działania programu PR64 dla różnych danych

Ulepszenie programu może polegać na przykład na rozpoczynaniu od sprawdzenia kilku punktów początkowych, a także na sprawdzeniu czy dla większego lub mniejszego kroku *d*, (na przykład dwukrotnie) uzyskuje się lepsze wyniki.

Spróbuj ulepszyć program.

6. ZADANIA DO ZAPROGRAMOWANIA

6.1. PROPONOWANE WARIANTY ROZWIĄZAŃ

Zamieszczone w tym rozdziale zadania mogą być wykorzystane do tworzenia różnych wariantów algorytmów i programów, w różnych językach programowania. A oto niektóre z tych wariantów:

a) **Obliczenia jednorazowe na zmiennych skalarnych**. Wariant podstawowy składa się tylko z poleceń wprowadzania danych, obliczania wartości zmiennych oraz wyprowadzania wyników. Programista musi jednak zapobiec próbom wykonywania działań niewykonalnych jak dzielenie przez zero czy pierwiastkowanie liczb ujemnych, a także dostarczyć użytkownikowi (wyświetlić na ekranie) niezbędne informacje o danych i wynikach. Bardziej zaawansowane warianty tego rodzaju programów powinny na żądanie dostarczać obszerniejszych informacji o zmiennych i sposobach ich przetwarzania oraz nie dopuszczać do wprowadzania danych niepoprawnych co do typu lub zakresu.

b) **Obliczenia wielokrotne** – powtarzane dowolną liczbę razy dla różnych danych, aż do wprowadzenia polecenia kończącego obliczenia na przykład wprowadzenia zera jako jednej z danych.

c) **Obliczenia cykliczne (iteracyjne)** – realizowane **bez użycia tablic** (wektorów, macierzy) a mające na celu wyświetlenie, wydrukowanie lub wyprowadzenie do pliku kilkukolumnowej tabeli, zawierającej kolumnę (ciąg) wartości zmiennej niezależnej, oraz kolumny zmiennych będących jej funkcjami. Użycie tablic jest zbędne jeśli nie jest potrzebne zapamiętywanie uzyskanych ciągów wartości celem późniejszego użycia w tym samym programie, a każda n-tka wartości jest wyprowadzana na urządzenie wyjściowe lub do pamięci masowej w tym samym cyklu w którym została wyznaczona.

   Danymi dla tego wariantu powinny być granice zakresu zmiennej niezależnej (np.: Xpocz, Xkonc) oraz jej przyrost lub liczba iteracji N (domyślnie np,: N=100), a także parametry funkcji.

   Jeśli realizowany będzie program obiektowy (np.: w MATLAB-ie lub Visual BASIC-u) to parametry funkcji mogą być dobierane przy pomocy suwaków lub list rozwijalnych.

d) **Obliczenia cykliczne wektorowe lub macierzowe** – różniące się względem poprzedniego wariantu tylko zastosowaniem tablic do zapamiętania każdej otrzymanej n-tki wartości.

e) **Generowanie wykresów** - albo bez użycia tablic (odcinkami) jako rozwinięcie wariantu (c), albo na podstawie tablic uzyskanych w wariancie (d).

W każdym z wymienionych wariantów należy zadbać o to aby działający już program wyświetlał komunikaty, żądania danych i objaśnienia wyników w zrozumiałej postaci.

Przy konstruowaniu algorytmów warto wykorzystać następujące zalecenia:

1. Określ jakie zmienne są niezbędne, sporządź ich listę z dokładnymi objaśnieniami roli każdej zmiennej np.:
   N = liczba obiegów (cykli) pętli
   I = numer obiegu
   . . . . . .
2. Zastanów się jakie operacje muszą być wykonane jednorazowo na początku czyli przed pętlą?
3. Określ które operacje mają być wielokrotnie powtarzane (czyli umieszczone wewnątrz pętli)?
4. Każdy próbny wariant algorytmu testuj na konkretnych danych liczbowych.

6.2. ZADANIA Z GEOMETRII

Z11. Wyznacz wysokość trójkąta równoramiennego względem podstawy, mając dane: długość podstawy oraz długość ramienia.

Z12. W dowolnym trójkącie mamy dane dwa boki oraz kąt między nimi. Wyznacz ze wzoru cosinusów długość trzeciego boku, oraz pozostałe kąty.

Z13. W dowolnym trójkącie mamy dany jeden bok oraz dwa przylegające do niego kąty. Wyznacz ze wzoru sinusów długości pozostałych boków.

Z14. Prostokąt jest wpisany w okrąg o danym promieniu R. Wyznacz pole prostokąta mając długość krótszego boku.

Z15. W graniastosłupie o podstawie kwadratowej dany jest bok podstawy oraz kąt między przekątnymi sąsiednich ścian bocznych. Wyznacz objętość i pole powierzchni graniastosłupa.

Z16. Oblicz pole równoległoboku jeśli dany jest jego obwód i dwie wysokości.

Z17. Na okręgu opisano trapez prostokątny. Dane są odległości środka okręgu od końców ramienia pochyłego. Oblicz pole trapezu.

Z18. Trapez równoramienny o danej przekątnej i danym obwodzie, jest opisany na okręgu. Oblicz promień okręgu wpisanego w ten trapez i okręgu opisanego na tym trapezie.

Z19. Oblicz pole romby o danym kącie ostrym, wiedząc, że romb ten opisany jest na okręgu o danym promieniu.

Z20. do Z34: Oblicz pole i obwód podanej figury z Rys. 6.1.

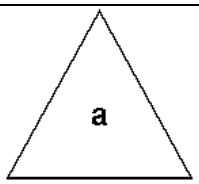


Fig.1. Trójkąt równoboczny o boku **a**

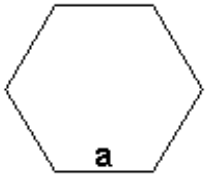


Fig.2. Sześciokąt o danym boku **a**

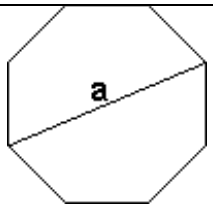


Fig.3. Ośmiokąt o przekątnej a

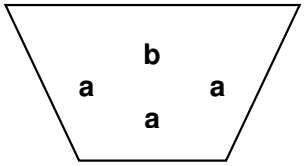


Fig.4. Trapez o bokach: **a, a, a, b**

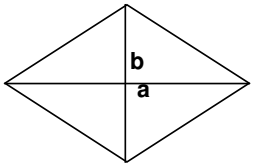


Fig.5. Romb o danych przekątnych **a, b**

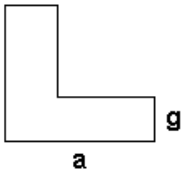


Fig.6. Wycinek kwadratu o boku **a**

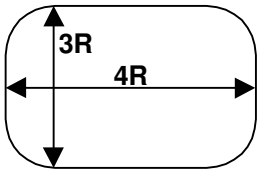


Fig.7. Prostokąt **3R** x **4R** zaokrąglony łukami o promieniu **R**

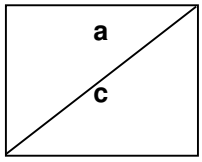


Fig.8. Prostokąt o danym boku **a** oraz przekątnej **c**

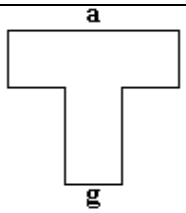


Fig.9. Wycinek kwadratu o boku **a**

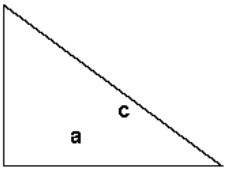


Fig.10. Trójkąt prostokątny o danej przyprostokątnej **a** oraz przeciwprostokątnej **c**

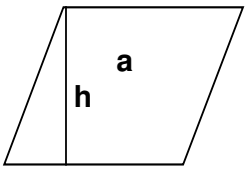


Fig.11. Romb o danym boku **a** i wysokości **h**

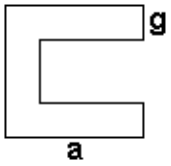


Fig.12. Wycinek kwadratu o boku **a**

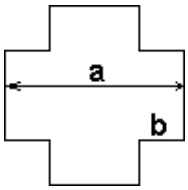


Fig.13. Krzyż – wycinek kwadratu o boku **a** i wcięciach **b**

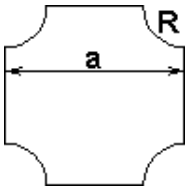


Fig. 14. wycinek kwadratu o boku **a** i wcięciach o promieniu **R**

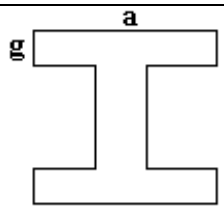


Fig.15 Wycinek kwadratu o boku **a**

Rys. 6.1. Figury geometryczne

6.3. ZADANIA Z FIZYKI

Z35. Kąt między płaszczyznami tworzącymi ostrze siekiery wynosi α. Jaka siła będzie rozłupywać pieniek jeśli siekiera naciska z siłą F. Oblicz dla: α=8°, F=50N.

Z36. Ciężarek upuszczony z okna domu spadał przez **ts** sekund. Z jakiej wysokości spadał?

Z37. Jaką prędkość przy ziemi uzyska ciężarek z poprzedniego zadania. Oblicz mając **ts**.

Z38. Jaką energię kinetyczną uzyska ciężarek z poprzedniego zadania uderzając w ziemię? Wyznacz mając dane **ts** i sprawdź prawo zachowania energii.

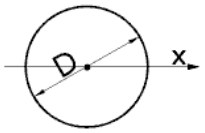
Z39. Oporniki R1, R2 połączone są równolegle. Wyznacz opór zastępczy Rz dla danej wartości R1 oraz kilku wartości R2. $\frac{1}{Rz} = \frac{1}{R1} + \frac{1}{R2}$

Z40. W kulistym zbiorniku o promieniu R, poziom cieczy (od dna), wynosi H metrów. Wyznacz objętość cieczy dla różnych H. $V = \frac{\pi \cdot H^2 (3R - H)}{3}$

Z41. Masa Ziemi wynosi Mz=5,9736*10^{24} kg, a jej przeciętna średnica 2r =12742 km. Oblicz siłę z jaką Ziemia przyciąga rakietę o masie Q: a) na powierzchni Ziemi, b) na wysokości 10 tys. km, jeśli stała grawitacyjna wynosi: γ=6,6725*10^{-11} [N*m^2/kg^2]. $F = \gamma \cdot \frac{Q \cdot Mz}{r^2}$

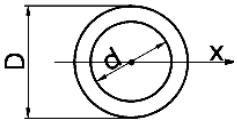
Moment bezwładności jest miarą bezwładności ciała w jego ruchu obrotowym względem określonej osi. Im jest on większy, tym trudniej zmienić prędkość kątową ciała. Moment bezwładności punktu materialnego obliczamy mnożąc jego masę przez kwadrat odległości od osi obrotu. Dla płaskich figur geometrycznych zamiast masy uwzględniane są rozkłady pola figur względem osi obrotu (więc wymiarem jest długość do potęgi czwartej).

Z42. Oblicz moment bezwładności koła względem średnicy mając dany promień koła R



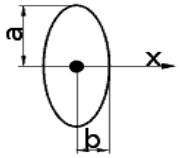
$I_x = \frac{\pi D^4}{64}$

Z43. Oblicz moment bezwładności pierścienia względem osi x przechodzącej przez środek, mając dane obie średnice D oraz d.



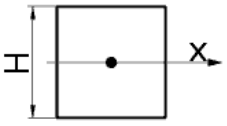
$I_x = \frac{\pi (D^4 - d^4)}{64}$

Z44. Oblicz moment bezwładności przekroju eliptycznego względem krótszej osi, mając dane obie półosie: **a** oraz **b**.



$I_x = \frac{\pi a^3 b}{4}$

Z45. Oblicz moment bezwładności przekroju kwadratowego o boku H, względem osi x przechodzącej przez środek i równoległej do boku.



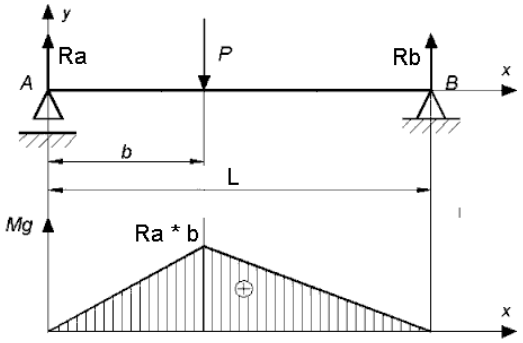
$I_x = \frac{h^4}{12}$

6.4. ALGORYTMY Z WARUNKAMI I WYBOREM

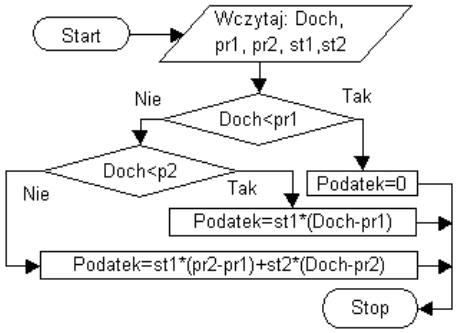
Z46. Scal obliczanie momentów bezwładności z zadań A13 – A16 w jeden program umożliwiający wybranie jednego z czterech wariantów.

Z47. Wyznacz pierwiastki równania kwadratowego i wykorzystując instrukcje IF przekształć wzór paraboli obrazującej to równanie tak aby część znajdująca się pod osią x została zastąpiona swoim odbiciem symetrycznym.

Z48. Belka ma długość **L** i jest obciążona punktowo siłą **F** w odległości **r** od swego początku. Wyznacz równanie momentu zginającego belkę w funkcji odległości **x** od poczatku belki (**0<x<L**).



Z49. Napisz algorytm w postaci opisu słownego dla obliczania należnego podatku. Dany jest dochód (Doch) oraz progi i stopy podatku.
Przykładowo: w roku 2010 były 2 progi podatkowe: **pr1**=3091zł, **pr2**=85528zł, oraz dwie stopy podatkowe **st1**=18%, **st2**=32%.



Z50. Dane są dwa prostokąty, każdy poprzez współrzędne swoich dwu przeciwległych wierzchołków (końców przekątnej). Należy wyznaczyć pole części wspólnej tych prostokątów.

6.5. CIĄGI I ITERACJE

Z51. **Rozszczepienie** ciągu
Dany jest ciąg dowolnych liczb zawierający N elementów. Należy rozdzielić go na dwa ciągi, tak aby w jednym umieszczane były tylko liczby ujemne a w drugim nieujemne. Na końcu algorytm ma podać ile było liczb ujemnych a ile nieujemnych. Przetestuj na odpowiednim przykładzie.

Z52. **Iloczyn** z warunkami
Dany jest ciąg N dowolnych liczb. Należy wyznaczać iloczyn niezerowych liczb, a przy każdym napotkaniu zera wydrukować dotychczasowy iloczyn i zacząć wyznaczać kolejny nowy iloczyn aż do napotkania kolejnego zera i tak aż do końca ciągu. Przetestuj na odpowiednim przykładzie.

Z53. **Zliczanie** liczb parzystych i nieparzystych
Dany jest ciąg N dowolnych liczb całkowitych dodatnich. Należy policzyć ile w tym ciągu jest liczb parzystych a ile nieparzystych. Do badania parzystosci wykorzystaj funkcję mod(x,y), która wyznacza resztę z dzielenia x przez y.

Z54. Warunkowe **wyznaczanie silni**
Narysuj schemat blokowy algorytmu, który: wczytuje dane: N oraz A, sprawdza czy N jest mniejsze od A i jeśli tak to oblicza w pętli silnię liczby N a jeśli nie to wyświetla odpowiedni komunikat. Sprawdź dla N=5, A=6 oraz N=5, A=4.

Z55. **Generowanie liczb**
Narysuj schemat blokowy algorytmu, który: dla danych: N oraz A, generuje w pojedynczej pętli N liczb naturalnych K= 1, 2, 3, ... , przy czym jeśli K mniejsze od A to wyświetla X=K, a jeśli nie to wyświetla X=100-K. Sprawdź dla N=8, A=5.

Z56. **Wydatki**
Narysuj schemat blokowy algorytmu, który: jeśli masz daną SUMĘ pieniędzy pozwala odejmować od niej kolejny WYDATEK pod warunkiem że nie przekracza on aktualnej wartości tej SUMY, która jeszcze ci pozostała, a w przeciwnym przypadku wyświetli odpowiedni komunikat.

Z57. **Długość wektora** N wymiarowego
Wyznacz długość (pierwiastek z sumy kwadratów) wektora N wymiarowego - tzn. posiadajacego N składowych: Dx1, Dx2, Dx3, Dx4, ...(każda składowa jest długością rzutu wektora na odpowiednią oś).
Sprawdź dla wektora dwu i trójwymiarowego.

Z58. **Wypadkowa** N wektorów
Oblicz wypadkową N wektorów dwuwymiarowych.

Z59. **Testowanie ciągu 1**. Sprawdź czy cztery wczytane liczby tworzą postęp arytmetyczny. Uogólnij na N liczb.

Z60. **Testowanie ciągu 2**. Sprawdź czy cztery wczytane liczby tworzą postęp geometryczny. Uogólnij na N liczb.

Z61. **Modelowanie przesiewania kulek.**
Dany jest bok oczka siatki sita oraz N kulek o różnych średnicach.
Wyznacz ile kulek przeleci przez sito a ile zostanie na nim.

Z62. **Histogram**. Dla danego ciągu N danych pomiarowych Xi (i=1,..N) wyznacz Xmin, Xmax. Zakres (Xmin, Xmax) podziel na P przedziałów.
Zlicz ile wyników pomiarów mieści się w każdym z tych przedziałów.

Z63. **Sortowanie**
a) Dla x=1,2,3, ...,15, wygeneruj ciąg 20*sin(x), a następnie posortuj ten ciąg.
b) Dla k=1,2,3,..., 17, wygeneruj ciąg i*(-1)^i, a następnie posortuj ten ciąg.

Rozwinięcia wybranych funkcji w szereg Maclaurina
Napisz algorytm lub program wyznaczający - dla wczytanej wartości ***X*** oraz dokładności ***D*** – wartość podanej funkcji, przy pomocy wyznaczenia sumy ***N*** wyrazów szeregu Maclaurina. Ostatni ***N***-ty wyraz szeregu powinien być mniejszy od ***D***. Oto niektóre szeregi:

Z64. funkcja **potęgowa** $(1+x)^m$ dla m>0 oraz |x|≤1 ma rozwinięcie w postaci szeregu:

$$1+mx+\frac{m(m-1)}{2!}x^2+\frac{m(m-1)(m-2)}{3!}x^3+...$$

Z65. funkcja **sinus x** ma rozwinięcie: $x-\frac{x^3}{3!}+\frac{x^5}{5!}-...+(-1)^i\frac{x^{2i+1}}{(2i+1)!}...$

Z66. funkcja **cosinus x** ma rozwinięcie: $1-\frac{x^2}{2!}+\frac{x^4}{4!}-...+(-1)^i\frac{x^{2i}}{(2i)!}...$

Z67. funkcja **tangens x** ma rozwinięcie: $x+\frac{1}{3}x^3+\frac{2}{15}x^5+\frac{17}{315}x^7+...$

Z68. funkcja $\mathbf{e^x}$ ma rozwinięcie: $1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+...$

Z69. si**nus hiperboliczny** x ma rozwinięcie: $x+\frac{x^3}{3!}+\frac{x^5}{5!}+...$

Z70. **cosinus hiperboliczny** x ma rozwinięcie: $1+\frac{x^2}{2!}+\frac{x^4}{4!}+...$

7. WPROWADZENIE DO PROGRAMOWANIA W VISUAL BASIC

7.1. TWORZENIE PROGRAMÓW DLA MS WINDOWS W JĘZYKU MS VISUAL BASIC

Programy pisane w języku BASIC można wprawdzie uruchamiać w Ms Windows, gdyż jest wiele wersji tego języka pracujących we współczesnych systemach operacyjnych – jak choćby wersja QB64 zaprezentowana w rozdziale 0 – ale generalnie, język ten dostosowany do systemu DOS, nie potrafi wykorzystać graficznego interfejsu i bogatych możliwości systemu Ms Windows.

Tworzenie programów obiektowo-zdarzeniowych wykorzystujących obiekty graficzne dostępne w Ms Windows umożliwiają takie języki jak C++, Java, Delphi oraz Visual BASIC.

Visual BASIC (w skrócie VB) należy do najprostszych języków pozwalających opracowywać programy dla systemów Ms Windows. Istnieje w wielu wersjach i odmianach. Jedą z nich jest **VBA** czyli *Visual BASIC for Applications*, wbudowany i poszerzający możliwości tak znanych produktów jak Autocad czy pakiet biurowy Ms Office.

Najnowsze wersje Visual BASIC-a wchodzą w skład pakietu **Microsoft Visual Studio**. Pakiet ten występuje w kilku, bardziej lub mniej zaawansowanych wersjach. Wersja Visual Studio Express 2010, zawierająca **Visual BASIC 2010 Express**, Visual C++ 2010 Express, oraz Visual C# 2010 Express, jest dostępna za darmo. Istnieje też wersja **Visual Studio 2010 Express for Windows Phone** – do tworzenia programów na smartfony i tablety.

Od roku 2002 Visual BASIC – początkowo przemianowany na Visual BASIC.NET – działa na platformie programistycznej „**.NET**" (czytaj „dot-net"). Oznacza to, że do uruchomienia programu napisanego w języku VB wymagana jest bezpłatna biblioteka uruchomieniowa *.NET Framework*. Programy napisane w dowolnym języku dostępnym w Visual Studio są kompilowane do kodu pośredniego wykonywanego w maszynie wirtualnej zgodnej z .NET. Zaadaptowanie Visual BASIC-a do platformy .NET poszerzyło jego możliwości i **zbliżyło funkcjonalność do C#, a kod produkowany przez kompilatory obu języków często jest identyczny.**

Do nauczania i ćwiczeń mogą jednak z powodzeniem służyć także starsze wersje jak np.: Visual BASIC 3. Przy poznawaniu Visual BASIC-a przyda się znajomość niektórych środków poznanych już w QB64, ale sporo nowych rzeczy trzeba się będzie nauczyć.

Ms Visual BASIC należy do systemów **RAD** (*Rapid Application Development*) – szybkiego tworzenia aplikacji – po części budowanych z gotowych komponentów.

Istnieją inne systemy RAD – oparte na innych językach programowania jak: Java Builder, C++ Builder, oraz Delphi – oparty na języku Pascal.

Komponenty z których buduje się programy w systemie Ms Windows to **obiekty** (*Objects*). Najważniejsze z nich to **formatki** (*Forms*) i **kontrolki** (*Controls*). Każdy obiekt posiada wiele **cech** (*Properties*). Wartości cech można zmieniać - albo dialogowo w IDE, albo poleceniami o składni:

```
Obiekt.NazwaCechy = WartośćCechy
```

Obiektowi przypisane są też określone **zdarzenia** (*Events*) - związane z akcjami użytkownika (myszką , klawiszami, ...) oraz zaistniałą sytuacją na ekranie. Od wybranych zdarzeń można uzależnić to co ma się dziać w programie pisząc własne **procedury** (*Sub*) **zdarzeniowe** o składni:

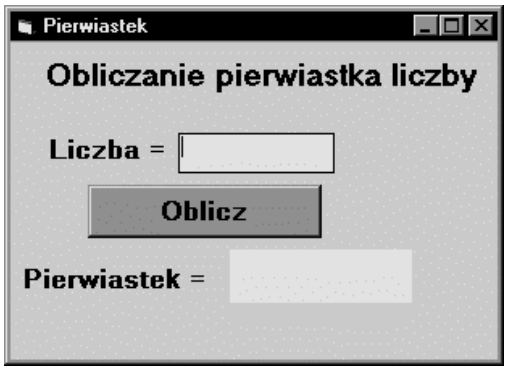
```
Sub NazwaObiektu_Zdarzenie()
   Instrukcje
End Sub
```

Można też wykorzystywać podprogramy standardowo przypisane obiektowi, zwane jego **metodami** (*Methods*). Wywołanie metody ma składnię: `Obiekt.Metoda`

7.2. TWÓJ PIERWSZY PROGRAM W VISUAL BASIC-U

7.2.1. KONCEPCJA PROGRAMU

Zanim przystąpi się do pisania jakiegokolwiek programu użytkowego (czyli tzw. aplikacji) trzeba opracować jego szczegółową koncepcję a więc przede wszystkim dokładnie wiedzieć do czego program ma służyć, jak będzie się prezentował na ekranie i porozumiewał z użytkownikiem (na przykład na które działania myszki ma reagować), jakie dane będą mu potrzebne do działania i w jakiej postaci je mu dostarczymy, jakie wyniki i gdzie mają być wyprowadzane itp, itd.



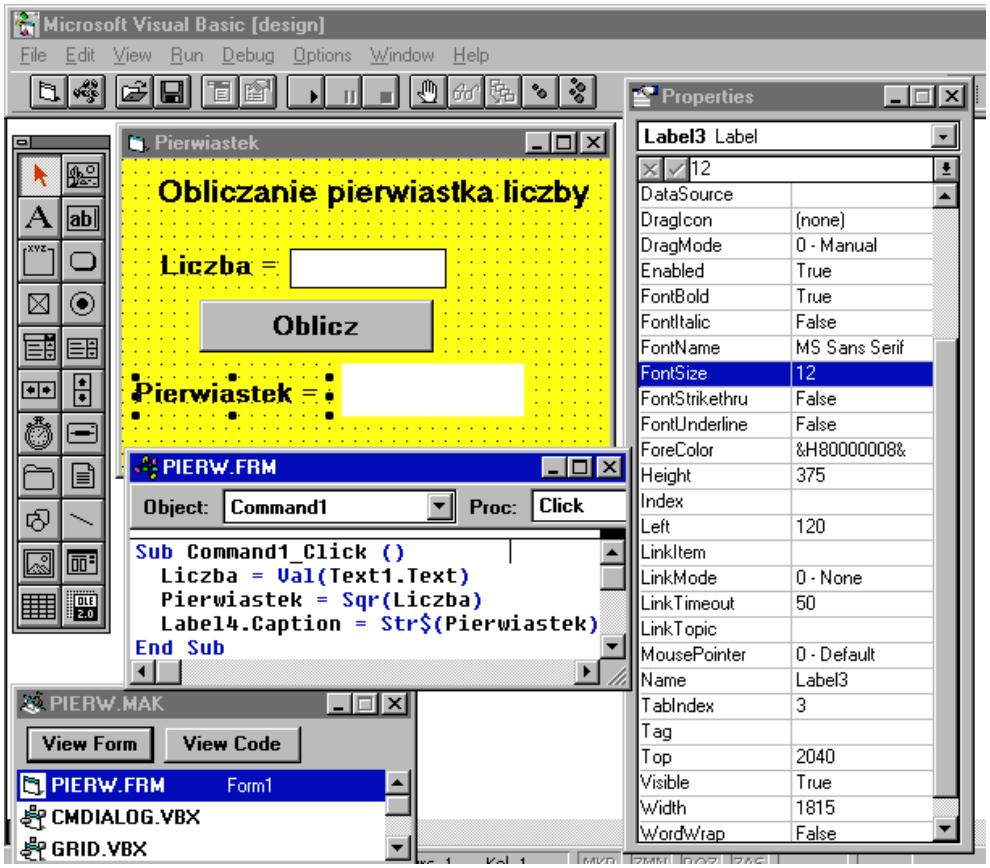
Rys. 7.1. Docelowy widok projektowanej aplikacji

Nasz pierwszy program będzie bardzo prosty. Otóż chcemy utworzyć aplikację do obliczania pierwiastków z liczb wprowadzanych przy pomocy klawiatury.

Docelowy wygląd okna naszej aplikacji ma być taki jak na Rys. 7.1, przy czym tło okna ma być żółte.

7.2.2. URUCHOMIENIE I OBSŁUGA VISUAL BASIC-A

Po uruchomieniu Visual BASIC-a (użyto VB3) pojawią się elementy widoczne na Rys. 7.2.



Rys. 7.2. Okna Visual BASIC-a z widocznym gotowym już projektem pierwszego programu (PR 65)

Element te to:

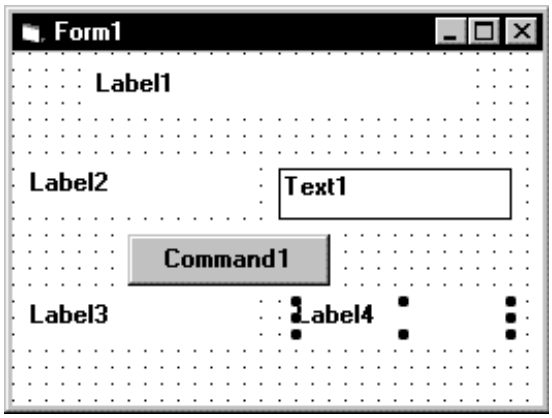
1. **pasek tytułowy** oraz **Menu główne** (u góry) z paskiem przycisków "standardowych";
2. pusta **Formatka (*Form1*)** projektowanej aplikacji (po środku) ;
3. **paleta Kontrolek** (po lewej) do wstawiania na formatkę (patrz Tabela 7.1);
4. **okno Własności (*Properties*)** wybranego kliknięciem obiektu;
5. **okno Projektu** (czasem schowane pod innymi) – na początku zatytułowane **Project1** a potem zawierające nazwę naszego projektu.

Tabela 7.1. Kontrolki

Ikona	Kontrolka	Funkcja
	PictureBox	Okno do rysowania, może też zawierać obraz wczytany z pliku.
	Label	Wyświetla tekst, którego użytkownik nie może zmienić.
	TextBox	Okno tekstowe edycyjne - do wpisywania i poprawiania.
	Frame	Ramka do grupowania innych elementów (kontrolek)
	Command Button	Przycisk do uruchamiania operacji (wywoływania podprogramu)
	CheckBox	Pola opcji do wyboru dowolnych opcji z wielu możliwych
	OptionButton	Przełącznik – wybór jednej opcji z wielu
	ComboBox	Lista rozwijalna
	ListBox	Lista do wyboru jednej lub kilku opcji
	HscrollBar	Poziomy pasek przewijania
	VScrollBar	Pionowy pasek przewijania
	Timer	Budzik – do uruchomiania operacji w określonych momentach czasu
	DriveListBox	Lista do wyboru dysku na którym będziemy szukać pliku
	DirListBox	Lista do wyboru katalogu dyskowego
	FileListBox	Lista do wyboru pliku
	Shape	Kreśli prostokąty, okręgi i inne kształty
	Line	Kreśli odcinek linii prostej
	Image	Wyświetla obraz typu bitmapa, ikona, lub metaplik
	CommonDialog	Okienko dialogu - do umieszczania innych kontrolek.
	Grid	Siatka arkusza kalkulacyjnego

Kontrolki widoczne na palecie po lewej stronie Rys. 7.2 objaśnia Tabela 7.1.

Na początku pracy **formatka** nosząca na Rys. 7.1 nazwę "Pierwiastek" nazywa się "Form1" i jest pusta a naszym zadaniem będzie wstawienie na nią odpowiednich elementów zwanych "**kontrolkami**" (ang.: *Controls*) aby otrzymać to co pokazano na Rys. 7.1.



Rys. 7.3 Formatka z wstawionymi kontrolkami

Aby wstawić wybraną kontrolkę wystarczy kliknąć jej przycisk na palecie a następnie zakreślić na formatce prostokąt określający położenie i wielkość kontrolki. Ostatnio wstawiona kontrolka (lub jedna z wcześniej wstawionych a zaznaczona przez kliknięcie) posiada uchwyty pozwalające zmieniać (myszką) jej wymiary – na Rys. 7.3 jest to ***Label4***. Ciągnąc myszką za środek zaznaczonej kontrolki można ją przemieszczać.

Klawisz Delete pozwala usunąć zaznaczoną a niepotrzebną kontrolkę.

Wstaw na formatkę kontrolki ***Label, TextBox, CommanButton*** tak aby otrzymać efekt podobny jak na Rys. 7.3. To co pokazuje Rys. 7.3 będzie potrzebne do pisania rozkazów programu – dlatego, **zawsze po wstawieniu kontrolek: wydrukuj (lub naszkicuj) ich rozmieszczenie wraz z nazwami** a przed dalszymi czynnościami.

7.2.3. USTAWIANIE CECH OBIEKTÓW

W oknie ***Properties*** (Rys. 7.2) widoczne są cechy jednego, wybranego wcześniej kliknięciem, obiektu, na przykład formatki lub jednej z wstawionych na nią kontrolek.

Każda z wybranych w oknie ***Properties*** cech ma **nazwę** oraz **wartość**, która może być zmieniona w linii edycyjnej tego okna (trzecia linia od góry).

Wartości cech mogą też być zmieniane (lub wykorzystywane) przy pomocy rozkazów języka BASIC w procedurach lub funkcjach. Wówczas nazwa cechy musi być poprzedzona kropką i nazwą obiektu którego ona dotyczy na przykład:

```
Form1.BackColor = RGB(150,255,150)
```

- ustawi kolor jasnozielony dla tła formatki ***Form1.***

Trzeci sposób zmiany, ale niektórych tylko cech, jak na przykład położenie i wielkość, to działanie myszką – przemieszczanie i rozciąganie obiektów – spowoduje zarazem zmianę liczbowych wartości cech Top, Left (położenie) oraz Width, Height (rozmiary).

Dla formatki jak i dla większości innych obiektów najważniejsze cechy to:

- **Name** – nazwa obiektu którą posługiwać się należy w rozkazach programu – w tym przypadku: *Form1*
- **Caption** – napis, którym pojawi się na ekranie (wstępnie jest taki sam jak nazwa).

Wartości obu tych cech można zmienić ale podczas gdy nazwy (*Name*) można pozostawić takie jakie VB nadał automatycznie (lub zmienić na bardziej adekwatne) to napisy na ekranie (*Captions*) na pewno chcemy mieć inne choćby dlatego aby zmienić je na polskie i niosące jak najwięcej niezbędnych informacji dla użytkownika programu

Zmieńmy więc wartość cechy **Caption** dla formatki z "Form1" na "Pierwiastek".

Zmieńmy wartość cechy **BackColor** klikając na niej podwójnie i wybierając odpowiedni kolor (jasnożółty). Jeśli chcielibyśmy zamiast koloru wstawić jakiś odpowiedni obraz jako tło to cecha Picture pozwoli wybrać plik obrazu, który ma być tłem formatki.

Inne cechy pozostawimy bez zmian.

Następnie trzeba zmienić wartości cech ***Caption*** dla wszystkich kontrolek ***Label1 - Label4*** oraz dla kontrolki ***Command1*** na takie jak na Rys. 7.1.

Dla okienka edycji tekstów czyli kontrolki typu **TextBox** (u nas noszącej nazwę ***Text1***) najważniejszą cechą jest cecha ***text***. W naszej kontrolce ***Text1*** wymażemy wartość cechy ***text*** aby pozostawić puste okienko do wpisywania liczb.

7.2.4. PROCEDURY ZDARZENIOWE

Teraz gdy mamy odpowiednie elementy w oknie naszej aplikacji i dostosowaliśmy ich postać do naszych potrzeb wygląda to jak program ale przecież musi jeszcze **działać** – reagować na nasze akcje i wykonywać poprawnie to co zaplanowaliśmy. Do tego potrzebny jest program zbudowany z deklaracji i instrukcji języka BASIC tworzących procedury i funkcje.

Programy dialogowe a w szczególności te obsługiwane głównie myszką muszą zawierać **procedury zdarzeniowe** to znaczy procedury wywoływane przez zaplanowane przez nas zdarzenia jak na przykład kliknięcie myszką

W naszym programie chcemy aby po wpisaniu liczby w okienku ***Text1*** i kliknięciu myszką przycisku ***Command1*** (z napisem "Oblicz") został obliczony i wyświetlony w okienku ***Label4*** pierwiastek z wpisanej liczby.

Aby otrzymać szkielet procedury reagującej na kliknięcie przycisku ***Command1*** wystarczy podwójnie kliknąć na nim. Otrzymamy wtedy:

```
Sub Command1_Click ()
. . .
End Sub
```

No tak ale resztę musimy wymyślić i wpisać sami (tam gdzie postawiono kropki) na podstawie znajomości języka, nazw kontrolek i ich cech oraz świadomości co ma program wykonać.

Gdy po uruchomieniu naszego programu wpiszemy liczbę w okienku ***tekst1*** to będzie ona stanowiła wartość jego cechy ***text***. W instrukcjach BASIC-a będzie to więc: ***text1.text***

Następnie trzeba z tej wpisanej liczby obliczyć pierwiastek funkcją SQR ale tu pojawi się problem. Otóż BASIC rozróżnia typy zmiennych i funkcji a w szczególności:

- **tekstowe** czyli typu ***string*** - których wartościami są dowolne ciągi znaków drukarskich,
- **liczbowe** – mogące brać udział w obliczeniach.

Naszym problemem jest to, że chcemy coś obliczyć ale to co wpisujemy w okienkach typu TextBox jest zawsze przyjmowane przez Visual BASIC jako **tekst** (dokładniej jest typu ***string***) a nie jako liczba (nawet gdy składa się z samych cyfr).

Na szczęście są odpowiednie funkcje konwersji typów:

- funkcja **Val** przekształcająca **tekst** złożony z cyfr **na liczbę**, oraz
- funkcja **Str$** przekształcająca **liczbę na tekst**

Tak więc ostatecznie po przypomnieniu sobie kilku funkcji lub wyszukaniu w pomocy (***HELP***) takich haseł jak ***Math functions, Data type conversion functions*** możemy wpisać treść procedury pokazaną poniżej:

```
Sub Command1_Click ()
    Liczba = Val(Text1.Text)
    Pierwiastek = Sqr(Liczba)
    Label4.Caption = Str$(Pierwiastek)
End Sub
```

Pamiętamy (mam nadzieję), że instrukcje o budowie:

zmienna = wyrażenie lub ***LET zmienna = wyrażenie***

- to nie żadne równania czy równości ale instrukcje wyznaczające wartość ***wyrażenia*** i podstawiające ją do ***zmiennej*** (traktowanej jak pojemnik).

Tłustym drukiem wyróżniono nazwy wspomnianych wyżej funkcji standardowych BASIC-a. Ponieważ cecha ***Caption*** okienka ***Label4*** też musi mieć typ tekstowy (***String***) więc wynik zamieniono na tekst funkcją ***Str$***.

Pamiętajmy, że zmienne i funkcje o wartościach tekstowych muszą mieć w BASIC-u nazwy zakończone znakiem dolara.

7.2.5. ZAPISANIE ORAZ URUCHOMIENIE I TESTOWANIE DZIAŁANIA APLIKACJI

Nawet jeśli nasze dzieło nie jest jeszcze całkiem gotowe to trzeba je zapisać do pliku, aby nie stracić po wyjściu z Visual BASIC.

Operacja ***File – Save Project As...*** zapisuje cały projekt z formatkami i modułami programowymi natomiast ***File - Save File As...*** – pozwala zapisać formatkę lub moduł.

Operacja ***File – Make EXE File...*** – utworzy samodzielny program działający bez potrzeby uruchamiania Visual BASIC-a, aczkolwiek wymagający istnienia pliku **vbrun300.dll,** jeśli działaliśmy w Visual BASIC 3, lub zainstalowanej odpowiedniej platformy „.NET Framework" jeśli działaliśmy w Visual Studio.

Jeśli chcemy sprawdzić jak działa aplikacja, którą utworzyliśmy, to wystarczy nacisnąć klawisz [**F5**] lub z menu wybrać ***Run-Start***. Gdy chcemy jednak coś poprawić to trzeba zatrzymać działanie programu wybierając z menu: ***Run-End***.

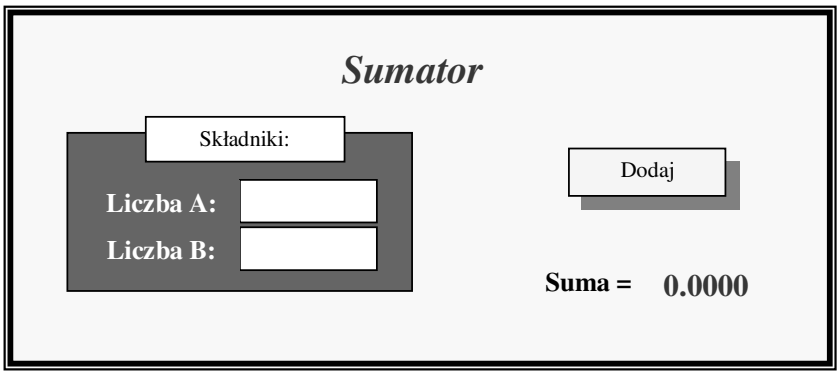
Po utworzeniu (jak wyżej) pliku z rozszerzeniem „.EXE" możemy także testować samodzielny program, po zakończeniu pracy z VB.

7.3. ZADANIA

Po zapoznaniu się z powyższym przykładem spróbuj samodzielnie wykonać zadania.

Z71. Suma dwu liczb

Napisz program do sumowania liczb, wyświetlający okienko podobne jak na Rys. 7.4:



Rys. 7.4. Okno programu "Sumator dwu liczb"

Działanie SUMATORA ma być bardzo proste: po wpisaniu dwu liczb i kliknięciu przycisku „Sumuj" powinna ukazać się ich suma (Rys. 7.4). Jest to oczywiście koncepcja nie pozwalająca sumować wielu liczb i w następnym programie postarasz się ją ulepszyć.

Z72. Sumowanie ciągu liczb

Zaprojektuj i napisz lepszy wariant poprzedniego programu, pozwalający sumować wiele liczb.

Z73. Kalkulator

Napisz program pozwalający wykonywać 4 lub więcej działań na wpisywanych liczbach. Do tego oczywiście trzeba będzie zwiększyć liczbę przycisków.

8. PODSTAWY PROGRAMOWANIA W MATLAB-IE

Ten rozdział ma pomóc w rozpoczęciu **użytkowania MATLAB-a** ale przede wszystkim pomóc w poznaniu **podstaw programowania w ogólności** – szczególnie w przypadku gdyby ze względu na bardzo ograniczony czas zajęć na Studiach Niestacjonarnych – zrezygnowano na zajęciach z języka BASIC. Dlatego **nauczanie MATLAB-a podzielono na etapy**. Niniejszy rozdział dotyczy tylko tych podstawowych środków MATLAB-a, które są podobne do odpowiadających im środków w tradycyjnych językach programowania (imperatywnego, strukturalnego i proceduralnego) jak BASIC, Algol, Pascal czy C. W szczególności, w tym rozdziale ograniczymy się do **działań na skalarach** oraz **elementach** wektorów i macierzy, realizowanych m.in. przy pomocy pętli programowych.

Część różnic między MATLABem a tradycyjnymi językami wyjaśniono w tym rozdziale ale wybrane istotne **środki specyficzne dla MATLAB-a** - jak **operacje macierzowe nie stosujące pętli programowych** oraz t.zw. „**grafika uchwytów**" – **zostały omówione w następnych rozdziałach**.

8.1. WPROWADZENIE

Programowanie w języku i środowisku MATLAB-a stanowi jeden z głównych tematów tego podręcznika.

MATLAB został wybrany z kilku przyczyn. Po pierwsze udostępnia dla początkujących rozbudowany system pomocy i okno komend w którym można testować – w trybie konwersacyjnym - działanie pojedynczych poleceń, a na poziomie elementarnym pozwala pisać programy bardzo podobne jak w BASIC-u – wyręczając użytkowników z konieczności deklarowania typów zmiennych.

Po drugie oprócz swej prostoty na poziomie podstawowym, jest równocześnie potężnym narzędziem do operowania na macierzach i do obliczeń naukowo-technicznych z bardzo wielu dziedzin, powszechnie stosowanym na uczelniach Świata.

Po trzecie zaś, umożliwia naukę programowania obiektowego i zawiera wiele zapożyczeń z popularnego wśród programistów języka C, co w przyszłości może ułatwić naukę tego języka.

Poznanie MATLAB-a na pierwszym roku studiów ułatwi korzystanie z jego wyspecjalizowanych zastosowań w ramach różnych przedmiotów nauczania na starszych latach studiów.

Czym jest MATLAB?

MATLAB (od angielskiej nazwy: MATrix LABoratory) to rozwijany od roku 1984 **pakiet oprogramowania matematycznego** firmy MathWorks Inc. wyposażony w:

- **język i środowisko programowania** do obliczeń naukowo-technicznych,
- zestaw wielu tematycznych bibliotek **podprogramów** (toolbox'ów),

– obszerną i przystępnie napisaną dokumentację w języku angielskim (w postaci plików typu PDF i HTML), złożoną z kilkudziesięciu **podręczników** i bardzo wielu **przykładowych programów** (demos) .

Oprócz oryginalnej anglojęzycznej dokumentacji istnieje szereg książek w języku polskim a między innymi, wymienione w spisie literatury pozycje [53] do [58].

Dlaczego warto poznać MATLAB-a?

Bo stanowi potężne i rozpowszechnione na Świecie narzędzie do obliczeń naukowo-technicznych i inżynierskich a zarazem - w podstawowym zakresie – jest łatwy w obsłudze i nadaje się do nauki programowania.

Posiada bogaty zestaw **funkcji matematycznych** i funkcji do prezentacji wyników - w postaci różnorodnych **wykresów** a także **animacji**. Specjalistyczne „toolbox'y" czynią go narzędziem dostosowanym do prawie każdej dziedziny, a przystępnie (choć po angielsku) objaśnione przykłady pozwalają poznawać metody matematyczne w praktyce.

W MATLAB-ie można stosować także programowanie **obiektowe** i tworzyć programy "okienkowe" **z graficznym interfejsem użytkownika**. Możliwa jest również współpraca programów MATLAB-a z programami w językach C oraz Fortran – dzięki **API** (*Application Programming Interface*) czyli interfejsowi programowania aplikacji.

Opracowany w MATLAB-ie **Simulink** to biblioteka modułów symulacyjnych z których budować można układy przetwarzania sygnałów i modelować układy dynamiczne, w podobny sposób jak niegdyś na maszynach analogowych.

Oprócz oryginalnego, komercyjnego MATLAB-a, istnieje i rozwija się szereg legalnie darmowych pakietów oprogramowania wzorowanych na MATLAB-ie, o czym wspomniano nieco dalej.

MATLAB umożliwia m.in:

- wykonywanie wszelkich obliczeń naukowych i inżynierskich,
- modelowanie i symulację,
- analizę danych (w tym: sygnałów i obrazów)
- graficzną wizualizację danych i wyników obliczeń.

Aby oglądać programy demonstracyjne napisane w MATLAB-ie wystarczy w oknie komend wpisać komendę: **demos,** a następnie wybrać program z wykazu.

8.1.1. FREEMAT I INNE DARMOWE ODPOWIEDNIKI MATLAB-A

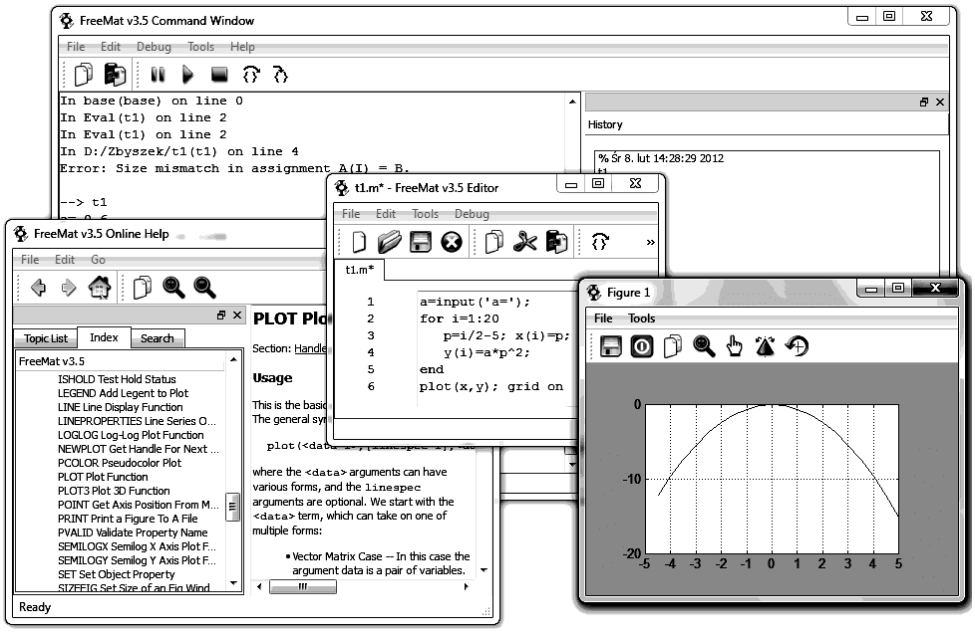
MATLAB jest kosztownym programem ale istnieją także jego legalnie darmowe odpowiedniki, do których należą m.in.:

– OCTAVE - http://octave.sourceforge.net/ także on-line: http://www.mathcloud.se
– SCILAB - www.scilab.org – częściowo spolszczony,
– Euler - http://euler.rene-grothmann.de

- Maxima - http://maxima.sourceforge.net
- **FreeMat** - http://freemat.sourceforge.net/;

Wygodną alternatywą MATLAB-a - dla początkujących - jest FreeMat. Jak twierdzą jego autorzy jest w ok. 95% kompatybilny z MATLABem, a w zakresie najprostszych elementów języka jest identyczny z MATLABem. W związku z tym może on z powodzeniem zastępować MATLAB-a, szczególnie w początkowej fazie uczenia się programowania w tym języku.

Informacje o programie FreeMat można przeczytać (w języku angielskim) na jego stronie internetowej i stamtąd można też pobrać wersję instalacyjną. Polecam wersje wcześniejsze niż 4 (np. 3.6 lub 3.5) gdyż mają niedużą objętość (ok. 10 MB) i wystarczające możliwości, podczas gdy wersja 4.1 ma ok. 10 razy większą objętość.

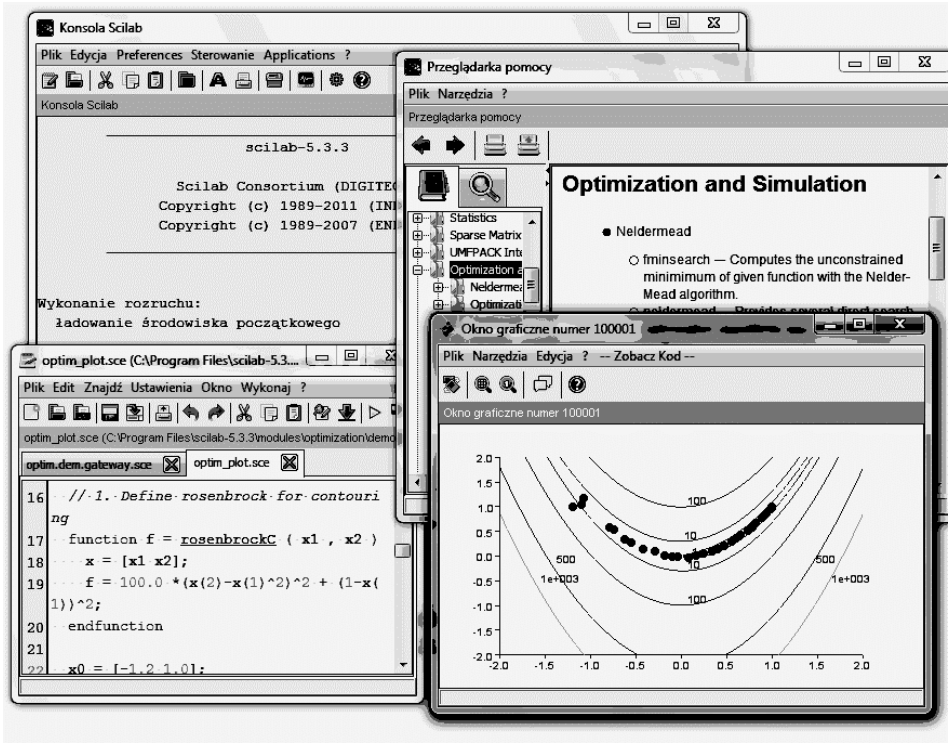


Rys. 8.1. Okno programu FreeMat oraz okno jego edytora

Po zainstalowaniu FreeMat-a trzeba jednorazowo wykonać operację powodującą zapamiętanie ścieżek do jego folderów, które będą przeszukiwane przy wywoływaniu funkcji lub opisów (help). W tym celu trzeba z menu ***TOOLS*** wybrać opcję ***Path Tool***, następnie w okienku, które się pojawi znaleźć i wybrać folder w którym zainstalowaliśmy FreeMat-a (np.: C:\Program Files\FreeMat...) - oraz kliknąć przycisk: [***Add with Subfolders***]. Dodatkowo swój folder w którym będziemy przechowywać programy możemy także dodać do wykazu zapamiętanych ścieżek. Zakończymy te operacje kliknięciem przycisków [***Save***] i [***Done***].

Wygląd okna programu FreeMat oraz okna jego edytora pokazuje Rys. 8.1.

Ciekawą alternatywą MATLAB-a jest także SciLab (Rys. 8.2), który jest dostępny także w częściowo spolszczonej wersji. Scilab posiada również odpowiednik Simulinka. W SciLabie edytor nosi nazwę SciNotes, toolbox'y nazywają się ATOMS a biblioteka modułów symulacyjnych podobna do Simulinka nazywa się Xcos, komentarze zaczynają od „//" a stałe systemowe od „%". Objętość pliku instalacyjnego wersji 5.3.3 (wraz z toolboxami) wynosi ok. 120 MB.



