Podstawy programowania.

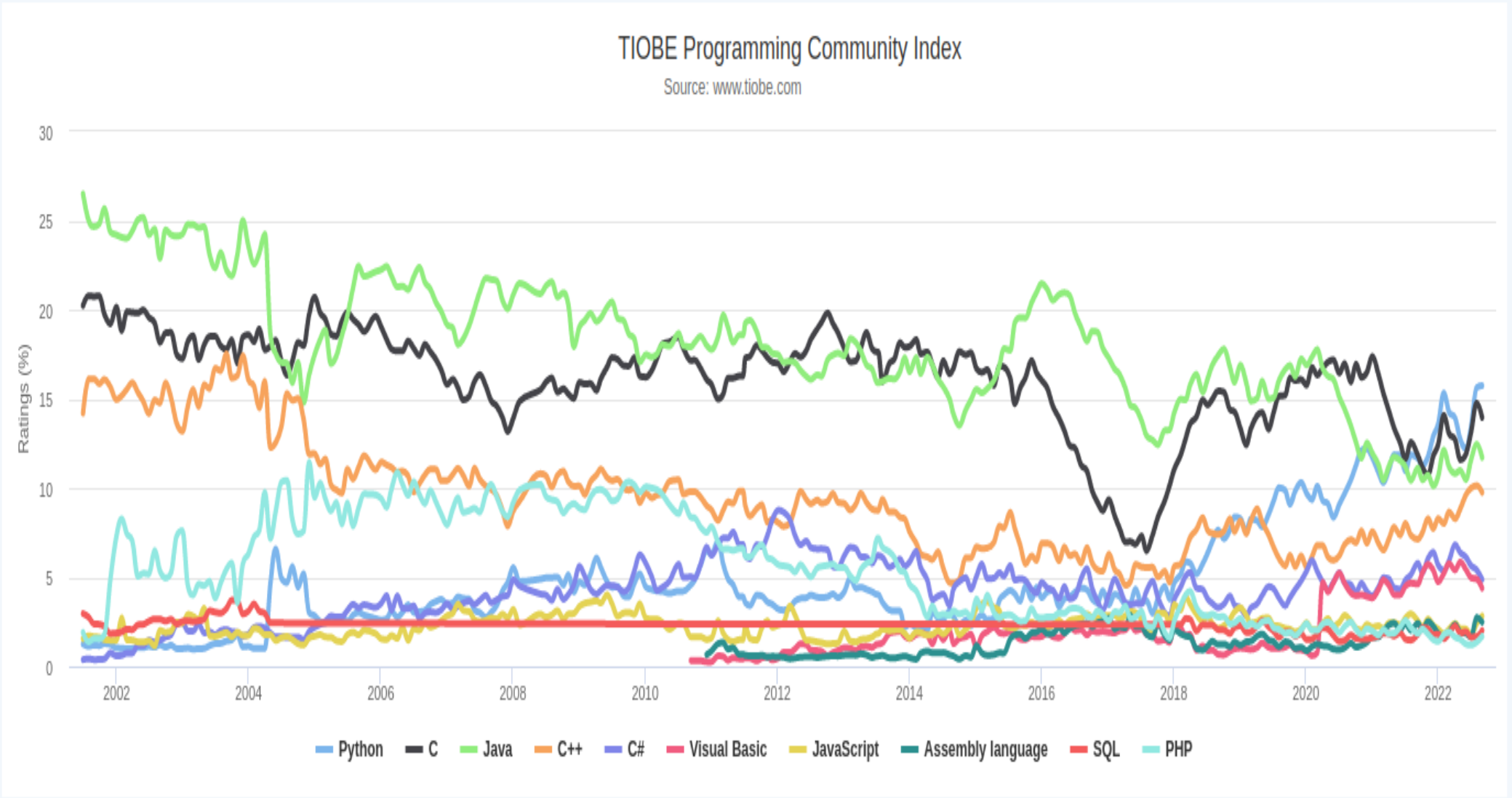
Wykład 1

Wstęp

Elementy historii informatyki

- “I think there is a world market for maybe five computers.”
  - Thomas Watson, chairman of IBM, 1943.
- “There is no reason for any individual to have a computer in their home”
  - Ken Olson, president and founder of Digital Equipment Corporation, 1977.
- “640K [of memory] ought to be enough for anybody.”
  - Bill Gates, chairman of Microsoft,1981.

