Programowanie imperatywne

dr inż. Piotr Szwed
Katedra Informatyki Stosowanej
C2, pok. 403

e-mail: pszwed@agh.edu.pl

http://home.agh.edu.pl/~pszwed/

Aktualizacja: 28.02.2023

Informacje o przedmiocie

Zakres

Czego powinniście Państwo nauczyć się?

- **Składnia** języka C
  - deklarowanie zmiennych
  - definiowanie funkcji
  - stosowanie instrukcji sterujących
  - zasady konstrukcji wyrażeń i ich interpretacji deklaracji typów danych
- **Semantyka** - zasady odwzorowania konstrukcji języka C w elementy wykonywalnego programu
  - lokalizacja zmiennych w pamięci
  - wykonanie instrukcji i interpretacja wyrażeń
  - przebieg wywołania funkcji,
  - sposób przekazywania parametrów

Zakres

- Podstawowe **funkcje standardowych bibliotek** języka C
- Przebieg **procesu tworzenia programu** (zastosowanie preprocesora, kompilacja i konsolidacja)
- Zasady konstruowania programów **wielomodułowych**
- Przetwarzanie **plików** oraz programy komunikujące się poprzez standardowe wejście i wyjście
- **Uruchamianie programów** (usuwanie błędów kompilacji, konsolidacji i wykonania)
- **Dynamiczna alokacja pamięci** - implementacja różnych struktur danych

Czego nie będzie?

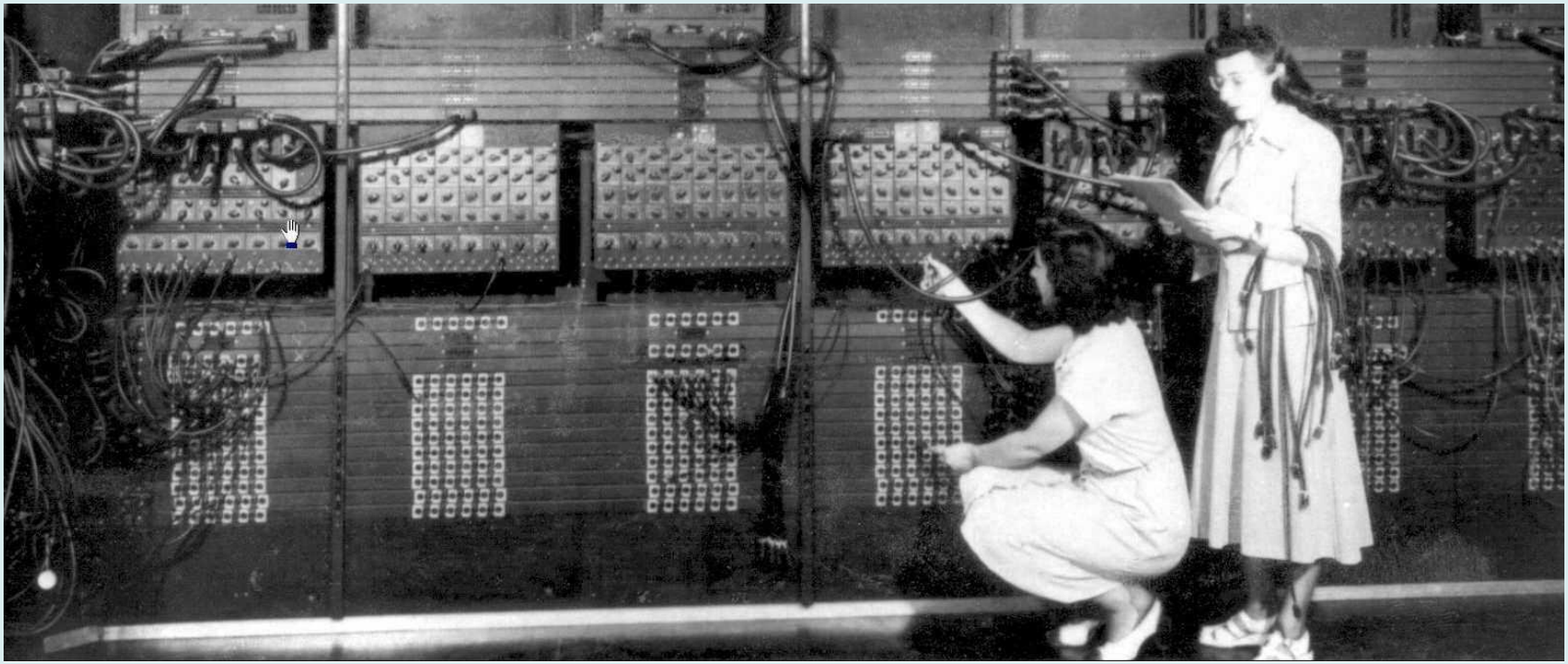
- Grafiki
  - nieprzenośna i uzależniona od platformy wykonania
- Graficznego interfejsu użytkownika (GUI)
  - interfejsy okienkowe można wydajnie realizować z użyciem gotowych bibliotek obiektowych dla języka C++, np. Qt, MFC
- Programy będą wykonywane na konsoli i mogą przekłamywać polskie znaki (przynajmniej w systemie Windows).