Rys. 8.2. Okna programu SciLab

8.1.2. ŁAGODNY START. OKNO KOMEND MATLAB-A

Po uruchomieniu MATLAB-a, na ekranie (Rys. 8.3) ukazuje się okno główne MATLAB-a, które może być podzielone na kilka okien podrzędnych (a, b, c, d, e) – zależnie od wersji MATLAB-a i ustawień. Przykładowo, dla wersji 7.9, na Rys. 8.3, widoczne są okna:

a) okno komend (Command Window)
b) okno plików w bieżącym folderze (*Current Folder*),
c) okno podglądu plików (*Details*),
d) okno zmiennych w przestrzeni roboczej (*Workspace*),
e) okno historii – z wykazem wydanych komend (*Command History*),

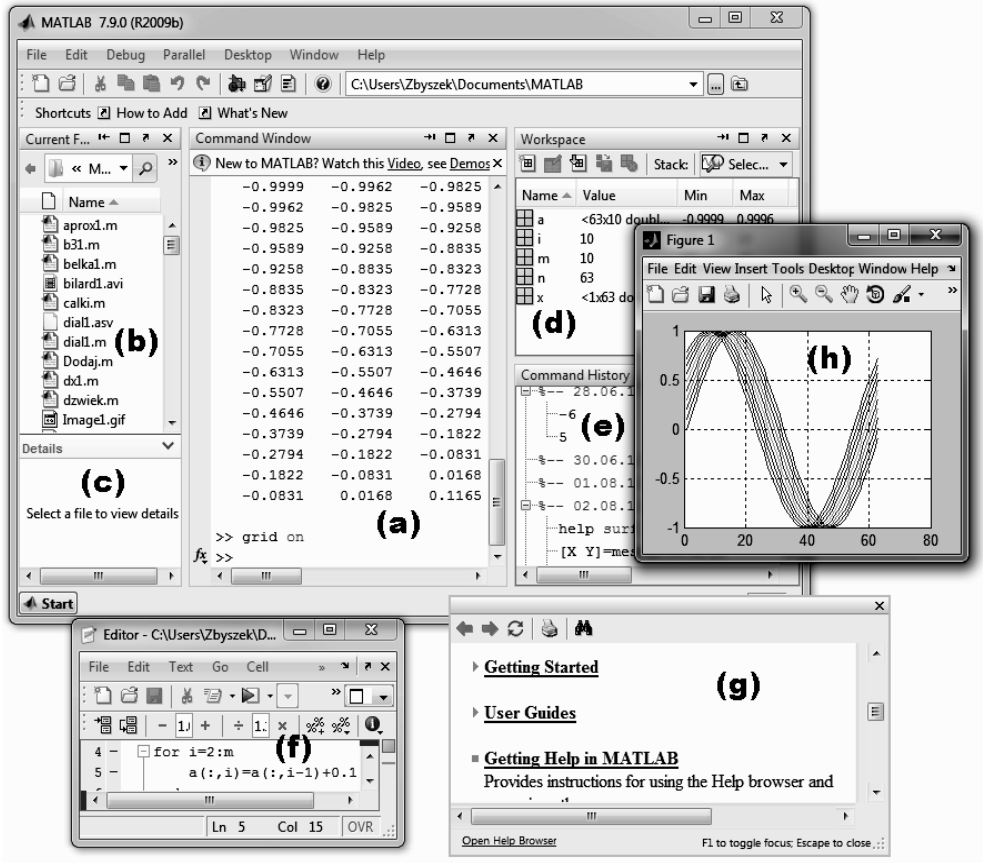
a także:

f) okno edytora do pisania i poprawiania programów (*Editor*),
g) okno pomocy (*Help*) – wywoływane z menu lub klawiszem F1,
h) okno graficzne (*Figure*) – takich okien może być wiele.

Najważniejsze jest **okno komend (a)**. Widać w nim znak gotowości do przyjmowania poleceń: **>>**. FreeMat ma nieco inny znak gotowości: **-->**.

Dla uniknięcia nadmiaru okien i gubienia się w nich – zalecam początkującym aby **w oknie głównym pozostawić tylko okno komend** wybierając z menu:

Desktop – Desktop Layout – Command Wndow Only.



Rys. 8.3. Okna MATLAB-a

W oknie komend można wykonywać obliczenia kalkulatorowe oraz wpisywać polecenia.

W pierwszych spotkaniach z MATLABem wypróbujemy najpierw jego najprostsze możliwości w zakresie działań podobnych jak na kalkulatorze.

Przykładowo, wpisanie 2+3 i zakończenie linii klawiszem [Enter] da od razu wynik:

```
>> 2+3
ans = 5
```

Nie wstawiliśmy wyniku do żadnej zmiennej dlatego MATLAB użył zmiennej „ans" (nazwa pochodzi od angielskiego: ***answer*** czyli odpowiedź). Prawie zawsze jednak będziemy wyniki obliczeń zapamiętywać w tworzonych przez nas **zmiennych**. Zmienne powstają w momencie pierwszego przypisania im wartości i nie muszą być deklarowane, jak w języku Pascal albo C. Na przykład zmienna liczbowa „a" o wartości 70 może powstać gdy MATLAB wykona instrukcję przypisania „a = 43+3*3^2":

```
>> a = 43+3*3^2
a = 70
```

Jak widać, znakiem mnożenia jest gwiazdka [*] a kolejność wykonywania działań była taka jak w wyrażeniu 43 + (3*(3^2)), to znaczy: jeśli nie użyjemy nawiasów to najpierw wykonywane są potęgowania, potem mnożenia i dzielenia a następnie dodawania i odejmowania.

Sprawdźmy teraz czy MATLAB wie co to sinus i czy poprawnie obliczy **sinus 30 stopni** (a wiemy, że powinno to być **0,5**, bo pamiętamy jaki jest stosunek najkrótszego do najdłuższego boku w ekierce nierównoramiennej). Jak zapisać sinus 30 stopni? – próbujmy najpierw „strzelać w ciemno".

np.: *b=sin(30)* - okaże się, że wynik jest błędny! (b=-0.9880) a inne zapisy takie jak: *c=sin30, d=SIN(30), e=sinus(30)* – nie dadzą w ogóle rezultatu tylko komunikat o błędnej funkcji. Żaden z tych zapisów **nie** jest poprawny bo:

(1) **argumenty każdej funkcji muszą być w nawiasach**,

(2) **argumenty funkcji trygonometrycznych muszą być w radianach**, a nie stopniach

(3) **nazwy funkcji standardowych** (z bibliotek MATLAB-a) **piszemy małymi literami.**

A więc poprawnie możemy napisać tak:

```
>> x = sin(pi/6)
x = 0.5
>>
```

Komenda zakończona średnikiem wykona się lecz nie będzie wyświetlony jej wynik na przykład:

```
>> y = sin(pi/2);
>>
```

Jednak wartość wstawiona do każdej zmiennej (mówi się także: „przypisana zmiennej") będzie pamiętana tak długo dopóki nie zastąpi jej nowa wartość wstawiona w późniejszym momencie do tej samej zmiennej (identyfikowanej nazwą).

Wpisanie samej nazwy zmiennej wyświetli jej aktualną wartość:

```
>> x
x = 0.5
```

```
>> y
y = 1

>> a
a = 70
```

Jak widać – zwyczajem anglosaskim – **część ułamkowa liczby poprzedzana jest kropką a nie przecinkiem**

Zmienne definiujemy po to aby używać je w wyrażeniach arytmetycznych a działania będą wykonywane zawsze na aktualnych wartościach tych zmiennych np.:

```
>> (a+y)*x
ans =
   35.5
```

Jeśli chcemy sprawdzić **jakie zmienne są** przechowywane aktualnie **w pamięci** MATLAB-a i jakie są ich typy oraz rozmiary, to możemy użyć w MATLAB-ie komendy **whos** a w FreeMat komendy **who**:

```
-->who
Variable Name      Type          Size
    a             int32         [1 1]
    ans           int32         [1 1]
    b             double        [1 1]
    x             double        [1 1]
    y             double        [1 1]
```

Bardzo pożyteczna jest możliwość przywoływania - przy pomocy klawisza "strzałki w górę" [↑] - poprzednio wpisywanych komend (tzw. „historii"), z możliwością ich poprawiania. Można też wyszukiwać poprzednio uruchamiane komendy w oknie "History" i albo je uruchamiać (podwójnym kliknięciem) albo kopiować do okna komend lub edytora, modyfikować i uruchamiać.

Okno komend można wyczyścić komendą **clc**, natomiast pamięć zmiennych wymazuje komenda **clear**. Zmienne zostaną skasowane także po zamknięciu MATLAB-a, chyba że zapiszemy je do pliku dyskowego, ale o tym później.

Na razie widać, że już na tym elementarnym poziomie potrzeba nam więcej informacji o obwiązujących w MATLAB-ie **regułach**, które dotyczą wszystkiego co wpisujemy w oknie komend czy edytorze MATLAB-a, począwszy od pojedynczych znaków i liczb, poprzez dopuszczalne typy poleceń i szczegóły ich budowy aż do kompletnych programów.

8.1.3. TRYBY UŻYTKOWANIA I STYLE PROGRAMOWANIA

Można rozróżnić cztery tryby wykorzystywania MATLAB-a:

1) **Tryb bezpośredni**, czyli **wpisywanie wyrażeń i poleceń w oknie komend** IDE MATLAB-a. W trybie tym pojedyncza sesja polega na wpisaniu w oknie komend linii zawierającej jedno lub kilka poleceń (komend), oddzielonych przecinkami lub średnikami. Zakończenie tej linii klawiszem ENTER spowoduje natychmiastowe wykonanie zawartych w niej komend lub wyświetlenie komunikatu o błędzie.

Średnik [;] umieszczony po poleceniu zapobiega wyświetleniu wyniku tego polecenia. Polecenie może być **wyrażeniem** (p.8.2.), którego wartość MATLAB ma wyznaczyć lub **instrukcją** o ściśle określonej budowie (p.8.7.).

Tryb bezpośredni jest używany raczej jako pomocniczy - głównie przy uczeniu się i testowaniu działania poszczególnych funkcji lub wartości zmiennych.

2) **Tryb pośredni, programowy** – jest trybem najczęściej wykorzystywanym, polega na pisaniu programów **w edytorze** a następnie ich uruchamianiu. Programista ma między innymi do dyspozycji: **strukturalne instrukcje sterujące**, bogate zbiory **funkcji** MATLAB-a (tak zwane **toolbox**'y), możliwość definiowania własnych funkcji oraz generowania wielu różnych typów **wykresów**.

   Edytor MATLAB-a można otworzyć z menu File-New lub przez kliknięcie ikony w postaci pustej kartki. Po wpisaniu programu w edytorze należy zapisać go do pliku. Nazwy plików podlegają takim samym regułom jak nazwy zmiennych (pierwszym znakiem musi być litera). Standardowo plik otrzyma rozszerzenie nazwy ".m" i będzie zapisany w folderze ustawionym jako bieżący. Program uruchamiamy albo z edytora (RUN) albo z okna komend przez wpisanie nazwy pliku bez rozszerzenia nazwy (.m). Programy można też pisać w innym prostym edytorze tekstowym (jak np.: "Notatnik" w Ms Windows).

3) **Tryb graficzny**. MATLAB posiada też środki do opracowywania programów z **graficznym interfejsem użytkownika** (***GUI*** – *Graphic User Interface*) i wykorzystujących obiekty stanowiące elementy okien dialogowych znane z Ms Windows.

   Programy mogą być całkowicie lub częściowo tworzone w edytorze jednak częściowo mogą być generowane w sposób półautomatyczny z wykorzystaniem narzędzia typu ***RAD*** (*Rapid Application Development*) nazywającego się ***GUIDE***. Polega to na wstawianiu elementów dialogu (suwaków, pól edycyjnych i t.p.) na formatki graficzne (*figure*) i modyfikowaniu ich cech, z równoczesnym automatycznym generowaniem odpowiednich poleceń. Ten tryb programowania oparty jest na **programowaniu obiektowym**, w którym można korzystać z szeregu gotowych klas obiektów (m.in. elementów okien graficznych) jak i definiować własne klasy obiektów.

4) **Tryb symulacyjny**. Rozszerzeniem MATLAB-a jest **SIMULINK** - pozwalający zestawiać z różnorodnych bloków funkcjonalnych modele "analogowe" układów dynamicznych mające postać schematów funkcjonalnych (blokowych) a następnie zadawać przebiegi wielkości wejściowych i obserwować przebiegi wielkości wyjściowych. SIMULINK pozwala więc badać modele w sposób zbliżony jak na maszynach analogowych. Jest on wprawdzie dodatkowym elementem MATLAB-a, jednak pracę z Simulinkiem można uznać za tryb czwarty.

W Simulinku istnieje biblioteka modułów mechanicznych **SimMechanics** pozwalająca budować funkcjonujące modele układów mechanicznych. Dokumentacja SimMechanics jest na stronie: *http://www.mathworks.com/help/toolbox/physmod/mech/*, natomiast filmy demonstrujące tworzenie i testowanie modeli są na stronach:
http://www.mathworks.com/products/simmechanics/demos.html

8.2. PODSTAWOWE ELEMENTY JĘZYKA MATLAB

Niezależnie od trybu w jakim używamy MATLAB-a – musimy znać jego język programowania - noszący tą samą nazwę co cały pakiet. Tak jak w każdym języku programowania – w MATLAB-ie treść programu tworzą głównie:

- **instrukcje** (zwane czasem komendami) – realizujące m.in. wprowadzanie i wyprowadzanie danych oraz rozgałęzienia i pętle programowe,
- **definicje podprogramów** (zwanych m-plikami skryptowymi) i **funkcji** użytkownika,
- **wywołania funkcji** należących do MATLAB-a lub jego toolbox'ów i funkcji użytkownika **oraz podprogramów**

Podstawowymi składnikami instrukcji są: **słowa kluczowe** (m.in. opisane w p.8.2.1.) oraz **wyrażenia** o wartościach liczbowych, logicznych, lub tekstowych. Składnikami wyrażeń są: **stałe**, **zmienne** (proste i złożone), **wywołania funkcji**, oraz **operatory** i **nawiasy okrągłe**. Nie trzeba w MATLAB-ie deklarować typów zmiennych.

Reguły budowy poprawnych elementów i konstrukcji języka oraz ich rola, są ściśle określone, poczynając od postaci stałych i zmiennych, a kończąc na strukturze wyrażeń, poleceń i programów.

8.2.1. SŁOWA KLUCZOWE

Podstawowe słowa kluczowe używane w instrukcjach MATLAB-a są następujące:

break - przerwij	***end*** - koniec	***if*** - jeśli
case - przypadek	***for*** - dla	***return*** - wróć
catch - przechwyć	***function*** - funkcja	***switch*** - przełącz
continue - kontynuuj	***global*** - globalne	***try*** - próbuj
else - w przeciwnym przypadku	***otherwise*** - w pozostałych przypadkach	***while*** - podczas

8.2.2. REGUŁY TWORZENIA NAZW

Użytkownik pisząc programy musi nadawać **identyfikatory** czyli **nazwy** definiowanym przez siebie zmiennym, tablicom, funkcjom i innym elementom programu (np.: obiektom oraz polom tablic strukturalnych), a w końcu także plikom dyskowym do których zapisuje programy, podprogramy i funkcje.

Każda nazwa powinna w czytelny sposób kojarzyć się z **rolą** jaką będzie pełnił reprezentowany przez nią element. Czytelniejsze są nazwy wieloliterowe ale nie zbyt długie. Jeśli na przykład zmienna ma reprezentować „naprężenie maksymalne" to nazwa „Napr_max" jest czytelniejsza niż „nm" i wygodniejsza niż „naprezenie_maksymalne".

MATLAB rozróżnia duże i małe litery.

Polecenia standardowe muszą być pisane **małymi literami** a dla wymyślanych przez nas identyfikatorów wskazane jest używanie tak małych jak i dużych liter.

Budowa nazwy musi spełniać wymagania obowiązujące w danym języku programowania. W MATLAB-ie tworzone nazwy zmiennych, funkcji, obiektów, pól oraz plików (z rozszerzeniami: .m, .mex, .mdl) muszą spełniać następujące wymagania:

1) nazwa może składać się jedynie: z **liter alfabetu angielskiego** oraz **cyfr** i **znaku podkreślnika** `[_]`,
2) **pierwszym znakiem** nazwy musi być **litera**,
3) długość nazwy nie powinna przekraczać liczby znaków określonej przez stałą ***namelengthmax***,
4) nazwy ustalane przez użytkownika nie mogą być identyczne z nazwami funkcji MATLAB-a gdyż wynikają z tego błędne operacje, aby spełnić ten warunek wystarczy w nazwie zastosować chociaż jedną dużą literę, gdyż nazwy systemowe pisane są zawsze małymi literami.

Pamiętajmy, że reguły te dotyczą także **nazw plików** do jakich będziemy zapisywać nasze programy, funkcje i podprogramy, ponieważ nazwa programu (podprogramu, funkcji) jest zarazem nazwą pliku dyskowego do którego zapisujemy ten program (podprogram, funkcję). Przykłady poprawnych i niepoprawnych nazw pokazuje Tabela 8.1.

Tabela 8.1. Poprawne i nieprawidłowe nazwy zmienych

Poprawne	Nieprawidłowe
Moc2	**2**Moc
MomentGn1	Moment**-**1
moment_gn_1	moment_gn**ą**cy1
SILA_Px1	SI**Ł**A_Px1

Jak widać - **nie wolno używać polskich liter** np.: ą, ć, ę, ł, ń, ... ani spacji ani żadnych znaków nie będących literami angielskimi, cyframi oraz podkreślnikiem. Prawidłową nazwą może być np.: *Moment_sily9.*

W sprawdzeniu jakie mamy nazwy i typy zmiennych mogą nam dopomóc niektóre komendy dotyczące pamięci zmiennych, a mianowicie:

1) **who** - wyświetla wykaz zmiennych (a w FreeMat także typy),
2) **whos** - wyświetla dokładne informacje o zmiennych,
3) **save** - zapisuje pamięć zmiennych do pliku matlab.mat,
4) **load** - wczytuje pamięć zmiennych z pliku matlab.mat,
5) **clear** *zmienna* - usuwa *zmienną* z pamięci,
6) **clear** - usuwa wszystkie zmienne z pamięci,
7) **iskeyword(S)** – sprawdza czy łańcuch znaków S jest słowem kluczowym,
8) **isvarname(S)** – sprawdza czy łańcuch S może być poprawną nazwą zmiennej.

8.2.3. TYPY WARTOŚCI

Każda stała i zmienna użyta w programie musi mieć zarezerwowany obszar w pamięci operacyjnej komputera, odpowiedni dla swego typu.

Typ stałej określony jest bezpośrednio przez postać jej zapisu – więc postać ta musi być zgodna z regułami języka, na przykład „trzy i pół" to nie 3,5 ale **3.5**. Zmienne są natomiast widoczne w programie tylko przez swoje nazwy i dlatego większość języków programowania (w szczególności Pascal oraz C) wymaga umieszczenia na początku programu deklaracji określających typ każdej zmiennej.

Inaczej jest w MATLAB-ie. **MATLAB** (podobnie jak BASIC) **nie wymaga** od użytkownika **deklarowania typów zmiennych** ponieważ **określa typy samoczynnie na podstawie wartości przypisywanych zmiennym**. W większości prostych programów użytkownik może prawie wcale nie interesować się typami zmiennych gdyż MATLAB sam daje sobie z nimi radę. Trzeba jednak znać podstawowe typy danych, gdyż ich nieznajomość może przysporzyć nam kłopotów.

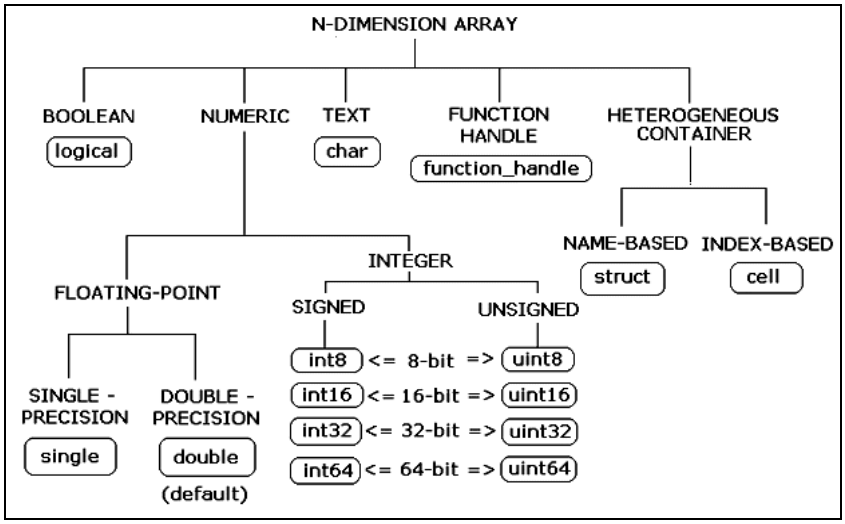
Wykaz standardowych typów wartości dostępnych w MATLAB-ie pokazuje Rys. 8.4, pochodzący z dostępnego w Internecie systemu pomocy MATLAB-a [59]. Oprócz tego użytkownik może definiować własne typy (zwane klasami), co jest wykorzystywane głównie przy programowaniu obiektowym.

Odpowiedniki najpopularniejszych typów występujących we wszystkich językach programowania to:

- grupa typów liczbowych - ***numeric***,
- typ znakowy (tekstowy) - ***char*** – znaki i łańcuchy znaków (*strings of characters*),
- typ logiczny – ***logical.***

Specyficznym dla MATLAB-a jest fakt, że wszystkie te typy domyślnie dotyczą tablic. Specyficzne dla MATLAB-a są także pozostałe trzy typy:

- ***function_handle*** – uchwyty funkcji,
- ***struct*** – struktury pozwalające definiować rekordy baz danych
- ***cell*** - tablice komórkowe, w których każdy element może być innego typu.



Rys. 8.4. Typy danych w MATLAB-ie (źródło: [59])

8.2.4. WYRAŻENIA

W ramach każdego typu wartości mogą występować **wyrażenia** zawierające **stałe, zmienne i funkcje** powiązane **operatorami** i pogrupowane przy pomocy **nawiasów**.

Wyrażenia arytmetyczne pozwalają dokonywać obliczenia na określonych wartościach liczbowych swych składników i uzyskiwać wynikowe wartości liczbowe.

Wyrażenia logiczne są potrzebne w instrukcjach warunkowych pozwalających programowi wybrać jeden z kilku wariantów dalszych działań.

Wyrażenia tekstowe pozwalają operować na tekstach, na przykład: wyszukiwać, wycinać oraz sklejać fragmenty tekstów.

W kolejnych podrozdziałach omówiono elementy i wyrażenia poszczególnych typów.

Najpierw zajęto się elementami i budową wyrażeń arytmetycznych (p.8.3.), a następnie wyrażeń logicznych (p.8.4.) oraz tekstowych (p.8.5.).

8.3. WARTOŚCI LICZBOWE, MACIERZE, WYRAŻENIA ARYTMETYCZNE

Omawianie budowy języka MATLAB zaczniemy od najprostszych jego „cegiełek” – a mianowicie składników wyrażeń arytmetycznych. Wyrażenia – jak to już pokazano na wstępnych przykładach – mogą występować w roli samodzielnych poleceń, ale najczęściej są elementami instrukcji języka.

Wyrażenia arytmetyczne mogą zawierać:

- stałe liczbowe,
- zmienne (nazwy zmiennych),
- operatory działań,
- nawiasy,
- funkcje.

Poszczególnym składnikom wyrażeń poświęcono kolejne podrozdziały.

Inaczej niż w innych językach, **w MATLAB-ie każda zmienna jest traktowana jak macierz**, a więc także działania i funkcje z założenia dotyczą macierzy co w pewnym stopniu sygnalizuje już nazwa języka *MATrix LABoratory* – laboratorium macierzowe. MATLAB ma rozbudowane i specyficzne dla siebie operatory i środki do operowania na macierzach. Oczywiście macierz może w szczególności mieć rozmiary 1x1 czyli być skalarem (jej wartość jest wtedy pojedynczą liczbą).

W tym rozdziale – dla ułatwienia i utrzymania stylu programowania podobnego jak w językach BASIC czy Pascal - ograniczymy się właśnie do **działań na skalarach** oraz **elementach wektorów i macierzy**, realizowanych przy pomocy pętli, wspominając jedynie o operacjach na całych macierzach.

Omówienie specyficznych „matlabowych" operacji macierzowych nastąpi w rozdziale następnym.

8.3.1. TYPY WARTOŚCI LICZBOWYCH

Podstawowym, domyślnym typem stałych i zmiennych o wartościach **liczbowych**, jest w MATLAB-ie tablica (***array***) typu ***double***, której elementami są liczby rzeczywiste „podwójnej precyzji" zapisywane przy pomocy 64 bitów. Wartości **skalarne**, czyli pojedyncze liczby, są traktowane w MATLAB-ie jako tablice o jednym wierszu i jednej kolumnie czyli wymiarach 1x1.

Zakres liczb typu ***double*** sięga od -***realmax*** do +***realmax*** (czyli ±1.7977 e+308)

Najmniejszy ułamek ma wartość: ***realmin*** (±2.2251 e-308)

Precyzja: Liczby są pamiętane z dokładnością 15 cyfr znaczących.

Wartość typu ***double*** zajmuje 8 bajtów (64 bity) w pamięci

Do dyspozycji są także typy jedno, dwu i cztero bajtowe: **int8, int16, int32 -** dla zmiennych o wartościach całkowitych ze znakiem oraz **uint8, uint16, uint32 -** dla zmiennych o wartościach całkowitych bez znaku. Na zmiennych tych typów nie można wykonywać działań arytmetycznych w starszych wersjach MATLAB-a (np. wersja 6). W nowszych wersjach (np. 7.9) jest to możliwe, ale wyniki są zaokrąglane i ograniczane do zakresu właściwego dla danego typu np.:

```
>> a=int8(100), b=int8(102)
a =
  100
b =
  102

>> a+b
ans =
  127

>> a/b
ans =
    1

>> 100/3.5
ans =
   28.5714

>> a/3.5
ans =
   29
```

Tak więc typy te są używane raczej do przechowywania pewnych informacji bo zajmują kilkakrotnie mniej objętości pamięci.

8.3.2. ZAPIS LICZB

Poprawnie zapisana w MATLAB-ie **liczba rzeczywista** czyli **stała liczbowa typu *double*** może zawierać:

- początkowy znak plus (na ogół pomijany) lub minus,
- **kropkę dziesiętną (NIE przecinek!)** poprzedzającą część ułamkową np.: -97.6397 ,
- może być stosowana tzw. **notacja naukowa** w której **e oznacza "dziesięć do potęgi ..."** np.: **-1.60210e-23** oznacza: -1.60210 razy 10 do potęgi –23.

W zapisie liczb zespolonych stosuje się symbole jednostki urojonej "i" albo „j" zawsze poprzedzone liczbą np.: **1i; 5.7- 3.149j; 2+3e5i**.

8.3.3. ZMIENNE. IDENTYFIKACJA I KONWERSJA TYPÓW

Zmienne, które będą przechowywały pojedyncze wartości lub tablice liczbowe, nie są przed użyciem deklarowane a ich nazwy także nie sygnalizują typu (jak może być w BASIC-u). MATLAB **określa typ zmiennej na podstawie wprowadzonej wartości**. Zmienne, którym przypisywane są liczby lub tablice liczb uzyskują domyślnie typ ***double*** co możemy sprawdzić komendą ***whos***:

```
>> a=11111111111111111111111111

a =
  1.1111e+025

>> a=127551327e12
a =
  1.2755e+020

>> c=[1 2 3]
c =
     1     2     3

>> whos
  Name       Size             Bytes  Class     Attributes

  a          1x1                  8  double
  b          1x1                  8  double
  c          1x3                 24  double
```

Do sprawdzania jakiego typu jest zmienna lub jaki rodzaj wartości przechowuje, służy obszerna grupa kilkudziesięciu (48) funkcji zaczynających się od „is..." (po polsku: „czy to jest...") np.: ***isnumeric, isreal, isequal, isstr, iskeyword, isinteger, isspace, ...***

W MATLAB-ie **typ zmiennej może być zmieniony w trakcie wykonywania programu** - inaczej niż w innych językach tradycyjnych programowania gdzie jest na ogół niezmienny. Jednym ze sposobów, jest przypisanie zmiennej wartości innego typu niż dotychczasowa.

Sprawdzimy to – poleceniem ***whos*** - na przykładzie, podstawiając najpierw zero a potem wartość tekstową (ciąg znaków) do zmiennej:

```
>>x=0; y=0.0;
>>whos
Name       Size   Bytes  Class
x          1x1    8      double array
y          1x1    8      double array
>> y='ala ma kota';
>>whos
Name       Size   Bytes  Class
x          1x1    8      double array
y          1x11   22     char array
```

Nieco inaczej jest w systemie FreeMat, ale także możliwe jest **przedefiniowanie typu zmiennej** przez wstawienie wartości innego niż poprzednio typu:

```
-->x=0; y=0.0;
-->who
Variable Name     Type                Size
   ans            double        [0 0]
    x             int32         [1 1]
    y             double        [1 1]
--> x=3.1;
--> y='ala ma kota';
--> who
Variable Name     Type                Size
   ans            double        [0 0]
    x             double        [1 1]
    y             string        [1 11]
```

Zmiana typu zmiennych lub stałych może być dokonywania także przy pomocy odpowiednich funkcji, na przykład:

- **double**(x) - przekształca wartość x na typ double,
- **num2str**(x) - przekształca wartość x na ciąg znaków,
- **str2num(tekst)** - przekształca tekst (ciąg cyfr) na liczbę,
- **uint8**(x)**, uint16**(x)**, uint32**(x) - konwertują do całkowitych bez znaku,
- **int8**(x)**, int16**(x)**, int32**(x) - konwertują do całkowitych ze znakiem.

Jak widać, nazwy tych funkcji albo są takie jak nazwy typów albo mają w środku „2" czytane po angielsku tak samo jak „to" czyli polskie „na".

Przy pomocy tych funkcji można nie tylko zmieniać typ stałych, ale także przedefiniowywać typy zmiennych. Na przykład:

```
>> a=3.75
a =
    3.7500
```

```
>> a=int8(a)
a =
     4

>> whos
  Name        Size              Bytes  Class     Attributes
  a           1x1                   1  int8
```

8.3.4. FORMATY WYŚWIETLANYCH LICZB

Domyślnie wyniki obliczeń wyświetlane są na ekranie z dokładnością czterech cyfr po kropce dziesiętnej czyli w formacie ***short*** (krótkim), jednak dokładność obliczeń jest większa.

Postać prezentowania liczb na ekranie można zmieniać przy pomocy dyrektywy:

format *parametr*

gdzie *parametr* jest jednym ze słów: **short, short e, long, ...**

Wpisz komendę ***help format*** dla uzyskania dokładniejszych informacji o formacie.

Przykłady formatów liczby:

a) Liczba w domyślnym formacie **short:**

```
» 2.5
ans =
     2.5000
```

b) Po zmianie na format **short e:**

```
» format short e
» 2.5
ans =
2.5000e+000
```

c) Po zmianie na format **long:**

```
» format long
» 2.5
ans =
     2.50000000000000
```

8.3.5. MACIERZE

Zmiennej można przypisać dwuwymiarową tablicę liczb czyli macierz. Postać zapisu macierzy ma spełniać następujące reguły:

- całość macierzy musi być ujęta w nawiasy prostokątne,
- elementy wierszy należy oddzielać przecinkami lub spacjami,
- wiersze oddzielane są od siebie średnikami.

A oto przykład jak przypisać zmiennej dowolną macierz:

```
>> Ug15 = [12.3, 11.4, 14.75;   20.55, 27.31, 31.44]
Ug15 =
   12.3000   11.4000   14.7500
   20.5500   27.3100   31.4400
```

Poszczególne elementy macierzy można wykorzystywać podając po nazwie macierzy indeksy elementu czyli numery wiersza i kolumny, oddzielone przecinkiem i umieszczone w nawiasach okrągłych:

```
>> Ug15(2,1)
ans =
   20.5500
```

```
>> Ug15(2,3)
ans =
   31.4400
```

Oprócz dwuwymiarowych można stosować tablice jednowymiarowe czyli wektory oraz wielowymiarowe i dokonywać na nich różnorodnych operacji - o czym obszerniej napisano w rozdziale 9. Oto przykład definiowania wektora wierszowego ***W***:

```
>> W=[1.73, 4.06, 6.23, 8.13]
W =
    1.7300    4.0600    6.2300    8.1300
```

Bardzo często potrzebne jest przypisanie zmiennej szczególnego wektora a mianowicie ciągu typu „postęp arytmetyczny". Wystarczy wówczas podać wartość pierwszego elementu, przyrostu i ostatniego elementu, oddzielając je od siebie dwukropkami, jak poniżej:

```
>> V=3:0.5:5
V =
    3.0000    3.5000    4.0000    4.5000    5.0000
```

8.3.6. OPERATORY DZIAŁAŃ ARYTMETYCZNYCH NA SKALARACH

W MATLAB-ie, oprócz operatorów arytmetycznych o postaci znanej z innych języków programowania czy programów matematycznych, jest dodatkowy komplet **operatorów poprzedzanych kropką**.

Co ciekawsze, ponieważ MATLAB-a zaprojektowano głównie z myślą o macierzach, więc domyślnie, operatory arytmetyczne (`+, -, *, /, ^`) są operatorami działań **macierzowych**, natomiast **operatory poprzedzone kropką** (`.+, .-, .*, ./, .^`) – zwane czasem tablicowymi - **dotyczą** odpowiadających sobie **par elementów** dwu **macierzy o identycznych wymiarach**.

W zakresie działań arytmetycznych na skalarach lub pojedynczych elementach macierzy jest jednak obojętne czy zastosujemy operator z kropką czy bez.

Dlatego – zgodnie z założeniami tego rozdziału - na razie pozostaniemy przy tradycyjnych operatorach dla których, w przypadku nie używania nawiasów, **działania arytmetyczne będą wykonywane w następującej kolejności**:

1) potęgowanie (`^`),
2) zmiana znaku (`-`),
3) mnożenie i dzielenie (`*, /`),
4) dodawanie i odejmowanie (`+, -`).

Sposób działania operatorów **na macierzach** omówiono w rozdziale 9 dotyczącym macierzy i tam też znajduje się tabela objaśniająca wszystkie operatory arytmetyczne.

8.3.7. PRZYKŁADY WYRAŻEŃ ARYTMETYCZNYCH

Ponieważ wyrażenie arytmetyczne (zwane czasem algebraicznym lub matematycznym) może zawierać dowolną liczbę opisanych wyżej składników - stałych, zmiennych, operatorów, funkcji i par nawiasów okrągłych - to w szczególności wyrażeniem jest także pojedyncza liczba oraz pojedyncza zmienna.

Jak już wspomniano – wyrażenia są najczęściej elementami instrukcji i argumentami funkcji MATLAB-a. Jednak wyrażenie może też być użyte samodzielnie jako osobne polecenie w oknie komend lub w linii programu i jeśli nie umieścimy po nim znaku średnika, a wszystkie zmienne w tym wyrażeniu mają już określone wartości, to MATLAB obliczy i wyświetli w oknie komend wartość tego wyrażenia.

Przykłady wyrażeń arytmetycznych:

```
p12_1, 1753.97, (2*x-1)^3,
sin(a)/sqrt(cos(alfa1+a)^2+cos(a)^2)
```

Wykaz funkcji elementarnych można uzyskać wpisując w oknie komend: **help elfun**. Niektóre z tych funkcji objaśnia Tabela 8.2.

8.3.8. WAŻNIEJSZE FUNKCJE ELEMENTARNE MATLAB-A

Tabela 8.2. Ważniejsze funkcje elementarne MATLAB-a

Nazwa funkcji	Objaśnienie
sqrt(x)	pierwiastek z x
abs(x)	wartość bezwzględna x
exp(x)	e do x
log(x)	logarytm naturalny z x
log2(x)	logarytm o podstawie 2
log10(x)	logarytm o podst 10
sign(x)	znak x
mod(x)	reszta z dzielenia
sin(x), cos(x), tan(x), cot(x)	fun. trygonometryczne – argumenty w **radianach**
sinh(x), cosh(x), tanh(x), coth(x)	funkcje hiperboliczne
asin(x), acos(x), atan(x), acot(x)	f. odwrotne do trygonometrycznych
round(x)	zaokrągla do najbliższej całkowitej
ceil(x)	zaokrąglenie w górę (dosłownie: sufit)
fix(x)	zaokrągla w stronę zera
floor(x)	zaokrągla w dół (dosłownie: podłoga)
imag(x)	część urojona liczby zespolonej
real(x)	część rzeczywista liczby zespolonej
gcd(x)	największy wspólny podzielnik
lcm(x)	najmniejsza wspólna wielokrotność

8.3.9. ĆWICZENIA – OBLICZENIA W TRYBIE BEZPOŚREDNIM

Z74. „Notacja naukowa" zapisu liczb

a) oblicz: $7{,}7^{310}$

b) Zastosuj tzw. „notację naukową" zapisu liczb w obliczaniu wyrażenia:

$$\frac{1{,}78\cdot 10^{14} + 9{,}76^{11}}{128000000} + 0{,}00005 =$$

Z75. Obliczenia kalkulatorowe.

Sprawdź w MATLAB-ie poprawność następujących obliczeń:

A) $\dfrac{\sin(6)+1}{3+e^{1,7}} = 0{,}085$

B) $\dfrac{2+\ln(5)}{\sqrt{\pi+2}} = 1{,}5918$

C) $\dfrac{2\cdot e^{7}-9}{1+\sqrt{3}} = 799{,}497$

D) $\left|e^{\pi}-25{,}1\right| = 1{,}9593$

E) $\dfrac{\sqrt{e}+1{,}2}{\ln(\pi-1)} = 3{,}7407$

F) $\dfrac{1}{\sqrt{3}} + \sin\left(\dfrac{\pi}{4}\right) = 1{,}2845$

G) $\dfrac{e^{\sin(1,1)}}{1+\sqrt{\pi-1}} = 0{,}9897$

H) $\sqrt{\dfrac{1+\sqrt{3}}{\ln(4{,}5)}} = 1{,}3477$

I) $\dfrac{\sin(1{,}23)+1}{2\cdot\cos\left(1+\dfrac{\pi}{7}\right)} = 7{,}981$

J) $\dfrac{1{,}35^{2,67}}{\sqrt{1.2+\pi}} = 1{,}0695$

K) $\sin\left(\dfrac{\pi}{\sqrt{3}}\right)^{1,3} = 0{,}962$

L) $e^{\sin\left(\frac{\sqrt{2}}{1,2}\right)} = 2{,}5194$

Z76. Stopnie i radiany, funkcje trygonometryczne

(*Radian to miara łukowa kąta: stosunek długości łuku do promienia, 180° to π radianów*)

Sprawdź w MATLAB-ie podane niżej wzory (wynikające m.in. z proporcji boków ekierek):

a) $\sin(30°) = \cos(60°) = 0{,}5$ b) $\cos(30°) = \sin(60°) = \dfrac{\sqrt{3}}{2}$ c) $\cos(45°) = \sin(45°) = \dfrac{1}{\sqrt{2}}$

Z77. Obliczenia z użyciem zmiennych.

(Do obliczania w programie wyrażeń zawsze brane są aktualne czyli ostatnio wyznaczone wartości zmiennych)

Oblicz wartość zmiennej: $y = \dfrac{2}{\sqrt{|x|-4}} - \dfrac{6}{x^2-5x}$

dla x = 8,167 oraz x = –8,167 (Powinno być: 0,748 i 0,924)

(Warto wykorzystać możliwość przywoływania w oknie komend, poprzednio wpisanych już poleceń, przy pomocy klawiszy strzałek pionowych)

8.4. WARTOŚCI I WYRAŻENIA LOGICZNE

Wartości **0** oraz **1** pełnią także rolę **stałych logicznych** ***false*** i ***true*** czyli „fałsz" oraz „prawda", a więc wszystkie prawdziwe relacje i wyrażenia logiczne mają wartość liczbową 1 a fałszywe mają wartość zero. Na przykład:

```
>> 2>3
ans = 0
>> 3>2
ans = 1
```

Także na odwrót – zmienne i wyrażenia arytmetyczne o wartości różnej od zera mogą pełnić rolę wyrażeń logicznych o wartości „prawda" a zerowe są traktowane jako wyrażenia logiczne fałszywe.

Wyrażenia logiczne umożliwiają sprawdzanie prawdziwości określonych zależności dla konkretnych danych liczbowych. Najprostszymi wyrażeniami logicznymi są relacje. Relacja to dwa wyrażenia arytmetyczne połączone operatorem relacji. Używane w MATLAB-ie operatory relacji podaje Tabela 8.3.

Tabela 8.3. Operatory relacji

Operator	Znaczenie
<	mniejsze
<=	mniejsze lub równe
>	większe
>=	większe lub równe
==	równe
~=	nierówne

Przykłady relacji:

```
x>5, b*b-4*a*c>=0, 5==x^2+y, 2*(m-3*h)^2<=sqrt(u^2+5*h),
x+a>x
```

Wartości logiczne tych relacji MATLAB może wyznaczyć dopiero gdy:

a) będą znane wartości liczbowe wszystkich występujących w nich zmiennych,

b) zostaną obliczone wartości liczbowe wyrażeń arytmetycznych.

Przetestuj w oknie komend niektóre z tych relacji, nadając wcześniej wartości występującym w nich zmiennym.

Tabela 8.4. Operatory logiczne i równoważne im funkcje

Operator	Funkcja	Znaczenie
A **&** *B*	**and**(*A*,*B*)	A i B
A **|** *B*	**or**(*A*,*B*)	A lub B
~ *A*	**not**(*A*)	nie A

Złożone wyrażenia logiczne można tworzyć przy pomocy relacji i operatorów oraz funkcji logicznych. Operatory logiczne MATLAB-a i odpowiadające im funkcje podaje Tabela 8.4, przy czym *A* i *B* reprezentują wyrażenia logiczne.

Wyrażenia logiczne znajdują zastosowanie głównie w instrukcjach warunkowych IF. Przykładem zastosowania instrukcji IF może być przypomniany poniżej fragment programu rozwiązywania równania kwadratowego:

```
if  b*b-4*a*c <0
    disp('Brak pierwiastkow rzeczywistych');
else
    x1=(-b-sqrt(b*b-4*a*c))/(2*a);
    x2=(-b+sqrt(b*b-4*a*c))/(2*a);
    disp('x1='); disp(x1); disp('x2='); disp(x2);
end
```

Inne przykłady podano przy opisywaniu instrukcji IF w p.8.7.4. .

8.5. WARTOŚCI I WYRAŻENIA TEKSTOWE

MATLAB umożliwia dokonywanie pewnych operacji na tekstach. Stała tekstowa to **łańcuch znaków** (ang.: *string of characters)* ujęty w apostrofy np.:

```
'  x1=', '  x2=',  'Ala ma kota'
```

W rzeczywistości łańcuch znaków jest tablicą znaków (ang.: ***array of characters***) a dokładniej wektorem, którego elementami są znaki drukarskie. W FreeMat typ ten określony jest jako ***string***. Długość tego wektora jest równa liczbie składowych znaków, łącznie ze spacjami. Do określenia rozmiarów macierzy (lub wektora) służy funkcja ***size(..)***. Inne wybrane funkcje do operowania na tekstach pokazuje Tabela 8.5.

Wartości tekstowe (łańcuchy znaków) można przypisywać zmiennym, np.:

```
S1='Ala ma kota';   S2='i chomika';
```

Możliwe jest **sklejanie łańcuchów** reprezentowanych przez stałe, zmienne oraz funkcje typu ***string***. Najprostszym sposobem sklejenia jest umieszczenie elementów przewidzianych do sklejenia we wspólnym nawiasie prostokątnym, przy czym należy oddzielać je od siebie przecinkami lub spacjami.

Na przykład:

```
S=[S1,'i psa',S2]
S =
Ala ma kotai psai chomika
```

Jak widać zabrakło odstępów przed spójnikami ”i”, a więc korygujemy wyrażenie tekstowe przypisane zmiennej S:

```
S=[S1,' i psa ',S2]
S =
Ala ma kota i psa i chomika
```

Sklejanie łańcuchów może być przydatne m.in. przy wykorzystywaniu funkcji ***disp***. Ponieważ funkcja ***disp*** może mieć jako argument albo wyrażenie numeryczne albo tekstowe więc w przykładzie dotyczącym rozwiązywania równania kwadratowego podano taką linię poleceń wyświetlających wyniki:

```
disp('x1='); disp(x1); disp('x2='); disp(x2);
```

Zamiast tej linii z czterema instrukcjami ***disp***, można użyć jednej instrukcji ***disp*** wyświetlającej jeden łańcuch znaków otrzymany przez sklejenie objaśnień z obliczonymi wartościami wyników x1, x2, jednak wyniki te trzeba najpierw przekonwertować z typu liczbowego do typu znakowego, przy użyciu funkcji ***num2str*****`(...)`** :

```
disp(['x1=',num2str(x1),'  x2=',num2str(x2)]);
```

Sklejanie łańcuchów znaków nazywane jest także **katenacją** lub **konkatenacją**, a zamiast nawiasów prostokątnych można do tego celu stosować funkcję ***strcat(...)*** (tabela 8.5).

Tabela 8.5. Wybrane funkcje do operowania na tekstach

Funkcja	Działanie
`z=char(n)`	zamienia wartość kodową *n* na odpowiedni znak z
`n=double('z')`	wyznacza kod znaku z
`deblank(S)`	usuwa spacje i znaki niedrukowane z końca łańcucha S
`eval(S)`	wykonuje polecenie zapisane w łańcuchu S
`ischar(x)`	sprawdza czy x jest tablicą znaków (1 oznacza tak)
`lower(S)`	zamienia w łańcuchu S duże litery na małe
`S=num2str(x,p)`	zamienia wartość wyrażenia **x** na łańcuch znaków **S**, opcjonalnie uwzględniając **p** miejsc dziesiętnych po kropce
`x=str2double(S)`	zamienia łańcuch znaków S reprezentujący liczbę na liczbę typu double
`S=strcat(S1,S2,..)`	skleja łańcuchy S1, S2, ... w jeden łańcuch S
`v=strfind(S,S1)`	podaje pozycje w łańcuchu S na jakich znalazł wystąpienia krótszego łańcucha S1
`strjust(S,j)`	wyrównuje S do: prawej (j=’right’), lewej (j=’left’) lub środka (j=’center’)
`strrep(S,S1,S2)`	w łańcuchu S zastępuje każdy podłańcuch S1 przez podłańcuch S2
`strtrim(S)`	usuwa początkowe i końcowe spacje z łańcucha S
`S=strvcat(S1, S2, ..)`	tworzy macierz w której łańcuchy S1, S2, .. są wierszami, uzupełniając krótsze łańcuchy spacjami na końcu do jednakowej długości
`upper(S)`	zamienia w łańcuchu S małe litery na duże

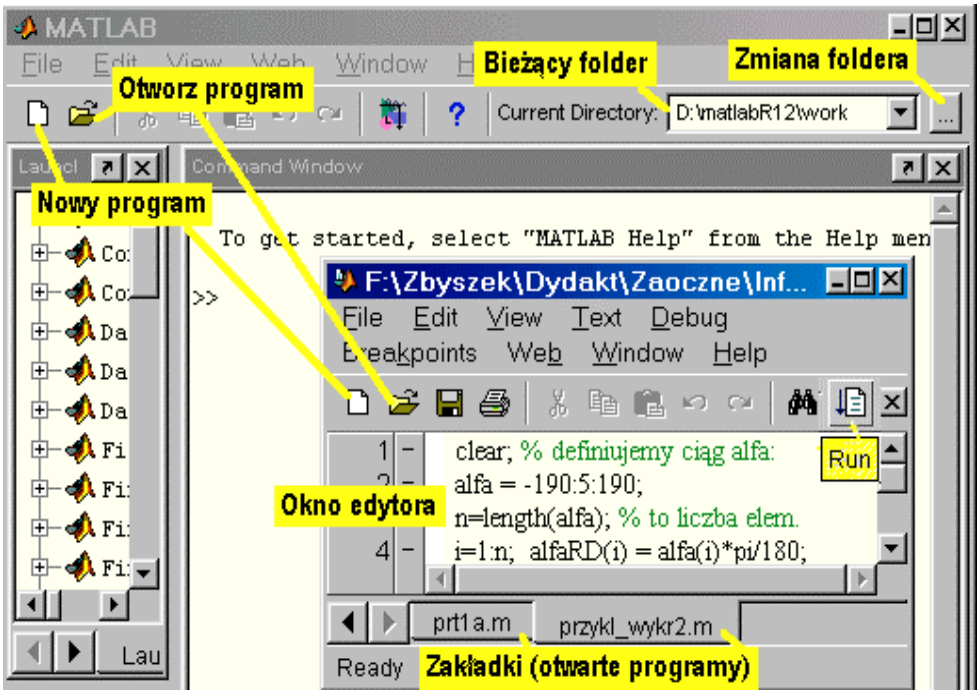
Informacje o łańcuchach znaków oraz o funkcjach do operowania na nich można wyświetlić w oknie komend poleceniami: ***help strings*** oraz ***help strfun***.

8.6. PISANIE I URUCHAMIANIE PROGRAMÓW

Jak już wspomniano, programy tworzone w MATLAB-ie, zapisywane są jako zwykłe pliki tekstowe choć ich nazwy posiadają rozszerzenie ".m". Można więc pisać te programy przy pomocy najprostszych edytorów tekstu jak „Notatnik" w Ms Windows, ale znacznie wygodniej to robić przy pomocy edytora MATLAB-a (Rys. 8.5).

Poszczególne instrukcje programu można pisać w oddzielnych liniach kończonych naciśnięciem klawisza ENTER. Można też pisać po kilka instrukcji w jednej linii oddzielając je wówczas przecinkami lub średnikami. Średnik na końcu instrukcji dodatkowo blokuje wyświetlanie jej rezultatu (tak zwanego „echa") na ekranie. Można jedną długą instrukcję rozmieścić w kilku kolejnych liniach i wtedy musimy przy pomocy trzech kropek, umieszczonych na końcu linii, zasygnalizować, że dalszy ciąg jest w następnej linii.

Po napisaniu programu trzeba zapisać go do pamięci masowej jako plik tekstowy z rozszerzeniem nazwy: ".m" (potocznie nazywany: "m-plikiem").



Rys. 8.5. Edytor MATLAB-a i bieżący folder dla programów

Istnieją **dwa rodzaje m-plików**:

- **skrypty** czyli procedury działające na zmiennych tworzonych w tzw. przestrzeni roboczej MATLAB-a,

- **funkcje** definiowane przez użytkownika w celu wielokrotnego wykorzystywania - obszerniej przedstawione w p.8.10. .

Zanim przystąpisz do pisania pierwszych programów:

- utwórz **swój folder** w którym będziesz gromadzić te programy,
- po otwarciu MATLAB-a kliknij przycisk [...] (patrz Rys. 8.5), znajdź swój folder i ustaw go jako **bieżący**,
- otwórz **okno edytora** przez wybranie z menu File-New lub wciśnięcie ikony z symbolem pustej kartki,
- aby sens instrukcji programu był zrozumiały, warto stosować mnemoniczne nazwy zmiennych i uzupełniać program objaśnieniami w komentarzach rozpoczynanych znakiem procentu [%];
- pamiętaj aby po napisaniu programu i po każdej jego modyfikacji **zapisać program do pliku** a dopiero potem **uruchamiać** (gwiazdka obok nazwy pliku w pasku tytułowym edytora przypomina, że zmodyfikowany plik nie został jeszcze zapisany); wymyślając nazwę pliku pamiętaj, że dotyczą jej te same reguły co nazw zmiennych i funkcji użytkownika (p.8.2.2.).

Edytor MATLAB-a (lub FreeMat-a) automatycznie nadaje zapisywanym plikom rozszerzenia nazw "**.m**". W edytorze można mieć otwartych kilka programów i wybierać je przy pomocy "zakładek" u dołu okna edytora.

Aby uruchomić napisany program wystarczy wpisać w oknie komend jego nazwę (tzn. nazwę pliku bez rozszerzenia ".m") lub w oknie edytora kliknąć przycisk opisany "RUN" (Rys. 8.5). Przy kolejnych uruchomieniach zamiast ponownie wpisywać nazwę w oknie komend można przywołać ją "z historii" naciskając klawisz strzałki w górę.

Po zasygnalizowaniu przez MATLAB-a błędów w trakcie wykonywania programu, należy zidentyfikować je i poprawić w edytorze a następnie zapisać i uruchomić zmodyfikowaną wersję.

Uwzględniając powyższe zalecenia napisz w edytorze MATLAB-a program (skrypt) obliczania trzeciej potęgi podanej liczby i zapisz do pliku, na przykład z nazwą: "pr56.m":

```
a = input('dana liczba=')
x=a^3
disp('szescian tej liczby='), disp(x)
```

Uruchom ten program (tak jak podano powyżej). Jeśli chcesz się pozbyć wyświetlania "echa" działania każdej instrukcji to musisz kończyć każdą instrukcję średnikiem [;].

Linie komentarzy – nie mające wpływu na działanie programu - rozpoczynane są znakiem [%].

Komentarze umieszczone na początku pliku (przed instrukcjami) wyświetlą się jako jego opis gdy w dowolnym momencie wpiszemy w oknie komend słowo: "help" a po nim nazwę pliku programu. Dodatkowo, program po uruchomieniu powinien się sam przedstawić, wyświetlając informacje o tym czego dotyczy i jak go użyć.