Wykres popularności języków



Język C

- Język C
  - jest jednym z najpowszechniej używanych języków programowania
    - specyfikacja języka zawarta jest w kolejnych międzynarodowych standardach (1989, 1999, 2011, 2018)
  - jest językiem relatywnie niskiego poziomu (blisko sprzętu)
    - pozwala na pisanie bardzo wydajnych programów
  - jest prosty i jednocześnie elastyczny
    - pozwala łatwo implementować rozmaite algorytmy
  - posiada konstrukcje występujące w wielu innych językach
    - nauka języka C jest dobrym wstępem do nauki innych języków
  - jest językiem programowania strukturalnego (proceduralnego)
    - podstawowym elementem programów w C są funkcje
    - C dostarcza własną bibliotekę podstawowych funkcji, której zawartość jest określona przez specyfikację języka C (standard)
      - praktycznie nie jest możliwe pisanie użytecznych programów bez wykorzystania funkcji z biblioteki standardowej

Języki i paradygmaty programowania

- C i programowanie strukturalne
  - zazwyczaj
    - małe programy (choć są też wielkie systemy)
    - pisane dla ściśle określonego celu
- C++ i inne języki obiektowe
  - zazwyczaj
    - nacisk na wielokrotne wykorzystanie fragmentów kodu
    - starannie zaprojektowana, rozbudowana, wielopoziomowa architektura
  - C++ zawiera w sobie C, ale pisanie programów strukturalnych w C++ przyczynia się do utrwalania błędnych nawyków przy programowaniu obiektowym
- Inne języki i paradygmaty (ponad 1000 języków)

Algorytmika – istota informatyki

- Algorytm - dokładny przepis podający sposób rozwiązania określonego problemu w postaci skończonej liczby uporządkowanych operacji.
- Algorytm – (inf.) ściśle określony ciąg kroków obliczeniowych, prowadzący do przekształcenia danych wejściowych w wyjściowe.
- Algorytm – (Wikipedia!) skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego rodzaju zadań.

- Przykłady algorytmów: codziennych i obliczeniowych – przepisy kulinarne, sortowanie, rozwiązywanie równań

Analiza algorytmów

- Dowodzenie poprawności algorytmów
  - Algorytm jest poprawny syntaktycznie, jeżeli jest skonstruowany zgodnie z zasadami wybranego języka (pseudo-języka) programowania.
  - Algorytm jest poprawny semantycznie, jeżeli dla każdego egzemplarza danych (tj. zbioru danych wejściowych, spełniającego warunki początkowe) daje prawidłowy wynik (spełniający warunki końcowe) i zatrzymuje się.
- Określanie złożoności obliczeniowej
  - Złożoność obliczeniowa - ilość zasobów komputerowych koniecznych do wykonania programu realizującego algorytm jako funkcja pewnego parametru, określającego rozmiar rozwiązywanego zadania.

Zapis algorytmów

- Zapis algorytmu w informatyce musi być:
  - jednoznaczny
  - zrozumiały dla wykonawcy:
    - sprzętu
    - narzędzia tłumaczącego na rozkazy sprzętowe
    - osoby piszącej program
- Zapis na etapie analizy i projektowania kodu
  - zrozumiały dla projektanta i programisty
    - zapis słowny
    - pseudokod – zapis zbliżony do zapisu w językach programowania, bez szczegółów technicznych, zrozumiały dla założonego odbiorcy
    - schemat blokowy
    - umowne specyfikacje (UML, itp.)

Zapis algorytmów

- Zapis w języku programowania
  - zgodny ze składnią języka (standardy)
    - sprawdzenie przez kompilator/interpreter
  - poprawny semantycznie
    - analiza kodu
    - testowanie
  - z założenia przenośny między systemami
- Ostateczny zapis programu
  - ma postać pliku wykonywalnego
  - zawierającego rozkazy procesora
  - oraz szereg informacji określających sposób wykonania programu
  - specyficzny dla konkretnego systemu i sprzętu

Proces programowania

- Proces wytwarzania oprogramowania:
  - specyfikacja i projektowanie programu
    - warunki początkowe i końcowe
    - schemat przetwarzania
  - tworzenie kodu źródłowego
    - implementacja algorytmu
  - uruchomienie
    - kompilacja (interpretacja)
      - poprawianie błędów składni
    - wykonanie
      - interakcja ze środowiskiem wykonania (uruchomieniowym)
  - dalsze etapy
    - testowanie – rozmaite egzemplarze danych
    - optymalizacja – oszczędność zasobów
    - modyfikacje, ewolucja