Nazwa przedmiotu?

Paradygmaty oprogramowania

- **Imperatywne** – program jest sekwencją rozkazów zmieniających stan pamięci oraz skoków
  - **Strukturalne** – korzysta ze strukturalnych instrukcji sterujących: sekwencja, wybór, iteracja
  - **Proceduralne** – instrukcje są grupowane w procedury lub funkcje
  - **Obiektowe** – obiekty odbierają i wykonują polecenia; funkcje są zgrupowane z danymi na których działają
- **Deklaratywne** – specyfikowany jest oczekiwany rezultat, a nie sposób jego wyznaczania
  - **Funkcyjne** – obliczenia polegają na wykonaniu funkcji, nie jest zmieniany stan programu, dane są niemodyfikowalne
  - **Logiczne** – program jest zbiorem reguł i faktów. W wyniku wnioskowania powstają nowe fakty

Pierwszy język imperatywny

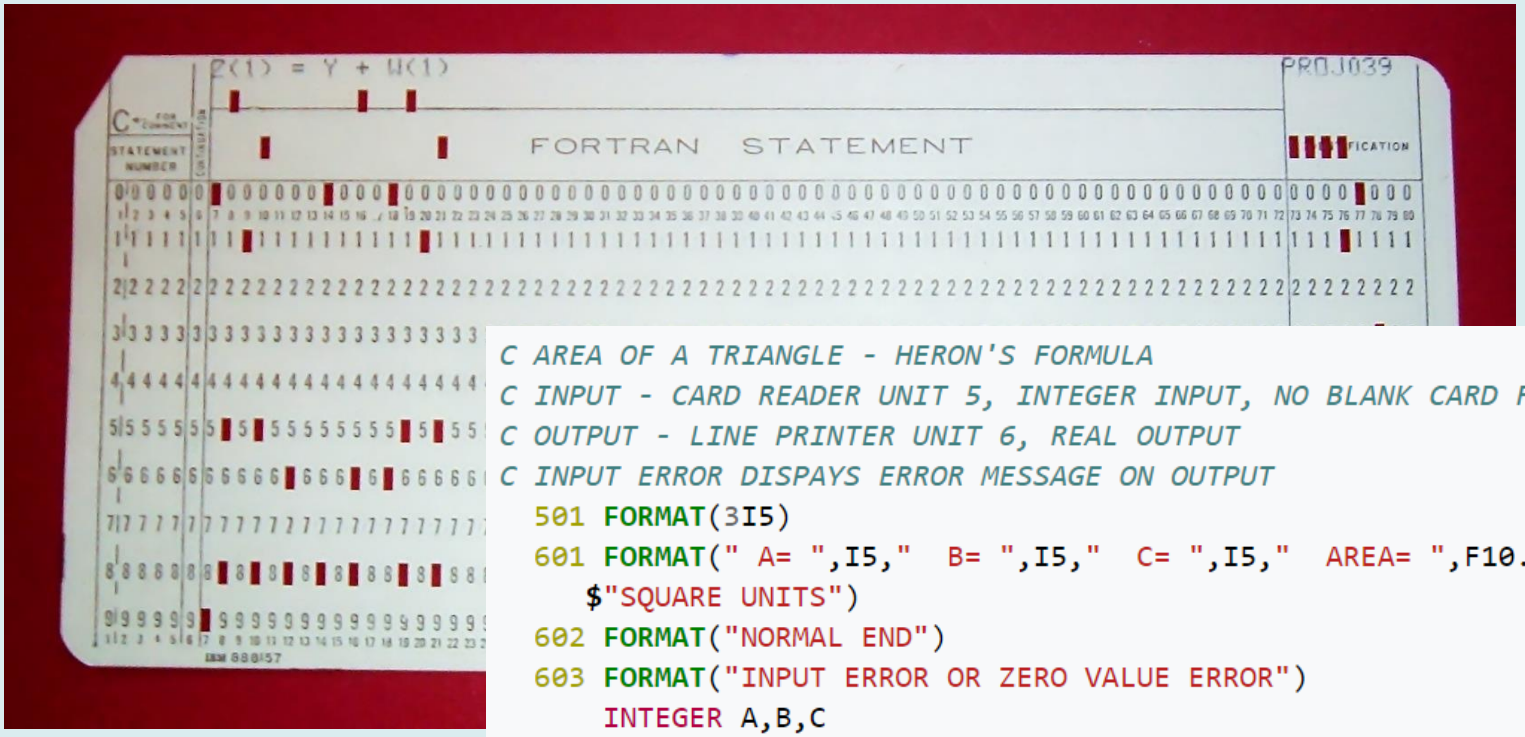


ENIAC - **Electronic Numerical Integrator and Computer** 1946
ENIAC could be programmed to perform complex sequences of operations, including loops, branches, and subroutines. However, instead of the stored-program computers that exist today, ENIAC was just a large collection of arithmetic machines, which originally had programs set up into the machine by a combination of plugboard wiring and three portable function tables (containing 1200 ten-way switches each). [Wikipedia]