Tak więc ulepszona wersja programu może wyglądać tak:

```
% PR 66 Program oblicza sześcian podanej liczby
disp('Obliczanie sześcianu danej liczby:');
a = input('dana liczba=');
x=a^3;
disp('szescian tej liczby='); disp(x);
```

Wpisz: „**help** *nazwa_pliku*" a następnie jeszcze raz uruchom program.

Jeśli chcemy aby program działał dla wielu kolejno podawanych liczb – trzeba zastosować jeden z kilku możliwych typów pętli programowych. W tym przypadku najodpowiedniejsza będzie pętla WHILE ... END powtarzająca operacje aż do momentu gdy zechcemy ją zatrzymać.

Instrukcje i funkcje użyte w tym programie, jak również instrukcje pętli opisano dokładniej w następnym podrozdziale. Po przeczytaniu ich opisów uzupełnij powyższy program pętlą WHILE ... END tak aby zatrzymał się po wprowadzeniu zera jako danej dla x.

8.7. PODSTAWOWE INSTRUKCJE

Podstawowe instrukcje i funkcje MATLAB-a odpowiadają podstawowym typom operacji omówionym w rozdziale o algorytmach (p.3.2.), a mianowicie: wprowadzaniu i wyprowadzaniu danych, nadawaniu wartości zmiennym przez przypisanie, instrukcji warunkowej i dwu rodzajom pętli oraz wywołaniom podprogramów. Omówiono je kolejno w tym podrozdziale. Przedstawiono też instrukcję SWITCH pozwalającą wybierać jedną z wielu możliwości, oraz instrukcje obsługi plików pozwalające wprowadzać dane i wyprowadzać wyniki.

Do wprowadzania danych i wyprowadzania wyników w MATLAB-ie podobnie jak w językach Pascal i C nie są używane instrukcje lecz wywołania **funkcji**.

8.7.1. INSTRUKCJA PRZYPISANIA

Instrukcja **przypisania** zwana też instrukcją **podstawiania** jest jedną z najczęściej używanych instrukcji. Pozwala ona **nadawać zmiennym wartości** i realizować **obliczenia**.

Instrukcja przypisania ma postać:

```
zmienna = wyrażenie
```

Jej sens jest następujący: „oblicz wartość ***wyrażenia*** i przypisz ją ***zmiennej***".

Przykłady:
```
a23 =7.45e6;  Obciazenie = 3500;  Alfa = -0.456
Sila_Rozc = Obciazenie*sin(Alfa);     k = k+1;
```

Należy pamiętać, że znak [=] nie jest w Matlabie znakiem równości lecz **znakiem przypisania**, który należy rozumieć jako „przypisz" lub „podstaw", dlatego instrukcja **`k=k+1`** ma następujący sens: „zwiększ o 1 dotychczasową wartość zmiennej **`k`** i uzyskany wynik przypisz zmiennej **`k`** jako jej nową wartość".

Zauważmy, że **wyrażenie** zawsze występuje **po stronie prawej** (a nie odwrotnie) i najpierw obliczana jest jego wartość a dopiero potem wartość ta jest nadawana **zmiennej** zapisanej **po lewej stronie**. Aby obliczenia były wykonalne, wszystkie zmienne występujące w wyrażeniu muszą już mieć określone wcześniej wartości.

Oprócz realizacji obliczeń, instrukcja przypisania jest używana do nadawania wartości początkowych oraz modyfikowania wartości **zmiennych sterujących** działaniem programu jak liczniki czy indeksy.

Można także przypisać zmiennym wartości danych, ale raczej tylko we wstępnych, testowanych wersjach programu. Nie należy natomiast używać instrukcji przypisania do wprowadzania danych w ostatecznej wersji programu, gdyż dane te byłyby wówczas fragmentem programu a ich zmiana wymagałaby dokonania zmian w programie. Byłoby to sprzeczne z regułą, że każdy program powinien działać dla wielu różnych danych, bez konieczności zmieniania poleceń tego programu.

Ćwiczenia:

Z78. Napisz (w edytorze MATLAB-a) program, który przypisuje dane zmiennym wejściowym oraz oblicza i wyświetla (z objaśnieniem) wartość naprężenia wg wzorów:

$\sigma_{\max} = \dfrac{F_{\max}}{A_{\min}}$ gdzie: $A_{\min} = \pi R_{\min}^2$, natomiast: $F_{\max}$ oraz $A_{\min}$ są dane.

Z79. Napisz w edytorze MATLAB-a, zgodnie z zaleceniami podanymi w podrozdziale 8.6. program złożony z instrukcji przypisania, realizujący obliczenia według wariantu (a) wybranego algorytmu z podrozdziału 5.1.

8.7.2. WPROWADZANIE DANYCH Z KLAWIATURY

Programy przetwarzające dane muszą pobierać je z klawiatury lub innego urządzenia wejściowego lub z pliku, tak aby program, bez potrzeby modyfikowania, mógł działać wielokrotnie, dla różnych zestawów danych. Pobieranie danych z urzadzenia wejściowego lub z pliku i przypisywanie ich określonym zmiennym nazywa się **wczytywaniem danych**.

Do wczytywania danych z klawiatury najdogodniejsza jest w MATLAB-ie funkcja ***input(..)***. Funkcję tą można używać w dwu postaciach.

Postać pierwsza:

```
zmienna = input('żądanie danych')
```

spowoduje wyświetlenie w oknie komend tekstu *żądania danych* i oczekiwanie na to aby użytkownik wprowadził z klawiatury **wartość liczbową**, która ma być przypisana *zmiennej*. MATLAB umożliwia także wpisanie **wyrażenia** zamiast liczby.

Funkcja input w drugiej postaci:

```
zmienna = input('żądanie danych','s')
```

także wyświetli *żądanie danych,* ale tym razem oczekuje na wpisanie przez użytkownika *łańcucha znakowego* (ang.: *string of characters*), który przypisze *zmiennej*.

Z80. **Ćwiczenie**: Poprzednie programy ulepsz wprowadzając do nich polecenia wprowadzania danych z klawiatury.

8.7.3. WYŚWIETLANIE WYNIKÓW – *DISP(), FPRINTF()*

Najprostszy - podany już - sposób wyświetlania wyników, polega na nie umieszczaniu średników [;] po poleceniach, których wynik ma być wyświetlony. Poleceniami tymi mogą być instrukcje przypisania ale także same nazwy zmiennych, albo wyrażenia.

Drugi sposób polega na wykorzystaniu funkcji:

```
disp(wyrażenie)
```

przy czym dopuszczalne jest użycie *wyrażenia* albo o wartości liczbowej albo tekstowej. Przykłady użycia tej funkcji podano już przy omawianiu wyrażeń tekstowych (p.8.5.).

Kolejnym, bardziej zaawansowanym (chociaż i bardziej skomplikowanym) sposobem jest wykorzystanie funkcji **fprintf(..)** – identycznej jak w języku C. Funkcja ta pozwala wyprowadzać sformatowane wyniki wraz z objaśnieniami na ekran lub do pliku dyskowego, przy czym ten drugi wariant będzie omówiony nieco dalej.

W systemach Freemat i Scilab oraz języku C na ekran wyprowadza funkcja **printf()**.

Dla wyświetlania wyników na ekranie używana jest następująca postać funkcji **fprintf**:

```
fprintf('format', lista_wyrażeń)
```

Lista_wyrażeń zawiera pooddzielane przecinkami zmienne lub wyrażenia, których wartości mają być wyświetlone. Sposób wyświetlania poszczególnych wartości opisany jest przez *'format'*. Tam także umieszcza się teksty objaśnień które mają być wyświetlone. Przykładowo instrukcja:

```
fprintf('\nLiczba skladników=%3d Suma=%7.2f',N,Suma)
```

może wyświetlić napis:

```
Liczba skladników= 23   Suma= 784.32
```

Objaśnienie:

[**\n**] to rozkaz przejścia do nowej linii, kolejne znaki będą przekopiowane do wyprowadzanego napisu, aż do znaku [%]. Przy każdym napotkaniu znaku [%] nastąpi wyprowadzanie wartości kolejnego wyrażenia z listy, w sposób opisany przez liczbę umieszczoną po znaku [%] i zakończoną literą (w naszym przykładzie **d** lub **f**).

Dokładniej: ***'format'*** – to łańcuch znaków ujętych w apostrofy, który może zawierać:

a) znaki sterujące - złożone z litery poprzedzonej „backslashem" [\]:
   \a *alarm* – sygnał dźwiękowy,
   \b *backspace* – cofnięcie o znak,
   \f *form feed* – przejście do nowej strony,
   \n *new line* – przejście do nowej linii,
   \r *carriage return* – skok na poczatek linii („powrót karetki")
   \t *horizontal tab* – tabulator poziomy,
   \v *vertical tab* – tabulator pionowy,
   \xN *hexadecimal number* N – liczba szesnastkowa,
   \N *octal number* N – liczba ósemkowa;

b) opisy formatów, które zaczynają się od **znaku %**
   - po znaku % jest liczba określająca **długość pola**,
   - następnie (dla wyświetlanych liczb rzeczywistych) jest kropka i liczba określająca **precyzję** - liczbę miejsc po kropce w wyświetlanej wartości,
   - format kończy **litera** okreslająca typ wyświetlanych wartości, tzw. **kod konwersji**, najczęściej stosowane kody to:
     d – dla liczb całkowitych – np.: **%5d**
     e – dla wyświetlania liczb rzeczywistych w postaci wykładniczej,
     f – dla wyświetlania liczb rzeczywistych w postaci stałoprzecinkowej,
     g – dla wyświetlania liczb rzeczywistych w formacie **e** lub **f** w zależności od tego, który z nich będzie odpowiedniejszy dla danej wartości,
     s – dla wyprowadzania wartości wyrażeń tekstowych (*string*).

c) pozostałe znaki ze spacjami włącznie są traktowane jak tekst który ma zostać wyświetlony i będzie w miejscach opisów formatów uzupełniany wartościami poszczególnych wyrażeń z ***listy_wyrażeń***.

W przypadku gdyby w tekście konieczne było wyświetlenie znaków: apostrof ['], procent [%] oraz backslash [\] wówczas należy w formacie wpisać dwa takie znaki ["], [%%], [\\].

Z81. **Ćwiczenie**:

Napisz w edytorze MATLAB-a, zgodnie z zaleceniami podanymi w podrozdziale 8.6. , program realizujący obliczenia według wariantu (a) wybranego algorytmu z rozdziału 6, zawierający: polecenia wprowadzania danych z klawiatury, instrukcje przypisania oraz funkcję ***disp()*** lub ***fprintf()*** wyświetlającą wyniki wraz z tekstami objaśniającymi.

8.7.4. INSTRUKCJA *IF*

W wielu przypadkach musimy w programie umieścić kilka alternatywnych sekwencji operacji a wybranie jednej z nich uzależnić od spełnienia określonego warunku.

Służy do tego instrukcja warunkowa nazywana instrukcją IF od występującego w niej słowa kluczowego (IF oznacza „jeżeli").

Instrukcja IF (w najprostszych wariantach) może mieć jedną z postaci:

% postać 1:	% postać 2:

if *warunek*
  instr1
end
instr3
	if *warunek*
	  instr1
	else
	  instr2
	end
	instr3

warunek – może być wyrażeniem logicznym lub wyrażeniem algebraicznym,
instr1, instr2, instr3 – to bloki zawierające jedną lub więcej instrukcji.

Działanie instrukcji IF jest następujące: jeżeli *warunek* jest spełniony (czyli wartość liczbowa wyrażenia logicznego lub algebraicznego jest różna od zera) to zostanie wykonany blok instrukcji *instr1*, w przeciwnym przypadku (tzn. gdy warunek ma wartość liczbową zero) – w postaci 1 wykonane będą instrukcje bloku *instr3* a w postaci 2 – bloki instrukcji *instr2* oraz *instr3*.

Pamiętajmy, że jeśli mamy w programie uwzględnić **dwa warianty** działań to wystarcza do tego **jedna instrukcja IF**, natomiast przy trzech lub większej liczbie wariantów trzeba będzie użyć kilku instrukcji IF odpowiednio zagnieżdżonych, co pokazano na przykładzie PR67. W przykładzie tym nie ma instrukcji wyświetlających wyniki a zamiast tego pominięto średniki po instrukcjach przypisujących wartość zmiennej **y**, co spowoduje wyświetlenie wartości tej zmiennej.

PR 67: Dla danej wartości x obliczyć y według wzoru:

$$y = \begin{cases} 1-x^2 & dla \quad x < -1 \\ x & dla \quad -1 \le x \le 1 \\ 1+x^2 & dla \quad x > 1 \end{cases}$$

```
clear
x = input('x=');
if x<-1
   y = 1-x^2
else
   if x>1
         y = 1+x^2
   else
         y = x
   end
end
```

Ćwiczenia:

Z82. Napisz program, który pyta o temperaturę za oknem i zależnie od wartości odpowiada jednym z komunikatów: „mróz”, „zimno”, „ciepło”, „gorąco”.

Z83. Napisz program, który pyta o liczbę i po jej wprowadzeniu bada czy jest parzysta oraz czy jest podzielna przez 3, a następnie wyprowadza odpowiednie komunikaty. Podpowiedź: wykorzystaj funkcję ***mod(x,n)*** zwracającą resztę z dzielenia x przez n;

Z84. Napisz program „ekspercki” pomagający stwierdzić przyczynę nie świecenia lampy (żarówka, bezpieczniki, kabel, ...), to znaczy wydający polecenia, zadający pytania oraz udzielający rad.

8.7.5. INSTRUKCJA WYBORU *SWITCH*

Instrukcja SWITCH (czyli „przełącznik") pozwala wybrać i wykonać jeden z wielu bloków instrukcji. Instrukcja SWITCH ma postać:

```
switch w
  case w1
    instr1
  case w2
    instr2
. . . . . .
  otherwise
    instrN
end
```

gdzie: *w* - zmienna lub wyrażenie, które musi mieć wartość skalarną lub znakową,
instr1, instr2, ... , instrN – to bloki instrukcji,
w1, w2, ... – wartości zapisane w postaci stałych

Instrukcja wyznacza wartość wyrażenia „*w*" umieszczonego po słowie „**switch**" a następnie wykonuje ten blok instrukcji, przed którym zapisano po słowie „**case**" stałą równą wartości wyrażenia „*w*". Jeśli żadna ze stałych „*w1, w2,* ..." nie jest równa wartości wyrażenia „*w*", to zostanie wykonany blok instrukcji zapisany po słowie kluczowym **otherwise**.

PR 68:

```
clear; disp('Kalkulator cztero-działaniowy:')
x=input('x='); y=input('y=');
zn=input('wpisz znak działania:', 's');
switch zn
    case '+'
      wynik=x+y
    case '-'
      wynik=x-y
    case '*'
      wynik=x*y
    case '/'
       if y==0
          disp('Błąd: Dzielenie przez zero!')
       else
          wynik=x/y
       end
    otherwise
      disp('Nierozpoznane działanie!')
end
```

Zauważmy, że instrukcja SWITCH nie może na ogół zastąpić instrukcji IF gdyż nie można przy jej pomocy testować prawdziwości nierówności czy dowolnych wyrażeń logicznych a jedynie można sprawdzić czy spełniona jest jedna z równości: *w==w1, w==w2,* ...

Nie można więc użyć instrukcji SWITCH dla realizacji podanego uprzednio zadania – wyznaczania wartości funkcji określonej w trzech przedziałach oddzielnymi wzorami. Dla rozgałęziania programu na podstawie nierówności i innych wyrażeń logicznych pozostaje więc używanie jednej lub wielu zagnieżdżonych instrukcji IF.

8.7.6. PĘTLA *FOR*. WYKRES TYPU XY

Pętla **for** ... **end** jest stosowana jest gdy z danych wynika ile razy zachodzić będą cykliczne powtórzenia instrukcji zawartych wewnątrz pętli czyli zapisanych między **for** a **end**. Pętla ta ma postać:

```
for zmienna = wart_p : krok : wart_k
    blok instrukcji
end
```

Pozwala więc ona powtarzać wykonywanie *bloku instrukcji* określoną liczbę razy przy czym dodatkowo *zmienna* kontrolna w tej pętli przyjmuje kolejno wartości od *wart_p* do *wart_k* z przyrostem (lub ubytkiem) *krok*. Jeśli *krok* jest równy 1 to można go pominąć w zapisie.

PR 69:
Chcemy otrzymać kwadraty liczb parzystych od 2 do 10 i podstawić je do kolejnych elementów wektora p:

```
for i = 1:5
    p(i) = (2*i)^2
end
p
```

PR 70: Generowanie tabeli i wykresu funkcji sinus.
W pętli FOR wyznaczamy i wstawiamy do wektora X ciąg wartości kąta (w radianach) od zera do 2*pi z przyrostem 0.2, a do wektora Y ciąg odpowiadających im wartości funkcji sinus.

```
k=0; % zerujemy licznik elementów tabeli
fprintf('\n  x  sin(x)'); % nagłowek tabeli
for x=0:0.2:2*pi
  k=k+1;
  X(k)=x; Y(k)=sin(x);
  fprintf('\n %7.4f %7.4f',X(k),Y(k));
end
plot(X,Y); grid on; % to wykres i siatka
title('Funkcja sinus'); % tytuł wykresu
xlabel('x'); ylabel('y'); % etykiety osi
```

PR 71: Chcemy wygenerować następującą macierz M o trzech wierszach i czterech kolumnach:

```
M =
   2   3   4   5
   3   4   5   6
   4   5   6   7
```

```
for w=1:3
   for k=1:4
      M(w,k)=w+k;
   end
end
M
```

Ćwiczenia:

Z85. Napisz program z pętlą FOR...END, który wyświetla na ekranie pierwiastki z kolejnych liczb nieparzystych od 1 do 9

Z86. Napisz program z pętlą FOR...END, który dla ciągu wartości φ od 3,6 do 13 co 0,4 oblicza wartości wyrażenia: **β = (φ − 0,5)/(1,1+sin φ)**

Z87. Napisz program realizujący tabelę i wykres funkcji:

$$Wz = \frac{3\sqrt{\alpha}}{1 + 2\sin\alpha\cos^2\alpha} \quad \text{dla } \alpha = 5°, 10°, 15°, ...175°$$

Z88. Napisz program do wybranego (lub podanego) algorytmu z rozdziału 6.5. .

8.7.7. PĘTLA *WHILE*

Jeden z dwu typów pętli programowych realizuje w MATLAB-ie instrukcja ***while ... end*** o następującej postaci:

```
while wyrażenie
   instrukcje
   . . . . .
end
```

Po słowie kluczowym **while** musi wystąpić wyrażenie logiczne lub algebraiczne.

Pętla typu **while** powtarza *instrukcje* zapisane między **while** i **end** tak długo jak długo *wyrażenie* jest różne od zera lub w sensie logicznym jest prawdziwe.

Instrukcję **while** stosuje się gdy liczba powtórzeń pętli nie jest z góry określona lecz powtórzenia mają zachodzić aż do wystąpienia pewnego zdarzenia, na przykład naciśnięcia określonego klawisza lub wystąpienia znacznika końca pliku dyskowego.

Pętlę typu **while** ogólnie omawiano m.in.w p.3.6.6. i 4.6.10. natomiast przykład jej zastosowania w MATLAB-ie pokazano i objaśniono poniżej.

PR 72:

```
clear;
p='T';
while p=='T'
  disp('Obliczanie wartości dowolnej funkcji f(x)');
  x=input('Podaj wartość x:');
  funkcja=input('Wpisz wyrażenie zależne od x','s');
  y=eval(funkcja);
  fprintf('\n x=%10.4f  f(x)= %10.4f\n',x,y);
  p=input('Czy nowe obliczenie? (T/N):', 's');
  p=upper(p);
end
```

Powyższy program rozpoczyna się od wyczyszczenia pamięci zmiennych (clear). Aby instrukcje wewnątrz pętli **while** mogły być po raz pierwszy wykonane musi być spełniony warunek podany po słowie **while**, to znaczy zmienna **p** musi zawierać znak **'T'**, (dokładniej: wartością zmiennej **p** ma być znak **'T'**) a więc przed pętlą musimy zmiennej **p** nadać taką wartość.

Wewnątrz pętli mamy instrukcje wyświetlające informacje o tym co program będzie robił a następnie, przy pomocy funkcji **input** program wprowadza z klawiatury wartość **x** oraz łańcuch tekstowy z wyrażeniem które należy obliczyć (np.: **sqrt(x)** – dla obliczenia pierwiastka z **x**). Funkcja **eval** pozwoli wykonać obliczenia według wprowadzonego z klawiatury wzoru i przypisać wynik zmiennej **y**.

Kolejna instrukcja **input** wyświetla pytanie `'Czy nowe obliczenie? (T/N):'` żądając naciśnięcia klawisza T (tak) lub N (nie). Ponieważ wprowadzony do zmiennej **p** znak może być dużą lub małą literą więc dla umożliwienia porównania go - po powrocie na poczatek pętli - ze znakiem **'T'** dokonano konwersji na duże litery, przy pomocy funkcji **upper.**

8.8. OPERACJE NA PLIKACH

MATLAB może korzystać z wielu różnych typów danych zawartych w plikach: tekstowych, binarnych, graficznych oraz audio i video. Potrafi m.in. wczytywać dane z arkuszy kalkulacyjnych Excela i Lotusa, filmy w formacie AVI oraz obrazy zapisywane w wielu różnych formatach plików graficznych. Wykaz typów plików obsługiwanych przez MATLAB-a oraz jego funkcji do tego przeznaczonych można uzyskać wpisując „*help fileformats*".

W przypadku prostych programow obliczeniowych wystarczy nam znajomość kilku funkcji umożliwiających (a) zapisywanie i odczytywanie zmiennych z przestrzeni roboczej i (b) obsługiwanie plików tekstowych. Większość funkcji służących do tych celów zostało do MATLAB-a zapożyczonych z języka C.

8.8.1. ZAPISYWANIE I ODCZYTYWANIE ZMIENNYCH Z PRZESTRZENI ROBOCZEJ

Zmienne – a w tym i macierze – używane w plikach skryptowych oraz poleceniach pisanych w oknie komend, są przechowywane w tak zwanej przestrzeni roboczej MATLAB-a (*MATLAB workspace*). Zmienne te można wyświetlić komendą **whos** lub wymazać komendą **clear**. Zmienne te można także zapisać do pliku na przykład komendą:

```
save plik lista_zmiennych
```

na przykład: `save wyniki x y`

Jeśli pominiemy listę zmiennych to zapisane będą wszystkie.

Jeśli nie podamy rozszerzenia nazwy pliku to powstanie plik binarny z rozszerzeniem **.mat.** Natomiast gdy damy rozszerzenie **.txt** to powstanie plik tekstowy.

Aby wczytać tak zapisane zmienne z pliku *.mat trzeba użyć komendy:

```
load plik
```

a w przypadku pliku tekstowego:

```
load plik.txt
```

w tym drugim przypadku wczytane dane utworzą macierz o nazwie *plik*.

8.8.2. OTWIERANIE I ZAMYKANIE PLIKÓW

Przed rozpoczęciem wyprowadzania informacji do pliku lub wczytywania z pliku należy ten plik **otworzyć** przy pomocy funkcji **fopen**, a gdy plik jeszcze nie istnieje to funkcja **fopen** może najpierw go utworzyć. Polecenie otwarcia pliku może mieć postać:

```
[id, kom] = fopen(nazwa_pliku,tryb)
```

Jako pierwszy parametr funkcji **fopen** musimy podać łańcuch znaków określajacy *nazwę_pliku*, wraz z ewentualną ścieżką dostępu. Drugi parametr, także będący łańcuchem znaków, określa *tryb* dostępu do pliku i jego typ. Tryby dostępu do plików przedstawia Tabela 8.6.

Funkcja **fopen** zwraca dwuelementowy wektor **[id, kom]**. Jako pierwszy element otrzymamy identyfikator **id** pliku, na który będziemy musieli powoływać się we wszystkich następnych poleceniach dotyczących tego pliku. Jeśli otwieralibyśmy kilka plików to oczywiście powinniśmy zastosować różne nazwy zmiennych dla ich identyfikatorów (np.: `id1, id2, ...`).

Identyfikator `id` po prawidłowym otwarciu pliku otrzyma wartość całkowitą dodatnią, natomiast w przypadku niemożności otwarcia pliku otrzyma wartość minus jeden. Dodatkowo, jako drugi element wektora zwracanego przez funkcję **fopen** otrzymamy w tym drugim przypadku komunikat (**kom**) o przyczynie błędu. Przykładowo – dysk docelowy może być zabezpieczony przed zapisem.

Tabela 8.6. Tryby dostępu do pliku tekstowego

Parametr	Tryb dostępu do pliku tekstowego
'rt'	otwarcie do odczytu (*read*) - domyślne gdy pominiemy ten parametr
'wt'	usunięcie zawartości istniejacego pliku lub utworzenie nowego (gdy plik z tą nazwą nie istnieje) i otwarcie go do zapisu (*write*)
'at'	otwarcie w celu dopisywania na końcu istniejącej zawartości (*append*)
'rt+'	otwarcie do odczytu i zapisu
'wt+'	wymazanie istniejacego pliku lub utworzenie nowego i otwarcie do odczytu i zapisu
'at+'	otwarcie w celu czytania lub dopisywania elementów na końcu (*append*)

Tak więc, jeśli chcemy być informowani o występowaniu błędów otwarcia plików, to po poleceniu otwarcia pliku powinniśmy napisać:

```
if id<0
    disp(kom);
end
```

Po otwarciu pliku można w programie umieścić wiele poleceń wprowadzania lub wyprowadzania informacji, odwołujących się do tego pliku poprzez jego identyfikator. Po zakończeniu tych operacji, plik na którym operujemy musimy zamknąć przy pomocy funkcji **fclose**(id).

W przypadku gdyby program – z powodu błędu - przerwał działanie przez wykonaniem **fclose**, najlepiej w oknie komend wpisać wówczas polecenie **fclose('all')** zamykające wszystkie otwarte pliki. Jest to istotne bo plik niezamknięty nie daje się również usunąć. Polecenie takie warto również umieszczać na początku lub na końcu programu.

8.8.3. WYPROWADZANIE WYNIKÓW DO PLIKU

Po otwarciu pliku w trybie umożliwiającym zapis (patrz Tabela 8.6) można rozpocząć wyprowadzanie informacji do tego pliku. Użyjemy do tego tą samą funkcję **fprintf**, którą używaliśmy do wyprowadzania informacji na ekran, ale tym razem pierwszym jej parametrem musi być identyfikator pliku (id) utworzony wcześniej funkcją **fopen**. Tak więc ogólna postać wywołania funkcji **fprintf** będzie następująca:

```
fprintf(id,'format', lista_wyrażeń)
```

gdzie:
- `id` - identyfikator pliku omówiony w p.8.8.1.
- `'format'` - argument funkcji fprintf omówiony w 8.7.3.
- `lista_wyrażeń` - argument funkcji fprintf omówiony w 8.7.3.

Program PR73 napisany z zastosowaniem omówionych funkcji obsługi plików przedstawiono poniżej.

```
% PR 73
% Program zapisuje do pliku wartości kąta x oraz jego funkcji sin(x), cos(x)
[id, kom] = fopen('wyniki1.txt','wt'),  % Tworzy plik wyników
fprintf(id,'%s\n', '  kąt x [stopnie]    sin(x)    cos(x)' );  % Nagłówek tabelki
for xs = 0 : 5 : 90
        x = xs*pi/180;    % kąt xs zamieniony na radiany
        y1=sin(x);  y2=cos(x);
        fprintf(id,'       %3d', xs);
        fprintf(id,' %15.4f   %12.4f\n', y1,y2);
end
fclose(id);
```

Program ten generuje do pliku tekstowego tabelkę wartości kąta oraz odpowiadających im wartości funkcji sinus i cosinus. Oznajmia o tym komentarz zawarty w pierwszej linii programu.

W linii drugiej funkcja **fopen** tworzy i otwiera folderze bieżącym (patrz Rys. 8.5) plik o nazwie ***'wyniki1.txt'*** bo taką nazwę pliku podaje jej pierwszy parametr.

Drugi parametr **'wt'** określa typ dostępu:

'w' = zapis (ang. write), **'t'** – plik typu tekstowego

Funkcja **fopen** zwraca dwie wartości, które w tym przypadku zostaną podstawione do zmiennych **[id, kom]** gdzie: **id** = identyfikator pliku, **kom** = komunikat o ewentualnej przyczynie niemożliwości otwarcia pliku.

W linii trzeciej wyprowadzono przy pomocy funkcji **fprintf** nagłówek tabelki do pliku tekstowego o identyfikatorze podanym jako pierwszy parametr tej funkcji (w naszym przypadku: **id**), przy czym przypomnijmy sobie, że:

%s - to format dla wyprowadzania tekstu podanego jako drugi parametr,

\n - polecenie przejścia do nowej linii.

Linia czwarta rozpoczyna pętlę **for**, w której będą powtarzane operacje obliczania i wyprowadzania do pliku poszczególnych wierszy tabeli, a słowo **end** zaznacza koniec tej pętli. Ponieważ argumenty funkcji trygonometrycznych muszą być podawane w radianach (w MATLAB-ie ale także w innych językach programowania), więc kolejna linia programu dokonuje takiej zamiany, a w następnej linii dokonuje się wyznaczenie wartości interesujących nas funkcji. Kolejne dwie linie to wywołania funkcji **fprintf** w celu „wydrukowania" do pliku wiersza tabeli, przy czym przypominam, że:

%3d - określa **3** miejsca dla liczby **całkowitej** (o czym świadczy litera **d**)

%12.4f - to format dla liczb rzeczywistych, a w nim **12** miejsc zadeklarowano dla całej liczby (i poprzedzających ją spacji) a w tych 12-tu zarezerwowano **4** miejsca po kropce dla części ułamkowej

\n - oznacza rozkaz zmiany linii na wydruku wyprowadzanym do pliku

fclose(id) - zamyka plik o identyfikatorze **id**

Po zakończeniu działania programu, łatwo sprawdzić – na przykład przy pomocy „Notatnika" systemu Ms Windows – że to co program wygenerował do pliku 'wyniki1.txt' ma następującą postać:

```
kąt x [stopnie]    sin(x)    cos(x)
          0        0.0000     1.0000
          5        0.0872     0.9962
         10        0.1736     0.9848
         15        0.2588     0.9659
         20        0.3420     0.9397
        . . .      . . . .    . . . .
```

Jak widać spacje w łańcuchu formatu są istotne bo przenoszą się na wydruk.

8.8.4. WCZYTYWANIE DANYCH Z PLIKU

Wyniki różnorodnych pomiarów - dokonywanych w praktyce inżynierskiej - przechowywane są z reguły w plikach dyskowych. Musimy więc umieć napisać program, który wczytuje dane z pliku, w celu dalszego ich przetwarzania.

Przed wczytywaniem danych z pliku – tak samo jak przed wyprowadzaniem - trzeba najpierw **plik z danymi otworzyć i przypisać mu identyfikator** przy pomocy tej samej co poprzednio funkcji **`fopen`**:

```
[id kom] = fopen(nazwa_ pliku)
```

Jak widać w tym przypadku pominięto drugi argument funkcji, co jest równoznaczne z zastosowaniem domyślnego trybu otwarcia pliku czyli „do odczytu".

Licząc się z możliwością wystąpienia błędu przy otwieraniu pliku, warto następnie wpisać polecenia wyświetlające ewentualny komunikat o przyczynie błędu:

```
if id<0
   disp(kom)
end
```

Teraz można już realizować wczytywanie. Jedną z funkcji do tego przeznaczonych jest funkcja **fscanf**, której wywołanie ma następującą składnię:

```
[macierz liczba_el] = fscanf(id,'format', [Lk Lw])
```

gdzie:
- `id` - to identyfikator pliku nadany przy otwarciu funkcja ***fopen***
- `'format'` - łańcuch opisu formatu dla wczytywanych danych, przy czym można nie podawać długości pól
- `[Lk Lw]` - rozmiary_macierzy
- `macierz` - wczytana macierz
- `liczba_el` - liczba wczytanych elementów

Jak widać funkcja **fscanf** pozwala wczytać od razu całą macierz danych.

Jeśli nie podamy rozmiarów [Lk Lw] to otrzymamy macierz jednokolumnową. Jeśli natomiast nie wiemy ile jest elementów ale wiemy, że chcemy otrzymać dwie kolumny to podamy rozmiar [2 *inf*], (przy czym *inf* to nieskończoność).

```
% Program PR 74:
x = 0:.1:1;        % generuje wektor wierszowy
y = [x; exp(x)];   % skleja dwa wektory wierszowe w macierz y
fid = fopen('exp.txt', 'w');          % otwiera plik do zapisu
fprintf(fid, '%6.2f %12.8f\n', y);    % wyprowadza do pliku macierz y
fclose(fid);       % zamyka plik
type exp.txt       % wyświetli treść pliku
% ======= a teraz wczytywanie  z pliku:
fid = fopen('exp.txt');               % otwiera plik
A = fscanf(fid, '%g %g', [2 inf]);    % wczytuje do macierzy A
fclose(fid);                          % zamyka plik
% ===== Potrzebna jest jeszcze transpozycja macierzy bo kolumny wczytały się do wierszy
A = A';
```

Po zakończeniu wczytywania trzeba zamknąć plik używając funkcji: **fclose(**`id`**).**

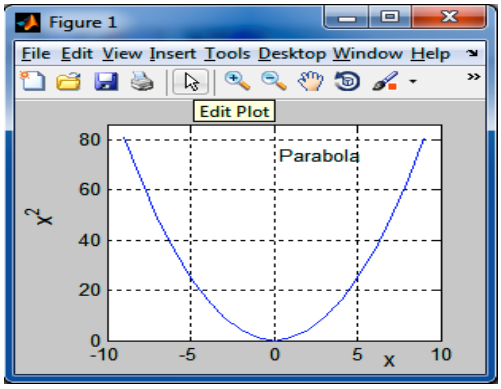
Jak widać jest to nieco skomplikowane dlatego czasem prościej jest użyć omówionego wcześniej polecenia **save**.

8.9. WPROWADZENIE DO WYKRESÓW TYPU XY

Podstawową funkcją do tworzenia w MATLAB-ie wykresów liniowych typu XY jest funkcja **plot**, która może być wywoływana w kilku wariantach, różniących się liczbą argumentów. Jednym z najprostszych wariantów jest wywołanie funkcji plot z dwoma parametrami: **plot(*x, y*)**. Parametry są wektorami o jednakowej długości, przechowującymi ciągi współrzędnych x(1), x(2), ... , x(n), oraz y(1), y(2), ... , y(n), punktów łamanej tworzącej wykres.

Przykład wykorzystania pętli FOR oraz funkcji **plot** dla narysowania wykresu funkcji sinus pokazano już w p.8.7.6. , natomiast przykład programu PR75 rysującego wykres paraboli – bez użycia pętli - pokazuje Rys. 8.6. Jak widać **x** jest tu macierzą jednowymiarową (wektorem – sprawdź jego wartości w pisując nazwę x w oknie komend), więc do obliczenia kwadratów **x** trzeba użyć operatora z kropką czyli potęgowania poszczególnych elementów wektora **x**. Operator potęgowania bez kropki oznaczałby mnożenie macierzowe wektora wierszowego przez taki sam wektor wierszowy co jest niewykonalne, bo jak wiemy przy mnożeniu macierzowym mnoży się „wiersz razy kolumna", a więc wiersz mnożnej musi mieć tyle samo elementów co kolumna mnożnika.

```
% Program PR 75
% Rysowanie wykresu paraboli
x=-9:9;
y=x.^2;
plot(x,y);
grid on
title('Parabola')
xlabel('x')
xlabel('x^2')
```



Rys. 8.6. Przykład wykresu XY

Oprócz wywołania funkcji plot użyto jeszcze pomocniczych poleceń ustalających szczegóły wykresu takie jak:

grid on – włączenie siatki,

title(*'Tytuł wykresu'*) – tytuł wykresu,

xlabel('*opis_x*'); **ylabel**('*opis_y*') – opisy osi.

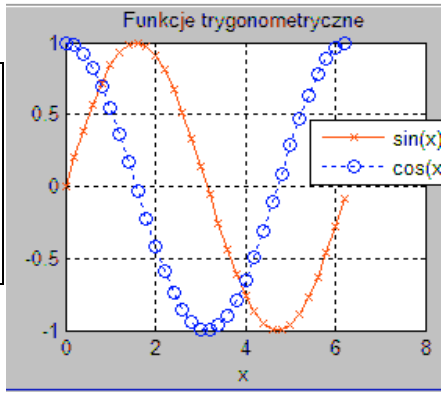
W pasku narzędzi okna graficznego widać **przycisk ze strzałką** [***Edit Plot***], którym można włączyć **tryb ręcznej edycji wykresu**. W trybie tym można wybierać myszką poszczególne elementy wykresu i zmieniać ich własności z menu podręcznego, tak samo jak robi się to na przykład w Excelu.

Jeśli w jednym układzie chcemy umieścić **kilka wykresów** to mamy dwie możliwości:

a) podać jako argumenty tyle par wektorów ile jest wykresów:
plot(*x1,y1, x2,y2*, ...), zdając się na automatyczny wybór rodzajów linii,

b) podać jako argumenty tyle trójek argumentów ile jest wykresów: **plot(*x1,y1,s1, x2,y2,s2,* ...)**, przy czym ostatni argument z każdej trójki (s1, s2, ...) jest łańcuchem symboli opisujących **typ linii, kolor linii** oraz rodzaj **znacznik**a punktów, tak jak opisuje to Tabela 8.7.

```
% Program PR 76
x=0 : 0.2 : 2*pi;
plot(x,sin(x),'-rx',x,cos(x),':bo');
grid on
title('Funkcje trygonometryczne')
xlabel('x')
legend('sin(x)','cos(x)');
```



Rys. 8.7. Przykład dwu wykresów XY

Przykład programu (PR 76) rysowania dwu wykresów typu XY w jednym układzie współrzędnych pokazuje kolejny rysunek (Rys. 8.7).

Tabela 8.7. Opis znaków definiujących parametry wykresu liniowego (typu PLOT)

Znak	Rodzaj linii
-	ciągła
--	kreskowana
:	kropkowana
-.	kreska-kropka

Znak	Kolor linii
y	*yellow* – żółty
m	*magenta* – karmazynowy
c	*cyan* – turkusowy
r	*red* – czerwony
g	*green* – zielony
b	*blue* – niebieski
w	*white* – biały
k	*black* – czarny

Znak	Znacznik punktu wykresu
+	+
*****	*
.	kropka
o	o
x	x
s	kwadrat
d	romb
p	gwiazdka pięcioramienna
h	gwiazdka sześcioramienna
v	trójkąt z wierzchołkiem w dół
^	trójkąt z wierzchołkiem w górę
<	trójkąt z wierzchołkiem w lewo
>	trójkąt z wierzchołkiem w prawo

Oprócz opisanych już poleceń występuje dodatkowo polece**nie legend** definiujące legendę opisującą poszczególne wykresy. Zastosowano opisy linii:

'-rx' – linia ciągła (-), czerwona (r), znaczniki „iksy” (x),

':bo' – linia kropkowa (:), niebieska (b), znaczniki „kółka” (o).

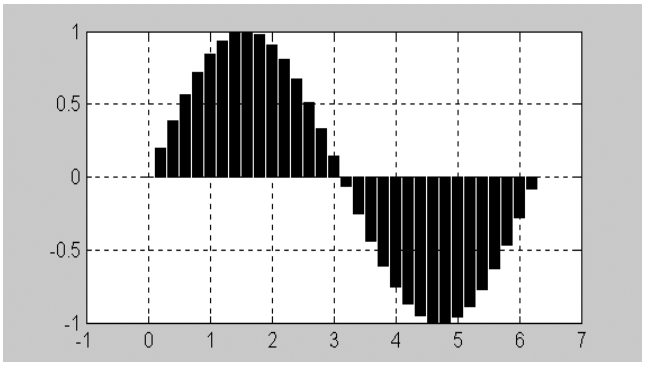
Wykres słupkowy typu XY (Rys. 8.8) można otrzymać przy pomocy funkcji **bar(*x, f(x)*).**

Jeśli w funkcji tej zastosujemy trzy argumenty:

bar(*x, y, s*)

to, ostatni argument ***s*** – określa stosunek szerokości słupka do odstępu między słupkami.

```
% Program PR 77
% Wykres słupkowy:

x=0 : 0.2 : 2*pi;
bar(x,sin(x));
grid on
```



Rys. 8.8. Wykres słupkowy

Bardzo przydatna, przy rysowaniu większej liczby wykresów lub innych obiektów graficznych, jest funkcja:

subplot(Lw, Lk, Nr);

która tworzy w oknie graficznym kilka regionów graficznych (okienek) ustawionych w **Lw** wierszach i **Lk** kolumnach, a następnie uaktywnia okienko o podanym numerze **Nr**. Tak więc liczba poleceń wywołujących funkcję subplot musi być równa liczbie okienek. Przykładowo gdy będą 4 okienka – do rysowania funkcji y1(x), y2(x), y3(x), y4(x) - czyli Lw=2, Lk=2 to muszą być 4 instrukcje **subplot**:

subplot(2, 2, **1**); plot(x,y1(x))

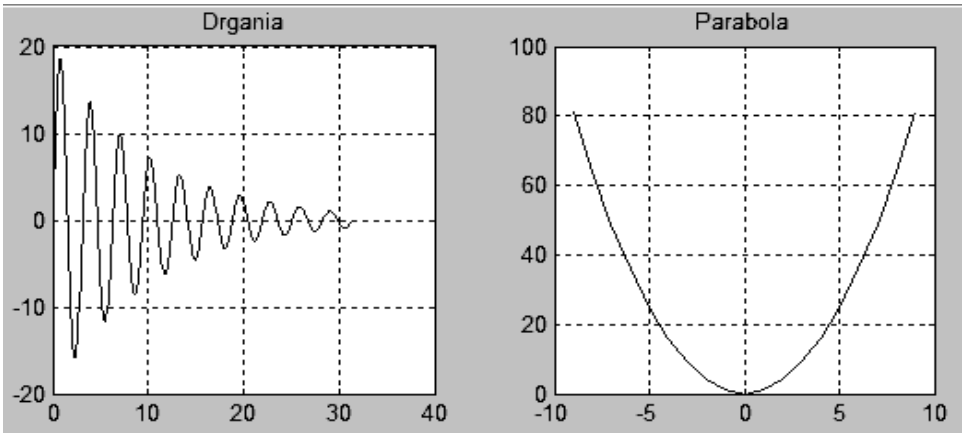
subplot(2, 2, **2**); plot(x,y2(x))

subplot(2, 2, **3**); plot(x,y3(x))

subplot(2, 2, **4);** plot(x,y4(x))

A oto konkretny przykład programu rysującego wykresy w dwu osobnych okienkach (regionach) tego samego okna graficznego (czy jak kto woli: tej samej „formatki").

```
% PR 78 – Program rysowania w dwu okienkach
clear
f=0:0.1:31.4;
x=-9:9;
y1=exp(3-0.1*f).*sin(2*f);
subplot(1,2,1); plot(f,y1);
title('Drgania'); grid on
subplot(1,2,2); plot(x,x.^2);
title('Parabola'); grid on
```



Rys. 8.9. Wykresy i podział okna graficznego na podrzędne okienka

Efekt działania tego programu (PR 78) pokazuje Rys. 8.9.
Więcej o wykresach i ich elementach napisano w rozdziale 10.

8.10. DEFINIOWANIE I WYWOŁYWANIE FUNKCJI UŻYTKOWNIKA

Zaletą MATLAB-a jest ogromne bogactwo gotowych funkcji z wszelakich niemal dziedzin zastosowań komputerów, jednak w razie potrzeby, użytkownik może **tworzyć własne funkcje** i wywoływać je identycznie jak funkcje MATLAB-a. Własne funkcje powinien użytkownik zdefiniować, m.in w przypadku:

1. gdy chcemy w programie wyodrębnić bloki funkcjonalne o ściśle określonym działaniu – co czyni program lepiej zrozumiałym i jest szczególnie zalecane w przypadku dużych programów
2. gdy funkcja będzie przynajmniej kilkukrotnie wywoływana w programie z różnymi argumentami, szczególnie gdy argumenty te nie zmieniają się w sposób, który pozwoliłby na użycie pętli;
3. gdy tworzymy program z interfejsem graficznym, gdzie elementom sterującym przypisane są zdarzenia, a zdarzeniom określone funkcje, które musimy zdefiniować.

Definicja funkcji w MATLAB-ie musi rozpoczynać się od linii o następującej strukturze:

function *wektor_zmiennych_wynikowych* = ***nazwa_funkcji****(parametry_wejściowe)*

Na przykład:

function [y1, y2, y3] = AL15(x1, x2, x3, x4)

W kolejnych liniach **po tym nagłówku, należy umieścić linie komentarzy** opisujących: do czego służy ta funkcja, jakie argumenty i w jakiej kolejności trzeba jej dostarczyć, jakie wyniki zwraca i jakie są możliwe warianty jej wywoływania. Te objaśnienia zostaną wyświetlone za każdym razem, gdy w oknie komend wpiszemy słowo "help", a po nim nazwę naszej funkcji.

W dalszych liniach muszą być zapisane polecenia prowadzące do wyznaczenia wszystkich *zmiennych wynikowych* (w przykładzie: [y1, y2, y3]) na podstawie *parametrów wejściowych* czyli **argumentów** funkcji (w przykładzie: x1, x2, x3, x4)

Wskazane jest kończyć funkcję słowem **end** umieszczonym w ostatniej linii, chociaż na ogół nie jest to obowiązkowe. Dla wcześniejszego wymuszenia wyjścia z funkcji można także użyć słowa kluczowego **return**.

Zmienne używane wewnątrz funkcji są **lokalne** (niedostępne z zewnątrz funkcji).

Jeśli chcemy wywoływać naszą funkcję z różnych programów i ewentualnie z okna komend to należy zapisać ją do **pliku o nazwie takiej samej jak nazwa tej funkcji**.

Inną możliwością jest umieszczenie funkcji podrzędnych wewnątrz pliku funkcji stanowiącej główny program. W takim przypadku funkcje podrzędne powinny być umieszczone poniżej bloku programu głównego oraz tych funkcji które ją wywołują. Takie funkcje podrzędne są dostępne tylko wewnątrz tego pliku programu (gdyż nazwa pliku nie zgadza się z nazwami funkcji podrzędnych i inne programy nie mogą ich znaleźć).

8.10.1. PRZYKŁADY

Przykład PR 79. Funkcja "silnia"

Zdefiniujmy funkcję do obliczania wartości silni z danej liczby:

```
function [wynik]= silnia(n)
% PR 79 – Funkcja silnia(n) wyznacza wartosc n!
wynik=1;
for i=1:n
   wynik=wynik*i;
end
return
```

Po zapisaniu funkcji do pliku o nazwie takiej samej jak nazwa funkcji, czyli: **silnia.m** można ją wywoływać z konkretnymi wartościami argumentu **n**:

```
>> silnia(5)
ans =
      120


>> silnia(9)
ans =
      362880
```

Przykład PR 80. Funkcja "pierwiastek z sumy kwadratów"

Napisanie własnej funkcji może być opłacalne gdy w wyrażeniach powtarzają się te same działania lecz dotyczące różnych danych,. Na przykład we wzorze:

$$p = \frac{1 - \sqrt{9 + x^2}}{\sqrt{4x^4 + 16}}$$

dwukrotnie występuje pierwiastek z sumy kwadratów. Można więc zdefiniować funkcję:

```
function c = pwsk(a,b)
% PR 80 dla dwu liczb danych jako argumenty oblicza pierwiastek z sumy ich kwadratów
c = sqrt(a^2+b^2)
```

Zapisujemy ją do pliku o nazwie **pwsk.m** (bo taką nadaliśmy jej nazwę) aby następnie wykorzystać ją – czyli **wywołać** - dwukrotnie w programie obliczającym wartości podanego wcześniej wyrażenia dla wielu wartości x różnych od zera:

```
x=1
while x ~ = 0
  x = input('x=');
  p = (1 – pwsk(3,x))/(pwsk(2*x^2, 4)
  disp(['p=',num2str(p)]);
end
```

Zauważmy, że **przy definiowaniu** użyto zmiennych “a” i “b” które nie były wcześniej zdefiniowane i nie posiadały określonych wartości ale wystąpiły w nawiasie po nazwie funkcji. Natomiast **przy wywoływaniu funkcji,** zamiast nich są dowolne wyrażenia ale takie, które muszą mieć określone już konkretne wartości.

Przykład PR81. Funkcja z instrukcją IF - rozwiązywanie równania kwadratowego

```
function [x1, x2] = prkw(a, b, c)
% PR 81 - ta funkcja oblicza pierwiastki x1, x2
% rownania: a*x^2 + b*x + c = 0
delta = b*b-4*a*c;
if delta<0
% dla delta<0 podstawimy NaN = "nieokreslone"
        x1=NaN;   x2=NaN
else
        x1=(-b-sqrt(delta))/(2*a);
        x2=(-b+sqrt(delta))/(2*a);
end
```

Ponieważ funkcję nazwano **prkw(a,b,c)** więc należy zapisać ją do pliku o tej samej nazwie czyli: **prkw.m.**

Funkcja **nie zawiera instrukcji wejścia/wyjścia** bo wprowadzenie do niej danych nastepuje przez wartości jakie umieścimy w miejsce parametrów (a,b,c).

Wyniki otrzymujemy jako wektor dwu składnikowy [x1, x2], zgodnie z zasadą że w MATLAB-ie zmienne domyślnie są macierzami a w szczególności wektorami lub skalarami.

Funkcja może być wywołana samodzielnie (z konkretnymi parametrami) ale najczęściej opłaca się ją napisać gdy będzie używana jako cegiełka większego programu.

Sprawdź wartość zmiennej delta po wykonaniu podprogramu (ewentualnie użyj polecenia ***whos***). Przypomnij sobie co to są **zmienne globalne i lokalne** (patrz p. 3.8).

Przykłady bezpośredniego użycia zdefiniowanej przed chwilą funkcji o nazwie **prkw**:

>> [x1, x2]=prkw(1,1,1)
x1 = NaN
x2 = NaN
W tym przypadku brak było pierwiastków rzeczywistych.
>> [x1, x2]=prkw(-1,1,1)
x1 = 1.6180
x2 = -0.6180

8.10.2. FUNKCJE O ZMIENNEJ LICZBIE ARGUMENTÓW

Jak łatwo było zauważyć, jedną ze specyficznych cech MATLAB-a jest **możliwość wywoływania funkcji z różną liczbą argumentów**. Dotyczy to zarówno funkcji standardowych, jak i funkcji definiowanych przez użytkownika. Przykładem funkcji która może być wywoływana z różną liczbą argumentów jest funkcja **plot**, przedstawiona w poprzednim podrozdziale.

Aby napisać funkcję o wielu wariantach działania uzależnionych od liczby dostarczonych argumentów, trzeba skorzystać ze zmiennej systemowej ***nargin*** (nazwa pochodzi od słów: *Number-of-ARGuments-IN* czyli "liczba argumentów wchodzących") podającej właśnie tą liczbę argumentów dostarczonych przy wywołaniu funkcji .

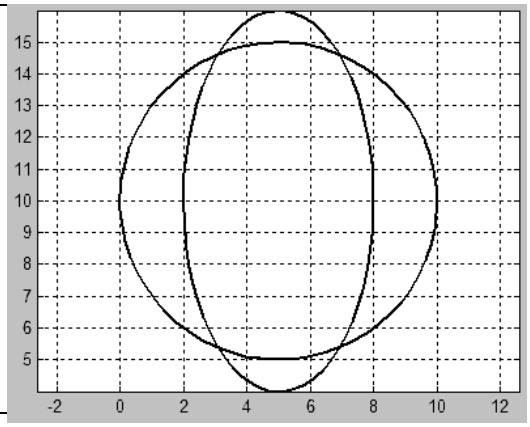
Przykład - chcemy napisać funkcję rysującą okrąg lub elipsę, która posiada następujące argumenty: **Xs** - współrzędna X środka okręgu lub elipsy, **Ys** - współrzędna Y tego środka, **Ra** - promień okręgu lub pierwsza półoś elipsy, **Rb** - druga półoś elipsy.

```
% PR 82 – Funkcja OkrElipsa(Xs,Ys,Ra,Rb)
% Gdy są 3 argumenty rysuje okrąg, a gdy 4 – to elipsę
% Xs, Ys – współrzędne środka,
% Ra – promień okręgu lub pierwsza półoś elipsy,
% Rb – druga półoś elipsy.
function OK=OkrElipsa(Xs,Ys,Ra,Rb)
if nargin<3
    'za malo argumentow'
else
    if nargin<4
        Rb=Ra;
    end
end
```

```
f=0:0.1:6.3;
x=Xs+Ra*cos(f);
y=Ys+Rb*sin(f);
plot(x,y);
axis equal;
grid on
```

Po zapisaniu funkcji **OkrElipsa** do pliku o tej samej nazwie, możemy w oknie komend wywołać objaśnienia komendą **help OkrElipsa**, oraz dwukrotnie wywołać funkcję **OkrElipsa** dla narysowania elipsy a potem okręgu, jak to pokazuje Rys. 8.10.

```
>> help OkrElipsa
Funkcja OkrElipsa(Xs, Ys, Ra, Rb)
 - rysuje okrąg - gdy są trzy
argumenty
 lub elipsę - gdy są 4 argumenty
 Xs - współrzędna X środka okręgu
lub elipsy,
  Ys - współrzędna Y środka okręgu
lub elipsy,
  Ra - promień okręgu lub pierwsza
półoś elipsy,
  Rb - druga półoś elipsy.

>> OkrElipsa(5,10,3,6)
>> hold on
>> OkrElipsa(5,10,5)
```



Rys. 8.10. Przykład wywołania własnej funkcji ze zmienną liczbą parametrów

Na uwagę zasługują dwa nowo wprowadzone polecenia:

axis equal - ustalenie jednakowej długości działek na obu osiach co pozwala odróżnić okrąg od elipsy.

hold on - utrzymanie poprzedniego wykresu gdy w tym samym oknie rysujemy nowy – inaczej nowy zamieniłby stary

8.10.3. DEFINIOWANIE FUNKCJI INLINE

Istnieje także w MATLAB-ie kilka innych sposobów definiowania funkcji. Jednym z nich jest zastosowanie instrukcji ***inline***.