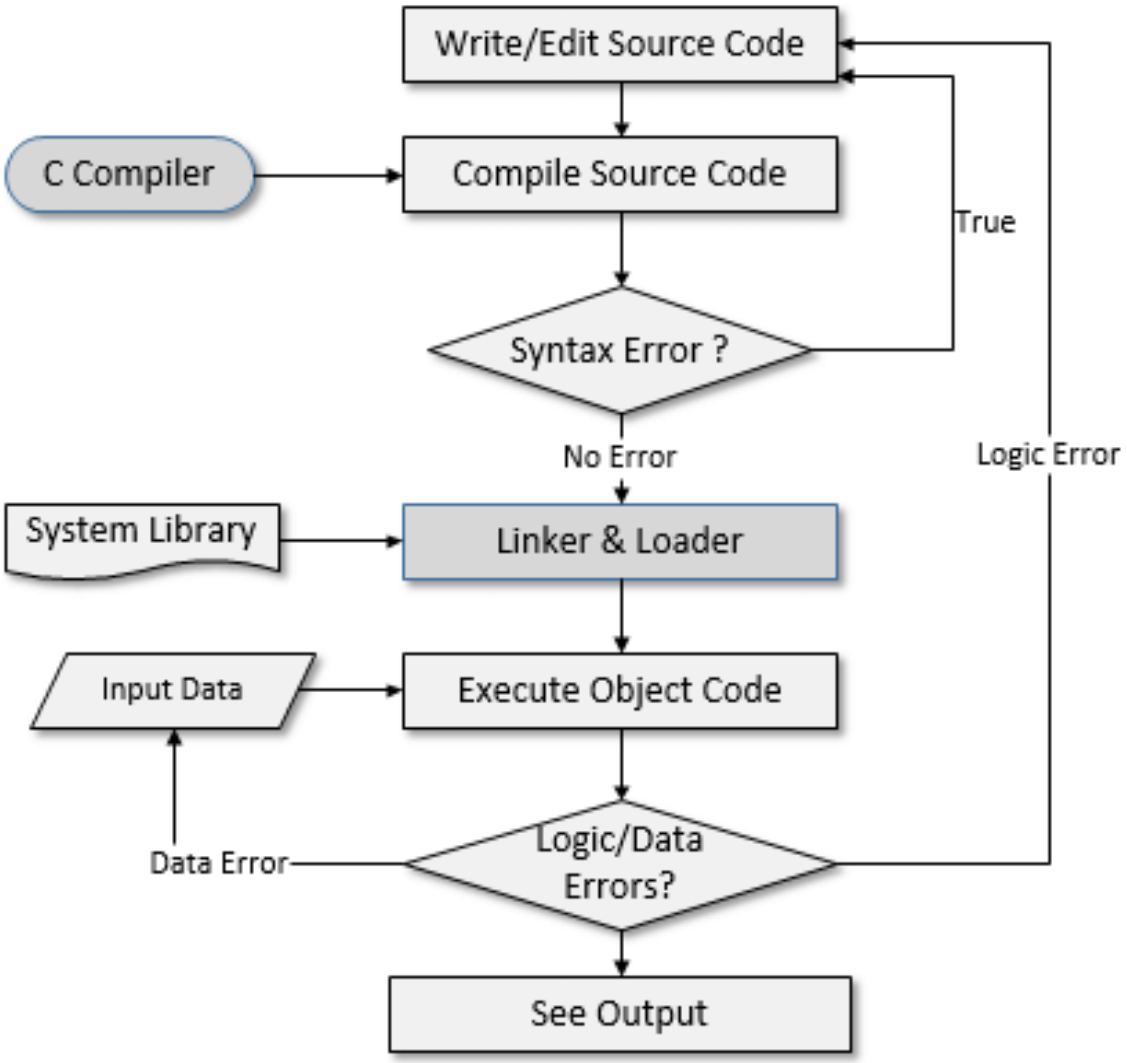
Proces programowania - kompilacja

- kod źródłowy -> [kod pośredni ->] kod binarny
- kompilacja – przeprowadzana etapami zamiana kodu źródłowego na kod wykonywalny
- kompilacja obejmuje m.in.:
  - możliwe wstępne przetworzenie kodu źródłowego
  - zamianę instrukcji języka na zestawy rozkazów procesora
  - zamianę zmiennych na adresy w pamięci
  - wprowadzenie rejestrów
  - wprowadzenie dodatkowych informacji, m.in. o sposobie wykonania
- kompilacja może zatrzymać się na etapie kodu pośredniego
- różne typy kodów pośrednich zależą od języka programowania, systemu operacyjnego
- kompilacja może obejmować etap konsolidacji