Asembler

```
Dump of assembler code for function is_prime:
   0x00000001004010f0 <+0>:     push   %rbp
   0x00000001004010f1 <+1>:     mov    %rsp,%rbp
   0x00000001004010f4 <+4>:     sub    $0x40,%rsp
   0x00000001004010f8 <+8>:     movaps %xmm6,-0x10(%rbp)
   0x00000001004010fc <+12>:    mov    %ecx,0x10(%rbp)
   0x00000001004010ff <+15>:    movl   $0x2,-0x14(%rbp)
   0x0000000100401106 <+22>:    jmp    0x100401120 <is_prime+48>
   0x0000000100401108 <+24>:    mov    0x10(%rbp),%eax
   0x000000010040110b <+27>:    cltd
   0x000000010040110c <+28>:    idivl  -0x14(%rbp)
   0x000000010040110f <+31>:    mov    %edx,%eax
   0x0000000100401111 <+33>:    test   %eax,%eax
   0x0000000100401113 <+35>:    jne    0x10040111c <is_prime+44>
   0x0000000100401115 <+37>:    mov    $0x1,%eax
   0x000000010040111a <+42>:    jmp    0x10040113a <is_prime+74>
   0x000000010040111c <+44>:    addl   $0x1,-0x14(%rbp)
   0x0000000100401120 <+48>:    cvtsi2sdl -0x14(%rbp),%xmm6
   0x0000000100401125 <+53>:    cvtsi2sdl 0x10(%rbp),%xmm0
   0x000000010040112a <+58>:    callq  0x1004011d0 <sqrt>
   …
```