Instrukcja ***inline*** pozwala na przypisanie zmiennej łańcucha odpowiadającego wyrażeniu matematycznemu.

Zmienną tą możemy dalej używać jak odpowiadającą jej funkcję.

Oto przykład:

```
f=inline('1+cos(x)');
f(pi)
fplot(f,[0 pi])
```

8.10.4. ZADANIA – DEFINIOWANIE I WYWOŁYWANIE FUNKCJI

Napisz funkcję do:

Z89. obliczania odległości między dwoma punktami A i B na płaszczyźnie, jeśli **dane** są współrzędne tych punktów: **xa, ya, xb, yb**. Zastosuj tą funkcję do obliczenia długości łamanej złożonej z trzech punktów o danych współrzędnych: (x1, y1); (x2, y2); (x3, y3). Uogólnij zadanie dla N punktów stosując odpowiednią pętlę.

Z90. wyznaczania współrzędnych (Xb, Yb), punktu B otrzymanego jako wynik obrotu (rotacji) na płaszczyźnie XY danego punktu A, wokół środka Xs, Ys, o dany kąt ALFA. Rozbuduj funkcję (lub napisz drugą), tak aby A i B mogły być ciągami punktów. Zastosuj tą funkcję w programie, który wczytuje współrzędne podstawy kwadratu a następnie rysuje ten kwadrat oraz kwadrat obrócony o 60 stopni: a) wokół punktu (0,0), b) wokół wierzchołka kwadratu.

Z91. wyznaczania współrzędnych (Xb, Yb), punktu B otrzymanego jako wynik odbicia symetrycznego, na płaszczyźnie XY, danego punktu A, względem osi symetrii o równaniu y=x. Rozbuduj funkcję (lub napisz drugą), tak aby A i B mogły być ciągami punktów. Przetestuj, tworząc program rysujący łamaną zamkniętą i jej odbicie symetryczne.

Z92. wyznaczania współrzędnych (Xb, Yb), punktu B otrzymanego jako wynik przesunięcia (translacji), na płaszczyźnie XY, danego punktu A, o dany wektor Tx, Ty.

Z93. wyznaczania współrzędnych (Xb, Yb), punktu B otrzymanego jako wynik symetrii środkowej, na płaszczyźnie XY, danego punktu A, względem środka symetrii S(Xs, Ys). Rozbuduj funkcję (lub napisz drugą), tak aby A i B mogły być ciągami punktów. Przetestuj, tworząc program rysujący łamaną zamkniętą i jej odbicie symetryczne.

9. MATLAB – OPEROWANIE NA TABLICACH

Tradycyjnie tablice jedno dwu lub więcej wymiarowe nazywa się w MATLAB-ie macierzami (wiąże się to z nazwą: MATLAB = MATrix LABoratory), choć matematycznie poprawniej byłoby nazywać macierzami tylko tablice dwuwymiarowe. Tak więc określenia „tablica" i „macierz" traktować będziemy jako synonimy.

Tablica to zbiór elementów jednakowego typu identyfikowanych przez wspólną nazwę oraz indywidualne numery. W tablicy dwuwymiarowej, elementy są ustawione w prostokątnym obszarze, w wierszach i kolumnach.

Nazwy wektorów i macierzy nie różnią się od nazw zmiennych skalarnych – bo w MATLAB-ie wszystkie zmienne są tablicami a skalar to tablica o wymiarach 1x1.

Dowolny element macierzy o nazwie ***A*** ma identyfikator:

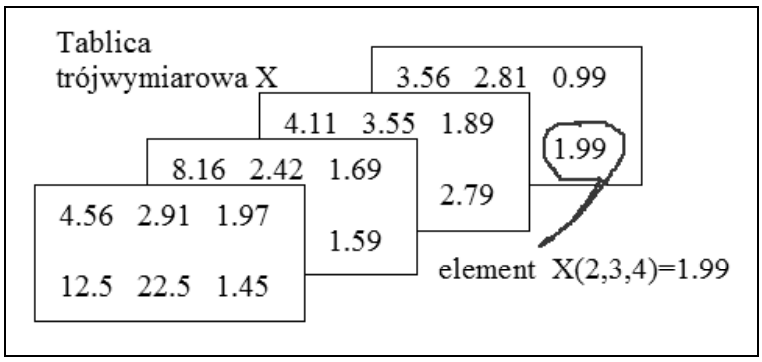
A(numer_wiersza, numer_kolumny)

Indeksy (numery wiersza i kolumny) zapisuje się po nazwie tablicy w nawiasach okrągłych. Każda tablica prostokątna (macierz) ma określoną liczbę wierszy i kolumn co można sprawdzić funkcją ***size***:

[Liczba_wierszy Liczba_kolumn] = **size**(*Macierz*)

Wektor w MATLAB-ie to domyślnie wektor kolumnowy - tablica o jednej kolumnie czyli rozmiarach [n, 1], aczkolwiek można posługiwać się też wektorami wierszowymi – czyli tablicami o jednym tylko jednym wierszu - rozmiar [1, n].

Dostępne są także **tablice o większej liczbie wymiarów**. Tablicę trójwymiarową pokazuje Rys. 9.1



Rys. 9.1. Tablica trójwymiarowa

Zauważmy następującą prawidłowość dotyczącą **kolejności indeksów**: na pierwszym miejscu jest numer wiersza bo **tablica jednowymiarowa** to **wektor kolumnowy** w którym tylko numery wierszy mają sens. Po zwiększeniu liczby wymiarów dochodzą kolejne indeksy: dla dwuwymiarowej tablicy numer kolumny a dla trójwymiarowej jeszcze numer strony: ***Tablica(nr_wiersza, nr_kolumny, nr_strony).***

Stąd można wnioskować, że dla czterowymiarowej tablicy będzie kolejny indeks reprezentujący „numer kostki”.

Przy operowaniu na wektorach i macierzach używane są znaki specjalne objaśnione w tabeli poniżej.

Tabela 9.1. Znaki specjalne używane dla macierzy i w innych rolach

Znaki	Objaśnienie
[ ]	**w nawiasach prostokątnych umieszcza się wartości elementów macierzy** przy czym elementy wiersza oddziela się **przecinkami** lub spacjami natomiast **średnik** odseparowuje poszczególne wiersze, na przykład: Macierz = [12.3, 14.1, 34.5; 2.76, 8.34, 7.12] to macierz o dwu wierszach i trzech kolumnach
{ }	nawiasy klamrowe są używane przy definiowaniu **macierzy komórkowych**
( )	nawiasy okrągłe używamy: 1) dla **indeksów** elementów macierzy czyli numerów wierszy i kolumn; 2) w wyrażeniach; 3) dla argumentów funkcji
:	dwukropek ma kilka znaczeń: 1. **w definicji ciągu** oddziela dwa lub trzy elementy na przykład: **5:2:13** oznacza: "ciąg od 5 z przyrostem 2 do 13" **5:10** oznacza: "ciąg od 5 do 10 domyślnie z przyrostem 1" 2. dwukropek **zamiast wskaźnika** wektora lub macierzy zastępuje wszystkie wartości tego wskaźnika na przykład **A(3,:)** oznacza wszystkie elementy trzeciego wiersza 3. dwukropek zamiast pary wskaźników macierzy np. A(:) to wektor kolumnowy otrzymany ze sklejenia wszystkich kolumn macierzy
=	przypisuje zmiennej wartość wyrażenia n.p.: x=2*sin(pi/6)
.	kropka poprzedza część ułamkową liczby (lub nazwę pola rekordu)
,	przecinek rozdziela 1)indeksy, 2)argumenty funkcji lub 3)poszczególne instrukcje (zamiast zmiany linii)
;	oddziela 1)wiersze macierzy, 2)instrukcje – blokując tzw. “echo”

Pomoc dotyczącą nawiasów można uzyskać wpisując***: help paren***

9.1. DEKLAROWANIE TABLIC

Deklarowanie typu i wymiarów tablic odbywa się automatycznie - przez rozpoznanie rodzaju wpisanych wartości oraz maksymalnych wskaźników. W trakcie wykonywania programu tablica może zmieniać wymiary a także typ wartości. Jeśli elementy tablicy generujemy w pętli to może niepotrzebnie, wielokrotnie, wystąpić operacja powiększania rozmiarów tablicy.

Prześledźmy to na przykładzie generowania elementów wektora:

```
>> for i=1:4; A(i)=2*i; A, end;
A =
     2
A =
     2     4
A =
     2     4     6
A =
     2     4     6     8
```

Aby nie marnować czasu komputera na przedefiniowywanie rozmiarów tablicy wskazane jest, przed generowaniem jej elementów, zarezerwowanie miejsca w pamięci przez wstawienie zera do elementu o maksymalnych wartościach wskaźników.

Na przykład:

```
>> clear
>> B(4)=0, for i=1:4; B(i)=2*i; B, end;
B =
     0     0     0     0
B =
     2     0     0     0
B =
     2     4     0     0
B =
     2     4     6     0
B =
     2     4     6     8
```

Jeszcze lepiej unikać pętli FOR - operujących na elementach macierzy - gdyż MATLAB jest zoptymalizowany dla specyficznych w tym języku operacji na całych macierzach.

Na przykład ten sam efekt co poprzednio można uzyskać bez pętli FOR:

```
>> clear
>> i=1:4; B=2*i
B =
     2     4     6     8
```

Uwagi powyższe dotyczą też tablic o większej liczbie wymiarów. Na przykład można zadeklarować macierz zerując jej ostatni element:

```
>> A(2,3)=0.
A =
    0.    0.    0.
    0.    0.    0.
```

Jeśli potem w programie wystąpią większe wartości wskaźników to macierz zostanie powiększona.

Na przykład instrukcja A(3,5)=2 spowoduje następującą zmianę macierzy:

```
>> A(3,5)=2.
A =
     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.
     0.     0.     0.     0.     2.
```

9.2. SPOSOBY WPROWADZANIA WEKTORÓW I MACIERZY

Wektory i macierze mogą być wprowadzane przez:

1. generowanie wektorów jako **ciągów typu postęp arytmetyczny**
2. generowanie elementów macierzy jako **funkcji ciągów indeksów** - wygenerowanych wcześniej sposobem 1;
3. **wpisywanie** wartości elementów wewnątrz nawiasów prostokątnych np.: *TAB_A=[3.56, 2.56; 0.1, 0.99]*;
4. **generowanie** przez operatory, funkcje i komendy MATLAB-a, np.: *B=ones(2,3)*;
5. **wczytywanie** z pliku dyskowego.

9.2.1. WEKTOR GENEROWANY JAKO POSTĘP ARYTMETYCZNY

Polecenie wygenerowania wektora (ciągu) typu postęp arytmetyczny, ma następującą postać: **nazwa_wektora = pierwszy element : przyrost lub ubytek : ostatni element**

Na przykład:

```
x=0:0.2:1
x =
    0    0.2000    0.4000    0.6000    0.8000    1.0000
```

Jeśli przyrost ma być równy 1 to można go pominąć w zapisie. Na przykład zdefiniujmy ciąg indeksów – który formalnie jest też wektorem:

```
 i = 1:6
 i =
      1     2     3     4     5     6
```

Następnie można zdefiniować wektor o elementach będących funkcją indeksu:

```
>> C(i)=3*i-2
C =
     1     4     7    10    13    16
```

9.2.2. WPROWADZANIE Z KLAWIATURY

Przykładem wektora kolumnowego wpisanego z klawiatury mogą być wyniki pomiarów napięcia w sieci elektrycznej w ciągu doby:

$$U = [220; 221; 220; 218; 218; 219; 220; 221]$$

Aby wprowadzić macierz z klawiatury należy elementy wiersza oddzielać **przecinkami** a poszczególne wiersze oddzielać **średnikami** np.:

```
>> A = [1.7, 2, 3.1, 0.1, 0.5;  7.34, 8.08, 9.1, 10, 0.06]
A =
    1.7000    2.0000    3.1000    0.1000    0.5000
    7.3400    8.0800    9.1000   10.0000    0.0600
```

9.2.3. ROLA DWUKROPKA W WYBIERANIU ELEMENTÓW MACIERZY

Tabela 9.1 podaje role dwukropka. Jedną z nich jest wykorzystanie dwukropka do wybierania elementów tabel.

Dwukropek w miejscu indeksu – oznacza **wszystkie wartości tego indeksu**.

Tak więc - ponieważ zawsze pierwszy indeks to numer wiersza a drugi to numer kolumny, to dla wprowadzonej wcześniej macierzy A:

A(: , 2) – oznacza wszystkie wiersze z kolumny 2:

```
>> A(:,2)
ans =
    2.0000
    8.0800
```

Druga rola dwukropka to **definiowanie zakresu od ... do ...**, na przykład
A(:, 2:4) – oznacza, że wybieramy wszystkie wiersze ale kolumny tylko od 2 do 4:

```
>> A(:, 2:4)
ans =
    2.0000    3.1000    0.1000
    8.0800    9.1000   10.0000
```

Dodatkowo można podać **przyrost** przy definiowaniu zakresu, a więc
A(:,1:2:5) – oznacza, że wybieramy kolumny od 1 **co 2** do 5:

```
>> A(:,1:2:5)
ans =
    1.7000    3.1000    0.5000
    7.3400    9.1000    0.0600
```

Podobnie można wybierać wiersze, na przykład **A(2,:)** – oznacza, że wybieramy drugi wiersz (a kolumny wszystkie):

```
>> A(2,:)
ans =
    7.3400    8.0800    9.1000   10.0000    0.0600
```

Jeśli zastąpimy dwukropkami oba indeksy czyli **A(: , :)** to otrzymamy całą macierz:

```
> A(:, :)
ans =
    1.7000    2.0000    3.1000    0.1000    0.5000
    7.3400    8.0800    9.1000   10.0000    0.0600
```

Wpisanie zamiast **obu indeksów pojedynczego dwukropka** wymusza natomiast **zamianę macierzy na wektor** kolumnowy. Zdefiniujmy nieco mniejszą macierz B:

```
>> B=[7.12, 0.1; 11.2,9.99]
B =
    7.1200    0.1000
   11.2000    9.9900
```

Pojedynczy dwukropek zamiast indeksów sklei kolumny w jeden słupek:

```
>> B(:)
ans =
    7.1200
   11.2000
    0.1000
    9.9900
```

9.2.4. GENEROWANIE MACIERZY

Do generowania pewnych macierzy można stosować funkcje:

zeros(w,k): macierz wypełniona zerami np.:

```
A = zeros(2,3)
ans =
     0     0     0
     0     0     0
```

ones(w,k): macierz wypełniona jedynkami np.:

```
A = ones(2,4)
ans =
     1     1     1     1
     1     1     1     1
```

rand(w,k): macierz liczb pseudolosowych o rozkładzie równomiernym np.:

```
A = rand(2,5)
ans =
    0.9501    0.6068    0.8913    0.4565    0.8214
    0.2311    0.4860    0.7621    0.0185    0.4447
```

eye(N): macierz jednostkowa (kwadratowa N x N z jedynkami na przekątnej głównej) np.:

```
A = eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

Wiele innych macierzy można generować programami z użyciem pętli lub ciągów indeksów.

9.2.5. WCZYTYWANIE MACIERZY Z PLIKU

Załóżmy, że mamy plik tekstowy o nazwie DANE1.TXT a w nim są dwie linie i w każdej po 4 liczby oddzielane odstępami:

2 4 6 8
3 6 9 12

Aby wczytać te liczby do macierzy można napisać następujące instrukcje:

```
[plik1 info] = fopen('DANE1.TXT');
A = fscanf(plik1, '%f %f %f %f', [4, 2])
close(plik1)
```

Ale **UWAGA: dane czytane są z pliku wierszami ale umieszczane w macierzy kolumnami,** dlatego po wczytaniu uzyskamy macierz:

```
A =
     2     3
     4     6
     6     9
     8    12
```

Aby uzyskać to samo co w pliku trzeba macierz transponować (jak w p.9.2.6.)

9.2.6. PODSTAWOWE OPERACJE NA MACIERZACH

Jak już wspomniano w p.8.3.6. , ponieważ MATLAB-a zaprojektowano głównie z myślą o macierzach, więc domyślnie operatory arytmetyczne (`+, -, *, /, ^`) są operatorami działań na macierzach, natomiast operatory poprzedzone kropką (`.+, .-, .*, ./, .^`) – zwane czasem tablicowymi - to operatory dotyczące elementów lub odpowiadających sobie par elementów dwu macierzy, które muszą wówczas mieć jednakowe wymiary, chyba, że jedna z nich jest skalarem.

W przypadku gdy oba operandy są skalarami (a więc także elementami macierzy), wtedy jest obojętne czy zastosujemy operator z kropką czy bez, a początkujący mogą spokojnie nie wiedzieć wówczas o istnieniu operatorów z kropką.

Generalnie, działania z kropką dotyczą dwu macierzy o identycznych rozmiarach lub operacji – jak potęgowanie - dokonywanych na wszystkich elementach danej macierzy.

Dla przypadków gdy jeden z operandów jest macierzą a drugi skalarem, najlepiej przyjąć zasadę stosowania operatorów bez kropki. Wyjątkami – czyli działaniami **niewykonalnymi dla skalara i macierzy** – są wówczas:

- dzielenie skalara prawostronnie przez macierz – na przykład: **`5/Macierz`**
- dzielenie macierzy lewostronnie przez skalar – na przykład: **`Macierz/5`**

Tabela 9.2. Operatory arytmetyczne w MATLAB-ie

Operator	Objaśnienie
+ lub **.+**	**dodawanie** skalarów lub odpowiadających sobie elementów dwu macierzy o identycznych rozmiarach
− lub **.−**	**odejmowanie** (jak wyżej) lub **zmiana znaku**
*****	**mnożenie macierzowe** (opisane w p.5.12.2.) - dla **A*B** liczba kolumn macierzy **A** musi być równa liczbie wierszy macierzy **B**
/	**dzielenie macierzowe prawostronne**: **X=A / B** (**A** dzielone prawostronnie przez **B** jest rozwiązaniem równania: **X*B=A**)
****	**dzielenie macierzowe lewostronne**, **X=A\B** (**B** dzielone lewostronnie przez **A** - jest rozwiązaniem równania **A*X=B;** zamiast: **inv(A)*B** lepiej użyć **A\B**)
^	**potęgowanie macierzowe A^2 = A*A** (mnożenie macierzowe macierzy kwadratowej przez identyczną macierz)
.*	**mnożenie tablicowe** czyli odpowiadających sobie par elementów macierzy o identycznych rozmiarach (opisane w p.5.12.2.)
./	**dzielenie tablicowe** - par elementów dwu macierzy, o identycznych rozmiarach
.^	**potęgowanie tablicowe** - dotyczy każdego elementu macierzy
.'	**transponowanie** macierzy (zamiana wierszy na kolumny)
'	**sprzężenie** (dla liczb zespolonych) lub transpozycja (dla liczb rzeczywistych)

Dla zilustrowania niektórych działań przeprowadźmy - w oknie komend MATLAB-a - kilka eksperymentów. Niech **j** oraz **c** będą wektorami wierszowymi czyli zarazem macierzami prostokątnymi o identycznych rozmiarach 1x3:

```
>> clear
>> j=1:2:5, c=[1.5, 0.1, 2.5]
j =
     1     3     5
c =
    1.5000    0.1000    2.5000
```

Przetestujmy operator mnożenia z kropką (lewa kolumna) i bez (prawa kolumna):

```
>> j.*c
ans =
    1.5000    0.3000   12.5000
```

```
>> j*c
??? Error using ==> mtimes
Inner matrix dimensions must
agree.
```

Mnożenie macierzowe (bez kropki), opisane w p.5.12.2. , wymaga aby liczba kolumn w pierwszej macierzy była równa liczbie wierszy w drugiej – stąd komunikat o błędzie.

Podobnie będzie przy próbie potęgowania macierzowego (operator bez kropki):

```
>> j.^2
ans =
     1     9    25
```

```
>> j^2
??? Error using ==> mpower
Matrix must be square.
```

Tym razem w komunikacie o błędzie napisano: “Macierz musi być kwadratowa”.

Transponowanie - to operacja polegająca na zamianie wierszy macierzy na kolumny. Operatorem transponowania jest w MATLAB-ie kropka i apostrof [.'], ale jeśli nie stosujemy liczb zespolonych to wystarcza sam apostrof ['].

Przykładowo jeśli:

```
A =
    2    3
    4    6
    6    9
    8   12
```

to macierz transponowana:

```
A' =
    2    4    6    8
    3    6    9   12
```

Suma i różnica macierzy wymagają identycznych rozmiarów obu macierzy i są działaniami na tyle oczywistymi, że nie będziemy tu dawać przykładów – przetestuj samodzielnie.

Dzielenie, lewo i prawostronne macierzy, będzie natomiast omówione przy rozwiązywaniu układów równań liniowych.

9.2.7. UKŁAD RÓWNAŃ LINIOWYCH. ODWRACANIE ORAZ DZIELENIE MACIERZY

Załóżmy że układ równań liniowych (p.5.12.3.) doprowadziliśmy do postaci macierzowej: zapisanej (w opisie a nie w MATLAB-ie) jako: **A*X=B**

gdzie: **A**=macierz współczynników przy niewiadomych,
X=wektor niewiadomych,
B= wektor wyrazów wolnych

Wtedy rozwiązanie czyli wektor niewiadomych **X** wyznaczamy przez lewostronne pomnożenie obu stron równania przez macierz odwrotną do A zapisywaną w MATLAB-ie jako **inv(A)**:

$$\text{inv}(A)*A*X = \text{inv}(A)*B$$

a ponieważ iloczyn macierzy danej i odwrotnej jest macierzą jednostkową którą można pominąć więc rozwiązanie dowolnego ukladu równań liniowych otrzymamy przy pomocy jednego wzoru:

X=inv(A)*B

Jednakże MATLAB nie zaleca stosowania funkcji **inv(..)** a zamiast niej poleca **dzielenie lewostronne macierzy** (operator „\” w odróżnieniu od dzielenia prawostronnego „**/**”) jako mniej pracochłonne dla komputera i mogące w większości przypadków zastąpić odwracanie macierzy.

W szczególności dla naszego układu równań liniowych stosując lewostronne dzielenie mamy (w opisie): **A\A*X=A\B** co po uproszczeniu trzeba zapisać w MATLAB-ie jako:

X=A\B

MATLAB stosuje wówczas wydajniejszą metodę eliminacji Gaussa zamiast pracochłonnego odwracania macierzy, co skraca czas obliczeń 2 do 3 razy i poprawia dokładność.

9.2.8. ĆWICZENIA

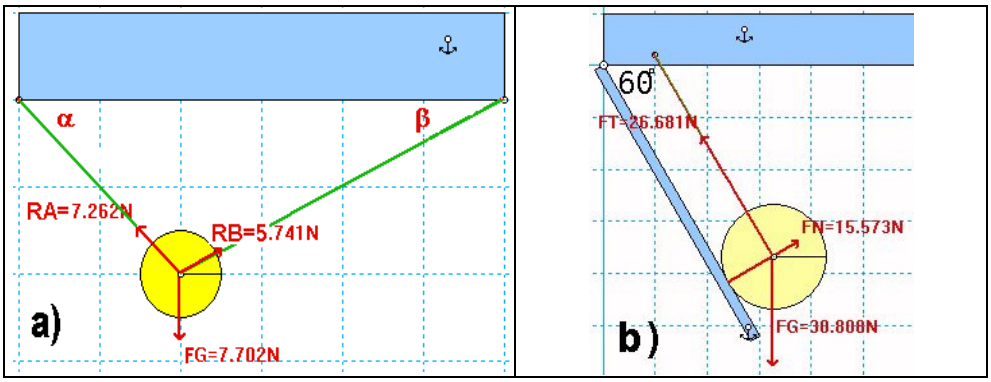
Z94. Rozwiąż w MATLAB-ie układ równań liniowych mając daną macierz współczynników M oraz wektor wyrazów wolnych C:

$$M = \begin{bmatrix} -2 & 0.5 & 4.2 & 8 \\ 0 & 4 & 8 & 2 \\ -5 & 7 & 3 & 1 \\ 10 & 12 & -6 & 4 \end{bmatrix} \quad C = \begin{bmatrix} 73.5 \\ 15.2 \\ -33 \\ 5 \end{bmatrix}; \qquad Rozw.: \; X = \begin{bmatrix} 3.258 \\ -4.499 \\ 1.817 \\ 9.329 \end{bmatrix}$$

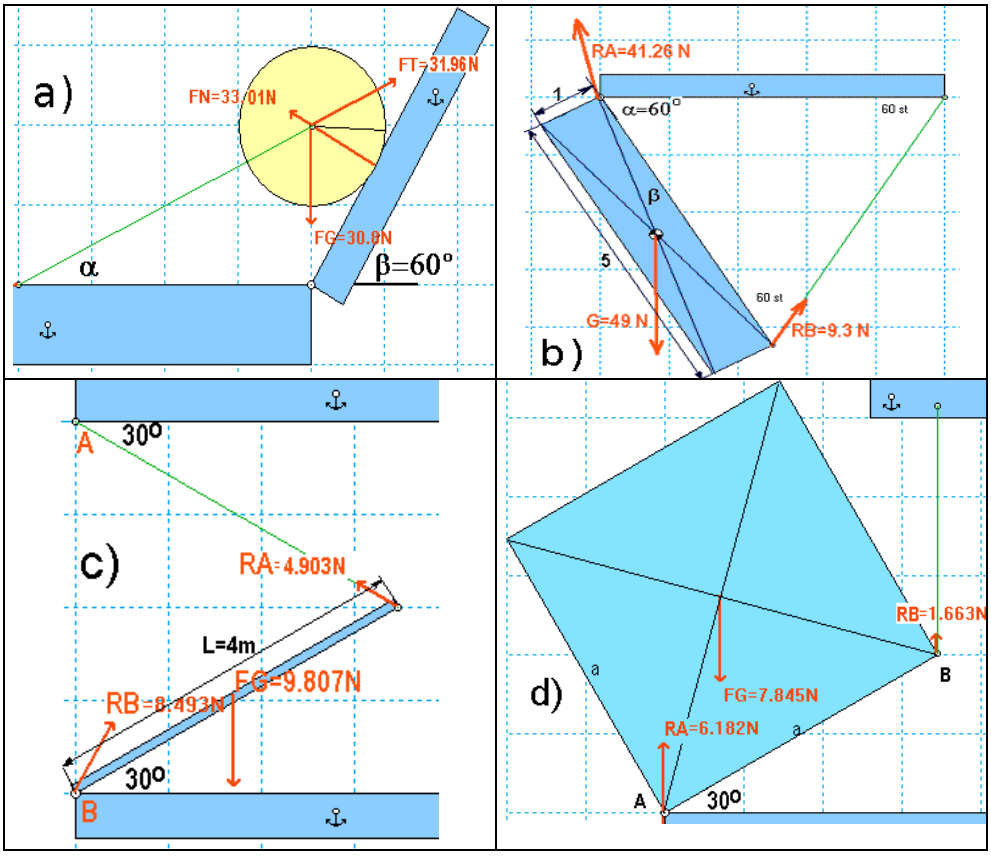
Z95. Dla poprzednich macierzy wpisz odpowiednią komendę aby wyswietlić:

a) macierz transponowaną względem M oraz C
b) wyznacznik macierzy M używając funkcji **det**(*macierz kwadratowa*)
c) pierwszą a potem drugą kolumnę macierzy M (wyrażenie z dwukropkiem)
d) elementy macierzy M z indeksami 1,1 oraz 2,2
e) liczbę elementów wektora C przy pomocy funkcji **length**(*wektor*)
f) rozmiary macierzy M oraz C przy pomocy funkcji **size**(*macierz*)

Z96. Mając daną wartość obciążenia FG, wyznacz siły reakcji w układzie wybranym z rysunków 9.2-9.3, korzystając z równań statyki. W przypadkach gdy na rysunku brak liczbowych wartości potrzebnych kątów, należy wyznaczyć je na podstawie widocznej siatki. Otrzymane wyniki porównaj z wynikami pokazanymi na tych rysunkach (uzyskanymi programem WorkingModel).



Rys. 9.2. Układy do obliczeń statyki – część 1



Rys. 9.3. Układy do obliczeń statyki – część 2

Z97. Napisz programy z pętlami FOR...END wykonujące: (a) mnożenie macierzowe, (b) mnożenie tablicowe dwu macierzy A i B i sprawdź zgodność wyników działań twoich programów z działaniami MATLAB-a: **A*B** oraz **A .* B**

9.3. WYKRESY FUNKCJI DWU ZMIENNYCH

Rysowanie wykresów trójwymiarowych lub konturowych dla funkcji dwu zmiennych Z(X,Y) w najprostszym przypadku przebiega dwuetapowo:

- przygotowanie siatki par współrzędnych **(x,y)** dla funkcji **z=f(x,y)** przy pomocy funkcji ***meshgrid***
- użycie jednej z wielu funkcji dla wykresów trójwymiarowych

Funkcji ***meshgrid*** podajemy jako argumenty wektory (ciągi) wartości **x** oraz **y** a w wyniku uzyskujemy dwie macierze zawierające łącznie wszystkie pary współrzędnych dla których mają być wyznaczane wartości funkcji zmiennych x,y. Na przykład:

```
>> x=0:0.1:0.3, y=1:3
x =
     0    0.1000    0.2000    0.3000
y =
     1     2     3

>> [X Y]=meshgrid(x,y)
X =
     0    0.1000    0.2000    0.3000
     0    0.1000    0.2000    0.3000
     0    0.1000    0.2000    0.3000
Y =
     1     1     1     1
     2     2     2     2
     3     3     3     3
```

Uzyskanie macierze mogą służyć jako argumenty dla funkcji Z(X,Y), która będzie wyliczana dla każdej pary X(w,k), Y(w,k). W powyższym przykładzie: (0, 1); (0.1, 1); (0.2, 1) ; (0.3, 1); (0, 2); (0.1, 2); (0.2, 2) ; (0.3, 2); ... i tak dalej.

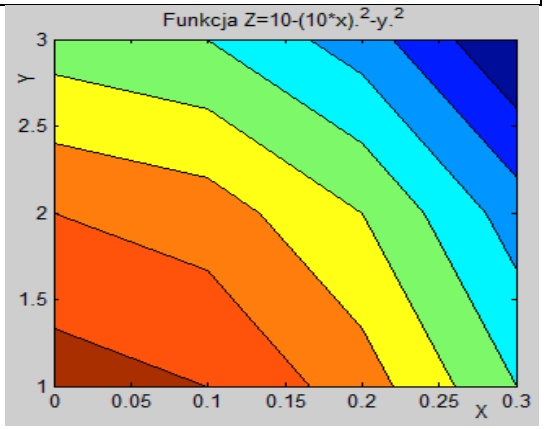
```
>> Z=10-(10*X).^2-Y.^2
Z =
    9.0000    8.0000    5.0000         0
    6.0000    5.0000    2.0000   -3.0000
    1.0000         0   -3.0000   -8.0000
```

Jak widać – **trzeba użyć potęgowania tablicowego** (z kropką).

Ostatecznie polecenia:

```
>> contourf(X,Y,Z); title('Funkcja Z=10-(10*x).^2-y.^2');
>> xlabel('X');ylabel('Y');
```

wyświetlą wykres warstwicowy wypełniony kolorami:



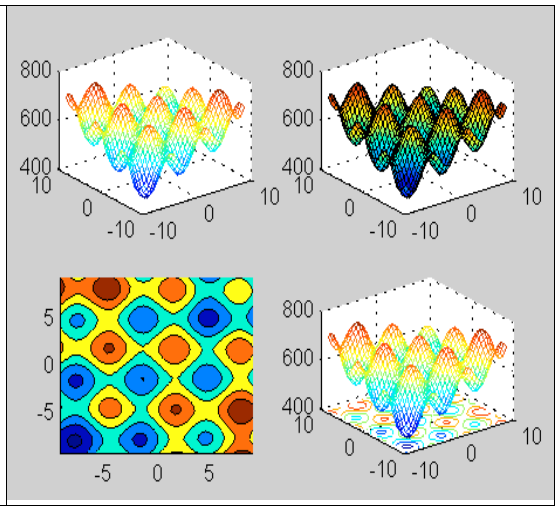
Rys. 9.4. Przykładowy wykres warstwicowy

Uzyskane macierze X, Y, Z mogą służyć jako argumenty dla takich funkcji generujących wykresy jak:

- **mesh**(X,Y,Z) – wykres siatkowy 3D, lub
  meshc(X,Y,Z) – wykres siatkowy z dodatkowymi warstwicami pod nim,
- **surf**(X,Y,Z) – wykres powierzchniowy 3D,
- **contour**(X,Y,Z) – wykres warstwicowy z samymi liniami poziomic,
- **contourf**(X,Y,Z) – wykres warstwicowy wypełniony kolorami,
- i innych

MATLAB posiada bardzo wiele funkcji dla wizualizacji linii i powierzchni trójwymiarowych. Niektóre z nich pokazuje przykład na Rys. 9.5.

```
% PR 83
% ciągi argumentów:
X = -3*pi:0.5:3*pi; Y=X;
% siatka punktów dla wykresu 3D:
[x,y]=meshgrid(X,Y);
% następnie definiujemy funkcję z(x,y):
z = 600 – x .* y + 50 * sin(x) + 50 * sin(y);
% przy pomocy funkcji subplot
% wybieramy ćwiartki okna graficznego
% i wyświetlamy w nich wykresy 3D:
% 1) wykres siatkowy:
subplot(2,2,1); mesh(x,y,z);
% 2) wykres powierzchniowy:
subplot(2,2,2); surf(x,y,z);
% 3)wykres warstwicowy:
subplot(2,2,3); contourf(x,y,z);
% 4)wykres siatkowy z warstwicami:
subplot(2,2,4); meshc(x,y,z);
```



Rys. 9.5. Przykłady wykresów funkcji dwu zmiennych

9.4. TABLICE KOMÓREK

Tablica komórek (*cell array*) - to tablica w której każdy element może być innego typu a w tym także typu złożonego. W odróżnieniu od tablic liczbowych, wpisywaną **zawartość tablic komórkowych ujmujemy w nawiasy klamrowe { }** (a nie prostokątne). Również przy odwoływaniu się do elementów tych tablic podajemy indeksy w nawiasach klamrowych.

Jako pierwszy przykład zdefiniujmy tablicę komórkową z nazwami przedmiotów i ocenami uzyskanymi z tych przedmiotów przez pewnego studenta:

```
>> oc={'fizyka', 3.5; 'chemia', 4.0; 'rysunek',5}
```

```
oc =
    'fizyka'     [3.5000]
    'chemia'     [     4]
    'rysunek'    [     5]
```

Z kolei, tą tablicę z ocenami możemy wstawić jako element do bardziej złożonej tablicy ST z danymi studenta, gdzie podane będą: imię, nazwisko, wiek, oceny oraz dokonywane w tym roku wpłaty:

```
>> ST={'Jan Kowalski', 23; oc, [230, 120, 600]}
ST =
    'Jan Kowalski'    [        23]
        {3x2 cell}    [1x3 double]
```

Dostęp do danych w tablicy komórkowej jest możliwy przez podawanie kolejnych poziomów indeksów, na przykład:

```
>> ST{2,1}{3,1}
ans =
   rysunek
```

Dane następnych studentów możemy gromadzić na następnych stronach trójwymiarowej tablicy ST:

```
>> ST(:,:,2)={'Adam Kos', 25;{},[]}

ST(:,:,1) =
    'Jan Kowalski'    [        23]
        {3x2 cell}    [1x3 double]
ST(:,:,2) =
    'Adam Kos'    [25]
              {}      []
```

A więc mamy dwie strony tablicy trójwymiarowej: pierwsza z danymi pana Kowalskiego a druga z niekompletnymi jeszcze danymi pana Kosa.

Jednym ze sposobów wprowadzania danych do tablicy komórkowej oraz strukturalnej może być pokazane niżej zastosowanie polecenia inputdlg:

```
% Program PR 84 wprowadza dane do tablicy komórkowej: osoba
% oraz do tablicy struktur o nazwie: baza
clear; clc
pola={'Imie'; 'Nazwisko'; 'Wiek'};
we={'...'};
i=1;
while ~isempty(we)
 we = inputdlg(pola, ['Student ',num2str(i)]);
 if ~isempty(we)
     osoba{i}=we;
     osoba{i}
     baza(i)=cell2struct(osoba{i},pola,1)
```

```
      i=i+1;
  end
end
```

Dostęp do elementów tablicy komórkowej mamy tylko przez indeksy. Nieco wygodniejszy może wydawać się dostęp do elementów **tablicy struktur**, gdzie po nazwie tablicy i po indeksach jej elementu podajemy **oddzieloną kropką nazwę pola.**

9.5. TABLICE STRUKTUR

Tablica struktur to tablica, której elementami są **rekordy** (jak w bazie danych) złożone z pól przechowujących wartości różnego typu i identyfikowanych przez swoje nazwy. Rekordy te nazywają się w MATLAB-ie strukturami (typ: *struct*).

Tablice struktur można definiować na dwa sposoby:

a) Definiowanie z użyciem „konstruktora" czyli funkcji *struct* ma postać:

```
nazwa_tablicy = struct('nazwa_pola1', wartość_pola1, 'nazwa_pola2', wartość_pola2, ...)
```

Przykład:

```
>> student = struct('Nazwisko', 'Kowalski',  'Imie', 'Jan',  'Wiek', 23)

student =
    Nazwisko: 'Kowalski'
        Imie: 'Jan'
        Wiek: 23
```

Zdefiniowanie zmiennej **student** jako ***struktury*** o określonych ***polach*** i ich **wartościach** utworzy na razie jednoelementową tablicę strukturalną, której elementem jest rekord dotyczący konkretnego studenta.

b) Definiowanie przez przypisywanie wartości kolejnych pól.

Nazwy pól oddzielamy kropką od nazwy tablicy lub jej elementu.

W ten sposób możemy definiować pola nowej struktury jak i dodawać nowe pole do już istniejącego rekordu:

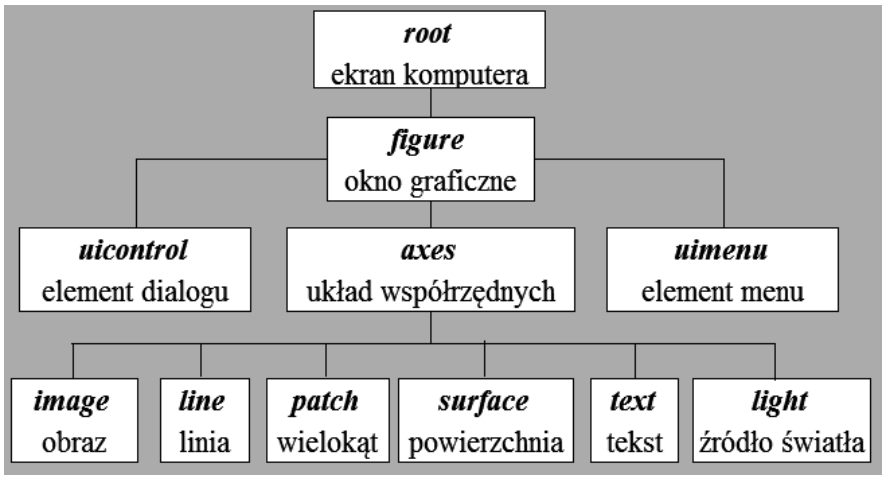
```
>> student.grupa=1

student =
    Nazwisko: 'Kowalski'
        Imie: 'Jan'
        Wiek: 23
       grupa: 1
```

Możemy także definiować kolejne elementy-rekordy dotyczące kolejnych studentów:

```
>> student(2).Nazwisko='Jankowski'
student =
    1x2 struct array with fields:
      Nazwisko
      Imie
      Wiek
      grupa

>> student(2)    % wyświetli zawartość drugiego elementu tablicy struktur
ans =
      Nazwisko: 'Jankowski'
          Imie: []
          Wiek: []
         grupa: []
```

10.MATLAB - OBIEKTY I PROGRAMOWANIE W TRYBIE GRAFICZNYM

Okna graficzne (figure), w których pojawiają się wszelkiego rodzaju wykresy, mogą być także wykorzystywane do tworzenia programów dialogowych z graficznym interfejsem użytkownika (*Graphic User Interface* - ***GUI***). Można w nich wówczas rozmieszczać graficzne obiekty dialogowe jak: menu, przyciski, listy rozwijalne i inne - obsługiwane najczęściej myszką. Rozmieszczanie i modyfikowanie obiektów graficznych może być zrealizowane albo przy pomocy poleceń programu albo przy pomocy myszki i wykorzystania kreatora zwanego GUIDE.



Rys. 10.1. Podstawowe typy obiektów graficznych w MATLAB-ie

Wykresy i inne obiekty graficzne w MATLAB-ie tworzą strukturę hierarchiczną pokazaną na Rys. 10.1. Struktura ta wraz z regułami jej dotyczącymi nazywa się **grafiką uchwytów** (*handle graphics*). Znajomość jej podstaw jest przydatna zarówno przy programowym generowaniu i modyfikowaniu wykresów jak i budowaniu programów z interfejsem graficznym.

Formatowanie i modyfikowanie wykresów można realizować albo przy pomocy poleceń, albo myszką, w trybie edycji, po wciśnięciu w oknie wykresu przycisku ze strzałką i korzystaniu z menu kontekstowego lub menu górnego (głównego).

Najistotniejsze zasady dotyczące obiektów graficznych są następujące:

1. Wszelkie obiekty graficzne powstają w **oknach graficznych** nazywanych **`figure`**. Okien takich może być wiele. Okno powstaje samoczynnie przy wykonywaniu poleceń graficznych lub może być utworzone poleceniem:

   `figure` albo *zmienna* = **`figure`**.

2. Przy tworzeniu zarówno okna jak każdego obiektu powstaje jego unikalny identyfikator zwany **uchwytem** (*handle*) .
   Uchwyt - jest to wielocyfrowy numer, który należy przypisać zmiennej jeśli chcemy się nim posługiwać, natomiast gdy nie posługujemy się uchwytem to operacje dotyczą zawsze "aktywnego" (bieżącego) elementu graficznego czyli tego który był przed chwilą utworzony albo ostatnio kliknięty. Na przykład komendy:

   `grid on, ylabel('sin(x)')`

   - włączą siatkę i utworzą opis osi Y na ostatnio utworzonym wykresie.

3. Jeśli nie przechowaliśmy uchwytu w zmiennej przy tworzeniu obiektu to możemy jeszcze korzystać z funkcji:

   `gcf` - uchwyt bieżącego okna (*get current figure*),

   `gca` - uchwyt bieżącego układu współrzędnych (*get current axes*),

   `gco` - uchwyt bieżącego obiektu (*get current object*),

4. Wszystkie **funkcje tworzące obiekty graficzne** mają podobną składnię:

   uchwyt = obiekt('Cecha1', Wartość1, 'Cecha2', Wartość, ...)

   przykład:

```
>>F1=figure('Position',[5 40 790 300], 'Name', 'TESTOWANIE');
```

5. Każdy obiekt ma określone własności, które można sprawdzać i pobierać ich wartości przy pomocy funkcji **get** oraz modyfikować (ustawiać) przy pomocy funkcji **set:**

```
>> get(F1,'Color')
ans =
    0.8000    0.8000    0.8000
>>set(F1,'Color', [0.9, 0.7, 0.95], 'MenuBar', 'none');
```

6. Każda własność ma swoją wartość domyślną, która będzie zastosowana jeśli użytkownik nie poda innej, na przykład kolor okna graficznego domyślnie będzie szary: [0.8 0.8 0.8].

7. Użytkownik może określać własności albo przy tworzeniu obiektu (patrz punkt 4), albo później korzystając z funkcji **set** oraz uchwytu obiektu – jak w p.5:

   `set`*(uchwyt_obiektu, 'Nazwa_cechy', wartość, 'Nazwa_cechy', wartość, . . .)*

8. Każdy obiekt reaguje na określone **zdarzenia**, którym można przypisywać określone **procedury** albo **funkcje** – zwane ogólnie CallBack (wywołania zwrotne) – realizujące potrzebne w danej sytuacji operacje,

9. Obiekty graficzne podlegają hierarchii określającej który obiekt jest dla danego nadrzędny (*parent* = rodzic) a który podrzędny (*child* = dziecko, potomek).

10.1. OKNO GRAFICZNE - *FIGURE*

Jeśli żadne okno graficzne (*figure*) nie było otwarte to przy tworzeniu wykresu samoczynnie powstaje nowe okno graficzne, jeśli jednak będziemy budować interfejs graficzny programu lub chcemy aby kolejny wykres powstał w nowym oknie to musimy je utworzyć komendą:

`figure` lub *`zmienna`* **`= figure;`**

Ta druga postać komendy, nie tylko tworzy i wyświetla okno ale również zapamiętuje jego identyfikator czyli **uchwyt** (*handle*) w *zmiennej*. Przykład:

```
F1 = figure;
```

Uchwyt F1 może być potem wykorzystywany zarówno do aktywacji tego okna (aby stało się „bieżącym" jeśli nie było ostatnio wygenerowane):

```
figure(F1)
```

Komenda ta zastępuje wybranie okna F1 przez kliknięcie myszką w przypadku gdy otwartych jest kilka innych okien. Uchwyt F1 potrzebny jest także gdy poleceniami chcemy zmienić cechy tego okna np.:

Przykład:

```
>>set(F1,'Position',[5  40  790  300],  'Name',  'TESTOWANIE'...
        , 'Color', [0.5, 0.5, 0.95], 'MenuBar', 'none');
```

- ustali położenie i rozmiary okna na ekranie, nazwę `'TESTOWANIE'`, która pojawi się u góry w pasku tytułowym, kolor tła (niebieski) oraz wyłączy standardowe menu.

Aby wyświetlić wykaz wszystkich cech okna graficznego oraz możliwych ustawień tych cech, trzeba użyć funkcji **set** podając tylko identyfikator okna jako parametr np.:

```
set(F1)
```

Funkcja **set** czyli „ustaw" służy więc do ustawienia lub zmiany wartości cech, natomiast odwrotną rolę ma funkcja **get** czyli „pobierz" – która odczytuje aktualne wartości cech. Na przykład **get(F1)** – wyświetli wszystkie nazwy cech obiektu **F1** i ich aktualne wartości.

Okno graficzne FIGURE - tak jak i pozostałe obiekty - ma wiele cech, niestety zbyt dużo aby je zapamiętać, ale trzeba znać przynajmniej niektóre często stosowane jak:

- ***Name*** - nazwa, która pokaże się w pasku tytułowym formatki i dobrze ją zmienić na nazwę naszego tworzonego programu,
- ***Tag*** – nazwa używana wewnątrz programu do której mogą odwoływać się instrukcje wykorzystujące lub zmieniające własności obiektu,
- ***Position*** - [Xp, Yp, Dx, Dy] - położenie lewego dolnego rogu Xp, Yp, oraz rozmiary Dx,Dy - domyślnie w pikselach ale można zadać inne jednostki zmieniając cechę ***Units***,

- cecha ***Units*** może mieć jedną z wartości:
  pixels | *normalized* | *inches* | *centimeters* | *points* | *characters.*
  Domyślnie: *'pixels'*. Wszystkie jednostki określają położenie względem lewego dolnego rogu okna. Przy użyciu jednostek *'normalized'* - lewy dolny róg okna ma współrzędne (0,0) a prawy górny (1,1)
- ***Color*** - kolor tła o składowych [red green blue] określany trzema liczbami z zakresu 0 do 1,
- ***MenuBar*** - który jeśli ma wartość *'figure'* jest menu standardowym, a gdy *'none'* to brak menu.

Polecenie ***help figure*** wyświetli ogólne objaśnienia dotyczące okna graficznego.

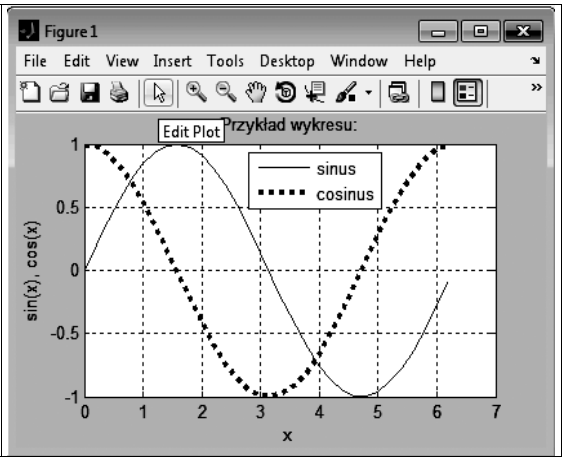
Jak wspomniano, okno ***figure*** może być używane zarówno do wykresów – które pojawią się w obiektach ***axes*** czyli układach współrzędnych – ale także do tworzenia dowolnych programów dialogowych, wykorzystujących różnorodne elementy sterujące - ***uicontrols*** - oraz elementy do budowy menu - ***uimenu***.

10.2. WYBRANE OBIEKTY I FUNKCJE GRAFICZNE

Wykresy – tak jak i inne obiekty graficzne są wyświetlane w oknach graficznych ***figure***, ale dodatkowo umieszczone są w **regionie** (okienku) **wykresu**, który jest obiektem graficznym o nazwie ***axes***. ***Axes*** jest potomkiem ***figure*** i posiada obiekty potomne: ***image, light, line, patch, surface, text***.

Wszystkie polecenia rysujące wykresy jak ***plot, surf, mesh, bar***, automatycznie tworzą obiekt ***axes***, jeśli dotychczas nie istnieje. Jeśli natomiast istnieje jeden lub więcej obiektów ***axes*** to zawsze jeden z nich (domyślnie ostatnio utworzony) jest aktywny ("bieżący") i w nim pojawi się efekt komendy rysującej wykres.