Proces programowania - konsolidacja

- kod źródłowy -> [kod pośredni ->] kod binarny
- konsolidacja (linkowanie)
  - łączenie różnych plików zawierających wstępnie skompilowany kod
    - m.in. biblioteki, w tym systemowe
  - możliwe rodzaje konsolidacji
    - statyczna – tworzony plik wykonywalny zawiera wszystkie potrzebne funkcje
      » konsolidacja statyczna częściowo uniezależnia od detali systemu operacyjnego dając kod bardziej przenośny
    - dynamiczna – plik wykonywalny zawiera informację o wymaganych funkcjach, które są dołączane w trakcie działania programu
      » konsolidacja dynamiczna produkuje kod o mniejszym rozmiarze

Kompilacja


Write/Edit Source Code
C Compiler
Compile Source Code
True
Syntax Error ?
No Error
Logic Error
System Library
Linker & Loader
Input Data
Execute Object Code
Data Error
Logic/Data Errors?
See Output

Proces programowania

- Istnieje szereg narzędzi wspomagających różne etapy procesu programowania
- Zintegrowane systemy tworzenia oprogramowania (*IDE – Integrated Development Systems*) dostarczają szereg funkcjonalności wymaganych w procesie tworzenia oprogramowania
  - przykłady: Visual Studio, NetBeans, KDevelop
- Alternatywnym modelem jest stosowanie niezależnych narzędzi, często związanych z systemem operacyjnym
  - środowisko Unixowe: edytory, kompilatory, debugery, profilery, programy kontroli wersji, itp.

Proces programowania - interpretacja

- kod źródłowy -> [kod pośredni ->] interpretacja
- interpretacja:
  - może dotyczyć kodu źródłowego
    - tłumaczenie dokonywane przez interpreter
  - lub kodu wstępnie przetworzonego (skompilowanego) do postaci pośredniej
    - tłumaczenie dokonywane przez maszynę wirtualną
      - przykład: bytecode dla języka Java
  - pojęcie maszyny wirtualnej jest bardzo szerokie, maszyny wirtualne mogą dokonywać interpretacji programów napisanych dla innych systemów operacyjnych, samych systemów operacyjnych itp.
  - brak znajomości kodu wykonywalnego oddala programistę od sprzętu i wiedzy o ostatecznym sposobie wykonania programu

Wykonanie programu

- Sposób wykonania programu musi prowadzić do efektów zgodnych z zapisem kodu w języku programowania
- Sposób wykonania programu jest zależny od systemu operacyjnego i sprzętu
- Interakcja z systemem operacyjnym jako środowiskiem wykonania obejmuje m.in.:
  - uruchomienie programu
  - wykorzystanie bibliotek systemowych
  - zarządzanie pamięcią
  - zarządzanie przydziałem sprzętu
    - obsługę urządzeń wejścia/wyjścia
  - mechanizmy zapewnienia bezpieczeństwa wykonania

Programy = algorytmy + struktury danych

- Algorytm najczęściej określa sposób przetwarzania dla rozmaitych egzemplarzy danych
- Zapis i sposób wykonania programu musi uwzględniać możliwość działania dla różnych egzemplarzy danych
  - zmienne jako sposób przechowywania danych
- Przebieg typowego programu:
  - wczytanie danych wejściowych
    - różne sposoby zależne od sprzętu i systemu operacyjnego
  - realizacja przetwarzania
  - zwrócenie wyniku, danych wyjściowych
    - różne sposoby zależne od sprzętu i systemu operacyjnego