FORTRAN




```
C AREA OF A TRIANGLE - HERON'S FORMULA
C INPUT - CARD READER UNIT 5, INTEGER INPUT, NO BLANK CARD FOR END OF DATA
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPAYS ERROR MESSAGE ON OUTPUT
  501 FORMAT(3I5)
  601 FORMAT(" A= ",I5,"  B= ",I5,"  C= ",I5,"  AREA= ",F10.2,
     $"SQUARE UNITS")
  602 FORMAT("NORMAL END")
  603 FORMAT("INPUT ERROR OR ZERO VALUE ERROR")
      INTEGER A,B,C
   10 READ(5,501,END=50,ERR=90) A,B,C
      IF(A=0 .OR. B=0 .OR. C=0) GO TO 90
      S = (A + B + C) / 2.0
      AREA = SQRT( S * (S - A) * (S - B) * (S - C) )
      WRITE(6,601) A,B,C,AREA
      GO TO 10
   50 WRITE(6,602)
      STOP
   90 WRITE(6,603)
      STOP
      END
```

[Wikipedia]

Strukturalne i proceduralne

- Instrukcje strukturalne: sekwencja, iteracja, wybór
- Podział programu na funkcje/procedury

```
int is_prime(int n){
    for(int i=2;i<=sqrt(n);i++){
        if(n%i==0)return 1;
    }
    return 0;
}

int main() {
    printf( "%s\n",is_prime(247) ? "tak": "nie");
    return 0;
}
```

Obiektowe

```
class Person{
    string surname;
    string name;
    double height;
public:
    Person(const char*sn,const char*n,double h)
    :name(n),surname(sn),height(h){}
    void change_name(const char*n){name = n;}
    double get_height()const {return height;}
};

vector<Person> select_sort(const vector<Person>&group){
    vector<Person> result;
    for(int i=0;i<group.size();i++){
        if (group[i].get_height()>150)
            result.push_back(group[i]);
    }
    sort(result.begin(),result.end(),
            [](auto&p,auto&p1){return p.get_height()>p1.get_height();});
    return result;
}
```

Deklaratywne

- SQL

```
SELECT * FROM Person WHERE height > 150 ORDER BY height
```

- Funkcyjne (tu Java)

```
List<Person> selectSort(List<Person> personList){
    return personList.stream()
            .filter(p->p.height>150)
            .sorted((p1,p2)->Double.compare(p1.height,p2.height))
            .collect(Collectors.toList());

}
```

If programming languages had honest titles, what would they be?

Fred Mitchell, I know Ruby, Rust, Python, Haskell, C++, Erlang, and more.
Updated Jan 30

1. C++ — A Force of Nature
2. Ruby —The Slow Scripting Language
3. Haskell — Academic Hardon
4. Python — 21st Century Basic
5. Erlang — The Dying Language
6. Elixir — It ain't Ruby!
7. C# — Java for Microsoft
8. Java — You will object, even if you object!
9. Kotlin — Java could never be so cool!
10. Rust — The Be Safe Language
11. Lisp — Parentitis
12. Clojure — Parentitis with Style!
13. C — Assembler for Fraidycats
14. Assembler — The Bit Twiddler Language
15. Perl — Mean and Lean Scripting Machine
16. PHP — The Ewwww Language

https://www.quora.com

17. Forth — Stack'em Up

18. BASIC — Useless

19. Visual Basic — Mostly Useless, except in the 3rd world.

20. Go — A Google Orgy

21. Javascript — Prototyping Nightmare

22. R — A data scientist's Wet Dream

23. Julia — Whoops! We forgot Concurrency!

24. Fortran — BASIC done right!

25. Lua — The "tuck me in anywhere" language.

26. Ada — Where Real Programmers just got Real about Real Time.

27. COBOL — It won't die because it can't die because it still runs your payroll.

28. Pascal — Teacher's old time favourite to learn you a useless language.

29. PL/1 — If you know this, you worked at IBM and are now retired.

30. ALGOL — Who's your daddy? Who's your dinosaur?

https://www.quora.com

Język C
1. Wprowadzenie

Literatura

- Brian Kernighan, Denis Ritchie – Język ANSI C, WNT, 2004
- K.N. King. Język C. Nowoczesne programowanie. Wydanie II, Helion 2011
- Stephen Prata, Język C. Szkoła programowania, Helion, 2016
- Składnia: http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html