```
% PR 85
x=0 : 0.2 : 2*pi;
plot(x,sin(x));
grid on
title('Przykład wykresu:');
xlabel('x');
ylabel('sin(x), cos(x)');

% a teraz drugi wykres
hold on % w tym samym układzie
P=plot(x,cos(x), ':m');
set(P,'Linewidth',3);
legend('sinus','cosinus');
```



Rys. 10.2. Wykresy XY. Widoczne okienko ***axes*** z białym wypełnieniem i siatką współrzędnych

Obiekt ***line*** tworzony jest przez większość poleceń rysujących wykresy, a w szczególności polecenie ***plot***. Możliwość ustawiania wartości cech obiektu ***line*** może być przydatna na przykład wtedy, gdy chcemy zmienić grubość (Rys. 10.2), typ lub kolor linii wykresu przy pomocy programu (a nie myszką). Pokazuje to przykład programu rysowania wykresów typu XY dla dwu funkcji na Rys. 10.2. Wykresy rysowane są przy pomocy oddzielnych wywołań funkcji ***plot***. Jak widać drugie wywołanie ***plot*** musi być poprzedzone poleceniem ***hold on*** jeśli drugi wykres nie ma zastąpić pierwszego lecz ma z nim współistnieć. W zmiennej ***P*** zapamiętany zostaje uchwyt do lini drugiego wykresu, dzięki czemu możemy programowo, poleceniem ***set*** odwołać się do tego wykresu i zmienić grubość jego linii (czyli cechy *Linewidth*) na 3.

Wciśnięcie przycisku ze strzałką (Rys. 10.2) umożliwia ręczną edycję wykresu (*Edit Plot*) czyli wybieranie (myszką) i modyfikowanie elementów wykresu, w tym również grubości linii. Cechy okna i układu współrzędnych można natomiast zmieniać wykorzystując menu i pasek narzędzi okna graficznego. A więc tak naprawdę, wiedza o „grafice uchwytów" staje się niezbędna dopiero przy budowaniu programów z interfejsem graficznym, wykorzystującym dialogowe elementy sterujące - UICONTROLS - jak przyciski, suwaki, listy rozwijalne i inne, które krótko nazywać będziemy „kontrolkami".

Podstawowe informacje o wykresach typu XY – m.in. funkcjach: ***plot, bar, subplot, grid, title, xlabel, ylabel*** - zamieszczono wcześniej, w p.8.9. , natomiast ważniejsze wykresy funkcji dwu zmiennych omówiono w p.9.3. . Tabela 10.1 zawiera opis innych, aczkolwiek nie wszystkich, funkcji dla wykresów. Dodatkowo każda z nich może mieć wiele wariantów wywołań.

Tabela 10.1. Wybrane funkcje dla wykresów

Polecenie lub funkcja	Opis
text(*x, y*, '*napis*')	Dodaje napis w układzie współrzędnych
legend(*s1, s2, s3,* **...)**	Legenda: *s1, s2, ...* to opisy do poszczególnych wykresów
fplot('*wyrażenie***',***[xmin, xmax]***)**	Wykres wyrażenia z jedną zmienną w podanym zakresie
ezplot('*wyrażenie***',***[xmin, xmax]***)**	Wykres wyrażenia z jedną zmienną w podanym zakresie
ezpolar('*wyrażenie***')**	Wykres biegunowy w zakresie 0 do 2π, np.: ezpolar('1+cos(t)')
hold on	Pozwala dodawać kolejne wykresy w tym samym układzie
axis([*xmin xmax ymin ymax***])**	Ustawia zakresy dla osi x oraz y
axis equal	Jednakowe skale na obu osiach
polar(*kąt, promień, typ_linii*)	Wykres biegunowy
loglog(*x, y, s*)	Wykres o skalach logarytmicznych na obu osiach
semilogx(*x, y*, *s*)	Wykres o skali logarytmicznej na osi x, s = typ linii
semilogy(*x, y, s*)	Wykres o skali logarytmicznej na osi x
plot3(*x, y, z, s*)	Wykres liniowy w trzech wymiarach
mesh(*x, y, z*)	Wykres - siatka 3D
surf(x, y, z)	Wykres - powierzchnia 3D

10.3. ELEMENTY STERUJĄCE – *UICONTROL'S*

Obiekty ***uicontrol*** czyli w polskim tłumaczeniu "elementy sterujące interfejsu użytkownika" (bo **ui** to *user interface*) będziemy krótko nazywać "kontrolkami".

Budowanie interfejsu graficznego trzeba zacząć od opisanego już definiowania okna graficznego i określenia jego cech. Następnie trzeba zaplanować jakie kontrolki będą nam potrzebne i zająć się ich definiowaniem.

Polecenie:

zmienna = **uicontrol**(F1, *'Nazwa_Cechy', Wartość, 'Nazwa_Cechy', Wartość, ...*)

- tworzy kontrolkę w oknie F1 i nadaje jej określone cechy (własności) oraz zapamiętuje jej uchwyt w *zmiennej.* Pierwszą i najważniejszą cechą jest cecha **'Style'**. Jej wartość określa typ kontrolki – jeden z następujących:

'**pushbutton**' - przycisk,
'**togglebutton**' - przełącznik,
'**radiobutton**' - przycisk wyboru jednego z wielu,
'**checkbox**' - pole wyboru (wiele z wielu),
'**edit**' - pole edycyjne do wpisywania lub zmiany treści,
'**text**' - napis (etykieta tekstowa),
'**slider**' - suwak,
'**frame**' - ramka,
'**listbox**' - lista,
'**popupmenu**' - lista rozwijalna.

Oto przykład utworzenia suwaka (*'Slider'*) o określonych wymiarach:

S1 **= uicontrol(*'Style',*** *'Slider',* ***'Units',*** *'pixels',* ***'Position',*** [10, 20, 15, 40])

Jeśli pominiemy cechę 'Style' to domyślnie przyjęty będzie przycisk - 'PushButton'.

Ponieważ wartości cechy ***Style*** (nazwy typów kontrolek) są typu ***string*** więc trzeba umieszczać je w apostrofach. Zazwyczaj pisze się w nich niektóre litery jako duże aby poprawić czytelność np.: 'ToggleButton', 'RadioButton', 'CheckBox' ale nie jest to obowiązkowe. Dla wszystkich kontrolek przydatna jest znajomość następujących cech:

'Position'*, [Xp, Yp, Dx, Dy]* - Położenie dolnego lewego rogu i rozmiar
'Units'*, 'normalized'* - Jednostki współrzędnych dla określenia położenia względem lewego dolnego rogu formatki. Domyślnie *'pixels'*. Dla jednostek *'normalized'* - lewy dolny róg formatki ma współrzędne (0,0) a prawy górny (1,1). Inne możliwe to: *'inches'* | *'centimeters'* | *'points'* | *'characters'*
BackgroundColor - kolor tła
Cdata - obrazek nakładany na obiekt (dany jako macierz)
ForegroundColor - kolor tekstu

SelectionHighlight - podświetlanie przy wybraniu (on | off)
Visible - widzialny (on | off)
TooltipString - opis w postaci dymku,
FontName - nazwa czcionki,
FontSize - rozmiar czcionki.

Oprócz tego pewne cechy mają szczególne znaczenie dla poszczególnych typów kontrolek. Tabela 10.2 objaśnia je tylko dla kilku kontrolek najprzydatniejszych dla naszych potrzeb.

Tabela 10.2. Ważniejsze cechy wybranych kontrolek

Wartość cechy 'Style'	Rodzaj kontrolki	Ważne cechy i działanie
pushbutton	Przycisk	***set****(uchwyt,**'String',** 'Napis na przycisku', **'Callback',** 'Polecenie')* - *polecenie* jest wykonywane po „wciśnięciu" przycisku myszką
edit	Pole edycyjne do wpisywania lub zmiany treści	***set****(uchwyt,**'String',** ' tekst')* wartość wpisaną w polu odczytamy przez: *x =* ***str2double(get****(uchwyt,* ***'String'));***
text	Dowolny napis	***set****(uchwyt,**'String',** ' Napis')*
slider	Suwak	ustawiamy: ***set****(uchwyt,**'Min',** 'wart.min', **'Max',** 'wart.maks')* odczytujemy: *x=**get****(uchwyt,**'Value');***
popupmenu	Lista rozwijalna	definiujemy pozycje menu: ***set****(uchwyt,* ***'String',*** *' opcja1 | opcja2 |opcja3');* odczytujemy: *x=**get****(uchwyt,**'Value'**);***

10.4. PRZYKŁAD PROGRAMU Z INTERFEJSEM GRAFICZNYM

Poniżej zamieszczam przykład zastosowania omówionych narzędzi w programie złożonym z dwu plików. Poszczególne instrukcje objaśniono komentarzami.

Plik **profil_01.m**

```
% PROFIL_01
% Uniwersalny program (PR 86) do oglądania przekrojów obrazów
% improfile(OBRAZ, [Xp Xk], [Yp Yk]) - to funkcja przekroju
naz=uigetfile('*.bmp','Wybierz plik obrazu do analizy');
A=imread(naz); % wczytuje obraz z pliku do macierzy A
[Lw, Lk] = size(A); % rozmiary obrazu
H=round(Lw/2); % rzędna linii przekroju
%--------------------
```

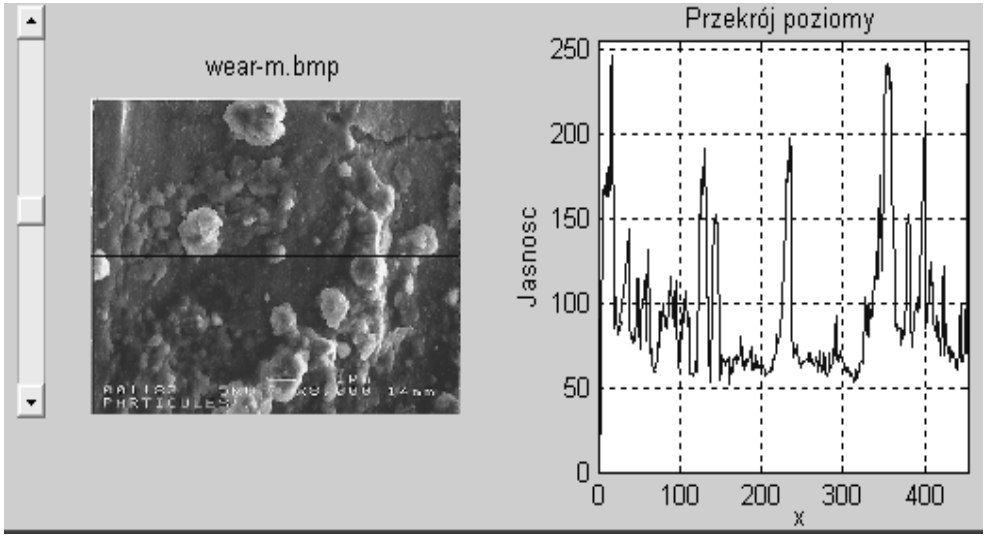
```
F1=figure; % wyswietla okno grafczne
  set(F1,'Position',[40 40 600 300]); % rozmiar i położenie figury F1
S1=uicontrol(gcf, 'Style','slider'); % definiuje suwak
  set(S1,'Position',[40 55 15 200]); % rozmiar i położenie suwaka
  set(S1,'CallBack','Ustaw'); % kliknięcie wywołuje procedurę „USTAW"
  set(S1,'Min',1,'Max',Lw); % okreslenie zakresu suwaka
  set(S1,'Value',H); % ustawienie pozycji suwaka
%---------------
Y=improfile(A, [1 Lk], [H H]); % wyznacza przekrój obrazu
A1=subplot(1,2,1); imshow(A);   % w jednym okienku: wyswietli obraz
  title(naz);
  L1=line([1 Lk],[H H]); % miejsce przekroju zaznaczone na obrazie
A2=subplot(1,2,2); PL1=plot(Y); % w drugim okienku: wyswietli przekrój
  grid on;  axis([0,Lk, 0, 256]); title('Przekrój poziomy');
  xlabel('x'); ylabel('Jasnosc');
```

Plik **Ustaw.m**

```
% Podprogram "USTAW" jest wywoływany przez kliknięcie suwaka S1 myszką
H = get(S1,'Value'); % wartość wynikająca z przemieszczenia suwaka
set(L1, 'XData',[1 Lk],'YData',[Lw-H Lw-H]); % nowa linia na obrazie
Y=improfile(A, [1 Lk], [Lw-H Lw-H]);
set(PL1, 'XData', 1:Lk-1,'YData',Y); % wykreśli nowy przekrój
```

Po uruchomieniu programu **profil_01** funkcja **uigetfile** wyświetli okno "OTWÓRZ" i pozwoli wybrać plik obrazu do analizy.

Następnie pojawi się obraz jak na Rys. 10.3.



Rys. 10.3. Okno przykładowego programu z interfejsem graficznym

10.5. ĆWICZENIA

Z98. Przeanalizuj poniższy program PR 88 a następnie utwórz w edytorze trzy podane niżej pliki a potem uruchom i przetestuj program „kontrolki1”:

```
% Program PR 87
% ten program zapisz do pliku „kontrolki1.m”
clear; clc;
y=20;
% ------ Definiowanie okna graficznego:
   f1=figure('position',[40,40,700,500]);
   set(f1,'color',[0.5 0.5 0]);
%------- Definiowanie przycisku:
   przycisk1=uicontrol('string','SKOCZ','Callback','przyrost');
%------- Definiowanie napisu
   napis1=uicontrol('Style','text', 'string','TU JESTEM')
   set(napis1,'position',[100,y,80,20])
%------- Definiowanie suwaka:
suwak1=uicontrol('style','Slider','position',[20,50,20,140]);
   set(suwak1,'Callback','kolor')
```

```
% tą procedurę zapisz do pliku  „kolor.m”
bb=get(suwak1,'Value');
set(f1,'Color',[1-sqrt(bb) bb bb^4]);
```

```
% tą procedurę zapisz do pliku „przyrost.m”
y=y+40;
if y>500
    y=0
end
set(napis1,'position',[100+y,y,80,20])
```

Uwaga: Jak dotychczas Freemat 3.5 oraz Scilab 5.3 nie obsługują w pełni występujących tu komend.

Z99. Napisz program PIERWIASTKI który wyświetla (w oknie f1=figure) wykres wielomianu trzeciego stopnia**:**

$$W(x) = ax^3 + bx^2 + cx + d$$

- jak na Rys. 10.4, przy czym:

$$\boldsymbol{a} = 1, \boldsymbol{c} = 2$$

natomiast wartości współczynników ***b*** i ***d***, mają być ustalane suwakami w określonych zakresach.
Wielomian wraz z wartościami współczynników ma się wyświetlać jako tytuł wykresu:

['x^3 + ', num2str(a), 'x^2 + ', num2str(b), 'x + ', num2str(c)].

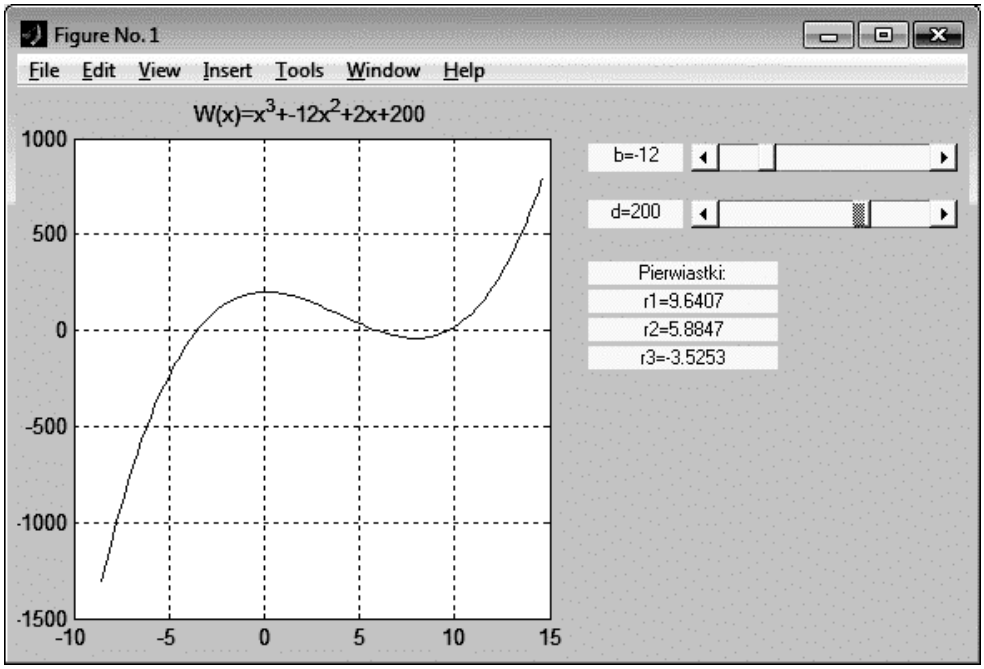
Do wyznaczania pierwiastków zastosowano funkcję **roots()**, której wyniki też mają się wyświetlić w oknie graficznym.

Poniżej masz zawartość trzech plików: **pierwiastki.m, rysuj.m, nowe.m**
Utwórz takie pliki i uzupełnij miejsca krzyżyków ##### odpowiednimi poleceniami., a nastepnie uruchom i sprawdź działanie. Zmodyfikuj kolor i tytuł okna graficznego.

```
% PROGRAM PIERWIASTKI (PR 88)
% cz.1 INICJACJA - jednorazowa
clear;clc
a=[1 5 2 20];
r=roots(a);
xp=min(real(r))-5; xk=max(real(r))+5;
x=linspace(xp,xk);
f1=figure;
a1=axes('position',[0.06,0.06,0.5,0.85]);
% ---- Tekst i suwak dla współczynnika b:
t1=uicontrol('style','text', 'units','normalized', 'position', [0.6,0.85,0.1,0.05]);
s1=uicontrol('style','slider', 'units','normalized', 'position', [0.71,0.85,0.28,0.05]);
set(s1,'Min',-20,'Max',20,'Callback','nowe');
% ---- Tekst i suwak dla wyrazu wolnego d:
t2= #####
s2= #####
set(s2,'Min',-500,'Max',500,'Callback','nowe');
%----
t03=uicontrol('style','text','units','normalized','position'...
,[0.6,0.65,0.2,0.04]);
set(t03,'string','Pierwiastki:')
%---- Wyświetlanie wartości pierwiastków
t3=uicontrol('style','text','units','normalized','position',[0.6,0.60,0.2,0.04]);
t4= #####
t5= #####
rysuj;
```

```
% procedura rysuj
% wyznaczenie ciągu wartości wielomianu
% dla danego wektora wspolczynnikow a
% oraz ciągu wartości x wyznaczonego przez: x=linspace(xp,xk)
y=polyval(a,x);
plot(x,y); grid on
st=['W(x)=x^3+',num2str(a(2)),'x^2+',num2str(a(3)),'x+',num2str(a(4))];
title(st);
set(t1,'string',['b=',num2str(a(2))]);
set(t2,'string',['d=',num2str(a(4))]);
set(t3,'string',['r1=',num2str(r(1))]);
set(t4,'string',['r2=',num2str(r(2))]);
set(t5,'string',['r3=',num2str(r(3))]);
```

```
% procedura nowe
a(2)=get(s1,'Value');
a(4)=get(s2,'Value');
r=roots(a); % wyznacza 3 pierwiastki (takze zespolone)
xp=min(real(r))-5; xk=max(real(r))+5;
x=linspace(xp,xk); % wyznacza ciąg 100 wartości x między xp xk
rysuj; % wywoluje procedure rysuj
```



Rys. 10.4. Ekran programu PIERWIASTKI

10.6. SAMODZIELNE OKIENKA DIALOGOWE

Oprócz opisanych wyżej kontrolek, osadzanych w oknie graficznym, istnieją funkcje generujące osobne okienka dialogowe dla swoich potrzeb. Funkcje te mogą być użyte zarówno jako polecenia w programach nie wykorzystujących kontrolek, jak i wywoływane interaktywnie - kliknięciem przycisku czy innym działaniem na określonej kontrolce.

Ważniejsze z tych funkcji to:

inputdlg - wyświetla okno z polami edycyjnymi do wprowadzania danych,
listdlg - wyświetla listę opcji do wyboru,

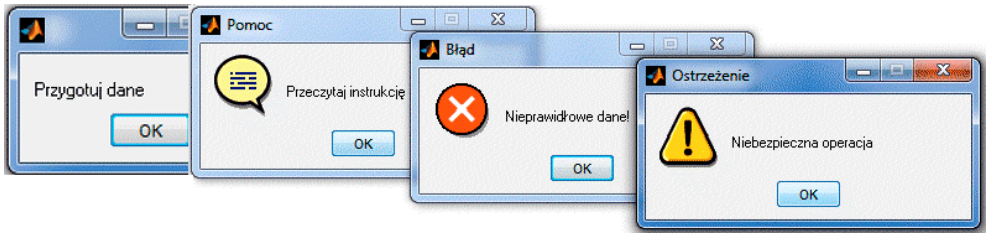
msgbox - wyświetla okienko z komunikatem,
printdlg - wyświetla okno do ustalania parametrów wydruku,
printpreview - wyświetla okno z podglądem wydruku,
questdlg - wyświetla okno z pytaniem i trzema przyciskami odpowiedzi,
uigetdir - wyświetla okno do wyboru dysku i folderu,
uigetfile - wyświetla standardowe okno otwierania plików,
uiputfile - wyświetla standardowe okno zapisywania plików,
uisetcolor - wyświetla okienko z kolorami do wyboru,
uisetfont - wyświetla okno do ustalania parametrów czcionki.

Przykłady użycia kilku z tych funkcji pokazano w kolejnych podrozdzialach

10.6.1. KOMUNIKAT - *MSGBOX*

Funkcja ***msgbox*** pozwala wyświetlić okienko z komunikatem. W najprostszym przypadku może ona mieć postać:

```
msgbox('Komunikat')
```



Rys. 10.5. Przykłady okienek z komunikatami

Funkcja ta może też przyjmować bardziej rozbudowane postaci (sprawdź w "help msgbox") na przykład:

```
msgbox('Komunikat','Tytuł','Ikona')
```

gdzie:
'*Komunikat*' - tekst komunikatu (wyrażenie typu *string*),
'*Tytuł*' - tytuł okna (na górnym pasku),
'*Ikona*' - to parametr który może mieć wartości:
'none', 'error', 'help', 'warn' lub 'custom',
domyślnie - brak ikony czyli 'none'.

Jeśli wystąpi parametr 'custom' to za nim powinny być jeszcze parametry określające obrazek ikony, a mianowicie nazwa pliku z obrazkiem oraz mapa kolorów.

Przykładowo polecenia:

```
msgbox('Przygotuj dane');
msgbox('Nieprawidłowe dane!','Błąd','error')
msgbox('Przeczytaj instrukcję','Pomoc','help')
msgbox('Niebezpieczna operacja','Ostrzeżenie','warn')
```

wyświetlą okienka z komunikatami pokazane na Rys. 10.5.

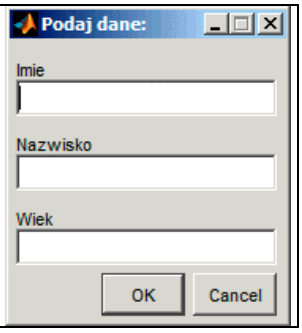
10.6.2. WPROWADZANIE DANYCH - *INPUTDLG*

Funkcja ta – w jednej z najprostszych postaci – może mieć budowę:

```
Wynik = inputdlg(Pytania, Tytuł)
```

Przykład pokazuje Rys. 10.6.

```
% Program PR 89 wprowadza dane trzech osób:
pytania={'Imie', 'Nazwisko', 'Wiek'}
for i=1:3
 osoba{i}=inputdlg(pytania,'Podaj dane:')
end
% ---- a teraz wyświetla te dane:
osoba{1}
osoba{2}
osoba{3}
```



Rys. 10.6. Przykład użycia funkcji ***inputdlg***

Dane wprowadzane są do tabeli komórkowej - w której każdy element może być innego typu, a nawet może być następną tabelą. Indeksy elementów tabeli komórkowej trzeba umieszczać w nawiasach klamrowych {}.

10.6.3. ZAPYTANIE – *QUESTDLG*

Wywołanie funkcji **questdlg** ma ogólną postać:

wybrany_przycisk = **questdlg**(*zapytanie, tytuł, O1,O1,O3,Odom*)

Pojawi się okienko (Rys. 10.7) z treścią *zapytania* a poniżej trzy przyciski podpisane zgodnie z treścią parametrów: *O1, O2, O3.* Ostatni parametr podaje który z przycisków ma być domyślny czyli wstępnie podświetlony. Napis w pasku tytułowym możemy określić przy pomocy parametru *tytuł.* Nazwa kliknięte go przycisku zostanie zapamiętana w zmiennej wynikowej *wybrany_przycisk*. Przykład użycia pokazuje Rys. 10.7.

```
% PR 90
tx='Czy Wilno jest stolicą Łotwy?';
wyb= questdlg(tx,'Pytanie:', 'Tak', 'Nie', 'Nie wiem', '')
   switch wyb,
       case 'Tak',
         disp('Odpowiedź błędna.');
       case 'Nie',
         disp('Odpowiedź poprawna!')
       case 'Nie wiem',
         disp('Wstyd nie wiedzieć!');
   end % switch
```

Rys. 10.7. Przykład użycia funkcji ***questdlg***

W najprostszym wariancie funkcja **questdlg** może mieć tylko pierwszy parametr, *zapytanie*, Wówczas przyciski będą miały napisy: ***Yes, No, Cancel***.

10.6.4. FUNKCJA *MENU*

Funkcja ***menu*** pokazana na Rys. 10.8 pozwala także realizować wybór przy pomocy kliknięcia myszką jednego z przycisków z tym, że liczba przycisków nie jest ograniczona do trzech a samo wywołanie tej funkcji, jak widać na Rys. 10.8, jest prostsze.

Ogólna postać tej funkcji to:

menu*('Nagłówek', 'Temat_1', 'Temat_1', ... , 'Temat_n')*

Kliknięcie przycisku nada funkcji wartość równą numerowi tego przycisku .

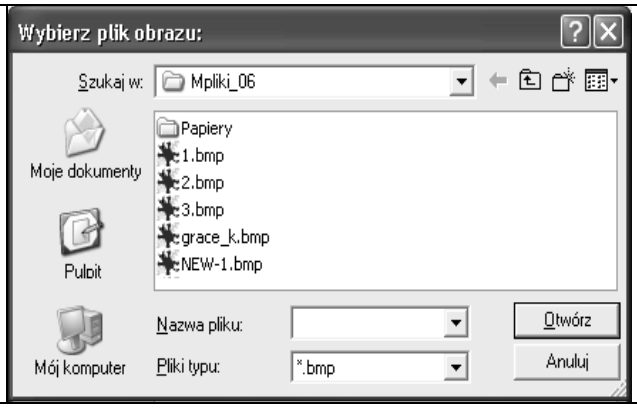
```
% PR 91
tx='Czy Wilno jest stolicą Łotwy?';
wyb_nr = menu(tx,'Tak', 'Nie', 'Nie wiem')
switch wyb_nr,
   case 1,
      disp('Odpowiedź błędna.');
   case 2,
      disp('Odpowiedź poprawna!')
   case 3,
      disp('Wstyd nie wiedzieć!');
end % switch
```

Rys. 10.8. Przykład użycia funkcji ***menu***

10.6.5. DIALOGOWY WYBÓR FOLDERÓW I PLIKÓW - *UIGETFILE*

Wywołanie funkcji ma postać: **[***nazwa_pliku, ścieżka***] = uigetfile(***filtr, tytuł***)**

```
% PR 92
% Dialog otwierania pliku
% przy pomocy UIGETFILE
% Przykładowy program
% pozwala wybrać plik obrazu
% typu "*.bmp"
% oraz wyświetla ten obraz.
clear;
t1='Wybierz plik obrazu:';
[naz,folder]=uigetfile('*.bmp', t1);
imshow(naz); % wyswietla obraz
```



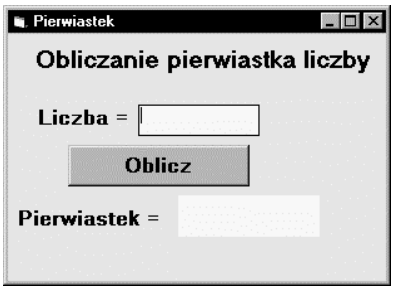
Rys. 10.9. Przykład użycia funkcji ***uigetfile***

Pierwszym parametrem *filtr* - jest szablon nazw plików – ograniczający wyświetlanie plików w oknie do określonego typu zgodnego z podanym rozszerzeniem nazwy.

Drugi parametr to napis jaki ma się pojawić w tytule okienka. W wyniku uzyskujemy nazwę wybranego myszką pliku oraz ścieżkę do jego foldera.

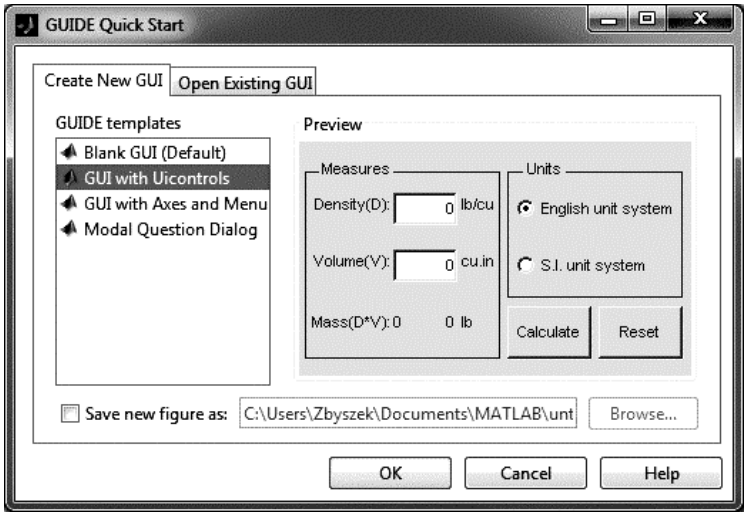
10.7. GUIDE – ŚRODOWISKO TYPU RAD

Mathcad umożliwia także korzystanie z tak zwanego "środowiska do szybkiego tworzenia aplikacji" (*RAD = Rapid Application Development*) – podobnego do Delphi, Visual BASIC-a czy C++ Buildera. Spróbujmy – jako najprostszy przykład – utworzyć taką samą aplikację jak poprzednio utworzona w Visual BASIC-u (Rys. 10.10).



Rys. 10.10. Docelowe okno aplikacji tworzonej w guide

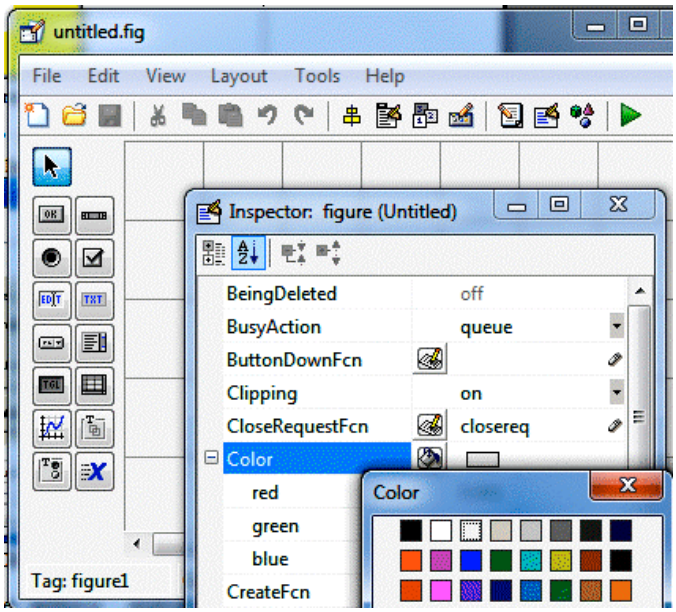
Po wpisaniu (w oknie komend) komendy „**guide”** MATLAB wyświetli okienko z kilkoma możliwościami startu (Rys. 10.11). Wybierzmy „Blank GUI”.



Rys. 10.11. Dialog startowy w GUIDE

Pojawi się puste okno graficzne (*figure*) projektowanej aplikacji - analogiczne jak "form" w Visual BASIC-u. Z lewej strony tego okna mamy **PALETĘ elementów dialogowych – kontrolek** - które możemy wstawiać do tego okna. Po podwójnym kliknięciu obszaru okna lub wstawionej kontrolki pojawi się t.zw. ***INSPECTOR*** wyświetlający **cechy** (*properties*) klikniętego elementu i umożliwiający ich modyfikowanie.

Kliknijmy podwójnie obszar okna *figure* i ustawmy wartości cech *COLOR* oraz *NAME* – a mianowicie: żółty kolor tła (Rys. 10.12) oraz nazwę okna „Pierwiastek".



Rys. 10.12. Zmiana cech w oknie INSPECTOR

Następnie zapiszmy projekt na dysk. Przy zapisywaniu utworzonego projektu okna graficznego do pliku z rozszerzeniem ".fig", domyślnie wytwarza się prototyp programu obsługującego to okno i zapisuje się do pliku z rozszerzeniem ".m". Nazwy obu plików są jednakowe a różne jedynie rozszerzenia nazw. A więc jeśli nasz projekt zapiszemy do pliku o nazwie „pierwiastek" to wytworzą się pliki **pierwiastek.fig** oraz **pierwiastek.m**. W pliku prototypu programu (pierwiastek.m), GUIDE wstawia automatycznie prototypy funkcji (podfunkcji) i ustawia własności "callback" tak aby wywoływały te funkcje.

Deklaracje funkcji mają postać:

***Nazwa_funkcji*(*h, eventdata, handles, varargin*)**

Nazwa_funkcji - jest złożona z *tagu* obiektu i typu wywołania oddzielonych przez '_', n.p.: '*edit1_Callback*', '*pushbutton1_Callback*';

h - jest uchwytem macierzystego obiektu, któremu przypisana jest ta funkcja;

eventdata - jest puste i zarezerwowane do użytku w przyszłości;

handles - jest strukturą zawierającą uchwyty do pozostałych komponentów budowanego interfejsu graficznego przy czym nazwami pól tej struktury są wartości własności ***tag*** tych komponentów;

varargin - to deklaracja dająca możliwość użycia dodatkowych argumentów przy wywoływaniu funkcji.

Kontynuujemy wstawianie kontrolek i ustawianie ich cech, tak jak pokazuje Tabela 10.3

Tabela 10.3. Ustawienia cech kontrolek w budowanej przykładowej aplikacji

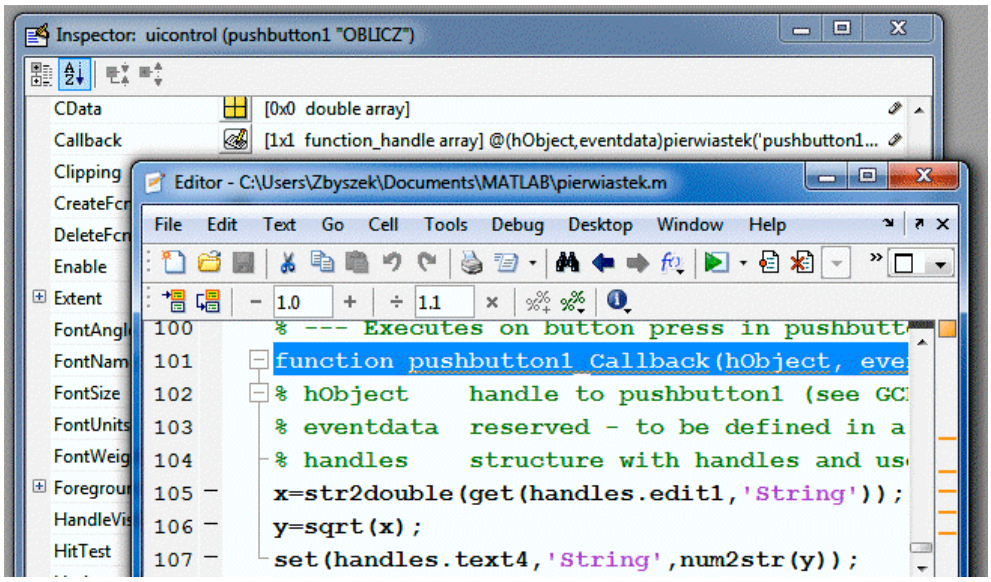
Typ	***TAG***	**Cecha**	**Wartość cechy**
Static Text	**text1**	*String*	'Obliczanie pierwiastka liczby'
		FontSize	11
		FontWeight	bold
		BackGroundColor	[1 1 0]
Static Text	**text2**	*String*	'Liczba ='
		FontSize	11
		FontWeight	bold
		BackGroundColor	[1 1 0]
Static Text	**text3**	*String*	'Pierwiastek ='
		FontSize	11
		FontWeight	bold
		BackGroundColor	[1 1 0]
Static Text	**text4**	*String*	' '
		FontSize	11
		FontWeight	bold
Edit Text	**edit1**	*String*	' '
		FontSize	11
		FontWeight	bold
Push Button	**pushbutton1**	*String*	'OBLICZ'
		FontSize	11
		FontWeight	bold
		Callback	- *ustawiona automatycznie*

Przycisk **pushbutton1** ma wywołać funkcję w której będą polecenia powodujące:

1. odczytanie tekstu wpisanego do pola **edit1** i przekonwertowanie go na liczbę,
2. obliczenie pierwiastka z tej liczby,
3. wyświetlenie wyniku w polu **text4**.

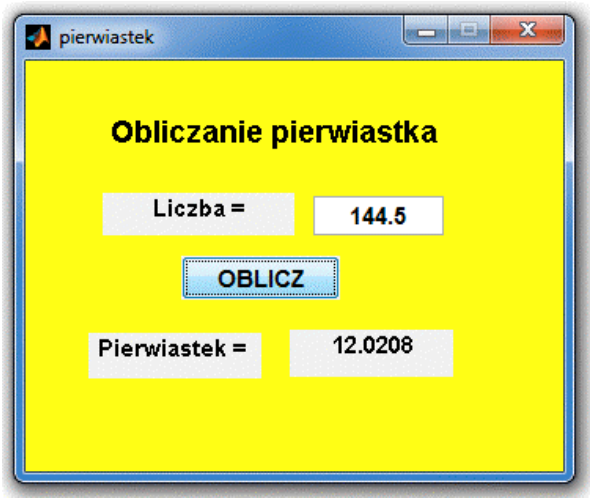
Wywołanie nastąpi po kliknięciu przycisku „Oblicz” (**pushbutton1**), dzięki ustawieniom cechy *Callback.* Podwójne kliknięcie w *Inspectorze* symbolu kartki z ołówkiem przy cesze *Callback* przeniesie nas do automatycznie utworzonego szablonu funkcji o nazwie *pushbutton1_Callback* (Rys. 10.13).

Wymienione powyżej trzy polecenia wpiszemy do tej funkcji w wierszach o numerach: 105, 106, 107 (Rys. 10.13). Wykorzystano tam m.in funkcje konwersji typów: *str2double, num2str,* oraz funkcję *sqrt* obliczającą pierwiastek.



Rys. 10.13. Funkcja realizująca działania wywołane przyciskiem “Oblicz”

Po wykonaniu tych operacji, ponownie zapisujemy projekt na dysk i uruchamiamy aplikację. Powinniśmy otrzymać efekt podobny jak na kolejnym rysunku.



Rys. 10.14. Okno utworzonej przykładowej aplikacji PR 93

11. ELEMENTY JĘZYKA C I PORÓWNANIE Z MATLABEM

Bardzo skrócony opis podstawowych elementów języka C zawarty w niniejszym rozdziale zamieszczono jako materiał nadobowiązkowy, dla pokazania z grubsza czym jest ten język o którym tak często się słyszy a na Studiach Stacjonarnych bywa nawet podstawowym językiem do nauki programowania.

11.1. JĘZYK C I JEGO NASTĘPCY

Język C jest dość starym językiem, opracowanym w latach 70-tych XX wieku, jednak wciąż jednym z najbardziej cenionych i często wykorzystywanych przez profesjonalnych programistów. Pierwszą wersję języka C opracował w Bell Laboratories Dennis Ritchie w roku 1972, a pełny opis został zawarty w książce Briana W. Kernighana i Dennisa M. Ritchie: "The C programming language" w roku 1978.

Język C powstał jako narzędzie dla profesjonalnych programistów, przy pomocy którego napisano system UNIX. Wcześniej, systemy operacyjne opracowywane były w asemblerze, ze względu na niedoskonałości istniejących języków wysokiego poziomu. Język C posiada więc zarówno cechy języków wysokiego poziomu, jak i języków niskiego poziomu, zbliżonych do asemblera, pozwalających operować na komórkach pamięci, rejestrach, portach. Jest więc język C trudniejszy, mniej intuicyjny i bardziej złożony niż BASIC - przeznaczony dla początkujących, czy MATLAB - przeznaczony dla szerokich rzesz użytkowników.

Dlaczego **język C i jego następcy: C++, C#, Java** zyskały i utrzymują tak dużą popularność?

Przyczyn jest kilka. Jedną z nich jest tworzenie szybko działających programów wynikowych o stosunkowo niedużej objętości, a więc t.zw. „wydajnego kodu”. Drugą przyczyną było powiązanie z wielodostępnym, sieciowym systemem operacyjnym UNIX z którym stanowił jedną całość. Trzecią przyczyną – bardziej współczesną – był fakt opracowania systemu Ms Windows właśnie w języku C, co w sposób naturalny powodowało, że najłatwiej współpracowały z nim aplikacje opracowane też w tym języku. Skutkiem tych przyczyn stała się moda na nauczanie języka C i pokrewnych, pomimo, że dla początkujących są one niezbyt przyjazne. Popularność języka C jest też utrzymywana dzięki bogatej literaturze oraz dzięki temu, że wiele firm programistycznych uczyniło go swym standardowym narzędziem, poszukując programistów znających właśnie język C.

Jednym z rozwinięć języka C jest **język C++** zaprojektowany przez Bjarne Stroustrupa w latach osiemdziesiątych. Jest on nadzbiorem C, to znaczy, że istnieje zgodność obu języków na poziomie podstawowym i że każdy program w języku C może być kompilowany przez kompilator C++. Język C++ wprowadził obsługę obiektów i stał się w latach 90-tych jednym z najpopularniejszych języków programowania.

Język C# (czytaj: „si szarp” lub „si hasz”) - dosłownie "C-krzyżyk" albo "cis" – to obiektowy język programowania zaprojektowany przez zespół pod kierunkiem Andersa Hejlsberga, wchodzący w skład środowiska programistycznego ***Visual Studio*** firmy Microsoft. Program napisany w tym języku kompilowany jest do kodu pośredniego w języku ***CIL*** (*Common Intermediate Language*), wykonywanego następnie w środowisku uruchomieniowym ***.NET Framework*** (patrz p.4.5.6.).

Język Java (zwany też Java C) – to obiektowy język programowania stworzony w firmie ***Sun Microsystems*** a obecnie (r.2013) stanowiący własność firmy ***Oracle Corporation***. Programy pisane w języku Java są kompilowane do kodu bajtowego, czyli postaci wykonywanej przez tak zwaną „wirtualną maszynę Java” czyli JVM (patrz p.4.5.5.).

Java przejęła podstawowe koncepcje z języków Smalltalk (maszyna wirtualna, zarządzanie pamięcią) oraz C++ (duża część składni i słów kluczowych).

Języka Java nie należy mylić ze skryptowym językiem ***JavaScript***, z którym wspólną ma jedynie składnię podstawowych instrukcji.

11.2. STRUKTURA PROGRAMU W JĘZYKU C

Poniższy opis języka C jest z założenia bardzo okrojony i uproszczony, mając na celu jedynie ogólną charakterystykę tego języka i porównanie go z MATLABem.

Program w języku C składa się z **funkcji**([1]). Funkcja jest wydzieloną częścią programu, realizującą pewne zadanie. Główny program musi mieć postać funkcji o nazwie **main()** - od niej rozpoczyna się wykonanie programu. Polecenia składające się na treść funkcji zwaną jej "ciałem" umieszcza się w nawiasach klamrowych **{ }**. **Nawiasy klamrowe { }** są także używane wszędzie **tam gdzie dwie lub więcej instrukcji ma utworzyć blok**, na przykład wewnątrz pętli lub w alternatywnych wariantach w instrukcji warunkowej.

Oto najprostszy przykład programu:

```
/* Program wyświetla napis: WITAM */
#include "stdio.h"
main()
{
 printf("WITAM \n");
}
```

Komentarze umieszczane są między znakami **/*** ... */, a więc pierwsza linia powyższego programu to komentarz wyjaśniający do czego służy ten program.

Linia druga, rozpoczęta znakiem #, jest „dyrektywą dla preprocesora” polecającą dołączyć (**`#include`**) plik zwany nagłówkowym, o nazwie „stdio.h”.

---

[1]) W innych językach program składa się z podprogramów wśród których jako **funkcje** definiowane są te które zwracają pojedynczą wartość (tak jest np. w językach BASIC oraz FORTRAN) lub macierz (w MATLAB-ie) a inne definiowane są jako **procedury**, które nie muszą dawać w wyniku wartości liczbowych . W języku C są **tylko funkcje**, dlatego wprowadzono typ **void** czyli "wartości których nie ma".

W trzeciej lini słowo „**main()**” jest nagłówkiem funkcji stanowiącej główny program. Po nazwie każdej funkcji muszą być nawiasy a jeśli nie istnieją argumenty funkcji to nawiasy te są puste. Definicje funkcji składają się z poleceń, które muszą kończyć się średnikiem [;].

Nawiasy klamrowe **{}** obejmują blok poleceń stanowiących „ciało funkcji”, a w powyższym przypadku blok ten składa się z jednego wywołania funkcji standardowej **printf()**. Funkcja **printf()** ma składnię taką samą jak opisana w p.8.7.3. funkcja **fprintf()**, z taką różnicą, że apostrofy zastąpione są w języku C cudzysłowami.

Tak więc **printf()** skopiuje na ekran znaki zawarte między cudzysłowami " ", za wyjątkiem „**\n**” - czyli polecenia zmiany linii, które spowoduje, że następny wydruk pojawi się (w tym przypadku na ekranie) w następnej linii.

Tabela 11.1. Porównanie programu RRK w MATLAB-ie i C (PR 94)

% Program RRK w MATLAB-ie	*/* Program RRK w języku C */*
	#include "stdio.h" **#include "math.h"** **#include "conio.h"** **double a, b, c, delta, x1, x2;** **char zn;** **int main()** **{**
clear; clc; zn='t';	**clrscr(); zn='t';**
while zn=='t'|zn=='T'	**while (zn=='t' | zn=='T')** **{**
fprintf('\n Równ.kwadratowe \n**');**	**printf("\n Równ.kwadratowe \n");**
a=input('a=');	**printf("\n a="); scanf("%f", &a);**
b=input('b=');	**printf("\n b="); scanf("%f", &b);**
c=input('c=');	**printf("\n c="); scanf("%f", &c);**
delta=b^2 - 4*a*c	**delta=b^2 - 4*a*c;**
if delta<0	**if (delta<0)**
fprintf('Brak rozw.rzeczywistych'**);**	**printf("**Brak rozw.rzeczywistych**");**
else	**else**
x1=(-b-sqrt(delta))/(2*a);	**{ x1=(-b-sqrt(delta))/(2*a);**
x2=(-b+sqrt(delta))/(2*a);	**x2=(-b+sqrt(delta))/(2*a);**
fprintf('\n x1=%g x2=%g',x1,x2);	**printf("\n x1=%8.2f x2=%8.2f ",x1,x2);**
end	**}**
zn=input('\n Nowe obliczenia? (T/N):**','s');**	**printf("\n** Nowe obliczenia? (T/N):**");** **zn=getch();**
end	**}** */* koniec pętli While */* **return 0** **}** */* koniec Main() */*

Tabela 11.1 zawiera kolejny przykład programu w języku C oraz – dla porównania - odpowiadający mu program w MATLAB-ie. Na przykładzie tego programu, w kolejnych podpunktach, omówiono niektóre podstawowe elementy języka C.

11.3. DOŁĄCZANIE PLIKÓW NAGŁÓWKOWYCH

Pliki nagłówkowe zawierają deklaracje (tzw. prototypy) różnorodnych funkcji z bibliotek standardowych, niezbędnych dla działania konkretnego programu. Najczęściej dołączane dyrektywą ***#include*** pliki nagłówkowe to: **stdio.h, conio.h, stdlib.h, math.h, float.h**. W powyższym programie mamy trzy dyrektywy ***#include***, dołączające: funkcje standardowego wejścia-wyjścia (stdio.h), funkcje matematyczne (math.h) oraz funkcje obsługi wejścia-wyjścia tzw. konsoli (conio.h) czyli klawiatury i ekranu w trybie tekstowym.

11.4. STAŁE, ZMIENNE, STRUKTURY DANYCH

11.4.1. STAŁE

Dopuszczalne postaci zapisu liczb nie różnią się od stosowanych w MATLAB-ie, natomiast w języku C rozróżnia się stałe znakowe - zawierające pojedynczy znak drukarski ujęty w apostrofy – od łańcuchów tekstowych ujmowanych w cudzysłowy i traktowanych jak tablice znaków.

11.4.2. NAZWY

Reguły tworzenia prawidłowych nazw zmiennych, tablic, funkcji i innych obiektów są takie same jak w MATLAB-ie (bo to MATLAB przejął reguły z C). W szczególności: rozróżniane są duże i małe litery, pierwszym znakiem musi być litera lub podkreślnik „_”, a poza tym dopuszczalne są tylko litery angielskie i cyfry. Nazwa nie może być słowem kluczowym (jak np.: **int** czy **while**).

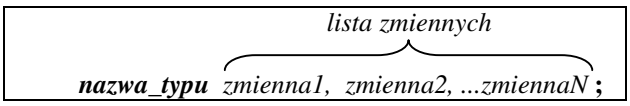
11.4.3. TYPY WARTOŚCI I DEKLARACJE ZMIENNYCH

Podobnie jak w innych językach, każda stała i zmienna ma określony typ i wynikające z niego własności. Poprzednio napisano już o typach stałych i zmiennych w języku BASIC (p.4.6.4.) oraz MATLAB (p.8.2.3.), ale tamte języki określają typy zmiennych w znacznym stopniu samoczynnie, podczas gdy w **języku C - to użytkownik musi sam zadeklarować typ każdej zmiennej**, przy pomocy deklaracji, a więc obowiązuje następująca reguła:

Wszystkie zmienne (proste i złożone) muszą być zadeklarowane przed ich użyciem

Zmienne proste to zmienne skalarne przechowujące pojedyncze wartości.

Deklaracja zmiennych prostych składa się z **nazwy typu** i **listy zmiennych**:



W języku C nazwy **podstawowych prostych typów wartości** to:

- **char** – wartość jest znakiem drukarskim,
- **int** – wartość jest liczbą całkowitą,
- **float** oraz **double** - wartości są liczbami rzeczywistymi,
- **void** - typ bezwartościowy (to znaczy brak zmiennej tam gdzie teoretycznie powinna być, na przykład brak argumentu funkcji lub jej rezultatu)

Repertuar typów jest poszerzony przez dodanie typów zmodyfikowanych, których nazwy zostały poprzedzone jednym lub dwoma z następującymi określeniami:

- **unsigned** - bez znaku;
- **short** - krótki
- **long** - długi

Wszystkie nazwy i własności prostych typów wartości w języku C pokazuje Tabela 11.2

Tabela 11.2. Proste typy wartości w języku C

Nazwa typu	Typ	Zakres	Bajtów
`void`	brak wartości		
Całkowite używane dla znaków drukarskich:			
`char`	całkowity	-128...127	1
`unsigned char`	całkowity	0...255	1
Całkowite			
`int`	całkowity	-2^31...2^31-1	4 (lub 2)
`unsigned int`	całkowity	0...2^32	4 (lub 2)
`short int`	całkowity	-32768...32767	2
`unsigned short int`	całkowity	0...65535	2
`long int`	całkowity	-2^31...2^31-1	4
`unsigned long int`	całkowity	0...2^32	4
Rzeczywiste			
`float`	rzeczywisty	6 znaków precyzji	4
`double`	rzeczywisty	10 znaków precyzji	8

Typy zmiennych muszą być odpowiednie do roli jaką te zmienne mają spełniać oraz operacji w jakich mają brać udział. Na przykład wskaźniki (indeksy) elementów wektorów

i macierzy muszą być typu całkowitego, a zmienne dla których ma sens dzielenie, pierwiastkowanie i inne działania dające wynik niecałkowity - muszą być typu rzeczywistego.

Przykład deklaracji zmiennych o wartościach całkowitych:

```
int jot, k, n;
```

Zauważmy, że deklaracja – tak jak instrukcja - musi kończyć się średnikiem
Kolejne przykłady masz w programie RRK (Tabela 11.1). Znajdź je i objaśnij.

Deklarowanie typów dotyczy również argumentów funkcji oraz wartości zwracanych przez funkcję, o czym napisano przy omawianiu funkcji.

11.4.4. WSKAŹNIKI

Szczególnym typem zmiennej jest wskaźnik (ang.: *pointer*). Wskaźnik to zmienna przechowująca adres innej zmiennej. Pobranie adresu zmiennej można wykonać przy pomocy **operatora referencji &** (referencja czyli odwołanie to właśnie adres). Na przykład:

```
wsk_x = &x;
```

spowoduje, że do zmiennej wskaźnikowej **wsk_x** pobrany zostanie adres **x**.

Wskaźniki (zmienne typu wskaźnikowego, pointery), tak jak i zmienne innych typów, **muszą być zadeklarowane**. Ponieważ różne typy zmiennych zajmują różną liczbę bajtów pamięci więc istotne jest to, jakiego typu zmienne będzie wskazywał dany wskaźnik.

Deklaracja składa się z nazwy typu zmiennych, które będą wskazywane, oraz nazw wskaźników poprzedzanych gwiazdkami. Na przykład deklaracja:

```
char *b, *c;
```

deklaruje wskaźniki **b** oraz **c** do zmiennych typu znakowego, a deklaracja:

```
int *p, *wsk_x;
```

deklaruje wskaźniki **p** oraz **wsk_x** do zmiennych typu całkowitego.

Zapisy ***b, *c** - oznaczają zmienne znakowe wskazywane przez **b** oraz **c**, natomiast zapisy ***p, *wsk_x** - oznaczają zmienne całkowite wskazywane przez **p** oraz **wsk_x**.

11.4.5. ZMIENNE LOKALNE I GLOBALNE

Zakres widoczności i dostępności zmiennej zależy od miejsca w którym ją zadeklarowano, a mianowicie:

- zmienne deklarowane wewnątrz definicji funkcji czyli po jej nagłówku – będą zmiennymi **lokalnymi** – czyli widzianymi i dostępnymi tylko w tym bloku definicji funkcji.
- zmienne, które mają być **globalne** – dostępne dla wielu funkcji – muszą być zadeklarowane przed tymi funkcjami (przed nagłówkiem pierwszej z nich)

Przykładowo: Tabela 11.1 pokazuje deklaracje zmiennych, rozpoczynające się od nazw typów: ***double*** (zmienne o wartościach liczbowych - liczby rzeczywiste „podwójnej

precyzji") oraz ***char*** (zmienna znakowa). Są one umieszczone **przed** nagłówkiem ***main()***, a więc są to zmienne **globalne**.

11.4.6. TABLICE NUMERYCZNE I ICH DEKLAROWANIE

Obowiązek deklarowania przed użyciem dotyczy także wszystkich zmiennych złożonych. Najczęściej wykorzystywanymi zmiennymi złożonymi są tablice.

Tablica jednowymiarowa czyli wektor jest zbiorem ponumerowanych elementów jednakowego typu. Inaczej niż w MATLAB-ie - numer pierwszego elementu w tablicy jest w języku C zawsze równy **zero**. Dla tablic stosuje się inne niż w MATLAB-ie nawiasy oraz inny sposób odwoływania się do elementów tablic wielowymiarowych (indeksy w osobnych nawiasach).

Deklaracja tablicy w języku C zawiera:

- nazwę typu,
- listę nazw tablic z ich wymiarami **-** czyli po każdej nazwie w nawiasach **prostokątnych** muszą być podane maksymalne przewidywane wartości wskaźników (indeksów).

Przykład deklaracji dwu tablic o wartościach całkowitych i nazwach "a" oraz "wektor":

```
int a[5], wektor[10];
```

Tablicę, przy okazji deklarowania, można także **inicjować** podając w deklaracji po jej nazwie i znaku równości listę wartości oddzielonych przecinkami i zamkniętych w nawiasach klamrowych. Przykład:

```
float Xp[5] = {1.2, 5.7, 0.3, 4, 27};
```

W programach posługujemy się elementami tablic określanymi przez **nazwę tablicy** i zapisane po niej **indeksy, ujęte w nawiasy kwadratowe**. Jako indeks czyli numer elementu może służyć stała całkowita, zmienna typu całkowitego lub dowolne wyrażenie, którego wynikiem jest liczba całkowita. Przykłady:

```
Sila[5] = 10.31;   b[i+1] = b[i]*2;
```

Możliwe jest zadeklarowanie tablicy tablic, odpowiadającej tablicy dwu- lub więcej wymiarowej:

```
double beta[10][15];
```

Powyższa instrukcja deklaruje 10-cio elementową tablicę **`beta`**, której składnikami są piętnasto elementowe tablice zmiennych typu **`double`**. Odwołanie do elementów tablicy następuje w sposób naturalny - najpierw podaje się numer tablicy, potem numer elementu wewnątrz tej tablicy:

```
beta[4][5] = 10.7;
```

Odpowiada to odwoływaniu się do wiersza [4] oraz kolumny [5].

Oprócz tablic język C posiada inne typy złożone jak: struktury (czyli rekordy), oraz unie i typy wyliczeniowe.

11.5. OPERATORY

Komplikacje i „udziwnienia" języka C zaczynają się już przy tak podstawowych elementach jak operatory. Język C posiada wiele operatorów, chociaż podobnie jak Pascal **nie posiada operatora potęgowania**, a tym bardziej operatorów działań macierzowych i tablicowych, które posiada MATLAB. Działania takie można uzyskać przy pomocy odpowiednich funkcji.

Jednym z najważniejszych operatorów jest operator przypisania „**=**", taki sam jak w MATLAB-ie, a odpowiadający stosowanemu w Mathcadzie i Pascalu symbolowi „**:=**".

Na przykład: `x = 34.7*alfa+5;`

Operator ten wyznacza wartość wyrażenia zapisanego z prawej strony i przypisuje ją zmiennej zapisanej po lewej.

Jednak w języku C operator ten można stosować dla kilku zmiennych czyli kaskadowo: `w = m = x = 34.7*alfa+5;`

Można też stosować tak zwane zapisy „uproszczone", które raczej wydają się „udziwnione", na przykład:

`a++;` zamiast `a = a+1;`

`a += 5;` zamiast: `a = a + 5;`

`a /= a + 2;` zamiast: `a = a / (a + 2);`

11.5.1. PODSTAWOWE OPERATORY ARYTMETYCZNE

Podstawowe operatory działań arytmetycznych pokazuje Tabela 11.3

Tabela 11.3. Podstawowe operatory działań arytmetycznych

Priorytet	Operator	Uwagi
3	`-`	jednoargumentowy -
2	`*`	mnożenie
2	`/`	dzielenie (rzeczywiste lub całkowite)
2	`%`	reszta z dzielenia liczb całkowitych
1	`-`	odejmowanie
1	`+`	dodawanie
	`++ oraz --`	objaśniono w tekście poniżej

Charakterystyczne dla języka C oraz C++ są operatory inkrementacji (**++**) czyli zwiększania wartości o 1 oraz dekrementacji (**--**) czyli zmniejszania wartości o 1.

Operator **++** użyty jako przyrostek to tzw. postinkrementacja, natomiast użyty jako przedrostek to tzw. preinkrementacja. Różnicę w ich działaniu pokażemy na przykładzie:

Załóżmy, że zmienna **a** ma wartość pięć i wykonujemy taką oto instrukcję:

```
c = a++;
```

W takim przypadku zmiennej **c** zostanie przypisana wartość zmiennej **a** (czyli 5) i dopiero po tym zmienna **a** zostanie zwiększona o jeden. W efekcie, po wykonaniu tej instrukcji zmienna **c** będzie miała wartość pięć, natomiast zmienna **a** będzie równa sześć.

Teraz przy założeniach takich samych jak powyżej wykonujemy taką instrukcję:

```
c = ++a;
```

W takim przypadku najpierw zmienna **a** zostanie zwiększona o jeden (czyli teraz będzie równa sześć) i następnie ta wartość będzie przypisana zmiennej **c**. Czyli w efekcie po wykonaniu tej instrukcji obie zmienne będą równe sześć.

Podobnie działa operator **--** tylko zamiast zwiększania, zmniejsza wartość o jeden.

11.5.2. INNE OPERATORY

Operatory relacji:

==	równe
>	większe
>=	większe lub równe
<	mniejsze
<=	mniejsze lub równe
!=	różne

Operatory logiczne:

&&	koniunkcja (AND)
||	alternatywa (OR)
!	negacja (NOT)

Bitowe operatory logiczne (dotyczą par odpowiadających sobie bitów):

&	bitowa koniunkcja (AND)
|	bitowa alternatywa (OR)
^	bitowa różnica symetryczna (XOR)
~	negacja bitów

Operatory przesuwania ciągów bitów:

<<	przesunięcie bitów w lewo
>>	przesunięcie bitów w prawo

Jednoargumentowy operator **referencji**:

& pobiera adres zmiennej, na przykład dzięki poleceniu:

```
wsk_x = &x;
```

zmienna wskaźnikowa **wsk_x** przechowa adres zmiennej **x**

Jednoargumentowy operator **dereferencji**:

***** pobiera wartość zmiennej na podstawie wskaźnika do tej zmiennej
na przykład wykonanie polecenia: **y = *wsk_x;**
pobierze do zmiennej **y** wartość zmiennej **x**

11.6. FUNKCJE WEJŚCIA I WYJŚCIA

W języku C podstawową funkcją dla wyprowadzania napisów i wartości zmiennych na ekran jest **printf()**, natomiast funkcja **fprintf()** pozwala wyprowadzać do pliku dyskowego i różni się od **printf()** tylko dodatkowym pierwszym argumentem – identyfikatorem otwartego wcześniej pliku.

W MATLAB-ie rolę obu tych funkcji przejęła – opisana w p.8.7.3. - funkcja **fprintf()**, która ma zmienną liczbę argumentów i przy wyprowadzaniu do pliku musi mieć identyfikator pliku jako dodatkowy pierwszy argument, a przy wyprowadzaniu na ekran - tego argumentu nie ma.

Dalszy opis funkcji **printf()** i **fprintf()** języka C a także funkcji **fopen()** – inicjującej pracę z plikiem i **fclose()** kończącej tą pracę - jest więc taki sam jak zawarty już w p.8.7.3. . oraz 8.8. .

Wprowadzanie w języku C danych z klawiatury można wykonywać przy pomocy funkcji **scanf()**, a z pliku dyskowego przy pomocy funkcji **fscanf()**. Funkcja **scanf()** ma składnię:

scanf("*format***", *&zm1, &zm2, &zm3,* ...);**

gdzie:

"*format*" – to łańcuch tekstowy z symbolami określającymi typy wczytywanych wartości,

&zm1, &zm2, &zm3,... – lista nazw zmiennych poprzedzanych operatorem & - czyli lista adresów zmiennych do których mają być przesłane wczytywane dane

na przykład:

scanf("%i", &K) - wczytuje liczbę typu ***int*** do zmiennej K,
scanf("%f", &XX) – liczbę typu ***float*** do zmiennej XX,
scanf("%s", TABZN) - ciąg znaków do tablicy znaków TABZN.

W ostatnim przykładzie przy nazwie tablicy znaków nie ma operatora & ponieważ, gdy argumentem funkcji jest wyrażenie typu tablicowego to zamieniane jest ono automatycznie na adres pierwszego elementu tablicy. A oto jeszcze jeden przykład:

```
/* PR 95 */
#include "stdio.h"
#include "conio.h"
#include "math.h"
int main ()
{
float a,b;  clrscr();
printf ("Podaj dwie liczby: ");
scanf ("%f %f", &a, &b);
printf ("Suma: %f+%f=%f\n", a, b, a+b);
return 0;
}
```

11.7. WYRAŻENIA LOGICZNE ORAZ INSTRUKCJA *IF*

Relacje i wyrażenia logiczne w językach MATLAB i C różnią się jedynie operatorem negacji, którym w języku C jest wykrzyknik a w MATLAB-ie tylda. W obu językach liczbowa wartość zero jest traktowana jako logiczny „fałsz" a wartości niezerowe jako „prawda".

Instrukcja warunkowa **if** ma w języku C najczęściej jedną z następujących postaci:

```
/* postać_1 */
    if (warunek)
        {
        blok instrukcji_1;
        }
```

```
/* postać_2 */
    if (warunek)
        { blok instrukcji_1 }
    else
        { blok instrukcji_2 }
```

Warunek jest wyrażeniem logicznym lub wyrażeniem o wartości liczbowej i musi być zawarty w nawiasach okrągłych, a *blok instrukcji* jest ujęty w nawiasy klamrowe, które są pomijane gdy blok składa się tylko z jednej instrukcji.

Analogicznie jak w innych językach - najpierw komputer określa czy *warunek* ma wartość logiczną "prawda" czy "fałsz". W tym pierwszym przypadku zostaje wykonany {*blok instrukcji_1*} i sterowanie przechodzi do instrukcji następnej po **if**. Jeśli natomiast warunek ma wartość „fałsz" to przy „*postaci_1*" nic nie będzie wykonywane a przy „*postaci_2*" wykona się tylko {*blok instrukcji_2*}.

Jak widać różnice w stosunku do MATLAB-a dotyczą nawiasów oraz braku w języku C słowa **end**.

Z100. **Ćwiczenie**: Napisz w języku C program obliczający ***y*** na podstawie wczytanego ***x***:

$$y = \begin{Bmatrix} 1-x^2 & dla & x < -1 \\ x & dla & -1 \le x \le 1 \\ 1+x^2 & dla & x > 1 \end{Bmatrix}$$

11.8. PĘTLA *FOR*

Pętla for ma składnię:

```
for(początek; warunek; zmiana)
  {
    blok_instrukcji
  }
```

W przypadku gdy *blok_instrukcji* jest tylko jedną instrukcją, wówczas nawiasy klamrowe mogą być pominięte.

Poniższy przykład pokazuje zastosowanie pętli **for** do wyznaczenia tabeli przeliczeń stopni Celsjusza na stopnie Fahrenheita:

```
#include "stdio.h"
#include "conio.h"
main() {
int t;  clrscr();
printf("\n Przeliczanie na stopnie Fahrenheita:");
for (t=0; t<=100; t=t+5;)
   printf("\n t = %3d[C] = %6.2f[F]",t,(9.0/5)*t+32);
}  /* koniec Main() */
```

Zauważmy, że w wyrażeniu musi wystąpić przynajmniej jedna liczba zapisana z kropką, ponieważ zmienna **t** jest całkowita (typu int) a do wyświetlenia wartości wyrażenia (9.0/5)*t+32 zastosowano format **%f** – właściwy dla liczb rzeczywistych. Gdyby wpisano (9/5)*t+32 to otrzymano by wynik w zakresie liczb całkowitych i byłby błąd. Można natomiast przerobić program tak aby zmienna **t** była typu **float**:

```
/* PR 96 */
#include "stdio.h"
#include "conio.h"
main() {
float t;  clrscr();
printf("\n Przeliczanie na stopnie Fahrenheita:");
for (t=0; t<=100; t=t+5;)
printf("\n t = %3.0f[C] = %6.2f[F]",t,(9.0/5)*t+32);
}
```

Ten sam program w języku MATLAB wygląda znacznie prościej:

```
clear; clc;
fprintf('\n Przeliczanie na stopnie Fahrenheita:');
for t=0:5:100
   fprintf('\n t = %3d[C] = %6.2f[F]',t,(9.0/5)*t+32);
end
```

11.8.1. PĘTLA *WHILE* ORAZ PĘTLA *DO*

Język C ma dwa typy pętli powtarzających operacje aż do zmiany wartości logicznej określonego wyrażenia. Pętla WHILE ma następującą składnię:

```
while (warunek){ instrukcje };
```

Pętla ta rozpoczyna i powtarza wykonywanie *instrukcji* tylko wtedy gdy *warunek* jest wyrażeniem logicznym o wartości „prawda” lub wyrażeniem liczbowym o wartości różnej od zera. Przykład zastosowania tego typu pętli pokazano dalej w programie SR_VAR.

Pętla typu DO ... WHILE ma składnię:

```
do { instrukcje } while (warunek);
```

Tutaj *instrukcje* są wykonane przynajmniej raz a sprawdzanie *warunku* zachodzi na końcu pętli i decyduje o dalszym jej powtarzaniu gdy *warunek* jest spełniony.

11.8.2. INSTRUKCJE ZMIENIAJĄCE DZIAŁANIE PĘTLI

Cykliczne działanie pętli może być zmienione przez umieszczenie - wewnątrz pętli - instrukcji warunkowej, wraz z instrukcją **`continue`** lub **`break`**.

Instrukcja **`continue`,** wewnątrz pętli, powoduje przeskok do końca pętli czyli do sprawdzenia możliwości dalszego wykonywania (kontynuacji) pętli.

Instrukcja **`break`** powoduje natomiast wyskoczenie z pętli do dalszych instrukcji zapisanych poniżej tej pętli.

11.9. FUNKCJE UŻYTKOWNIKA

Jak już napisano – program w języku C składa się z funkcji. Tak jak w innych językach rozróżnić należy **definiowanie funkcji** od ich **wywoływania** czyli wykorzystywania w wyrażeniach lub jako osobne polecenia. Funkcje wbudowane w język lub istniejące w pakietach oprogramowania, jeśli mamy zamiar wykorzystywać, to musimy je dołączyć do programu przy pomocy umieszczonych na początku dyrektyw **`#include " .. "`**.

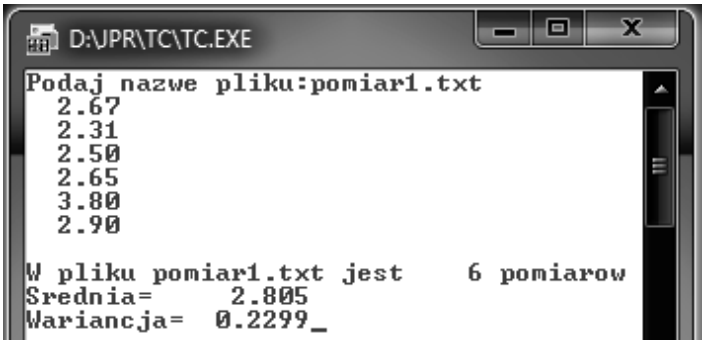
Oprócz wykorzystywania gotowych funkcji, użytkownik może definiować własne funkcje, a przy długich programach lub powtarzaniu pewnej kategorii operacji, nawet niezbędne będzie wyodrębnienie podprogramów, właśnie w postaci funkcji.

Bloki definiujące funkcje powinny poprzedzać bloki, w których te funkcje będą wywoływane (podobnie jak w języku Pascal). Nie musi jednak tak być, a nawet czasem nie jest to możliwe – na przykład gdy funkcje nawzajem się wywołują. W przypadkach gdy definicja funkcji wystąpi poniżej miejsca jej wywołania, wówczas kopię nagłówka tej definicji należy umieścić powyżej miejsca wywołania (czyli przed tym miejscem). Taka kopia nagłówka definicji funkcji nazywa się **deklaracją funkcji** albo **prototypem funkcji**.

W zamieszczonym poniżej przykładzie – opartym o algorytm z punktu 5.5. – występują dwie funkcje użytkownika: **`wczyt()`** – do wczytania ciągu wyników pomiarów, oraz **`srednia()`**. Definicja funkcji **`srednia()`** została umieszczona na końcu, a ponieważ jej dwukrotne wywołanie następuje w funkcji **`main()`**, więc przed funkcją **`main()`** umieszczono deklarację (inaczej: prototyp) funkcji **`srednia()`** – jak widać jest to kopia nagłówka jej definicji. Zauważmy, że: typy argumentów deklarowane są wewnątrz nawiasu, po nazwie funkcji, natomiast typ wyniku funkcji deklarowany jest przed jej nazwą.

```
/* PR 97: SR_VAR Obliczanie średniej i wariancji z pomiarów */
#include "stdio.h"
  int N; /* liczba pomiarow */
  float b[1000]; /* wyniki pomiarow*/
/*===== To jest definicja FUNKCJI WCZYT: =====*/
int wczyt(char *nazpli)
{   FILE *f;    int i;
  if ((f=fopen(nazpli,"r"))==NULL) {
```

```
    printf("Nie mozna otworzyc pliku!\n");
    exit(1); }
  i=1;
  while(fscanf(f,"%f",&b[i])!=EOF)
    { printf("%6.2f\n",b[i]);
      i=i+1; }
  fclose(f); N=i-1;
  return 0;
}
/* === To jest deklaracja funkcji SREDNIA: === */
float srednia(float nn, float tab[1000]);

/*===== PROGRAM GŁÓWNY czyli FUNKCJA MAIN =====*/
int main()
{ int j;  char naz[255];
  float sr, war, x, odch[1000];
clrscr();
printf("Podaj nazwe pliku:");
scanf("%s",&naz);  wczyt(naz);
printf("\nW pliku %s jest %4d pomiarow",naz,N);
sr=srednia(N,b);
printf("\nSrednia=%10.4g",sr);
for (j=1; j<=N; j++)
  { x=b[j]-sr;
  odch[j]=x*x;}
war=srednia(N,odch);
printf("\nWariancja=%8.4g",war);
return 0;
}
/* === A to jest definicja funkcji SREDNIA: === */
float srednia(float nn, float tab[1000])
{  int j; float sum;  sum=0.0;
 for (j=1; j<=nn; j++)
    sum=sum+tab[j];
 return sum/nn;
}
```



Rys. 11.1. Wyniki obliczeń programu SR_VAR

12. BAZY DANYCH

Od zarania cywilizacji ludzie odczuwali potrzebę tworzenia i wykorzystywania różnorodnych kartotek, katalogów czy wykazów. Były już one zapisywane na papirusach lub glinianych tabliczkach w starożytnych państwach Mezopotamii, Egiptu oraz Chin i stanowiły zaczątek baz danych.

Baza danych składa się ze zbiorów danych, zapisanych zgodnie z określonymi regułami i opisujących pewien wybrany fragment rzeczywistości. Bazy danych gromadzą informacje o określonych kategoriach obiektów, osób, czynności lub zdarzeń oraz związkach między nimi. Proste bazy danych, zwane kartotekami, występują jeszcze czasem w postaci papierowej jak choćby kartoteki biblioteczne czy personalne, a w postaci elektronicznej mamy je między innymi w telefonach komórkowych.

Komputerowe bazy danych (zwane też elektronicznymi) - stanowią jedną z głównych dziedzin zastosowań komputerów od początku ich istnienia. Przykładami komputerowych baz danych są katalogi książek, wykazy produktów, rejestry sprzedaży, inwentarze magazynowanych materiałów czy elementów maszyn, kartoteki personalne pracowników, wykazy klientów itp.

Oprócz samej bazy danych, niezbędne jest oprogramowanie stanowiące **system zarządzania bazą danych** (SZBD a w języku angielskim *Database Management System - DBMS*) - pozwalające gromadzić, aktualizować i wykorzystywać informacje stanowiące zawartość bazy. Baza danych wraz z SZBD nazywa się **Systemem Bazy Danych** lub czasem Bankiem Danych.

12.1. MODELE I EWOLUCJA STRUKTUR BAZ DANYCH

Modelem bazy danych [4] nazywany jest zbiór zasad opisujących strukturę i dozwolone operacje w bazie danych. Podstawowe modele to:

- hierarchiczny,
- relacyjny,
- sieciowy (grafowy),
- obiektowy,

W **modelu hierarchicznym** dane mają strukturę drzewiastą. Przykładem takiego modelu jest struktura katalogów na dysku twardym komputera. W systemie hierarchicznym zachodzą relacje rodzic-potomek. Każdy rekord może mieć wielu potomków ale tylko jednego rodzica. Hierarchiczna baza danych zakłada następujące podstawowe warunki integralności danych:

- każdy rekord (z wyjątkiem korzenia drzewa) musi posiadać jednego rodzica,
- jeżeli dany rekord miałby posiadać więcej rodziców to musi być skopiowany dla każdego rodzica oddzielnie tak, aby prawdziwa była poprzednia zasada,
- jeżeli usunięty zostaje dany rekord oznacza to, że usunięte zostają również wszystkie wywodzące się z niego rekordy – potomkowie.

Model sieciowy jest zmodyfikowaną wersją modelu hierarchicznego, mającą rozwiązać występujące tam problemy. Model sieciowy pozwala mianowicie na definiowanie relacji wiele-do-wielu, niedostępnej w modelu hierarchicznym. Model ten nie przyjął się w profesjonalnych bazach danych, jednak w praktyce jest zbliżony do wzajemnych powiązań stron internetowych.

Pierwsze komputerowe SZBD, oparte o model sieciowy oraz hierarchiczny, opracowano w latach sześćdziesiątych XX wieku.

W 1970 Anglik Edgar Frank Codd, krytykując dotychczasowe modele, zaproponował **relacyjny model danych**. Jednak pierwsze komercyjne rozwiązania relacyjnych SZBD (RSZBD) dla dużych komputerów: Oracle i DB2, pojawiły się dopiero około roku 1980. Pierwszym udanym RSZBD dla komputerów personalnych pracujących z Ms-DOS był dBASE firmy Ashton-Tate. Relacyjny model bazy danych, ostatecznie opracowany jako standard ANSI X3H2, zyskał największe rozpowszechnienie i zdecydowana większość istniejących obecnie profesjonalnych baz danych w instytucjach i przedsiębiorstwach jest zbudowana według tego modelu. Dlatego tylko model relacyjny będzie omawiany szerzej w tym rozdziale.

Obiektowy model danych – wzorowany na obiektowych językach programowania – zapoczątkowała w roku 1991 organizacja Object Database Management Group (ODMG), która opublikowała pierwszy rys standardu. W roku 2002 opublikowano standard Java Data Objects 1.0 w którym jako podstawa użyty został model obiektu używany przez język Java. W modelu obiektowym dane nie są gromadzone w tabelach lecz w obiektach, posiadających dodatkowo metody (procedury) do operowania na tych danych. Model ten – jako samodzielny - nie uzyskał większej popularności, natomiast został dołączony jako rozszerzenie do modelu relacyjnego, tworząc model **relacyjno-obiektowy**.

Wsparcie dla modelu relacyjno-obiektowego stanowi język SQL3 opracowany jako standard ANSI oraz ISO.

12.2. ARCHITEKTURA I EWOLUCJA SZBD

Historyczny rozwój baz danych i SZBD polegał zarówno na zmianach sposobów komunikowania się z bazami danych, jak i zmianach struktur danych - wynikających z opisanych wyżej modeli – oraz zmianach architektury oprogramowania.

Na początku ery komputerów - w latach sześćdziesiątych i siedemdziesiątych XX wieku - bazy danych gromadzono głównie na taśmach magnetycznych, w dużych centrach obliczeniowych a oprogramowanie pisano w języku **COBOL**. Przetwarzanie informacji odbywało się w sposób „wsadowy". Polegało to na dostarczaniu operatorowi komputera pakietu kart perforowanych z zakodowanymi na nich programami i danymi, a następnie odbieraniu, na następny dzień, paczki wydruków. Nie istniały bowiem, lub nie były jeszcze rozpowszechnione, urządzenia do dialogu z komputerem

Po wprowadzeniu komputerów osobistych, w latach 80-tych, dominowały SZBD oparte o język **dBase** (dBase I, II, III, IV, FoxBase, Clipper).

Wiele profesjonalnych systemów obsługujących banki czy kadry i płace w przedsiębiorstwach, opracowywano w językach **Clipper** lub **FoxPro** stanowiących rozwinięcia dBase. Były to systemy dialogowe o dużym stopniu niezawodności, działające w trybie tekstowym i obsługiwane głównie klawiaturą, pisane tak aby były „idioto-odporne". Niewielki, przykładowy system tego rodzaju przedstawiono dalej w p.12.9. . Wprowadzenie Ms Windows początkowo znacznie pogorszyło tą niezawodność i wprowadziło nadmiar różnych nie zawsze potrzebnych możliwości.

W systemie Ms Windows największą popularność, dla niezbyt dużych baz danych, zyskał **Ms Access** – program do baz danych wchodzący w skład pakietu biurowego **Ms Office**. Posiada on odpowiedniki również w darmowych pakietach biurowych jak **OpenOffice**.

Ms Acces jest przykładem **systemu typu „desktop"** czyli mieszczącego się na biurku i przewidzianego najczęściej tylko dla osób wykorzystujących dany komputer.

Wielodostępne, duże i kosztowne bazy danych, działające na serwerach w sieciach lokalnych lub korporacyjnych, są tworzone często z udziałem kilku języków programowania przy czym, językiem wykorzystywanym bezpośrednio do działań na danych, najczęściej jest język SQL (*Structured Query Language*). W tego typu systemach występuje **architektura dwu lub wielo-warstwowa**.

Przykładem **architektury dwuwarstwowej** są **systemy klient-serwer**.

Klientem nazywana jest ta warstwa oprogramowania, z którą ma styczność użytkownik i poprzez którą zgłasza tak zwane „zapytania" (ang.: *query*) określające sposób wyszukiwania i selekcji danych, oraz zleca wykonywanie różnorodnych operacji (transakcji) na danych. Dalsze funkcje klienta to odbieranie danych z serwera oraz ich przetwarzanie i prezentowanie.

Warstwa serwera m.in. umożliwia bezkonfliktowy dostęp wielu użytkowników (wielu klientów) do baz danych, przyjmuje od nich zlecenia i realizuje te zlecenia lub wysyła komunikaty o niemożliwości realizacji, a także dba o zapewnienie integralności i nienaruszalności danych, nawet w przypadku awarii.

Krótko mówi się, że warstwa serwera realizuje **procesy serwera** a warstwa klienta - **procesy klienta**. Zazwyczaj oprogramowanie warstwy serwera umieszczone jest na innym komputerze niż klienta, a komunikują się one poprzez sieć komputerową.

Wraz z rozwojem Internetu wzrasta tendencja do rezygnacji z oddzielnych sieci korporacyjnych i specjalnych aplikacji bazodanowych po stronie klientów, na rzecz zastąpienia ich wykorzystywaniem Internetu lub intranetu i ogólnodostępnych przeglądarek internetowych. Rozwiązania, które to umożliwiają, wykorzystują równocześnie trzy języki: HTML do budowy formularzy po stronie klienta, oraz PHP i SQL działające na serwerach. W tego typu systemach występuje kilka warstw:

- warstwę klienta („front-end") stanowi przeglądarka internetowa,
- warstwy pośrednie stanowią: serwer WWW oraz interpreter PHP generujący także polecenia w języku SQL
- warstwę końcową („back-end") stanowi SZBD obsługujący polecenia w języku SQL.

12.3. RELACYJNE BAZY DANYCH

Dane w relacyjnych bazach danych przechowywane są w tabelach. Konkretna **tabela** powinna gromadzić informacje o przedstawicielach określonego zbioru podmiotów (obiektów, zdarzeń, stanów, ...), reprezentowanych przez opis określonych n cech. Takie n-tki cech zwane są **encjami** co odpowiada angielskiemu terminowi ***entity***.

Przykładowo jedna tabela może dotyczyć autorów książek, druga wydawnictw, a trzecia egzemplarzy książek w bibliotece, albo: jedna tabela dotyczy poszczególnych maszyn zamontowanych w zakładzie, a druga rejestruje awarie maszyn w tym zakładzie.

Wiersze tabeli bazy danych nazywane są **rekordami** (ang.: *record* znaczy zapis).

Rekord tabeli zawiera opis pojedynczego obiektu, zdarzenia lub procesu - na przykład konkretnego egzemplarza książki, konkretnego studenta czy konkretnej awarii - w postaci wartości jego określonych atrybutów (cech), rozmieszczonych w osobnych **polach** odpowiadających **kolumnom** tabeli i mających ustalony **typ** (np.: numeryczny, logiczny, tekstowy, ...).

Każda tabela bazy danych ma więc stałą liczbę kolumn (pól), o określonych nazwach i typach, oraz zmienną (np. przyrastającą wskutek gromadzenia informacji) liczbę wierszy (rekordów).

Nazwisko	Imię	Nr_albumu	Rok_st	Grupa
Kolasiński	Tadeusz	128649	3	5
Szykulska	Ewa	128650	4	6a
Waraszkiewicz	Jan	128654	3	7
Piotrowski	Paweł	128644	1	4b
Piotrowski	Paweł	128645	1	4b

Rys. 12.1. Przykładowa tabela bazy danych

W tabeli z Rys. 12.1 rekord dotyczy konkretnego studenta i ma pola: *Nazwisko, Imię, Nr albumu, Rok studiów, Grupa*. Dość częste bywają przypadki wystąpienia dwu różnych studentów o tych samych nazwiskach a nawet i imionach. W powyższej tabeli dwa razy występuje Paweł Piotrowski. W takich sytuacjach przeważnie zastanawiamy się czy to jest omyłkowe powtórzenie tej samej osoby czy może dwie różne osoby.

Poprawna baza danych musi posiadać środki dla **jednoznacznej identyfikacji** rejestrowanych podmiotów co pozwoli rozróżnić osoby nawet w takiej sytuacji. Jednoznacznym identyfikatorem w przypadku studenta powinien być jego PESEL albo wartość pola Nr_albumu – jeśli jesteśmy pewni, że nie wydano mu dwu indeksów.

Każda tabela musi posiadać **jednoznaczny identyfikator rekordów** czyli **klucz podstawowy** zwany też kluczem **głównym**. Najczęściej jest to pole liczbowe zawierające kolejne numery rekordów. Unikalność numerów w Ms Access zapewnia wybranie dla tego pola typu „autonumer". Dla rejestrowanych osób kluczem głównym może być pole zawierające PESEL.

Każde pole musi mieć ustaloną i niezmienną **nazwę**, **typ** oraz **długość** odpowiednie dla wartości przechowywanego atrybutu. Niektóre systemy dopuszczają zmienną długość pól typu tekstowego. Jak widać typ tekstowy jest odpowiedni nie tylko dla nazwiska i imienia ale także dla grupy studenckiej a nawet roku studiów – gdyż zdarza się, że występują przy nich pomocnicze litery. Numer albumu może być wartością typu numerycznego chociaż tak naprawdę numerycznymi muszą być te pola, których wartości mogłyby brać udział w operacjach arytmetycznych. Długość pola – jeśli musi być stała – powinna być dostosowana do najdłuższej z możliwych wartości tego pola.

Projektowanie bazy danych polega nie tylko na zaprojektowaniu odpowiednich **tabel** ale także **relacji** (powiązań) między nimi, **operacji** jakie będą wykonywane oraz **sposobu komunikowania się** z użytkownikiem (interfejsu graficznego) i **prezentowania wyników** (m.in. postaci wydruków).

12.4. PROSTE BAZY W ARKUSZU KALKULACYJNYM

Najprostsze bazy danych, szczególnie jedno-tabelowe, (kartotekowe) można z powodzeniem tworzyć i obsługiwać w arkuszu kalkulacyjnym jak Excel (z Ms Office) czy Calc (z OpenOffice.org). Środki jakie posiada arkusz dla takich prostych baz danych prześledzimy na przykładzie tworzenia *spisu inwentarza*, pokazanego na Rys. 12.2.

a)

	A	B	C
1	**Nr_inw**	**Przedmiot**	**Pokój**
2	W14231	krzesło	213
3	W14232	krzesło	213
4	W14233	stolik	213
5	W14234	szafa	213
6	W14235	krzesło	214
7	T13238	komputer	214
8	T13239	komputer	214

b)

	A	B	C
1	**Nr_inw**	**Przedmiot**	**Pokój**
2	W14231	krzesło	213
3	W14232	krzesło	213
4	W14233	stolik	213
5	W14234	szafa	213
6	W14235	krzesło	214
7	T13238	komputer	214
8	T13239	komputer	214
9	W15237	krzesło	
10			
11			
12			
13			
14			

komputer
krzesło
stolik
szafa

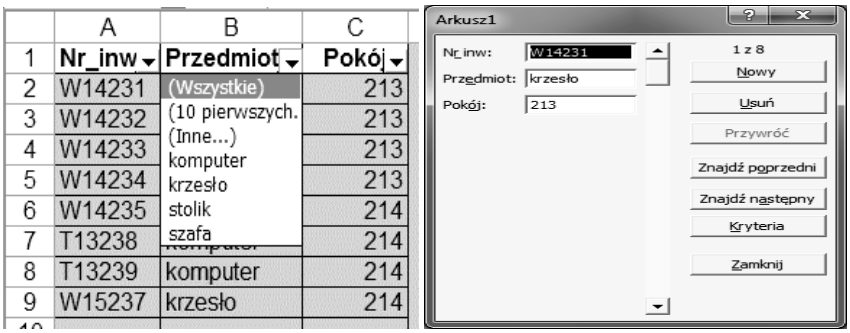
c)

Rys. 12.2. Tabela w arkuszu: a) podpowiedzi, b) wybór z listy, c) sortowanie

Do środków tych należą:

- **podpowiedzi** - przydatne przy wpisywaniu tych samych słów (Rys. 12.2a) – po wpisaniu początkowych liter komputer wyświetla wpisane już w tej kolumnie słowa rozpoczynające się od tych liter i jeśli podpowiedź jest trafna to wystarczy ją zaakceptować klawiszem ENTER;
- zamiast wpisywania słów **wybieranie ich z listy** utworzonej automatycznie z wcześniej wpisanych już w tej kolumnie słów (Rys. 12.2b) – listę uzyskuje się z menu kontekstowego (opcja „Wybierz z listy...”), po kliknięciu wypełnianej komórki prawym przyciskiem myszy.

- **sortowanie** według zawartości maksimum trzech wybranych pól (Rys. 12.2c);
- **filtrowanie** danych przy pomocy **autofiltrów** z menu Dane-Filtr (Rys. 12.3a);
- możliwość wprowadzania, przeglądania i wyszukiwania danych za pomocą automatycznie tworzonego **formularza** (Rys. 12.3b)



Rys. 12.3. Tabela w arkuszu: a) autofiltr, b) formularz

Możliwe są także operacje bardziej zaawansowane – przy użyciu takich środków jak:

- bogaty zestaw różnorodnych funkcji (m.in. do zliczania czy wyszukiwania);
- możliwość wstawiania do arkuszy tak zwanych „formantów formularzy";
- możliwość użycia języka VBA - Visual BASIC for Applications.

12.5. WYMAGANE CECHY BAZ DANYCH I SZBD

Niezależnie od przyjętego modelu struktury danych (jednego z opisanych w następnym podrozdziale) dane powinny dobrze odzwierciedlać opisywany fragment rzeczywistości, a więc być **prawdziwe, aktualne** i wzajemnie **niesprzeczne**; określają to także takie pojęcia jak **integralność** (kompletność, nienaruszony stan) i **spójność** (niesprzeczność) danych (*data integrity* - definiowane jako połączenie cech: *accuracy, correctness*, *validity*);

System zarządzania bazami danych musi posiadać szereg cech [30], [31], a w szczególności ma zapewnić:

- **tworzenie nowych baz danych** i określanie ich struktury czyli t.zw. *schematu*;
- możliwość **dialogowego uzyskiwania informacji** – w możliwie najkrótszym czasie - poprzez formułowanie pytań oraz możliwość **bezkolizyjnego wielodostępu** do danych przez wielu użytkowników równocześnie;
- **trwałe przechowywanie** danych i **możliwość ich odzyskiwania** w przypadku błędów pracy systemu;
- **ochronę danych przed niepowołanymi** użytkownikami i dostęp do danych oraz operacji zgodny z uprawnieniami użytkowników.

Oprócz tego większość baz danych posiada dodatkowe cechy charakterystyczne wymienione w pracy [30]:

- **niezależność aplikacji i danych** – modyfikacja danych nie wymaga modyfikacji aplikacji i odwrotnie;
- **abstrakcyjna reprezentacja danych** - twórca aplikacji nie musi interesować się sposobem reprezentacji danych - dzięki stosowaniu języka deklaratywnego precyzującego jedynie warunki selekcji informacji czyli „co uzyskać", a nie „jak";
- **różnorodność sposobów prezentacji danych** - te same dane zawarte w bazie mogą być prezentowane w różny sposób dla jednego lub różnych użytkowników.

12.6. PODSTAWOWE OPERACJE

Dostęp do wykonywania różnego typu operacji - jakie umożliwia system zarządzania bazą danych – jest inny na etapie budowy bazy a inny po jej przekazaniu do eksploatacji, przy czym wówczas zależy od **uprawnień** nadanych przez **administratora**.

Generalnie można wyróżnić następujące kategorie operacji:

- **projektowanie i modyfikowanie schematu (struktury) bazy** oraz **interfejsu użytkownika** do jej obsługi – to domena projektantów i programistów we współpracy z przedstawicielami użytkowników;
- **wprowadzanie i edycja danych** – prowadzona przez uprawnionych do tego użytkowników;
- **przeglądanie, wyszukiwanie**, sporządzanie wykazów (t.zw. raportów) i ich drukowanie – dostępne dla szerszego grona użytkowników, także dla tych, którzy nie posiadają uprawnień do edycji.

Łatwość dokonywania tych operacji zależy od interfejsu użytkownika. Generalnie interfejs łatwy w obsłudze bywa bardziej ograniczony co do zakresu działań, a interfejs bardziej uniwersalny – na przykład język zapytań – jest trudniejszy w użytkowaniu.

Poniżej opisano ogólnie najważniejsze operacje, natomiast dokładniej można się z nimi zapoznać nieco dalej gdzie omówiony jest przykład budowy bazy danych w systemie Ms Access.

12.6.1. PROJEKTOWANIE I BUDOWA BAZY

Pierwszy etap polega na **zdefiniowaniu** poszczególnych **tabel**, w taki sposób aby każda informacja wystąpiła tylko w jednym miejscu – przy czym tabele złożone trzeba wówczas dekomponować na **proste tabele** składowe **powiązane** odpowiednimi **relacjami**. Ten proces nazywany jest **normalizacją bazy danych**.

Dla każdej tabeli musi być określony t.zw. **klucz główny** – jednoznacznie identyfikujący poszczególne rekordy, a więc mający nie powtarzające się (unikalne) wartości.

Muszą też być określone **cechy** każdego **pola** każdej tabeli, takie jak:

- **nazwa**,
- **typ** wartości pola i jego **długość,**
- warunki **kontroli poprawności** wprowadzanych danych,
- dopuszczalność (lub nie) pozostawienia **pustej** zawartości,
- ewentualna potrzeba **indeksowania** według tego pola czyli ustalania porządku rekordów według zawartości kolumny odpowiadającej temu polu.

Pole do którego wartości będzie użytkownik wybierać z wykazu zawartego w innej tabeli - zwanej wtedy słownikową - będzie odwoływało się do wartości klucza głównego (najczęściej: numeru rekordu) tej tabeli słownikowej. Wartość klucza obcej tabeli, wstawiona jako wartość pola w danej tabeli nazywana jest **kluczem obcym**.

Mogą występować **trzy typy relacji** wiążących określone pola dwu tabel: jeden-do-jednego, jeden-do-wielu oraz wiele-do-wielu. Omówiono je dalej w rozdziale o Ms Access.

Jak wspomniano – w innej terminologii - podmioty reprezentowane w tabelach nazywane są **encjami** a relacje między nimi (a więc i między tabelami) - **związkami encji**.

12.6.2. WPROWADZANIE I EDYCJA DANYCH

Wprowadzanie i edycja danych możliwa jest zazwyczaj przez dostęp albo bezpośrednio do **tabel** albo do ich poszczególnych rekordów **przez formularz** wyświetlający pola jednego rekordu. Wprowadzanie danych bardzo może być bardzo ułatwione przez możliwość **wybierania wartości z wykazu**. Dlatego jest to powszechnie stosowane i zgodne z postulatem unikalności danych czyli występowania każdej danej tylko w jednym miejscu.

W bazie danych powinny być też wykorzystywane wszelkie możliwe **sposoby kontroli poprawności danych** – co do typu, zakresu i logicznej zgodności z innymi danymi.

Zakres edycji danych może w wielu przypadkach być ograniczany dla uniknięcia niespójności, dublowania czy sprzeczności danych.

Szczególnym typem edycji danych są transakcje**.**

Transakcja to grupa operacji, które albo muszą być wykonane wszystkie, albo jeśli choć jedna nie została wykonana to unieważnione zostają wszystkie. Przykładowo: dokonywanie przelewu polega na odjęciu określonej kwoty od stanu jednego konta, a dodaniu jej do stanu innego konta. Jest oczywistym, że w razie wyłączenia prądu między tymi operacjami nie może być wykonana żadna z nich.

Ogólnie cechy transakcji są określane akronimem ***ACID***:

- ***Atomicity*** - **niepodzielność** - cała transakcja ma być przeprowadzona, albo żaden z jej elementów nie zostanie uwzględniony;
- ***Consistency*** - spójność czyli **brak sprzeczności** – np:. miejsce w samolocie nie może być przydzielone dwu różnym pasażerom;
- ***Isolation*** - **izolacja** - brak wpływu kilku transakcji na siebie przy próbie jednoczesnej ich realizacji.
- ***Durability*** – **trwałość** - po zakończeniu transakcji jej wynik nie może zostać utracony.

Dla realizacji tych cech stosuje się takie środki jak:

- **blokada danych** na których wykonywana jest dana transakcja, co czyni je niedostępnymi dla innych transakcji;
- **rejestrowanie**, w plikach zwanych **logami**, przebiegu operacji operacji – czasu rozpoczęcia, dokonanych zmian oraz czasu zakończenia;
- **zatwierdzanie transakcji** - gdy transakcja kończy działanie i jest gotowa do zatwierdzenia to zmiany kopiowane są do logu, a dopiero potem następuje aktualizacja danych oraz zapisanie czasu końca transakcji do logu, a jeśli to się nie uda to operacje są wycofywane zgodnie z zapisem w logu.

12.6.3. PRZEGLĄDANIE, WYSZUKIWANIE, DRUKOWANIE

Przeglądanie i wyszukiwanie danych wymaga zazwyczaj ustawienia rekordów w odpowiedniej kolejności.

Sortowanie czyli fizyczne przestawianie rekordów w celu uporządkowania jest dość czasochłonne i na ogół nieopłacalne gdyż znacznie wygodniej otrzymać określony porządek przeglądania wykorzystując indeksowanie.

Indeksowanie polega na sporządzeniu plików (tabel) indeksowych odpowiadających różnym sposobom uporządkowania. Porządkowanie odbywa się według klucza indeksowego, którym może być jedno pole tabeli lub wyrażenie złożone z kilku pól. Przykładowo można utworzyć plik indeksowy pozwalający przeglądać spis studentów według roczników a w ramach rocznika np. alfabetycznie. Dzięki indeksowaniu można bardzo szybko przestawić przeglądanie tabeli na inny porządek – według innego pliku indeksowego. Do wyboru są dwa sposoby indeksowania:

- indeksowanie bez powtórzeń (unikalne) nie dopuszcza do zaistnienia dwu lub więcej rekordów z tą samą wartością klucza indeksowego,
- indeksowanie z powtórzeniami – dopuszcza występowanie rekordów o takim samym kluczu indeksowym

Selekcja to operacja wyszukania wszystkich rekordów, w których wartości pól spełniają określone warunki. Przykładowo: wyszukanie wszystkich studentów o średniej ocen powyżej 4 i średnim dochodzie na członka rodziny poniżej 2 tys. zł.

W wybranych rekordach mogą nie interesować użytkownika wszystkie a jedynie określone pola. Wybranie rekordów z pominięciem pewnych pól nazywa się **projekcją** tabeli.

Proste tabele zawarte w bazie danych mogą być łączone relacjami na różne sposoby – tworząc **tabele wirtualne**.

12.7. JĘZYKI PROGRAMOWANIA SZBD

Do tworzenia programów pozwalających korzystać z baz danych stosowane są różne metody i narzędzia.

W rozdziale 2.5. opisano zasady budowy i działania dynamicznych stron WWW, a w szczególności **obsługi formularzy internetowych** - w oparciu o języki HTML i PHP. Tego typu strony są współcześnie często używane w roli aplikacji klienckich, wykorzystujących wielodostępne internetowe bazy danych. Program w języku PHP rezydujący na serwerze WWW generuje dla klienta formularze i pobiera z nich dane oraz polecenia a równocześnie komunikuje się z serwerem baz danych generując dla niego odpowiednie polecenia w języku SQL.

Jak więc widać, budowanie oprogramowania dla internetowej bazy danych wymaga w tym przypadku znajomości aż trzech języków.

Aplikacje klienckie nie muszą jednak być dynamicznymi stronami WWW, mogą być też tworzone jako samodzielne programy (aplikacje) „okienkowe" w dowolnym języku zaimplementowanym w danym systemie operacyjnym, na przykład: Visual BASIC, Java, C++, Delphi i in. Komunikują się wówczas przez sieć komputerową z serwerem baz danych, także generując polecenia w języku SQL. W tym przypadku wystarcza więc znajomość dwu języków.

Czy koniecznie trzeba znać kilka języków do programowania baz danych i skąd to wyniknęło?

Głównym powodem używania conajmniej dwu języków jest architektura klient-serwer pozostawiająca swobodę twórcom aplikacji klienckich, które mogą różnić się wieloma aspektami m.in. celami użytkowania, systemami operacyjnymi, koncepcją interfejsu, a także narzędziami w których je utworzono. Z drugiej strony nastąpiła unifikacja serwerów baz danych, gdyż praktycznie niemal wszystkie wykorzystują do obsługi baz język SQL. Równocześnie trzeba zaznaczyć, że **SQL nie jest kompletnym językiem programowania** gdyż nie posiada środków do budowy graficznego interfejsu użytkownika i stąd wynika konieczność użycia również innego języka.

Nieco inaczej jest w systemach obsługujących lokalną bazę danych, które mogą być tworzone (z pominięciem SQL) w jednym z uniwersalnych języków programowania, jeśli posiada on środki do obsługi baz danych. Przez kilkanaście lat były takimi – dziś nieco zapomniane - języki z rodziny xBase, których pierwowzorem był dBase III. Nieco szczegółów, zarówno o SQL, jak i Clipperze z rodziny xBase, zamieszczono poniżej.

12.7.1. CHARAKTERYSTYKA JĘZYKA SQL

SQL czyli „strukturalny język zapytań" (*Structured Query Language*) nie jest samodzielnym językiem programowania lecz właściwie „podjęzykiem" używanym jedynie **do tworzenia, modyfikowania i wykorzystywania baz danych**. Jest to język deklaratywny, nieproceduralny - to znaczy, że podajemy co chcemy uzyskać a nie w jaki sposób i przy pomocy jakich operacji.

SQL jest już językiem leciwym bo jego pierwszą wersję opracowano w firmie IBM w latach siedemdziesiątych XX w. Pierwszą firmą, która włączyła SQL do swojego produktu komercyjnego, był Oracle. Już od 1986 roku SQL stał się oficjalnym standardem ISO i ANSI, a kolejne jego wersje ogłoszone jako standardy to SQL92 oraz SQL:2003.

Obecnie SQL jest powszechnie stosowany w komunikacji systemów klienckich z serwerami relacyjnych baz danych.

Instrukcje SQL tradycyjnie zapisywane są wielkimi literami, choć nie jest to obowiązkowe. Każde zapytanie w SQL-u musi kończyć się średnikiem [;].

Polecenia języka **SQL** (zwane zapytaniami) tworzą trzy kategorie (języki składowe):

- ***DDL*** - *Data Definition Language* - Język Definicji Danych, do tworzenia, usuwania, modyfikowaniastruktur baz danych
- ***DCL*** - *Data Control Language* - Język Zarządzania Danymi, stosowany m.in do nadawania uprawnień użytkownikom
- ***DML*** - *Data Manipulation Language* - Język Manipulacji Danymi, a w szczególności do selekcji, aktualizacji, wstawiania i usuwania danych.

Najważniejsze polecenia SQL to:

CREATE - tworzenie struktury (bazy, tabeli, indeksu, itp.), np. CREATE TABLE ..., CREATE DATABASE, ...
DROP - całkowite usunięcie struktury, np. DROP TABLE ..., DROP DATABASE, ...
ALTER - - zmiana struktury (dodanie kolumny do tabeli, zmiana typu danych w kolumnie tabeli), np. ALTER TABLE ...ADD COLUMN ...
GRANT - nadawanie uprawnień użytkownikom
SELECT - wybieranie podzbiorów spełniających określone warunki,
INSERT - wstawianie danych,
UPDATE - aktualizacja danych,
DELETE - usunięcie danych z bazy.

Przykłady:

```
CREATE TABLE pracownicy (imie varchar(255),
     nazwisko varchar(255), pensja float, staz int);
```

- tworzy tabelę "pracownicy" zawierającą tekstowe pola **imię** i **nazwisko**, typu ***varchar(255)*** (zmiennej długości pole tekstowe o maksymalnej długości 255 znaków), pole **pensja** typu ***float*** (liczby rzeczywiste) oraz pole **staz** typu ***int*** (liczby całkowite).

```
DROP TABLE pracownicy;
```

- usuwa z bazy całkowicie tabelę "pracownicy".

```
SELECT * FROM pracownicy WHERE pensja > 2000 ORDER BY staz DESC;
```

- wyświetla z tabeli pracownicy (FROM pracownicy) wszystkie kolumny (*) dotyczące tych pracowników, których pensja jest większa niż 2000 (WHERE pensja > 2000) i sortuje wynik malejąco według stażu pracy (ORDER BY staz DESC).

```
INSERT INTO pracownicy (imie, nazwisko, pensja, staz)
       VALUES ('Jan', 'Kowalski', 5500, 1);
```

Dodaje do tabeli pracownicy (INTO pracownicy) wiersz (rekord) zawierający dane pojedynczego pracownika.

```
UPDATE pracownicy SET pensja = pensja * 1.1 WHERE staz > 2;
```

- podnosi o 10% (SET pensja = pensja * 1.1) pensję pracownikom, których staż jest większy niż 2 lata.

```
DELETE FROM pracownicy WHERE imie = 'Jan' AND nazwisko =
'Kowalski';
```

- usuwa z tabeli "pracownicy" wiersz (rekord) dotyczący pracownika o imieniu "Jan" i nazwisku "Kowalski".