Programowanie - przykład

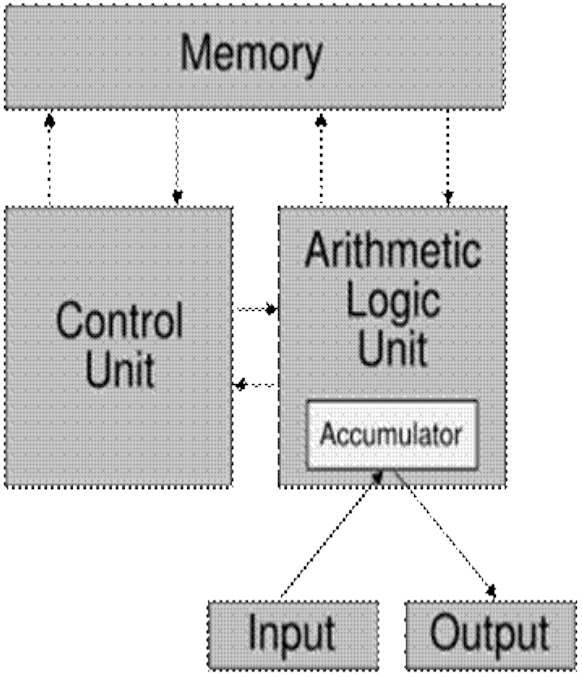
- Problem obliczeniowy:
  - dane wejściowe:
    - dwie liczby
  - dane wyjściowe:
    - liczba będąca sumą danych wejściowych
- Zapis symboliczny: **c = a+b**
- Pseudokod:
  - pobierz liczby **a** i **b**
  - oblicz **c = a+b**
  - zwróć (wypisz) **c**

Architektura von Neumana

- Wszystkie współczesne procesory można uznać za rozszerzenia i modyfikacje architektury von Neumana
- Podstawowe cechy maszyny von Neumanna
  - kod i dane zapisane w tej samej pamięci
    - zapis dwójkowy kodu i danych
  - pamięć adresowalna o dostępie swobodnym
    - dostęp do pojedynczej komórki pamięci poprzez jej adres
  - procesor – jednostka centralna, CPU
    - pobieranie, dekodowanie rozkazów
    - przetwarzanie rozkazów
    - rejestry (wewnętrzna pamięć procesora)

Maszyna von Neumanna

- Maszyna von Neumanna składa się m.in. z następujących elementów:
  - adresowalna pamięć operacyjna
  - magistrala – kanał komunikacyjny do przesyłania danych pomiędzy procesorem a pamięcią
  - procesor – jednostka centralna, CPU:
    - jednostka sterująca
    - jednostka arytmetyczno-logiczna (przetwarzająca)
    - rejestry (wewnętrzna pamięć procesora)
    - zegar
    - wewnętrzna magistrala procesora



Kod maszynowy i język asemblera

- Kod w języku asemblera
  - jest produktem w procesie kompilacji najbliższym binarnemu plikowi wykonywalnemu, a jednocześnie relatywnie łatwym w analizie przez programistę
  - może być tworzony bezpośrednio przez programistę
  - zawiera symboliczny zapis rozkazów procesora realizujących kod źródłowy
    - może istnieć wiele wariantów kodu w języku asemblera dla tego samego kodu źródłowego

Kod maszynowy i język asemblera

- Kod w języku asemblera zawiera
  - informacje specyficzne dla systemu operacyjnego
  - informacje specyficzne dla sprzętu
  - rozkazy procesora

➔ Interakcja ze sprzętem – rozkazy procesora
  - zasoby procesora bezpośrednio wykorzystywane
    - rejestry
    - pamięć operacyjna
  - typy rozkazów
    - podstawowe operacje
    - wersje rozkazów dla różnych typów danych

Wykonanie programu

- fragment kodu w języku asemblera dla procesorów rodziny x86 dla przykładowego programu:

```
movl    -12(%ebp), %eax
addl    -16(%ebp), %eax
movl    %eax, -8(%ebp)
```

Języki wysokiego poziomu

- Język asemblera jest językiem programowania najniższego poziomu
  - jest specyficzny dla konkretnej architektury sprzętu – odpowiada rozkazom maszynowym
- Języki wyższego poziomu wprowadzają konstrukcje mające ułatwić programowanie, między innymi takie jak:
  - złożone typy i struktury danych
  - konstrukcje sterujące wykonaniem programów
  - funkcje
  - obiekty
- Języki wyższego poziomu zapewniają przenośność kodu źródłowego programów
  - wymagają programów narzędziowych specyficznych dla konkretnego systemu operacyjnego i architektury sprzętu

Programowanie strukturalne

- Programowanie strukturalne (proceduralne) jest jednym z podstawowych paradygmatów programowania
- Wykorzystanie
  - procedur (funkcji)
  - podstawowych konstrukcji sterujących
- Funkcje
  - wygodny mechanizm implementacji algorytmów
    - dane wejściowe
    - dane wyjściowe
  - efekty uboczne
    - modyfikacje wewnętrznych i zewnętrznych struktur danych, operacje wejścia/wyjścia
    - w wielu programach efekty uboczne są podstawowym rezultatem realizacji funkcji
    - efekty uboczne utrudniają dowodzenie i zapewnianie poprawności programów