Historia

- Jest produktem ubocznym rozwoju systemu UNIX w Bell Labs (od 1969)
- Jeden z członków zespołu zdecydował się na zastąpienie asemblera językiem wyższego poziomu nazwanym B
- Ritche rozpoczął programowanie systemu w B; język stopniowo ewoluował i zmienił nazwę na C
- W 1973 roku system UNIX został przepisany w języku C

Historia standaryzacji

- 1978 – pojawia się książka Keringhan&Ritchie *The C Programming Language*
- 1989 opublikowany standard ANSI (American National Standards Institute)
- 1990 standard ISO (International Organization for Standardization)
- 1999 – rozszerzenia nowy standard ISO (najważniejsze zmiany dotyczą rozszerzenia zestawów znaków)
- 2011 – wątki, anonimowe struktury i unie

Cechy języka

- C jest językiem niskiego poziomu
  - Pozwala na operacje na bajtach, bitach, adresach
  - Wiele konstrukcji jest wprost przeniesionych z języka maszynowego
  - Blisko związany z architekturą sprzętu, np. wielkość typu całkowitoliczbowego int odpowiada długości słowa maszynowego
- C jest niewielkim językiem
  - Większość usług przeniesiona do bibliotek funkcji
- Kompilator C jest permisywny
  - Słaba kontrola typów, mało ograniczeń
  - Zakłada, że programista jest świadomy, tego, co robi

Cechy języka - zalety

- Wydajność – duża szybkość i małe zużycie pamięci
- Przenośność – dzięki standaryzacji, bliskim związkom z systemem UNIX, umieszczeniu nieprzenośnych elementów w bibliotekach
- Ekspresywność – możliwość definiowania dowolnych typów danych i funkcji
- Elastyczność – konstrukcje C mogą być bardzo oszczędne, np. odejmowanie liczb i znaków…
- Standardowa biblioteka – zbiór użytecznych funkcji
- Integracja z systemem UNIX (Linux)

Cechy języka - wady

- C jest językiem podatnym na błędy (adresy, liczby, typy logiczny mogą być mieszane)
- Programy w C mogą być trudno zrozumiałe:

  Czy to naprawdę rozwiązanie problemu 8 hetmanów?

```
v,i,j,k,l,s,a[99];
main()
{
  for(scanf("%d",&s);*a-s;v=a[j*=v]-a[i],k=i<s,j+=(v=j<s&&
(!k&&!!printf(2+"\n\n%c"-(!l<<!j)," #Q"[l^v?(l^j)&1:2])&&
++l||a[i]<s&&v&&v-i+j&&v+i-j))&&!(l%=s),v||(i==j?a[i+=k]=0:
++a[i])>=s*k&&++a[--i])
    ;
}
```

- Programy w C mogą być kłopotliwe w modyfikacji (mało mechanizmów organizujących kod)

Gdzie stosuje się C ?