```
ALTER TABLE pracownicy ADD COLUMN dzial varchar(255);
```

- dodaje do struktury tabeli "pracownicy" kolumnę "dzial" (dział), jako pole tekstowe o długości max. 255 znaków.

12.7.2. DBASE, CLIPPER, CA VO, HARBOUR

W latach 80-tych XX wieku – epoce komputerów personalnych i systemu operacyjnego DOS – najpopularniejszym narzędziem do tworzenia SZBD był **dBASE** firmy Ashton-Tate. System dBase składa się z języka programowania umożliwiającego operowanie na bazach danych, oraz środowiska IDE zawierającego interpreter tego języka i inne narzędzia programisty.

Polecenia systemu dBase można wpisywać i wykonywać w oknie komend – podobnie jak w MATLAB-ie – lub przy pomocy edytora tworzyć programy a następnie uruchamiać je w oknie komend przez wpisywanie ich nazw. Powstawały kolejne wersje dBase. Do wersji dBase IV dołączono dodatkowo obsługę SQL, a wersja dBase V była już zaprojektowana do pracy w Ms Windows. System dBASE nie zyskał jednak popularności w środowisku Windows, gdzie został szybko wyparty przez nowe produkty, jak Clipper i FoxPro bazujące na języku dBase, a potem największą popularność uzyskały wielodostępne systemy bazujące na standardzie SQL.

Clipper to system złożony z języka - będącego rozwinięciem dBase - oraz kompilatora tego języka. Kolejne wersje Clippera były rozwijane i sprzedawane w latach 1985-2000 najpierw przez firmę Nantucket Corporation, później (w r. 1992) przejętą przez firmę Computer Associates (CA). Oficjalnym następcą Clippera jest system CA-Visual Objects.

Zaletą języków dBase oraz Clipper jest połączenie możliwości operowania na bazach danych z możliwościami klasycznych języków programowania a w szczególności środkami do konstruowania interfejsu użytkownika, kontroli poprawności wprowadzanych danych oraz możliwości definiowania postaci wydruków. Dodatkowo Clipper pozwala tworzyć programy w postaci skompilowanej. Dzięki takim właściwościom Clipper był bardzo użytecznym narzędziem (między innymi dla autora tej książki) nie tylko do tworzenia SZBD – jak choćby opisywany dalej (p.12.9) system obsługi małej biblioteki - ale także do tworzenia inżynierskich programów obliczeniowych, jak wspomniany w p.1.2. program BGR-OS przeznaczony do obliczeń związanych z suwnicami.

Dla zobrazowania cech języka Clipper (a szerzej – języków z rodziny dBase), poniżej pokazano trzy charakterystyczne dla niego polecenia.

Wprowadzanie danych z klawiatury – w dBase i opartych na nim językach – realizowane jest przy pomocy polecenia @ **... Say ... Get ...** , które ma postać:

```
@ w,k SAY "napis" GET zm PICTURE format RANGE a,b VALID war
```

gdzie:

@	- symbol „AT" oznaczający w tym przypadku „od miejsca na ekranie ..."
w,k	- wiersz i kolumna podające miejsce ekranu w którym zacznie się "*napis*"
SAY	- słowo kluczowe oznaczające „wyświetl" (dosłownie „powiedz")
"napis"	- tekst stanowiący żądanie danych, który pojawi się przed polem wejściowym (do wprowadzania danych);
GET	- słowo kluczowe oznaczające „pobierz" i wyświetlające pole wejściowe
zm	- zmienna do której zostanie wprowadzona wartość z pola wejściowego;
PICTURE	- słowo kluczowe oznaczające „w postaci" (wg. „obrazu")
format	- wyrażenie tekstowe zawierające symbole formatowania i konwersji danych wprowadzanych w polu wejściowym;
RANGE	- słowo kluczowe oznaczające „dopuszczalny zakres";
a,b	- zakres [a,b] w jakim ma się zmieścić wprowadzana wartość;
VALID	- słowo kluczowe oznaczające „ważne gdy";
war	- warunek logiczny, któremu mają odpowiadać wprowadzane dane.

Jak widać polecenie to pozwala nie tylko precyzyjnie określić miejsce na ekranie ale także zapewnia objaśnienia i wszechstronną kontrolę wprowadzanych danych a także ich automatyczne formatowanie lub konwertowanie – na przykład wprowadzany tekst może być zamieniany na duże litery, numery telefonów mogą być rozbite na kilka pól numerycznych oddzielanych kreskami i t.p.

Drugą interesującą konstrukcją jest **zestaw poleceń tworzących menu ekranowe**, pozwalające dokonywać wyboru opcji. Do zestawu tego należą polecenia: **SET MESSAGE ..., @ w,k PROMPT ...,** oraz **MENU TO ...** , przy czym:

set message to *nr_wiersza*	- określa nr linii do wyświetlania objaśnień „*message*"
@ *w,k* **prompt "***t1***" message "***t2***"**	- wyświetla w wierszu *w* i kolumnie *k* tekst *t1* opcji do wybierania, a równocześnie jej objaśnienie (message) *t2*
menu to *zmienna*	- określa zmienną która w wyniku wybrania jednej z opcji uzyska wartość równą jej numerowi

Przykładowe obrazy menu uzyskanych tymi poleceniami zawierają m.in: Rys. 1.1, Rys. 12.21 i Rys. 12.22, przy czym objaśnienia uzyskiwane poleceniem **message** wyświetlane są w lini 22.

Trzecim z wybranych, interesujących poleceń Clipper-a, jest funkcja **DBEDIT()**. Funkcja ta może mieć nawet 12 parametrów, jednak są one opcjonalne. DBEDIT() wyświetla aktualnie otwarty (w bieżącym obszarze) plik bazy danych – w postaci tabelarycznej – i pozwala przeglądać ten plik oraz wybrać określony wiersz (rekord) do edycji.

Ponieważ tabela na ogół nie mieści się w oknie ekranowym więc może być przewijana w tym oknie, zarówno w pionie jak i w poziomie. Parametry pozwalają m.in. określić rozmiary okna, wybrać kolumny do wyświetlania, wyświetlić nagłówki inne niż nazwy pól, określić typ separatorów i t.p.

Jak widać z przytoczonych przykładów poleceń, Clipper to język dość wysokiego poziomu, dogodny do pisania aplikacji. Dlatego pomimo odejścia epoki DOS-a do historii, doczekał się szeregu kontynuacji dostosowanych do systemów Ms Windows.

Jedną z tych kontynuacji jest język **CA Visual Objects** (CA VO) - obiektowy język programowania wysokiego poziomu opracowany na bazie Clippera w firmie Computer Associates, a później odsprzedany firmie GrafxSoft, która opracowała m.in. jego wersję dla platformy .NET, pod nazwą **Vulcan.NET**.

Rozwijane są także darmowe, obiektowe kontynuacje Clipper-a i jedną z nich jest język **Harbour** (wzorowany na wersji CA-Clipper 5.2). W języku angielskim „clipper" to rodzaj okrętu a „harbour" to port, do którego ten okręt dopływa.

Harbour (http://harbour-project.sourceforge.net/) należy do oprogramowania Free Open Source Software i jest wieloplatformowy – posiadający kompilatory dla bardzo wielu systemów operacyjnych (także mobilnych), wielowątkowy, zorientowany obiektowo, wyposażony w biblioteki zapewniające m.in graficzny interfejs użytkownika.

12.8. BAZY DANYCH W PROGRAMIE MS ACCESS

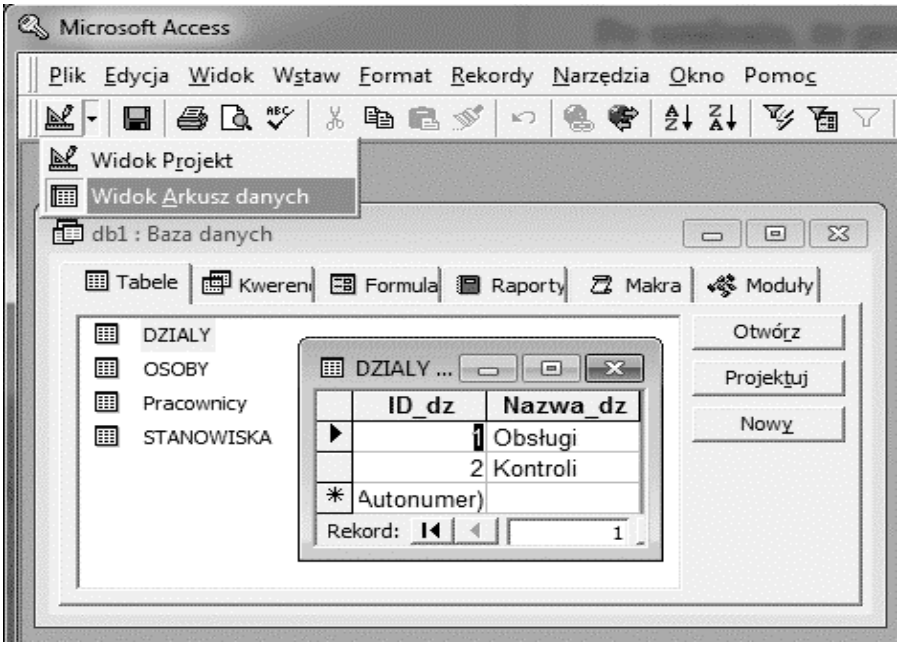
Ms Access to program do tworzenia lokalnych baz danych i dokonywania na nich operacji. Jest on jednym ze składników pakietu Microsoft Office. Średnio dokładne omówienie możliwości i bogactwa środków tego programu, wymaga osobnego podręcznika, i szereg takich podręczników jest ogólnie dostępnych, dlatego w niniejszym podrozdziale omówiono jedynie realizację podstawowych operacji bazodanowych w programie Ms Access.

Projektowanie bazy danych w programie Ms Access (z pakietu Ms Office), lub analogicznym programie z pakietu OpenOffice.Org – składa się z następujących etapów:

- **zaprojektowanie tabel** i ich **normalizacja** (rozkład na prostsze tabele),
- **utworzenie tabel** w programie Access lub analogicznym, z uwzględnieniem: określania nazw, typów i długości pól, wymagań i ograniczeń co do zakresu wartości i ewentualnej wartości domyślnej, wyboru klucza podstawowego, wyboru sposobów uporządkowania;
- określenie i utworzenie odpowiednich typów **relacji** między tabelami;
- opracowanie **zapytań** czyli tzw. **kwerend** (do selekcji, projekcji, ...) ;
- opracowanie **transakcji** (w Accessie zwanych **kwerendami funkcjonalnymi**);
- opracowanie **formularzy** – jako interfejsów do operowania na danych,
- opracowanie postaci różnorodnych wydruków zwanych **raportami**.

Aby utworzyć bazę danych, po uruchomieniu programu Ms Access, wybieramy opcję tworzenia nowej, pustej, bazy danych i zapisujemy ją wymyślając jej nazwę i wybierając folder. W głównym oknie bazy danych (Rys. 12.4) tworzone obiekty bazy danych czyli: tabele, kwerendy, formularze i raporty, będą pogrupowane na oddzielnych planszach wybieranych zakładkami. Przed dokonywaniem operacji na nich trzeba wybrać odpowiedni tryb czyli „widok" a mianowicie:

- **„Widok Projekt"** - pozwala tworzyć i modyfikować strukturę obiektów,
- **„Widok Arkusz danych"** – pozwala korzystać z utworzonych obiektów – wprowadzać, wybierać i modyfikować dane.



Rys. 12.4. Główne okno bazy danych w programie Ms Access

Przy omawianiu etapów tworzenia bazy danych w programie Ms Access pominiemy szereg szczegółów m.in. dotyczących obsługi interfejsu użytkownika (zmieniającego się wraz z wersjami Accessa). Skupimy się natomiast na istocie operacji i znaczeniu podstawowych pojęć – w dużej mierze niezależnych od wersji wykorzystywanego programu. Zainteresowanych szczegółami odsyłam do podręczników baz danych i Accessa, których jest bardzo wiele. Polecam krótki i dobry podręcznik [32].

12.8.1. PROJEKTOWANIE I NORMALIZACJA TABEL

Przy projektowaniu bazy danych nie zawsze łatwo jest **określić ile ma być tabel** i co mają rejestrować poszczególne tabele. Teoria baz danych definiuje postacie **normalne** bazy danych i określa jak do nich doprowadzić.

Praktycznie chodzi m.in o to **aby:**

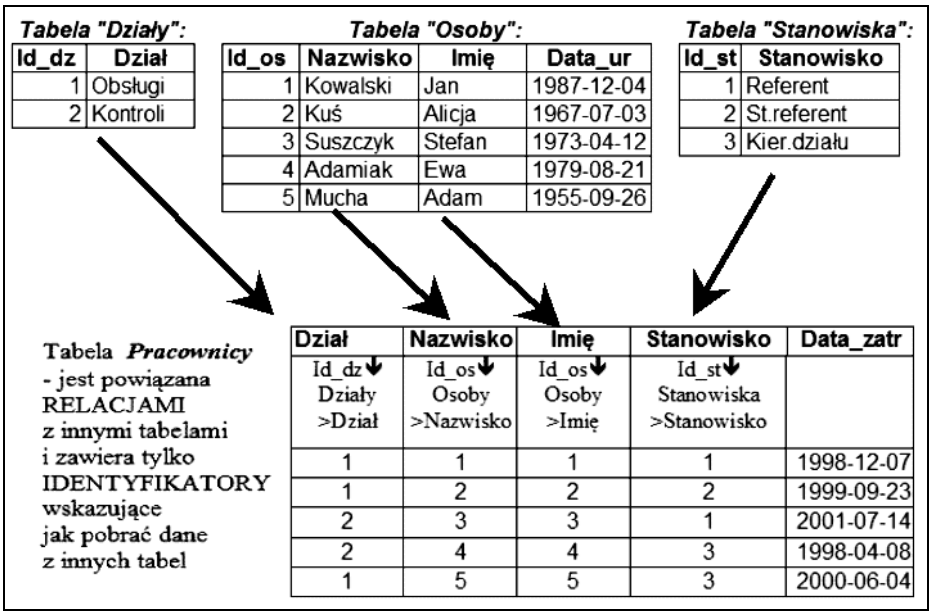
- w żadnej z tabel **informacje nie powtarzały się**,
- **każda tabela dotyczyła tylko jednego tematu** – a więc gromadziła informacje o określonej grupie: obiektów, zdarzeń, cech, stanów, ról i t.d.,
- **elementy tabeli były niepodzielne** (atomowe).

Załóżmy, że w pewnym przedsiębiorstwie sporządzono tabelę jak na Rys. 12.5.

Dział	Nazwisko	Imię	Stanowisko	Data_zatrudn
Obsługi	Kowalski	Jan	Referent	1998-12-07
Obsługi	Kuś	Alicja	St. referent	1999-09-23
Kontroli	Suszczyk	Stefan	Referent	2001-07-14
Kontroli	Adamiak	Ewa	Kier.działu	1998-04-08
Obsługi	Mucha	Adam	Kier.działu	2000-06-04

Rys. 12.5. Tabela *Pracownicy*

Jak widać, w tabeli tej powtarzają się nazwy stanowisk oraz działów, a więc nie jest to tabela znormalizowana i trzeba będzie ją rozłożyć kilka na prostszych tabel, jak to pokazano na Rys. 12.6. Obowiązuje zasada według której: **liczba tabel powinna być równa liczbie kategorii rejestrowanych podmiotów** (encji).



Rys. 12.6. Dekompozycja tabeli "Pracownicy" na tabele znormalizowane

Dla tabeli *Pracownicy* zrealizowano to tak jak pokazuje Rys. 12.6.

Relacyjna baza danych to zbiór wielu wzajemnie powiązanych tabel.

Tabele pokazane na Rys. 12.6 powiązane są **relacjami** przy pomocy identyfikatorów: „Id_dz", „Id_os", „Id_st". W efekcie każda informacja wystąpi tylko w jednym miejscu, a jedynie pewne identyfikatory (nie będące kluczami głównymi) będą się powtarzać.

Taki proces dekompozycji tabel na możliwie najprostsze tabele składowe, powiązane relacjami, nazywa się **normalizacją tabel.**

Dyskusyjne jest wyodrębnienie tabel: ***Osoby*** oraz ***Pracownicy***, szczególnie, jeśli obie dotyczą tych samych osób. Rzeczywiście, gdyby połączyć te tabele w jedną to i tak spełniałaby ona podane na początku tego podrozdzialu wymagania. Jedynym powodem rozdzielenia tabel może być fakt, że osobą jest się całe życie a pracownikiem tylko przez pewien okres a więc są to dwie różne role i jeśli nawet aktualnie połączymy obie tabele relacją „jeden-do-jeden" to nie musi tak być w przyszłości, gdy chcielibyśmy pozostawić w bazie dane osób, które przestały już być pracownikami.

Po ustaleniu, że przed utworzeniem tabeli ***Pracownicy*** potrzebne są trzy inne tabele, a mianowicie: ***Działy, Stanowiska*** i ***Osoby***, możemy przystąpić do tworzenia tych tabel.

Przebieg procesu tworzenia (definiowania) tabel w programie Ms Access omówiono na kolejnych stronach.

12.8.2. DEFINIOWANIE TABEL

Po uruchomieniu programu Ms Access i wybraniu opcji tworzenia nowej bazy danych, wybieramy zakładkę „Tabele" i opcję „Utwórz tabelę w widoku Projekt".

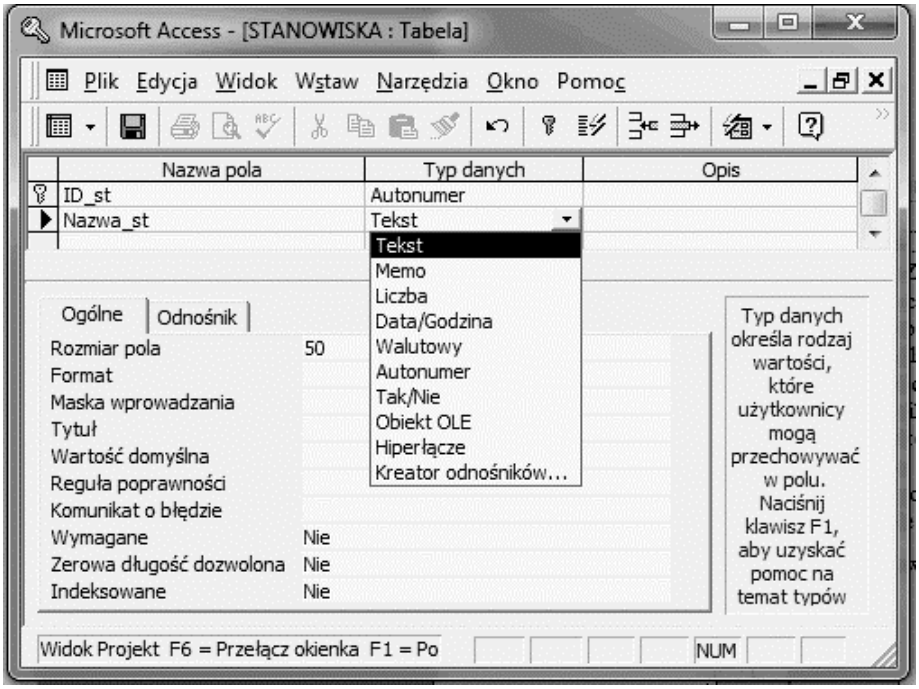
Definiowanie tabel najlepiej zacząć od **tabel podstawowych** – niezależnych od innych - stanowiących tak zwane „słowniki", to znaczy zawierających wykazy wykorzystywane przy wstawianiu wartości do pól innych tabel. W naszym przypadku najpierw zdefiniujemy tabele: ***Działy, Stanowiska*** i ***Osoby*** a dopiero potem (p.12.8.3.) tabelę ***Pracownicy***, która z nich korzysta, a więc jest wobec nich **tabelą zależną**.

Przy definiowaniu tabeli w widoku „Projekt" (Rys. 12.7) trzeba ustalić szereg **własności** dla **poszczególnych pól tej tabeli**, a mianowicie:

- **nazwę pola** – która powinna kojarzyć się z jego zawartością;
- **typ danych** – wybrać z listy rozwijalnej jak na Rys. 12.7, w przypadku klucza głównego często wykorzystywany jest typ „autonumer" – nie dopuszczający do powtórzeń i niejednoznaczności, przy nawiązywaniu relacji między tabelami można wykorzystać z listy typów „kreator odnośników";
- **rozmiar pola** – określa maksymalną liczbę znaków, którą można wprowadzić do pola (nie więcej niż 255);
- **format** - określa sposób wyświetlania danych w polu po zakończeniu ich wprowadzania (na przykład format daty: *dzień miesiąc rok* lub tylko *rok miesiąc);*
- **tytuł** - określa nagłówek pola wyświetlany podczas oglądania danych; jeśli komórka zostanie pusta, to tytuł jest tożsamy z nazwą pola;
- **indeksowane** – jeśli „**Tak**" – to powstanie plik indeksowy umożliwiający szybkie ustawienie uporządkowania tabeli według wartości tego pola, a jeśli wybierzemy opcję „**bez powtórzeń**" to indeks będzie miał unikalne (nie powtarzające się) wartości;

- **maska wprowadzania** - ustala wzorzec dla wprowadzania danych, uwzględniający miejsca na wprowadzane znaki oraz stałe znaki jak pauzy dla dat czy dwukropki dla godzin;
- **wartość domyślna** - ustala wartość wpisywaną automatycznie w nowym rekordzie;
- **reguła poprawności** - określa warunek (na przykład nierówność), który musi być spełniony, aby dana wprowadzana do pola została zaakceptowana;
- **komunikat o błędzie** - to tekst wyświetlany w przypadku próby wprowadzenia w polu danej, która nie spełnia reguły poprawności;
- **wymagane** - określa, czy pole musi być obowiązkowo wypełnione, czy też nie;
- **zerowa długość dozwolona** - określa, czy dopuszczalny jest ciąg znaków o zerowej długości złożony z dwóch cudzysłowów: " ".

Utworzoną tabelę trzeba **zapisać** na dysk. Jednak Access nie dopuści do zapisania, jeśli nie zdefiniowano klucza podstawowego tabeli, który będzie jednoznacznym i unikalnym identyfikatorem każdego rekordu tabeli.



Rys. 12.7. Tworzenie pól tabeli ***Stanowiska***

Zdefiniowanie klucza podstawowego (inaczej: głównego) dla każdej tabeli jest więc konieczne. Najlepiej zrobić to przed zapisywaniem – wystarczy w tym celu wskazać pole które ma być kluczem, kliknąć prawym przyciskiem myszy i wybrać z menu podręcznego odpowiednią opcję.

Jeśli żadne z pól nie może stanowić jednoznacznego identyfikatora rekordów tabeli to trzeba wstawić dodatkowe pole - jako identyfikator – i nadać mu typ „**autonumer**". Wówczas Access automatycznie zadba o to aby wartości tego identyfikatora nie powtarzały się, oraz były zwiększane przy dopisywaniu nowych rekordów. Jeśli usuniemy jakiś rekord to jego identyfikator także nie będzie już użyty dla innego rekordu.

12.8.3. WPROWADZANIE DANYCH DO TABEL SŁOWNIKOWYCH

Aby wprowadzić dane do zdefiniowanych już tabel ***Działy, Stanowiska*** i ***Osoby*** trzeba przełączyć widok „Projekt" na widok „Arkusz danych" wykorzystując menu główne lub pasek narzędzi. Pojawią się rubryki tabel do których trzeba wpisać dane. Klawisz [Tab] pozwoli wygodnie przechodzić do następnych pól.

12.8.4. USTALANIE RELACJI MIĘDZY TABELAMI

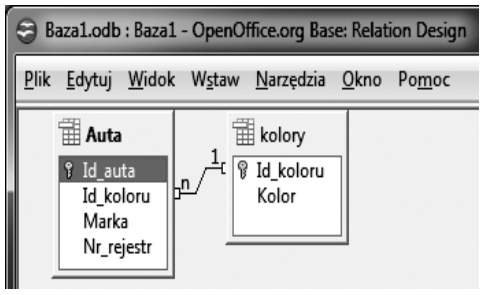
Po zdefiniowaniu tabel ***Działy, Stanowiska i Osoby***, możemy przystąpić do tworzenia tabeli ***Pracownicy***.

Jak już powiedziano – tabela ***Pracownicy*** ma wykorzystać dane z pozostałych trzech tabel. Będzie to możliwe dzięki **definiowaniu relacji** między tabelami.

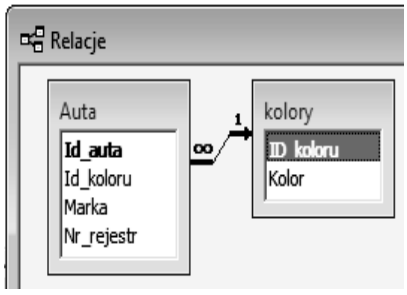
Relacja polega na połączeniu dwu tabel przez określenie pola wspólnego. Dzięki temu, jako wartości komórek z określonej kolumny (pola) jednej z tabel, wykorzystywane będą wartości określonej kolumny (pola) i określonych rekordów drugiej tabeli (wykazu, słownika), poprzez **odwoływanie się do ich identyfikatorów,** czyli wartości klucza podstawowego. Pole wykorzystujące wartości klucza podstawowego innej tabeli w Accesie będzie mieć taką samą nazwę jak tamten klucz i jest wtedy **kluczem obcym**.

Przykładowo: w tabeli „AUTA" (Rys. 12.8), pole „Id_koloru" ma wykorzystywać identyfikatory i nazwy kolorów opisanych w osobnej tabeli „KOLORY", zawierającej wykaz kolorów i posiadającej tylko dwa pola: „Id_koloru" oraz „Kolor". Po ustaleniu relacji, pole „kolor" w tabeli „AUTA" zmieni nazwę na „Id_koloru" i będzie kluczem obcym bo wykorzystuje wartości klucza podstawowego „Id_koloru" tabeli „KOLORY".

a)



b)



Rys. 12.8. Przykład relacji typu "jeden do wielu" a) w OpenOffice.org Base, b) w Ms Access

Możliwe są **trzy typy relacji**:

Relacja „jeden do wielu” (Rys. 12.8) - najczęściej stosowana– polega na takim powiązaniu pewnego pola tabeli nadrzędnej z tabelą słownikową (podstawową, podrzędną), że zamiast wpisywać wartość tego pola można wybrać jego wartość z listy jaką jest tabela słownikowa. Jeden rekord tabeli słownikowej może być wykorzystywany w wielu rekordach tabeli nadrzędnej – stąd nazwa „jeden do wielu”. Na przykład: ten sam kolor może posiadać wiele samochodów w tabeli ***Auta***, a stanowisko „Referent” wykorzystywane jest dla kilku pracowników z tabeli ***Pracownicy***.

Relacja „jeden do jednego” – polega na powiązaniu kluczy podstawowych dwu tabel dotyczących tych samych podmiotów na przykład tabeli danych personalnych ***Osoby*** z tabelą ***Pracownicy***. Może stworzyć efekt sklejenia tabel rozdzielonych z jakichś względów lub być wykorzystywana w tabeli będącej podzbiorem tabeli głównej [32]

Relacja „wiele do wielu” – w rzeczywistości składa się z dwu relacji „jeden do wielu” i wymaga istnienia tabeli pośredniczącej zwanej **tabelą łącza**, w której klucz podstawowy składa się z dwóch pól - kluczy obcych z dwu tabel wiązanych tą relacją.

Relacje między tabelami możemy definiować w programie Ms Access na dwa sposoby.

Pierwszy sposób – opisany nieco dalej - to definiowanie relacji już po zdefiniowaniu wszystkich tabel, w oknie „Relacje” (otwartym z menu: Narzędzia-Relacje).

Drugi sposób – prostszy **-** to definiowanie relacji w trakcie tworzenia tabeli zależnej (np.: ***Pracownicy***) – przez wykorzystanie „kreatora odnośników” z listy typów danych. Sposób ten jest możliwy jeśli tabele niezależne (podstawowe) - np.: ***Działy, Stanowiska, Osoby*** - są już zdefiniowane.

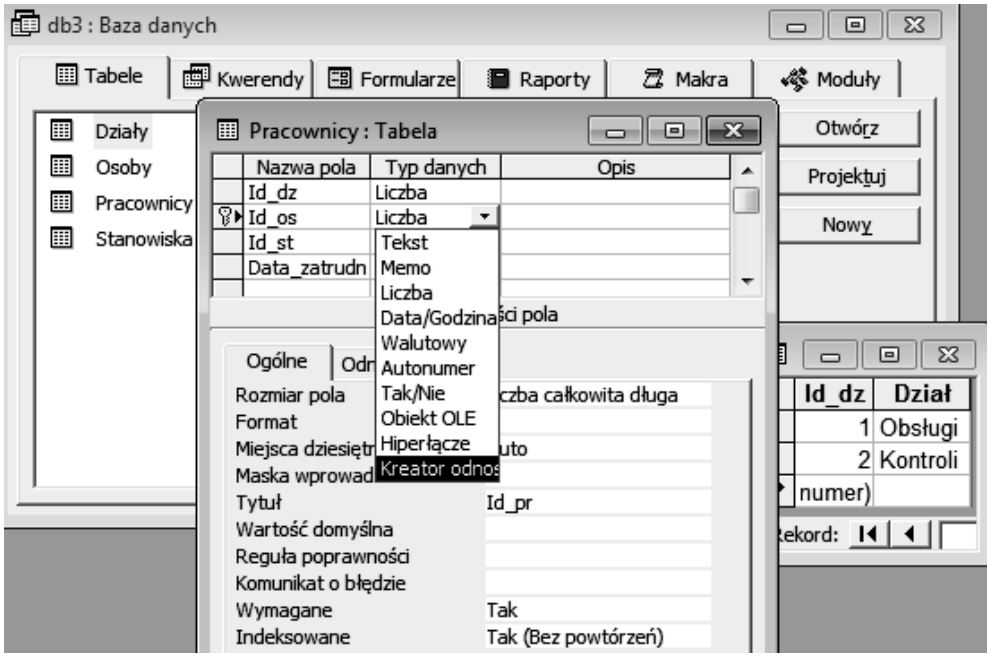
12.8.5. TWORZENIE RELACJI „KREATOREM ODNOŚNIKÓW”

„Kreator odnośników” użyjemy przy tworzeniu tabeli ***Pracownicy***:

- tworzymy nową pustą tabelę w widoku „Projekt”;
- pierwsze pole „*Dział*” ma wykorzystywać wartości z tabeli ***Działy***, więc po wprowadzeniu nazwy pola, wybieramy jako jego typ (z listy typów danych) „kreator odnośników”
- następnie wybieramy domyślną opcję: „*... kolumna odnośnika ma pobierać wartości z tabeli lub kwerendy*” oraz wybieramy tabelę ***Dzialy***;
- wybieramy dwa pola: „*Id_dz*” oraz „*Nazwa_dz*”;
- w następnych oknach pozostawiamy zaznaczoną opcję „ukryj kolumnę klucz” oraz nazwę Dział jako etykietę i kończymy przyciskiem”Utwórz”

Zauważymy, że Accesss zmienił nazwę pola *Dział* w tabeli ***Pracownicy*** na *ID_dz* czyli identyczną jak klucz podstawowy tabeli ***Działy*** do którego się to pole odwołuje. Jednakże, chociaż w obu tabelach jest teraz pole *ID_dz* ale tylko w tabeli ***Działy*** pełni rolę klucza podstawowego natomiast w tabeli ***Pracownicy*** pełni rolę tak zwanego **klucza obcego**.

A więc relacja jeden do wielu jest relacją między polem klucza podstawowego tabeli słownikowej (podstawowej, niezależnej) a polem drugiej tabeli (zależnej), nazywanym kluczem obcym. Klucz obcy przechowuje wartość klucza podstawowego tabeli słownikowej, identyfikującą rekord z którego zostaną pobrane informacje do tabeli zależnej. Access domyślnie zmienia nazwę pola klucza obcego na taką samą jak powiązany z nim klucz podstawowy tabeli słownikowej. Jednakże nie jest to obowiązkowe.



Rys. 12.9. Kreator odnośników

Kolejne pole ma odwoływać się do tabeli ***Osoby*** i pobierać stamtąd nazwisko i imię osoby. Ponownie wykorzystujemy jako typ: „Kreator odnośników", ale tym razem wybierając z tabeli ***Osoby*** trzy pola: *Id_os, Nazwisko, Imię*.

Następnie zaznaczamy wiersz *Id_os* i czynimy to pole kluczem podstawowym tabeli (wybierając w tym celu z menu kontekstowego pozycję „Klucz podstawowy").

Trzeci raz można by wykorzystać „Kreator odnośników" dla kolejnego pola „*Id_st*", wiążąc je w ten sposób relacją z tabelą ***Stanowiska***. Na razie ustawimy jednak dla pola „*Id_st*" typ „Liczba" aby móc nieco dalej pokazać inny sposób tworzenia relacji – w oknie „Relacje".

Dla ostatniego pola „*Data_zatrudn*" wybieramy **typ** „*Data/godz*", **format** „*Data krótka*" i **maskę wprowadzania**: „*0000-00-00*", która wymusza wprowadzanie ośmiu cyfr. Dodatkowo w polu „**Reguła poprawności**" możemy wpisać „***<=Date()***" co spowoduje nie dopuszczenie do wprowadzenia daty późniejszej od daty bieżącej.

Tabelę zapisujemy z nazwą ***Pracownicy***.

Warto – po wstawieniu odnośnika – obejrzeć jego własności, wybierając zakładkę „Odnośnik" jak to pokazano na Rys. 12.10. Jako własność „źródło wierszy" pola *Id_os* zostało wygenerowane polecenie języka SQL:

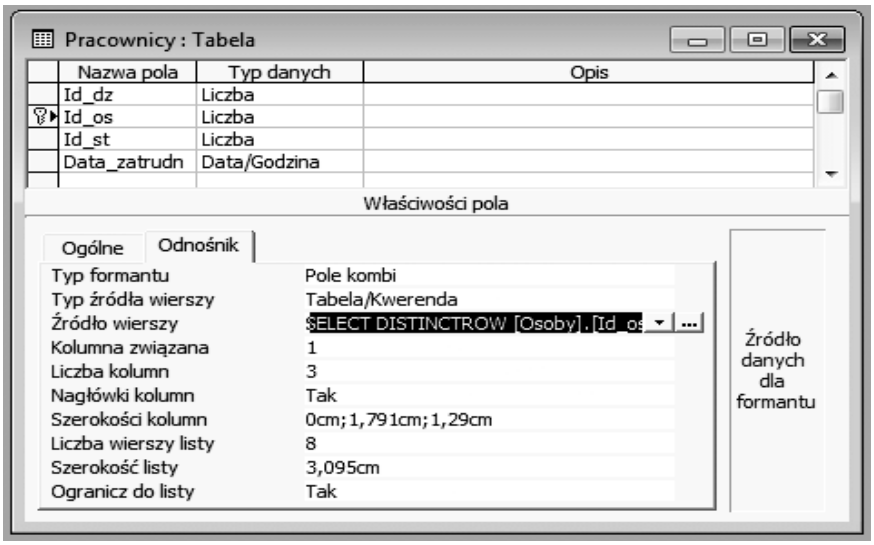
```
SELECT DISTINCTROW
[OSOBY].[Id_os],[OSOBY].[Nazwisko],[OSOBY].[Imie]
FROM [OSOBY];
```

Jeśli po wybraniu z listy rozwijalnej nazwiska i imienia pokaże się jedynie nazwisko to trzeba w tym poleceniu skleić wartości pól *Nazwisko* i *Imie* stosując znak „&":

```
SELECT DISTINCTROW
[OSOBY].[Id_os], ([OSOBY].[Nazwisko]&' '& [OSOBY].[Imie])
FROM [OSOBY];
```

lub znak „+" do sklejania łańcuchów znaków:

```
SELECT DISTINCTROW
[OSOBY].[Id_os], ([OSOBY].[Nazwisko]+' '+ [OSOBY].[Imie])
FROM [OSOBY];
```

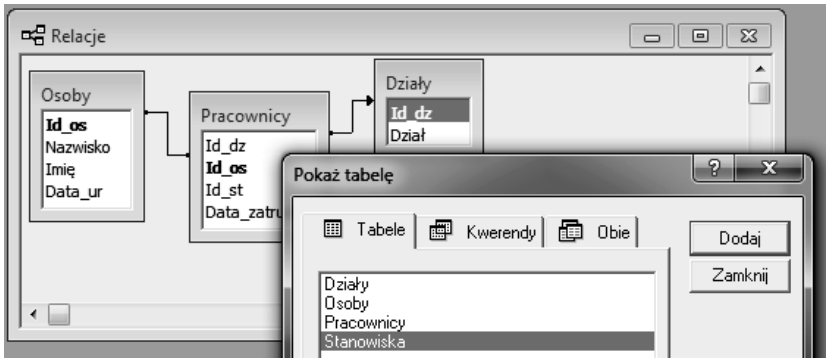


Rys. 12.10. Własności odnośnika tworzącego relację

12.8.6. TWORZENIE RELACJI W OKNIE „RELACJE"

Powyżej opisano jak użyć kreatora odnośników do powiązanie tabeli ***Pracownicy*** z tabelami ***Działy*** oraz ***Osoby***. Pozostało utworzenie relacji tabeli ***Pracownicy*** z tabelą słownikową ***Stanowiska***. Zrobimy to przy pomocy okna „Relacje" otwartego z menu „Narzędzia".

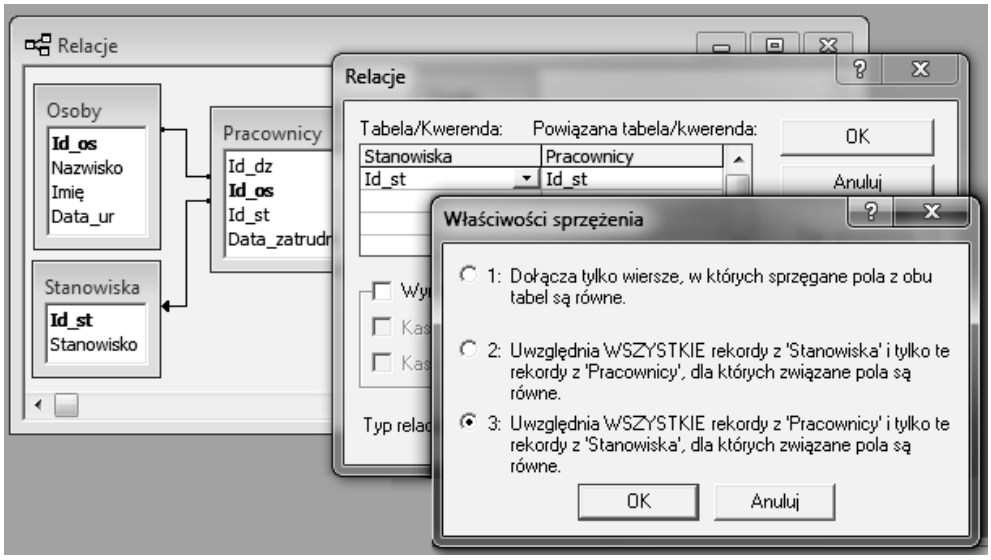
Po otwarciu tego okna i wybraniu z menu kontekstowego „Pokaż wszystko”, ukażą się trzy tabele i dwie utworzone już między nimi relacje. Wybieramy z menu kontekstowego opcję „Pokaż tabelę” aby wstawić jeszcze tabelę ***Stanowiska*** (Rys. 12.11).



Rys. 12.11. Wstawianie tabeli do okna relacji

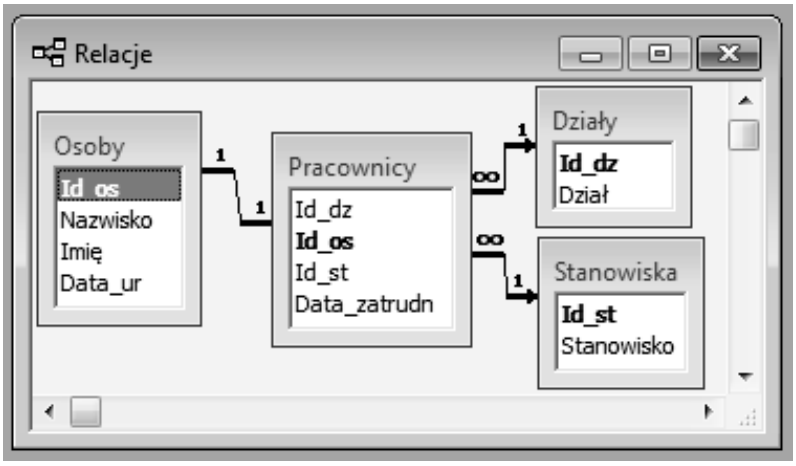
Następnie – aby utworzyć nową relację - przeciągamy myszką pole *Id_st* z tabeli ***Stanowiska*** do pola *Id_st* w tabeli ***Pracownicy***.

Właściwości utworzonej relacji możemy modyfikować po jej zaznaczeniu (przez kliknięcie) i wybraniu z menu kontekstowego „Edytuj relację”. W oknie tym widać, że utworzono relację typu „jeden do wielu”. W szczególności możemy ustalać typ sprzężenia – co pokazuje Rys. 12.12. Zaznaczona jest trzecia opcja bo: dla każdego pracownika musi być określone stanowisko, natomiast spis stanowisk może być dowolnie długi i nie wszystkie stanowiska muszą być wykorzystane a niektóre wykorzystywane będą wielokrotnie.



Rys. 12.12. Ustalanie właściwości relacji

Po ustaleniu właściwości wszystkich trzech relacji oraz po dodatkowym zaznaczeniu opcji „Wymuszaj więzy integralności”, otrzymujemy schemat relacji jak na Rys. 12.13.



Rys. 12.13. Relacje przykładowej bazy po włączeniu opcji „Wymuszaj więzy integralności”

12.8.7. KWERENDY CZYLI ZAPYTANIA

Kwerendy - zwane też zapytaniami - umożliwiają **pozyskiwanie informacji** z baz danych a także **wykonywanie operacji na danych**.

Zagadnienia dotyczące kwerend są bardzo rozbudowane i omówione obszernie w wielu ogólnie dostępnych podręcznikach Accessa, a w ramach niniejszego podręcznika zmieści się jedynie bardzo ogólne ich omówienie.

Główne kategorie operacji realizowanych przy pomocy konstruowania kwerend są następujące:

- wybieranie informacji z jednej lub wielu tabel (kwerendy wybierające),
- dołączanie, usuwanie, aktualizowanie danych oraz tworzenie nowych tabel na podstawie dotychczasowych (kwerendy funkcjonalne),
- przetwarzanie danych, wyznaczanie podsumowań i innego typu danych zbiorczych (kwerendy podsumowujące i krzyżowe)

Dowolne operacje można także realizować definiując kwerendy w języku SQL.

Elementarne przykłady konstruowania niektórych kwerend pokazano dalej.

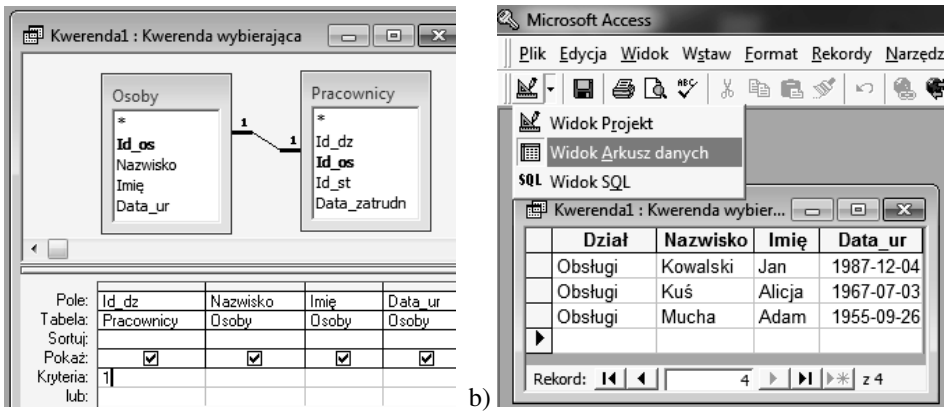
12.8.8. KWERENDY WYBIERAJĄCE

Załóżmy że chcemy mieć uzyskać wykaz nazwisk, imion i dat urodzenia tylko osób pracujących w dziale „Obsługi”. W tym celu trzeba utworzyć **kwerendę wybierającą**.

W oknie bazy danych wybieramy zakładkę „*Kwerendy*" i klikamy przycisk „Nowa" a następnie korzystamy z opcji „Widok projekt" lub „Kreator prostych kwerend" – aby wybrać z odpowiednich tabel interesujące nas pola: *Pracownicy.Id_dz, Osoby.Nazwisko, Osoby.Imię, Osoby.Data_ur* (nazwa tabeli połączona jest kropką z nazwą wybierane go pola).

W siatce projektowania kwerend, dla pola *Pracownicy.Id_dz* wstawiamy jeszcze jako „Kryterium" wartość 1 czyli identyfikator interesującego nas działu „Obsługi".

Otrzymujemy efekt jak na Rys. 12.14.



Rys. 12.14. Przykład kwerendy wybierającej: a) widok projekt, b) widok arkusz danych

Trzecia postać widoku kwerendy to „Widok SQL" czyli wygenerowane polecenia w języku SQL, które są następujące:

```
SELECT DISTINCTROW Pracownicy.Id_dz, Osoby.Nazwisko, Osoby.Imię, Osoby.Data_ur
FROM Osoby INNER JOIN Pracownicy ON Osoby.Id_os = Pracownicy.Id_os
WHERE (((Pracownicy.Id_dz)=1));
```

12.8.9. KWERENDY OBLICZENIOWE

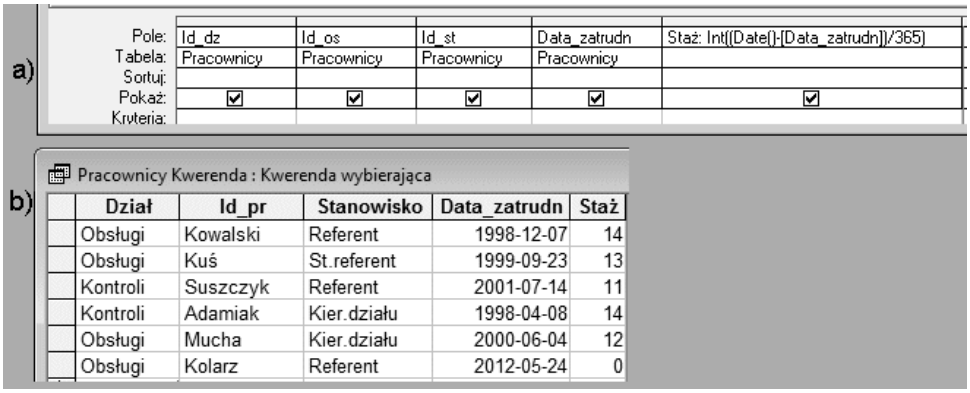
Przy pomocy odpowiednich kwerend można wykonywać wiele różnych obliczeń, jak wyznaczanie sumy lub średniej, działania na wartościach określonych pól, czy operacje na datach. Wyniki wyznaczane będą przy otwieraniu kwerendy, zawsze na podstawie aktualnych danych zawartych w bazie.

Standardowo, dla każdego pola – w wierszu *Podsumowanie* - zdefiniowane są obliczenia zwane **podsumowaniami**, jak: suma, średnia, minimum, maksimum, odchylenie standardowe, wariancja oraz zliczanie elementów. Jeśli taki wiersz jest niewidoczny to trzeba go włączyć przy pomocy menu podręcznego.

Inne **obliczenia** - **definiowane przez użytkownika** - umożliwiają wykonywanie operacji na danych liczbowych, datach i danych tekstowych, z jednego lub z wielu pól. W tym celu należy utworzyć **nowe pole obliczeniowe** bezpośrednio w siatce projektu. W komórce nagłówkowej nowego pola należy **wpisać nazwę** jaką nadajemy temu polu, **a po niej dwukropek i wyrażenie obliczeniowe.**

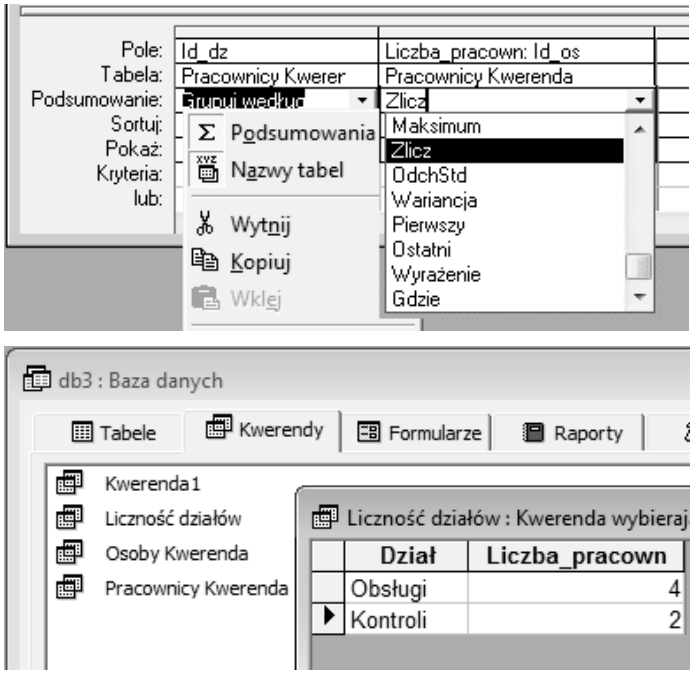
Jako przykład, utworzymy kwerendę zawierającą wszystkie pola tabeli ***Pracownicy***, a następnie, w osobnym polu „*Staż*” (Rys. 12.15), chcemy otrzymać liczbę pełnych lat pracy, na podstawie daty zatrudnienia i daty bieżącej, według wzoru:

```
Staż: Int((Date()-[Data_zatrudn])/365)
```



Rys. 12.15. Kwerenda obliczająca liczbę pełnych lat pracy. a) projekt, b) arkusz danych

Jak widać na Rys. 12.15a, wzór ten wpisano w nagłówku nowego pola, dzięki czemu, po przełączeniu do widoku „arkusz” (Rys. 12.15b), otrzymaliśmy wartości kolumny „*Staż*”.



Rys. 12.16. Kwerenda zliczająca pracowników w działach: a) projekt, b) arkusz

Drugim przykładem jest kwerenda **zliczająca** pracowników w poszczególnych działach, przedstawiona na Rys. 12.16.

Tworzymy kwerendę wybierając z tabeli „Pracownicy" tylko dwa pola: *Id_dz* i *Id_os.*

Następnie w siatce projektowej musi pojawić się wiersz zatytułowany „Podsumowania". W tym celu klikamy prawym przyciskiem myszki dowolny wiersz i wybieramy z menu kontekstowego: „*Podsumowania*" (jak na Rys. 12.16.a). W polu *Id_dz* pozostawiamy „*Grupuj według*", natomiast w polu *Id_os* wybieramy opcję „*Zlicz*". Dodatkowo, aby nagłówek tego pola odpowiadał zawartości, zmieniamy go na:

`Liczba_pracown: [Id_os]`.

Efekt uzyskany po przełączeniu do widoku „Arkusz" pokazano na Rys. 12.16.b.

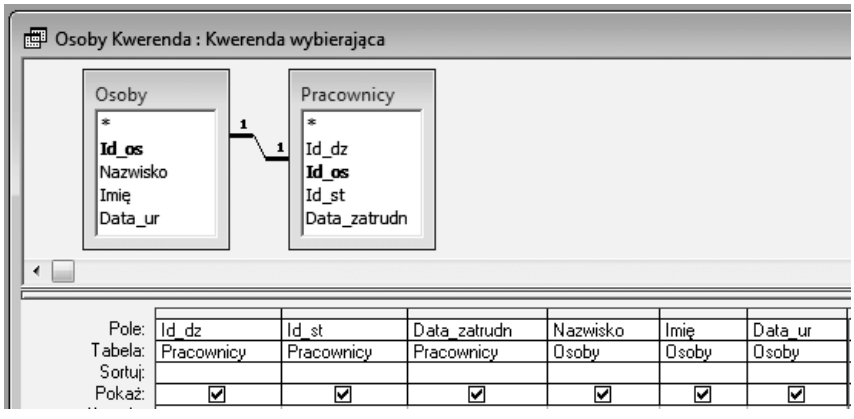
12.8.10. KWERENDY FUNKCJONALNE

Kwerenda funkcjonalna pozwala na wprowadzenie zmian w serii rekordów. Istnieją cztery rodzaje kwerend funkcjonalnych:

- **usuwająca** - usuwa grupę rekordów z jednej lub kilku tabel;
- **aktualizująca** – pozwala zmienić wartości grupy rekordów tabeli lub kilku tabel, na przykład dokonać zmian cen produktów czy podwyżek płac;
- **dołączająca** – pozwala dopisać nowe rekordy, kopiując je z innych tabel, lub dołączyć dane z pól wybranych na podstawie określonych kryteriów.
- **tworząca nową tabelę** z całości lub części danych znajdujących się w innych tabelach, przykładowo może to być **kopia zapasowa**.

12.8.11. TWORZENIE FORMULARZA

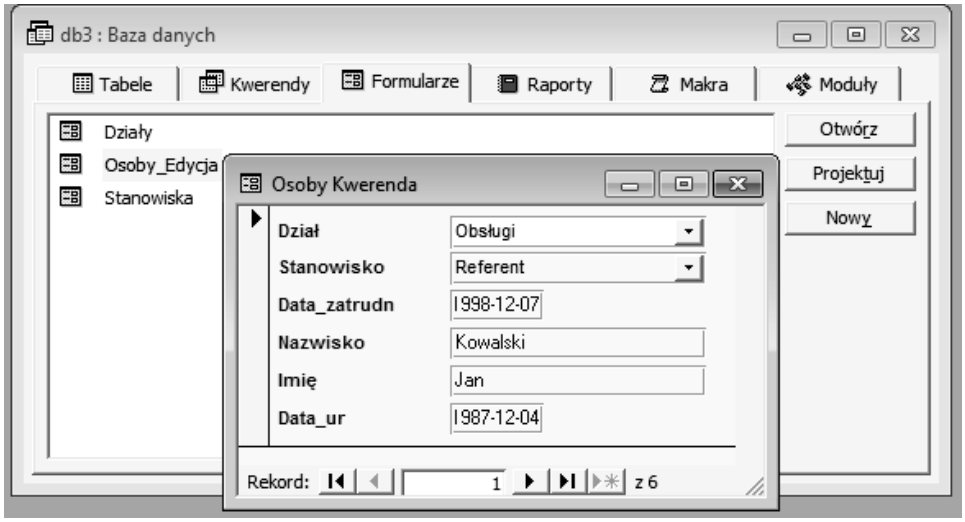
Formularz pozwala pokazać na ekranie pojedynczy rekord i jest dogodnym środkiem do przeglądania, edycji lub dopisywania rekordów. Przy pomocy formularza mamy więc „blankietową" postać danych, gdzie każdy rekord jest oddzielną „kartą", podobnie jak w papierowej kartotece.



Rys. 12.17. Kwerenda "Osoby" dla formularza wprowadzania danych

Formularze można tworzyć na podstawie tabel lub kwerend.

Aby utworzyć formularz do wprowadzania danych o osobach (pracownikach) utworzymy najpierw kwerendę jak na Rys. 12.17. Po wybraniu zakładki „Formularze" i kliknięciu przycisku „Nowy", bardzo łatwo – przy użyciu kreatora – otrzymać z tej kwerendy formularz pokazany na Rys. 12.18.



Rys. 12.18. Formularz do wprowadzania danych oparty na kwerendzie "Osoby"

Formularz ten pozwala przeglądać oraz wprowadzać wszystkie dane nowych osób, natomiast nie pozwala uzupełniać spisów „***Działy***" oraz „***Stanowiska***". Do tego można utworzyć oddzielne formularze lub dopisywać dane do odpowiednich tabel.

12.8.12. GENEROWANIE RAPORTÓW

Raportami nazywane są w Ms Access różnorodne wydruki, najczęściej wielostronicowe. Zawartość informacyjna raportu pobierana będzie z określonych przez projektanta tabel lub kwerend, które dodatkowo mogą być poddane filtrowaniu według określonych kryteriów. Oprócz informacji pobieranych z bazy, raport zawiera takie elementy jak: tytuły tabel oraz nagłówki kolumn, nagłówki i stopki stron z uwzględnieniem dat i numeracji, a także różnego typu podsumowania.

Projektowanie postaci raportów jest bardzo podobne do projektowania formularzy. W najprostszych przypadkach można wykorzystać „kreator raportów" i wybrać jedną ze standardowych postaci. Posługiwanie się „kreatorem raportów" do ustalenia jego wstępnego projektu jest korzystne także dla zaawansowanych użytkowników, którzy następnie mogą dostosowywać jego postać do własnych potrzeb.

Proponowane przez kreator standardowe postacie to:

- autoraport kolumnowy – w którym każde pole wystąpi w oddzielnym wierszu, z etykietą (objaśnieniem) umieszczonym z jego lewej strony,

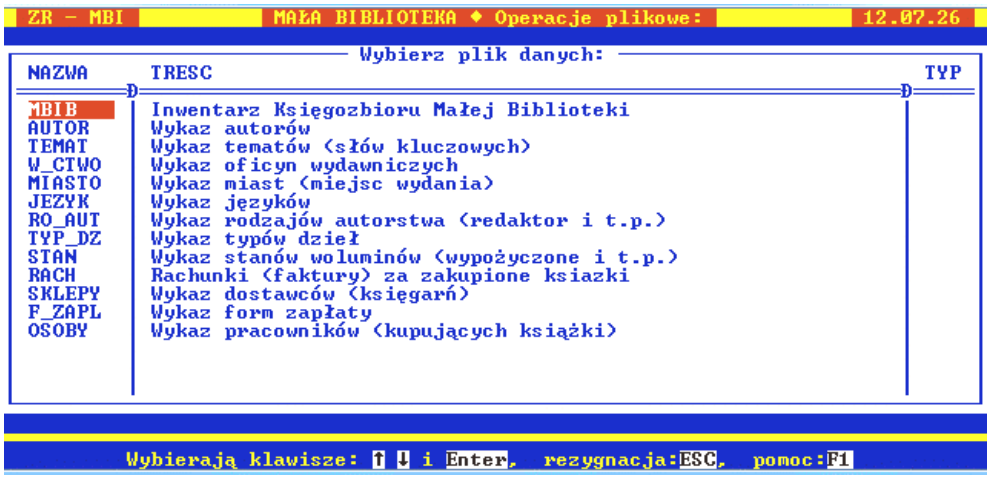
– autoraport tabelaryczny – w którym kolumny odpowiadają polom a rekordom odpowiadają wiersze, natomiast etykiety są tylko u góry stron.

Szczególnymi rodzajami raportów są etykiety adresowe oraz identyfikatory personalne.

12.9. PRZYKŁAD MAŁEJ LOKALNEJ BAZY DANYCH

Opisywany tu przykładowy system zarządzania bazą danych małej biblioteki, zrealizowano jako jeden program o nazwie ZR-MBI (Rys. 12.19), napisany w języku Clipper.

Rys. 12.19. Plansza tytułowa programu ZR-MBI

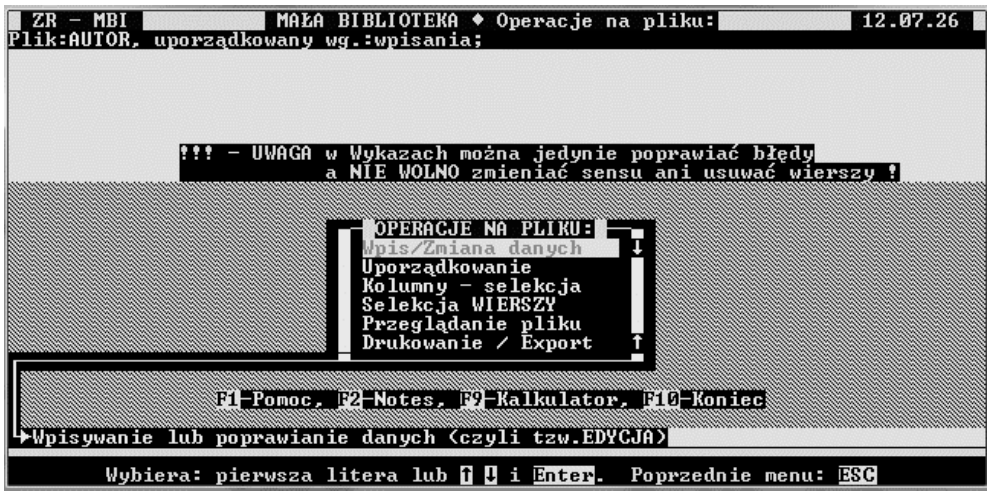


Rys. 12.20. Menu wyboru pliku bazy danych

Program ZR-MBI powstawał w okresie gdy większość oprogramowania pełniącego odpowiedzialne funkcje, jak choćby obsługa banków, tworzono tak, aby mogło pracować zarówno w systemie Ms Windows jak i w trybie tekstowym w Ms DOS – gdzie zapewniało większą niezawodność. Tak też pracuje opisywany program.

Pierwszą wersję programu ZR-MBI tworzono do pracy pod kontrolą systemu Ms DOS, jednak opisywana poniżej wersja może pracować w Ms Windows.

Po wybraniu pliku bazy danych (Rys. 12.20) pojawia się menu główne (Rys. 12.21).

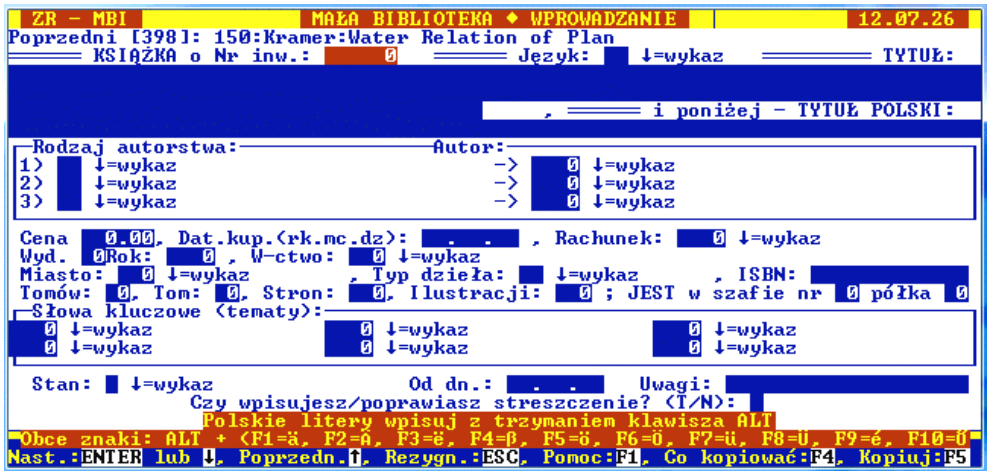


Rys. 12.21. Menu główne programu ZR-MBI

Gdy wybranym plikiem jest wykaz, do którego odwołują się wartości pól pliku głównego, wówczas pojawia się także (Rys. 12.21) informacja o ograniczeniach operacji.

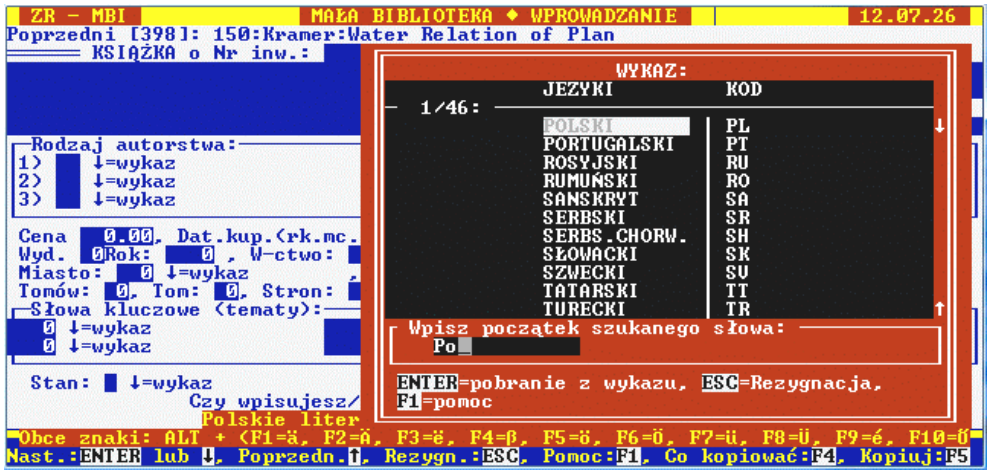
Rys. 12.22. Menu edycji wybranego zbioru

Istotną cechą programu jest możliwość wielokrotnego wykorzystywania raz wpisanych informacji przez wybieranie ich z wykazów (Rys. 12.23) zamiast powtórnego wpisywania. Dotyczy to autorów, języków, rodzajów autorstwa, wydawnictw, słów kluczowych i in. Program umożliwia też wpisywanie charakterystycznych znaków drukarskich (tzw. diakrytycznych) polskich i z większości języków europejskich (Rys. 12.23).



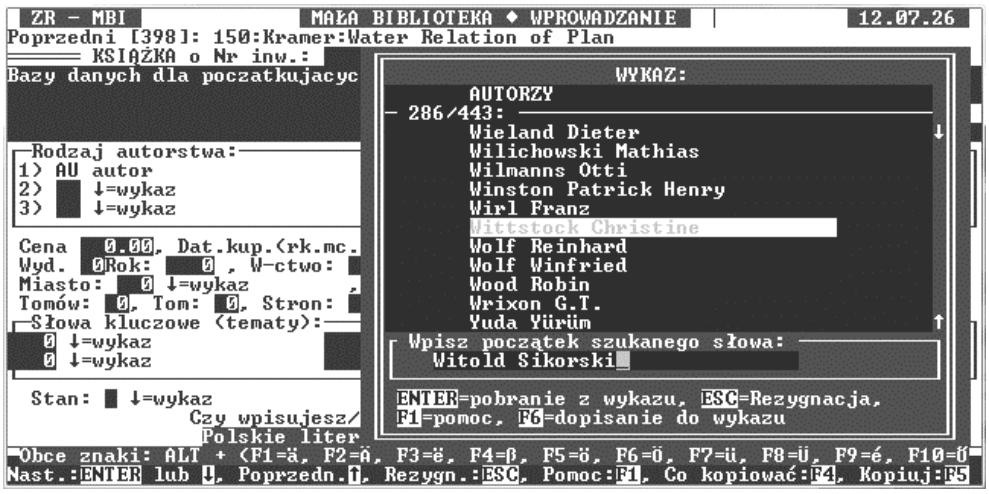
Rys. 12.23. Korzystanie z wykazów przy wprowadzaniu informacji o książce

Wyszukiwanie w wykazie odbywa się automatycznie przez wpisanie początkowych liter (Rys. 12.24) i po znalezieniu zatwierdzenie klawiszem ENTER.



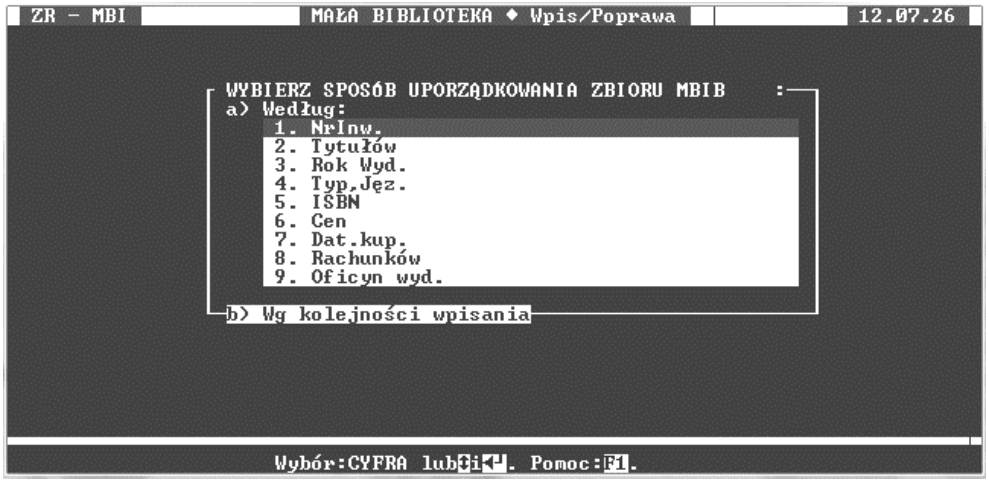
Rys. 12.24. Wyszukiwanie języka w wykazie

W przypadku nie znalezienia danej pozycji (np. autora) należy dokończyć wpisywanie a następnie (Rys. 12.25) nacisnąć klawisz F6 co spowoduje automatyczne dopisanie do wykazu tego co właśnie wpisaliśmy.



Rys. 12.25. Dopisywanie nie znalezionego autora do wykazu

Dla każdego z wybranych do edycji zbiorów można określić jeden z możliwych porządków przeglądania przez wybranie odpowiedniego zbioru indeksów (Rys. 12.26).

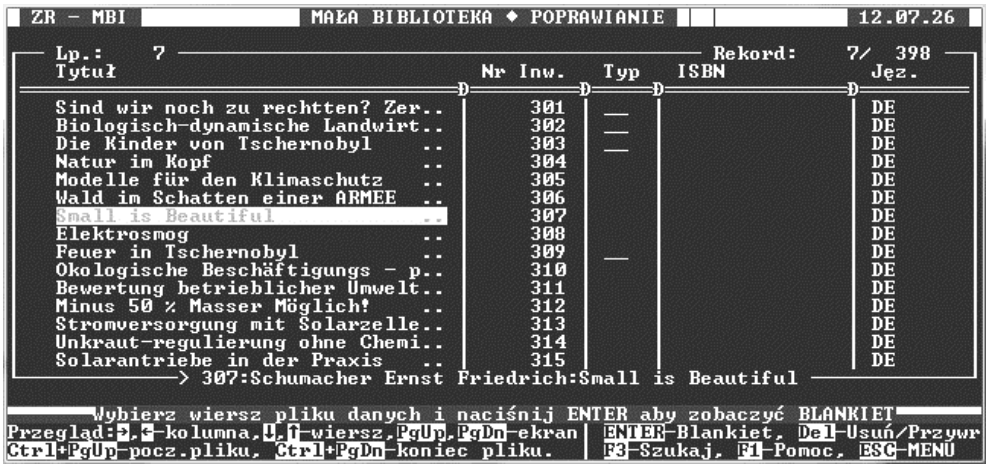


Rys. 12.26. Wybór porządku przeglądania

Zarówno sam program jak i wszystkie pliki baz danych i pliki pomocnicze zajmują bardzo niewiele miejsca. Przykładowo program wraz z plikami zawierającymi dane 398 książek, wykazy: 443 autorów i 131 oficyn wydawniczych zajmuje 511 KB.

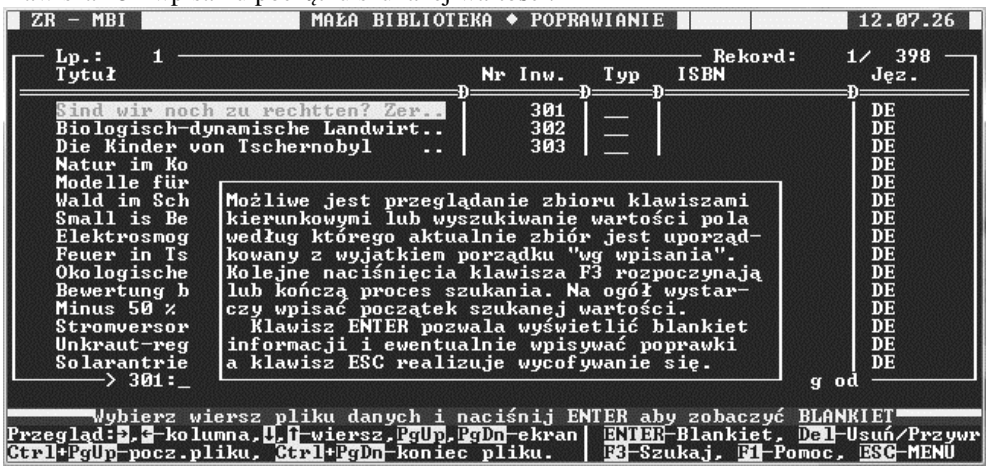
Szybkość zarówno uruchamiania się jak i działania programu jest natomiast znacznie większa niż programów działających w trybie graficznym i dedykowanych wyłącznie dla MS-Windows jak chociażby aplikacje bazo-danowe Ms Accessa, co było dla istotne dla „słabych” konfiguracji sprzętu

System pozwala rejestrować książki z długimi tytułami oryginalnymi (max.199 znaków) i osobno ich polskimi tłumaczeniami (max.106 zn.) a niezależnie od tego można wpisywać dowolnej objętości (max. 64 KB) streszczenia. Maksymalną liczbę rejestrowanych dla jednej książki autorów ograniczono do trzech.



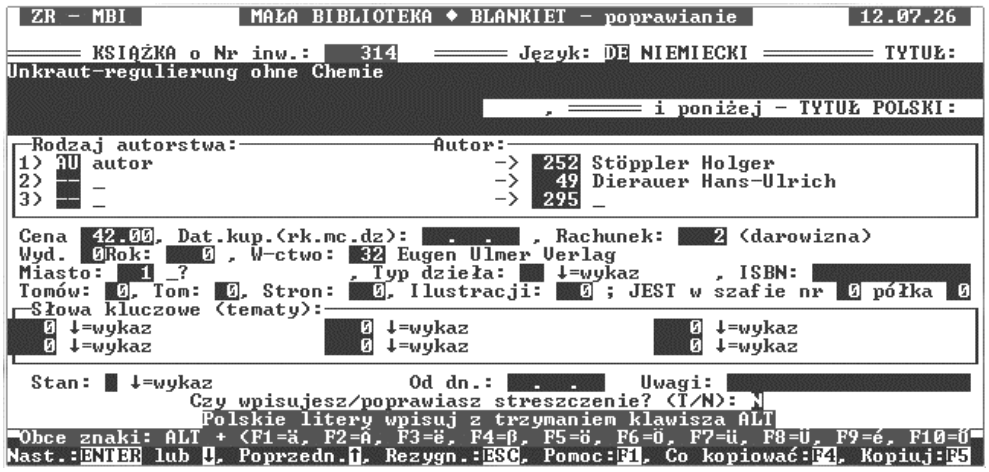
Rys. 12.27. Przeglądanie księgozbioru w postaci tabeli

Przy przeglądaniu lub poprawianiu, ukazuje się najpierw (Rys. 12.27) **tabela** tytułów księgozbioru. W tabeli tej można prowadzić automatyczne wyszukiwanie (Rys. 12.28) wartości tego pola według którego zbiór jest aktualnie uporządkowany – po naciśnięciu klawisza F3 i wpisaniu początku szukanej wartości.



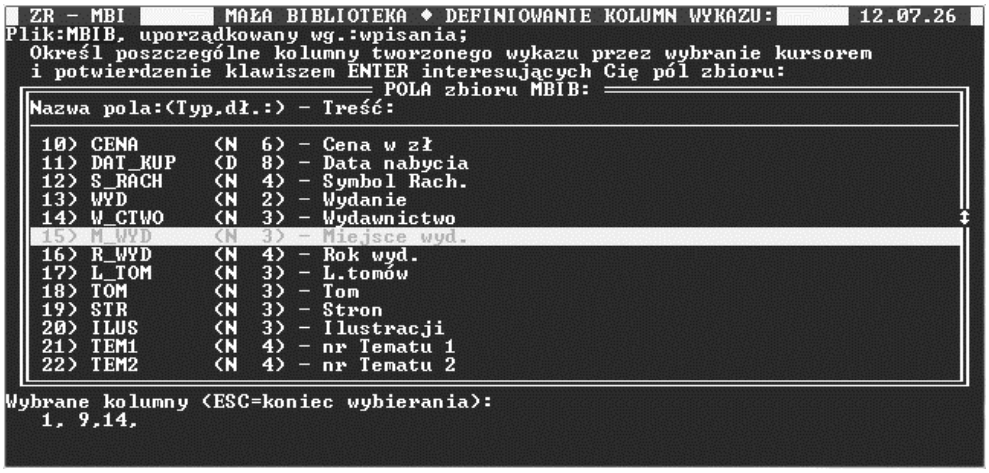
Rys. 12.28. Wyszukiwanie w tabeli

Po znalezieniu szukanego tytułu książki i zatwierdzeniu klawiszem ENTER, ukazuje się **blankiet** (Rys. 12.29) z danymi wybranej książki (a dokładniej woluminu czyli jej egzemplarza w bibliotece).



Rys. 12.29. Blankiet danych wybranego woluminu

Dla uzyskania wydruku dowolnej listy wybranych pozycji należy wcześniej dokonać **wyboru** interesujących nas **pól** zbioru (np.: autor , tytuł, rok wydania), co pokazano na Rys. 12.30.



Rys. 12.30. Wybór interesujących pól do wydruku

Oddzielnie musimy dokonać **selekcji podzbioru wierszy** z bazy danych księgozbioru, podając warunek lub warunki, które muszą być spełnione przez wybierane pozycje.
Najpierw wybieramy z menu sposób selekcji (Rys. 12.31) według:

- autora, albo
- tematu czyli słów kluczowych (Rys. 12.32), albo
- według dowolnych warunków przez skonstruowanie wyrażenia logicznego

Przykładowe wyniki selekcji na temat „Czarnobyl” pokazuje RYS. 12.33

Rys. 12.31. Wybór sposobu selekcji wierszy

Rys. 12.32. Selekcja książek według słów kluczowych (tematyki)

Rys. 12.33. Wynik selekcji na temat “Czarnobyl”

Po wybraniu z menu głównego pozycji: **„Drukowanie/Export”** pojawi się informacja o wybranych wcześniej wierszach (rekordach) i kolumnach (polach).

```
_ ZR - MBI ___ MAŁA BIBLIOTEKA ♦ DRUKOWANIE LIST __ 97.11.16 _
Plik:RACH, uporządkowany wg.:wpisania;
+-------------- DRUKOWANE BĘDĄ WYBRANE WCZEŚNIEJ: -----------+
|-> KOLUMNY (  3 z   9) uzupełnione o "L.p.":                |
| L.p.; SYMBOL; ILP; SUMA                                    |
|                                                            |
|-> WIERSZE - wybrany podzbiór:    15 z    20                |
|  dla warunku:SUMA>50                                       |
+------------ możesz zrezygnować klawiszem ESC --------------+
```

Rys. 12.34. Informacja o wybranych rekordach i polach.

Następnie pojawiają się puste pola dla wpisania dwu linii nagłówka wydruku, oraz pytania dotyczące: postaci wydruku, standardu polskich liter i numeru pliku wydruku. Jeśli już plik o tej nazwie i numerze istnieje, to system pyta czy zastąpić go nowym.

```
_ ZR - MBI ___ MAŁA BIBLIOTEKA ♦ DRUKOWANIE LIST __ 97.11.16 _
Plik:RACH, uporządkowany wg.:wpisania;

+--Wpisz 2-gą i 3-cią linię tytułową wydruku (max.60 zn.):---+
| Szkoła Ochrony i Inżynierii Środowiska im.Walerego Goetla  |
| Wykaz rachunków za zakupione książki    +DATA              |
| dla sumarycznych kwot powyżej 50 zł     + STR              |
+------------------------------------------------------------+
  Czy każde pole drukować od nowego wiersza (w słupku) T/N: N
  Czy drukować: 1=z ramkami, 0=bez ramek,  0 czy 1: 1
  Wybierz standard kodowania polskich liter w którym działa
twoja drukarka:
                            +------------------+
                            | Mazovia          |
                            | IBM-LATIN2       |
                            | Windows          |
                            | Tag              |
                            | Ventura          |
                            +------------------+
                         TEST DRUKOWANIA POLSKICH ZNAKOW
Wydruk do pliku dyskowego o nazwie "LISTAnr.il2",
gdzie "nr" jest liczbą którą teraz musisz podać, nr:  1
Istnieje już plik o tym numerze! Czy zastąpć go nowym (T/N): N
```

Rys. 12.35. Ekran generowania wydruku

Opisywany powyżej przykład programu do obsługi małej lokalnej bazy danych, zamieszczono w celu pokazania ważniejszych zagadnień w praktyce oraz wybranych możliwości ich rozwiązywania, łącznie ze stosowaniem ułatwień istotnych dla użytkownika. Zastosowany język programowania i narzędzia programistyczne są natomiast dla użytkownika mało istotne.

13. ZARYS METOD SZTUCZNEJ INTELIGENCJI

Ludzi myślących zawsze fascynowała doskonałość i funkcjonalność organizmów żywych, ich zdolności do adaptacji, rozmnażania i ewoluowania. Skłaniało to do badań a także marzeń o stworzeniu sztucznego życia, na przykład inteligentnych maszyn. Obecnie zastępowanie ludzi komputerami i robotami - także tam gdzie niezbędne jest inteligentne działanie – staje się możliwe a w wielu przypadkach konieczne. Komputerowe analizatory obrazów i złożonych sygnałów, ułatwiają stawianie diagnoz lekarzom, sterują rakietami, rozpoznają banknoty i wydają resztę w automatach sprzedających bilety; internetowe systemy ekspertowe ułatwiają wybór towarów i odpowiadają na pytania klientów, telefony i komputery rozpoznają polecenia wydawane głosem i potrafią z nami rozmawiać, programy (np. Picasa z Google) wyszukują na zdjęciach twarze naszych znajomych. Przykłady takie można mnożyć.

Jak wspomniano na początku tej książki - działania komputera wynikają nie tylko z wykonywania poleceń zawartych w programach, ale mogą zależeć również od wprowadzanych danych, a te z kolei mogą reprezentować stany otoczenia i różnorodne zdarzenia, ale także parametry sterujące programem. W tym sensie komputer może działać w sposób adaptacyjny, modyfikując te parametry na podstawie analizy danych i oceny własnych działań – może dostosowywać swe działania do zadań i sytuacji.

Programy komputerowe nie zawsze więc muszą być niezmienne. Istnieją języki programowania umożliwiające działającemu programowi wczytywanie tekstów, które następnie zinterpretuje jako rozkazy do wykonania. Program może też generować teksty komend innego programu, modyfikować lub kopiować inne programy lub samego siebie, inaczej mówiąc działanie programu może ewoluować. Czyż to nie przypomina zachowań istot żywych?

Istnieje wiele definicji inteligencji, między innymi :

- według Nowej Powszechnej Encyklopedii PWN, inteligencja jest to:
  „*cecha umysłu odpowiadająca za sprawność w zakresie myślenia, rozwiązywania problemów i in. czynności poznawczych, od której zależy poprawność rozumienia złożonych problemów i skuteczność poszukiwania trafnych rozwiązań a także sprawność działania w sytuacjach nowych i trudnych*";

- Słownik Języka Polskiego podaje natomiast, że:
  „inteligencja to zdolność rozumienia otaczających sytuacji i znajdowania na nie właściwych, celowych reakcji”.

Tak więc istota lub maszyna inteligentna powinna umieć:

- pobierać, selekcjonować i rozumieć informacje, szczególnie dotyczące istotnych dla niej celów, oraz zapamiętywać je w postaci użytecznej wiedzy;
- kojarzyć bieżące informacje istotnymi celami i z zapamiętaną wiedzą oraz wyciągać wnioski;

- opracowywać możliwe plany działania dla osiągania celów i dokonywać ich oceny w konfrontacji z bieżącą oceną sytuacji;
- podejmować decyzje wyboru optymalnych działań oraz realizować i kontrolować przebieg tych działań;
- porozumiewać się z innymi istotami inteligentnymi i koordynować działania wspólne lub podejmować działania przeciwstawne.

Czy więc komputer może działać inteligentnie? Na ten temat wiele osób - szczególnie o wykształceniu humanistycznym – od lat lubi się spierać, zazwyczaj wzdragając się przed odpowiedzią pozytywną. Dla nas jednak takie filozoficzne i terminologiczne spory nie są istotne. Po prostu pewną grupę metod komputerowych, które realizują przynajmniej niektóre z wymienionych działań, przyjęło się zaliczać do metod sztucznej inteligencji.

Metodami sztucznej inteligencji nazywa się ogólnie komputerowe metody rozwiązywania zagadnień – często trudnych, dla których nie istnieją skuteczne algorytmy – sposobami podobnymi do inteligentnych działań człowieka oraz procesów spotykanych w przyrodzie i organizmach żywych.

Przykładami takich zagadnień są:

- podejmowanie decyzji w warunkach niekompletnych danych;
- analiza i synteza mowy (w językach naturalnych);
- wnioskowanie i automatyczne dowodzenie twierdzeń;
- klasyfikowanie i rozpoznawanie obrazów i sygnałów;
- zagadnienia optymalizacji w przestrzeniach wielu zmiennych;
- sterowanie obiektami w czasie rzeczywistym;
- zarządzanie wiedzą, diagnozowanie i zastępowanie ekspertów.

Do metod i systemów sztucznej inteligencji zalicza się między innymi:

- **systemy ekspertowe** (bazy wiedzy i systemy wnioskowania logicznego),
- **logikę rozmytą**,
- **sztuczne sieci neuronowe**,
- **algorytmy genetyczne**,
- automaty komórkowe,
- hurtownie danych i systemy **eksploracji danych**,
- **systemy agentowe** i „boty” (internetowe „roboty”).

W kolejnych podrozdziałach scharakteryzowano pokrótce tylko dwie z tych dziedzin: systemy ekspertowe oraz sztuczne sieci neuronowe, zwane krócej sieciami neuronowymi.

13.1. SYSTEMY EKSPERTOWE

Najprościej mówiąc - **system ekspertowy to program komputerowy zastępujący człowieka będącego ekspertem z określonej dziedziny**.

Nieco dokładniej - jest to program wyposażony w bazę wiedzy i system wnioskujący, który przyjmując pytania użytkownika lub zadając mu pytania, potrafi na przykład zidentyfikować obiekt, postawić diagnozę lub sporządzić prognozę albo doradzić co czynić w danej sytuacji.

System ekspertowy pozwala **gromadzić wiedzę** ludzi **ekspertów** w formie **faktów i reguł** wnioskowania oraz **udostępniać** do dialogowego wykorzystania przez użytkowników nie będących ekspertami. Powinien także umieć rozwijać tę wiedzę, **wywodząc** w drodze wnioskowania **nowe reguły i fakty**.

Typowy system ekspertowy składa się z:

- banku wiedzy oraz
- dialogowego interfejsu użytkownika.

Bank wiedzy zawiera:

- bazę danych (faktów) – ang.: *data base*
- bazę reguł (jeżeli .. to ..) – ang.: *rule base*
- system (maszynę) wnioskowania – ang.: *inference engine*

Jedną z różnic systemów ekspertowych w stosunku do klasycznych programów jest więc to, że gromadzi reguły w osobnej bazie reguł i przy wprowadzaniu nowych reguł nie trzeba modyfikować samego programu a także to, że system wnioskowania pozwala wywodzić nowe fakty.

Oto przykład dialogu z systemem SCHOLAR przytaczanego w [60].

Pytanie użytkownika:

Ilu stroicieli fortepianów jest w Nowym Yorku?

Odpowiedź systemu:

Ponieważ w New Haven liczącym około 300 000 mieszkańców jest 3 lub 4 stroicieli fortepianów, co w stosunku do liczby mieszkańców wynosi 1 na 100 000 a Nowy York liczy 7 milionów mieszkańców, stąd szacunkowo można określić liczbę stroicieli fortepianów na około 70, no powiedzmy 50 lub 60.

Pytanie użytkownika:

Czy w Rio Grande jest uprawiana kukurydza?

Odpowiedź systemu:

Nie wiem czy kukurydza jest uprawiana w Rio Grande, chociaż wiem że roślinę tę uprawia się w pampasach. Ponieważ Rio Grande ma klimat oraz warunki glebowe podobne do pampasów, więc można się spodziewać, że na obszarze Rio Grande kukurydza jest jedną z rośli uprawnych.

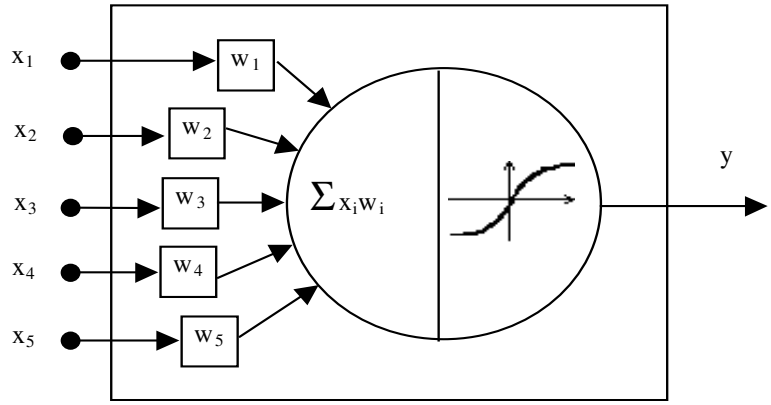
Kolejny przykład to dialog użytkownika z prostym edukacyjnym systemem ekspertowym napisanym w języku CLIPS. Użytkownik chce wiedzieć dlaczego silnik nie działa prawidłowo. System zadaje pytania a użytkownik tylko wpisuje „t" czyli TAK lub „n" czyli NIE. Oto co będzie ekranie:

```
SYSTEM DIAGNOSTYKI SILNIKA SPALINOWEGO
Implementacja w języku Clips
Proszę odpowiadać na pytanie tylko t lub n.
Zaczynamy!

Czy jest paliwo w zbiorniku? t
Czy kranik dopływu paliwa jest zamknięty ? n
Odczekaj 5 minut. Czy następna próba się powiodła? n
Czy paliwo dopływa do gaźnika? n
Czy paliwo wypływa z przewodu doprowadzającego je do pompy? t
MOIM ZDANIEM: Uszkodzenie pompy paliwa
EKSPERTYZA ZAKONCZONA
Czy chcesz następnej diagnozy ? t

Czy jest paliwo w zbiorniku? t
Czy kranik dopływu paliwa jest zamknięty? n
Odczekaj 5 minut. Czy następna próba się powiodła? n
Czy paliwo dopływa do gaźnika? t
Czy ustawienie przesłony rozruchowej jest prawidłowe? t
Czy główna dysza paliwa jest drożna ? t
MOIM ZDANIEM: Zanieczyszczona dysza biegu jałowego lub awaria
w układzie zapłonowym
EKSPERTYZA ZAKONCZONA
Czy chcesz następnej diagnozy? n
```

13.2. SZTUCZNE SIECI NEURONOWE

Sztuczne sieci neuronowe (SN) [61]-[64] naśladują w wielkim uproszczeniu działanie biologicznych systemów nerwowych a więc w jakimś, niewielkim stopniu, także niedoścignionego wzorca - ludzkiego mózgu, zawierającego około 100 miliardów komórek nerwowych - neuronów. SN tworzone są z wielu połączonych wzajemnie **sztucznych neuronów**, stanowiących uproszczone modele neuronów biologicznych. SN mogą być wytwarzane jako samodzielne układy elektroniczne o bardzo dużej szybkości działania, dzięki **równoległemu** przetwarzaniu sygnałów przez każdy z wielu sztucznych neuronów. Najczęściej jednak funkcjonowanie modelu sieci neuronowej symulowane jest na komputerze przy pomocy odpowiedniego programu.

Każdy ze sztucznych neuronów posiada wiele wejść i jedno wyjście co pokazano na Rys. 13.1. Sygnały wejściowe tworzą **wektor wejść x** = $[x_1, x_2, ..., x_n]$. Z każdym wejściem neuronu skojarzony jest współczynnik zwany **wagą** tego wejścia. Wartości wag wyznaczane są najczęściej automatycznie w procesie **uczenia sieci**. Wagi wejść neuronu można zapisać jako **wektor wag w** = $[w_1, w_2, ..., w_n]$.



Rys. 13.1. Sztuczny neuron

Przetwarzanie sygnałów w neuronie składa się z dwu faz.

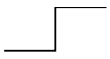
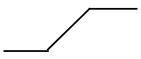
W pierwszej fazie – na podstawie wartości wejściowych **x** oraz wartości wag **w -** wyznaczana jest **zagregowana wartość wejściowa** *s*. Sposób obliczania wartości *s* określony jest przez funkcję *g(***x,w***)* zwaną **funkcją agregującą**, wyrażoną najczęściej jako suma iloczynów wejść przez odpowiadające im wagi:

$$s = \sum_{i=1}^{n} w_i x_i \tag{1}$$

lub jako suma kwadratów różnic wag i wejść czyli odległość euklidesowa pomiędzy wektorem **w** oraz **x** :

$$s = \sum_{i=1}^{n} (w_i - x_i)^2 \tag{2}$$

Tabela 13.1. Funkcje aktywacji sztucznych neuronów

Funkcja aktywacji:	liniowa	skok	liniowa z ograniczeniem	sigmoidalna
Kształt wykresu:				

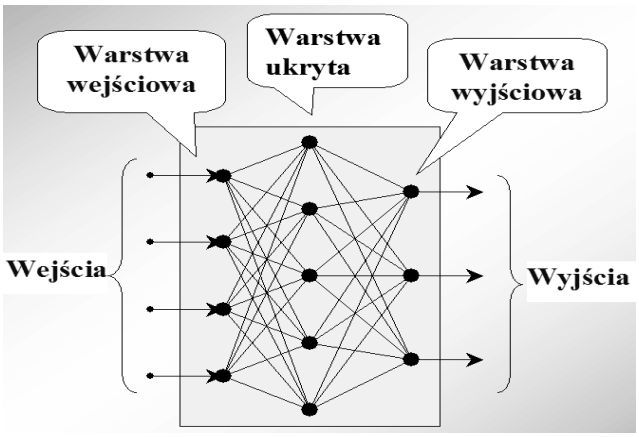
W fazie drugiej wartość *s* zostaje przetworzona na **wartość wyjściową** neuronu *y* przy pomocy funkcji *f(s)* nazywanej **funkcją aktywacji** lub **funkcją przejścia**.

Niektóre typy funkcji aktywacji pokazuje w formie graficznej Tabela 13.1.

Najczęściej stosowane są funkcje ciągłe: liniowa oraz sigmoidalna.

Każda sieć neuronowa potrafi na wiele sposobów przetwarzać sygnały, przy czym - co najważniejsze - sposoby przetwarzania **nie** są a priori ustalane przez człowieka, jak w klasycznych programach komputerowych, lecz zostają ustalone w procesie zwanym **uczeniem** lub **trenowaniem** sieci.

Istnieje wiele rodzajów SN, różniących się m.in. charakterystykami neuronów, ich liczbą, topologią połączeń oraz sposobami uczenia, jednak najogólniej każdą sieć neuronową można potraktować jako czarną skrzynkę o "n" wejściach i "m" wyjściach (Rys. 13.2).



Rys. 13.2. Przykład sieci neuronowej typu perceptron wielowarstwowy.

Liczba wejść i wyjść jest określana przez projektującego sieć ale wynika na ogół z istoty problemu. Na przykład przy filtrowaniu sygnału wystarczy jedno wejście i jedno wyjście a przy klasyfikowaniu obiektów potrzeba tyle wejść ile cech obiektu będzie analizowanych oraz tyle wyjść ile jest klas do których zaliczane będą badane obiekty.

Liczba warstw ukrytych jest dobierana zazwyczaj eksperymentalnie metodą prób i błędów. Najczęściej wystarcza **jedna lub dwie** warstwy ukryte. Zbyt mała liczba warstw nie pozwala osiągnąć dostatecznie dokładnego działania sieci natomiast zbyt duża ich liczba pogarsza zdolność do generalizacji (uogólniania) to znaczy sieć bardzo dokładnie działa dla zbioru danych uczących (uczy się „na pamięć") ale znacznie gorzej dla innych danych.

Uczenie sieci to proces kształtowania jej własności **na podstawie wzorcowych danych**. Zachodzi więc pewne podobieństwo do metod statystycznych, które wymagają danych statystycznych do określenia tendencji oraz wypracowania prognoz.

Zbiór danych eksperymentalnych – niezbędny do procesu uczenia SN - najczęściej dzieli się na trzy zbiory:

1) **zbiór uczący** - dla wypracowania wag neuronów,
2) **zbiór walidacyjny** - dla sprawdzania zdolności do uogólniania i podjęciu decyzji o zakończeniu uczenia lub korekcie topologii sieci,
3) **zbiór testowy** - do oceny sieci po procesie uczenia.

Zbiór uczący powinien być reprezentatywny dla wszelkich danych jakie mogą wystąpić w trakcie przyszłej pracy sieci wystarczająco obszerny - zależnie od liczby węzłów i złożoności problemu.

Istnieją **dwie główne metody uczenia** SN:

- uczenie **nadzorowane** albo uczenie "**z nauczycielem**" wymaga dostarczenia sieci zarówno serii przykładowych wartości wejściowych jak i prawidłowych dla nich wartości wyjściowych, a korygowanie wag neuronów odbywa się w taki sposób aby zminimalizować odchyłki sygnałów wyjściowych od zadanych wzorców wyjść;
- w uczeniu **nienadzorowanym** czyli "**bez nauczyciela**" sieć dostaje tylko serię wartości wejściowych, grupuje je według podobieństw, przypisując grupom skupiska neuronów i odpowiednie wyjścia.

Po nauczeniu sieci można ją używać do klasyfikowania lub przetwarzania cech lub sygnałów nieznanych, odmiennych niż podawane przy uczeniu.

Sieci neuronowe mogą być wykorzystywane do takich operacji jak:

- **generowanie** różnorodnych **sygnałów**,
- **przekształcanie sygnałów** a wśród nich także filtrowanie, wygładzanie, regeneracja, aproksymacja,
- **rozpoznawanie i klasyfikowanie** otrzymywanych na wejściach n-tek sygnałów odpowiadającym cechom analizowanych obiektów lub stanów – na przykład rozpoznawanie obrazów rakiet czy samolotów nieprzyjacielskich, podpisów lub linii papilarnych, podejrzanych przedmiotów w prześwietlanych walizkach, ocena wiarygodności dłużników, sygnalizacja konieczności wymiany stępionego narzędzia i t.p.
- **przewidywanie dalszego przebiegu zjawiska** na podstawie dotychczasowych przebiegów (predykcja, ekstrapolacja)

Prosty przykład zastosowania SN w tribologii – wchodzącej w skład zagadnień inżynierii mechanicznej – stanowi opisane w [66] monitorowanie i predykcja zużycia wiertła na podstawie danych: siły nacisku, momentu obrotowego, średnicy wiertła, szybkości obrotowej oraz posuwu.

Eksperymenty z sieciami neuronowymi można realizować w pakiecie **MATLAB** zawierającym **Neural Network Toolbox**. Innym profesjonalnym narzędziem do SN jest STATISTICA Neural Networks.

Można też eksperymentować z sieciami neuronowymi bez użycia tych kosztownych narzędzi, korzystając programów udostępnianych bezpłatnie w Internecie. Bogatym źródłem takich programów oraz wykładów, prezentacji i publikacji – dotyczących sieci neuronowych - są strony internetowe [62] prof. **Ryszarda Tadeusiewicza**. W szczególności są tam darmowo dostępne do pobrania programy, opisane szczegółowo w książkach tego autora [63] i [64], pozwalające krok po kroku poznawać właściwości różnego typu sztucznych sieci neuronowych. Zachęcam do czytania tych książek oraz eksperymentowania z opisywanymi w nich programami. Programy te wraz z książkami, stanowią spójną całość, pozwalającą nie tylko zdobyć wiedzę z zakresu sztucznej inteligencji, ale także rozwinąć własną dociekliwość i pasję badawczą.

LITERATURA

[1] Zbigniew Rudnicki: Techniki Informatyczne. Tom 1: Podstawy i wprowadzenie do CAD. AGH Uczelniane Wydawnictwa Naukowo-Dydaktyczne, Kraków 2012.

[2] Zbigniew Rudnicki: Techniki Informatyczne. Tom 2: Obliczenia bez programowania – arkusze kalkulacyjne, Mathcad ; Cyfrowe sieci komputerowe, telekomunikacyjne i nawigacyjne. AGH Uczelniane Wydawnictwa Naukowo-Dydaktyczne, Kraków 2012.

[3] Materiały dydaktyczne na stronie internetowej Katedry Konstrukcji i Eksploatacji Maszyn AGH [http://www.kkiem.agh.edu.pl/dydakt/index.html] (dostęp 2013.I.21)

[4] Polska Wikipedia [http://pl.wikipedia.org/wiki/] (dostęp 2012.I.21)

[5] Wikipedia [http://en.wikipedia.org/wiki] (dostęp 2012.I.21)

[6] Software for Multibody System Simulation (dostęp 2013.V.21) [http://real.uwaterloo.ca/~mbody/#Software]

[7] Zbigniew Rudnicki: Ważniejsze programy komputerowe ... http://www.kkiem.agh.edu.pl/staff/Rudnicki/progr/index.html (dostęp 2012.I.21)

[8] Komputerowe wspomaganie proj. zespołów i elementów maszyn w przykładach. Praca zbiorowa. Cz. 1, Połączenia / pod red. Marii Porębskiej. Wydaw. AGH, 1991.

[9] Komputerowe wspomaganie proj. zespołów i elementów maszyn w przykładach. Praca zbiorowa. Cz. 2, Przekładnie / pod red. Marii Porębskiej. Wydaw. AGH, 1991.

[10] Komputerowe wspomaganie proj. zespołów i elementów maszyn w przykładach. Praca zbiorowa. Cz. 3 / pod red. Marii Porębskiej. Wydaw. AGH, 1992.

[11] Stare komputery: [http://starekomputery.uibs.com.pl] (dost.2012.I.21)

[12] Kuliński S., Maziarz M., Rudnicki Z., Warszyński M.: "Metodyka projektowania łączników przegubowych walcarek przy wspomaganiu komputerowym." - Materiały XIII Sympozjonu PKM, Świnoujście 1987.

[13] Kuliński S., Rudnicki Z.: "Dialogowe wyznaczanie momentu skręcającego łącznik napędu walcarki." VI Konf.Metody i Środki Projektowania Wspomaganego Komputerowo. Zbiór referatów, W-wa 1987.

[14] Porady – programowanie smartfonów. CHIP 2009. (dost.2013.I.21) [http://www.chip.pl/artykuly/porady/2009/09/programowanie-smartfonow]

[15] Narzędzia dla systemu Android [http://developer.android.com/sdk/index.html]

[16] Narzędzia dla systemu iOS [https://developer.apple.com/](dost.2013.I.21)

[17] App Inventor – projektowanie aplikacji. [http://mobility.com.pl/app-inventor-projektowanie-aplikacji-stworz-wlasny-soundboard.html] (dost.2013.I.21)

[18] Strona języka “RFO BASIC!” [http://laughton.com/basic/] (dost.2013.I.21)

[19] Sławomir Kokłowski: HTML dla Zielonych [http://www.kurshtml.boo.pl/html/zielony.html] (dost.2013.I.21)

[20] Sławomir Kokłowski: Kurs HTML [http://www.kurshtml.edu.pl/] (dost.2013.I.21)

[21] Paweł Wimmer: HTML dla początkujących [http://webmaster.helion.pl/index.php/html-dla-poczatkujacych] (dost.2013.I.21)

[22] Włodzimierz Gajda: Kontrolki formularzy
[http://www.gajdaw.pl/html/kontrolki-formularzy/p1.html] (dost.2013.I.21)

[23] Poradnik webmastera.
[http://www.poradnik-webmastera.com/kursy/html/formularze.php] (dost.2013.I.21)

[24] Adam Golański:
[http://webhosting.pl/Badania.home.pl.pokazuja.ze.Joomla.kroluje.wsrod.polskich.webmasterow] (dost.2013.I.21)

[25] Karol Wierzchołowski: Programowanie w PHP. Poradnik dla webmasterów. Wyd. Axel Springer. Biblioteczka "Komputer Świat" nr 2/2006 (37) czerwiec.

[26] Dorota Wdzięczna: Bazy Danych. Programowanie w SQL i PHP. Wyd. Ringier Axel Springer. Biblioteczka „Komputer Ekspert". 06/2011.

[27] Leszek Krupiński: *Kurs PHP. [http://phpkurs.pl]* (dostęp 2012-07-08)

[28] Marcin Lis, Rafał Wileczek: Kurs PHP [http://webmaster.helion.pl/index.php/kurs-php] (dostęp 2012-07-12)

[29] Zbigniew Rudnicki: Opis systemu „ZR-MBIB - Mała Biblioteka". Kraków 1997.
[http://www.kkiem.agh.edu.pl/staff/Rudnicki/progr/mbi-opis.pdf]

[30] Tomasz Kubik: Relacyjne bazy danych. Politechnika Wrocławska.
[http://tomasz.kubik.staff.iiar.pwr.wroc.pl/dydaktyka/RelacyjneBazyDanych]

[31] Robert Wrembel :Wprowadzenie do problematyki baz danych
[http://wazniak.mimuw.edu.pl/index.php?title=BD-2st-1.2-w01.tresc-1.1-toc]

[32] Mirosława Kopertowska: ECDL Moduł 5. Bazy Danych. Wyd. MIKOM. 2007

[33] Internetowe materiały dydaktyczne na serwerze *wazniak.mimuw.edu.pl* opracowane przez 4 polskie uczelnie w ramach projektu finansowanego przez Unię Europejską: a) Wstęp do programowania; b) Algorytmy i struktury danych; c) Metody numeryczne; d) Matematyka dyskretna; e) Paradygmaty programowania

[34] Open AGH – Algorytmy i struktury danych (dostęp 2012.II.12)
[http://zasoby1.open.agh.edu.pl/dydaktyka/informatyka/c_algorytmy_i_str_danych]

[35] Kurs Algorytmiki - dostęp 2012.II.12
[http://www.algorytm.org/kurs-algorytmiki/rzedy-wielkosci-funkcji.html]

[36] Daniel Gontarek: Porównanie algorytmów sortowania (program)
[http://www.home.umk.pl/~abak/wdimat/s/Info.html] – dostęp 2010.I.25

[37] Jerzy Wałaszek: Serwis edukacyjny I LO Tarnów. Algorytmika.
[http://edu.i-lo.tarnow.pl/inf/index.php] – dostęp 2010.I.25

[38] Encyclopedia Britannica [http://www.britannica.com/] (dost.2012.I.21)

[39] Wydawnictwa Szkolne i Pedagogiczne – Klub Informatyka
[http://www.wsipnet.pl/kluby/informatyka_ekstra.php?k=69] (dost.2012.I.21)

[40] [http://encyclopedia2.thefreedictionary.com/Numerical+methods] (dost.2012.I.21)

[41] Numerical Analysis [http://math.fullerton.edu/mathews/numerical.html] (dost.2012.I.21)

[42] Andrzej Brozi: Materiały dydaktyczne: Metody numeryczne w fizyce
(dostęp 2012.II.21) [http://phys.p.lodz.pl/materialy/abrozi/mnf2%2707.pdf]

[43] Piotr A. Dybczyński: Metoda eliminacji Gaussa (dostęp 2012.I.25)
[http://apollo.astro.amu.edu.pl/PAD/index.php?n=Dybol.DydaktykaEliminacjaGaussa]

[44] Wstęp do metod numerycznych (dostęp 2012.I.21)
[http://www.l5.pk.edu.pl/~slawek/WWW/SMilewski_Num_Met1.pdf]

[45] Tadeusz Burczyński: Metody numeryczne w mechanice oraz ich wpływ na rozwój mechaniki w Polsce
[http://fundacjarozwojunauki.pl/res/Tom2/Burczy%C5%84ski.pdf] (dostęp 2012.I.21)

[46] Stanisław Białas, Andrzej Olajossy: Różnicowe metody rozwiązywania równań różniczkowych. Część I Równania różniczkowe Zwyczajne. Wyd. AGH. Kraków 1987.

[47] Materiały opracowywane przez absolwentów Informatyki Politechniki Poznańskiej
[http://www.algorytm.org] (dostęp 2012.II.20)

[48] Zbigniew Osiński, Jerzy Wróbel: Teoria konstrukcji. PWN, Warszawa 1995

[49] M. Komosiński: Optymalizacja. Programowanie Matematyczne (dostęp 2012.IV.21)
[http://www.cs.put.poznan.pl/mkomosinski/materialy/optymalizacja/MP.pdf]

[50] Materiały dydaktyczne KKiEM AGH – Pro/Desktop
[http://www.kkiem.agh.edu.pl/dydakt/prodesk/cysterna/index.html]

[51] George W. Collins: Fundamental Numerical Methods and Data Analysis
[http://ads.harvard.edu/books/1990fnmd.book], (dostęp 2012.III.25)

[52] Historia komputerów [http://edu.i-lo.tarnow.pl/inf/hist.php], dostęp 2013.10.22

[53] Anna Kamińska, Beata Pańczyk: „MATLAB - przykłady i zadania" - Wyd. Mikom 2002, z serii „ćwiczenia z..." (150 stron)

[54] J.Brzózka, L.Dorobczyński: „Programowane w MATLAB". Wyd. Mikom 1998.

[55] Bogumiła Mrozek, Zbigniew Mrozek: MATLAB i Simulink. Wyd. Helion 2012

[56] M. Stachurski: Metody numeryczne w programie MATLAB. Wyd. MIKOM 2003

[57] Wiesława Regel: Wykresy i obiekty graficzne w MATLAB. Wyd.MIKOM 2003

[58] B. Mrozek, Zb.Mrozek: „MATLAB 5.x, Simulink 2.x"., Wyd. PLJ 1998

[59] MATLAB R2012a Documentation. Programming Fundamentals
http://www.mathworks.com/help/techdoc/matlab_prog/bqjgwp9.html

[60] James Martin: Dialog człowieka z maszyną cyfrową. Wyd. WNT, 1976

[61] P. Lula, G. Paliwoda-Pękosz, R. Tadeusiewicz: Metody sztucznej inteligencji i ich zastosowania w ekonomii i zarządzaniu. Wyd. Akad. Ekonomicznej. Kraków 2007

[62] Strony WWW prof. Ryszarda Tadeusiewicza: http://www.uci.agh.edu.pl/uczelnia/tad/, oraz http://home.agh.edu.pl/~tad

[63] Ryszard Tadeusiewicz: Elementarne wprowadzenie do techniki sieci neuronowych z przykładowymi programami. Akad. Oficyna Wyd. PLJ. Warszawa 1998

[64] Ryszard Tadeusiewcz i in. : Odkrywanie właściwości sieci neuronowych przy użyciu programów w języku C#. Wyd. Polskiej Akademii Umiejętności. Kraków 2007

[65] Systemy agentowe: http://www.agentlink.org/index.php, http://www.robocup.org/, http://www.agentland.com/**,** (dostęp 2012.XI.25)

[66] Rudnicki Z., Figiel W. - Neural nets application in tribology research. Zagadnienia Eksploatacji Maszyn. Zeszyt 3 (131) 2002, str 97-110