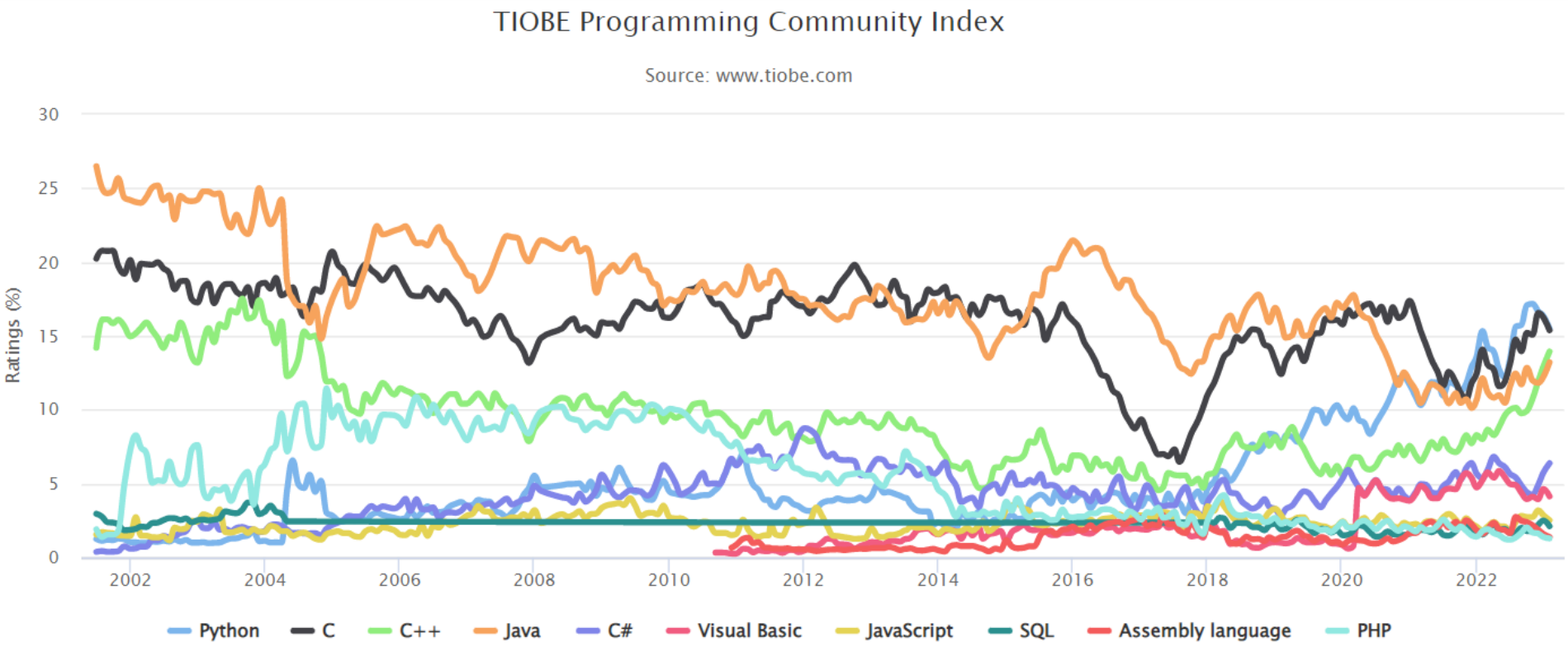
- Systemy różnej skali – od wbudowanych do dużych systemów
- Oprogramowanie systemów operacyjnych (Linux)
- Przenośne biblioteki (np. algorytmów kompresji, manipulacja formatami obrazów)
- Środowiska wykonawcze (np. maszyna wirtualna Java)
- Tam gdzie liczy się szybkość:
  - Oprogramowanie uruchamiane na kartach graficznych (GPU)
  - Implementacja funkcji wołanych z innych języków: Python, Java

Python:

```
>>> a = [[1, 0], [0, 1]]
>>> b = [[4, 1], [2, 2]]
>>> np.dot(a, b)
```

Jak często stosuje się C?

Statystyki z https://www.tiobe.com/tiobe-index/ (luty 2023)



Jak często stosuje się C?

Feb 2023	Feb 2022	Change	Programming Language	Ratings	Change
1	1		Python	15.49%	+0.16%
2	2		C	15.39%	+1.31%
3	4	^	C++	13.94%	+5.93%
4	3	v	Java	13.21%	+1.07%
5	5		C#	6.38%	+1.01%
6	6		Visual Basic	4.14%	-1.09%
7	7		JavaScript	2.52%	+0.70%
8	10	^	SQL	2.12%	+0.58%
9	9		Assembly language	1.38%	-0.21%
10	8	v	PHP	1.29%	-0.49%
11	11		Go	1.11%	-0.12%
12	13	^	R	1.08%	-0.04%

Następcy C

Język stał się źródłem inspiracji dla takich języków jak:

- C++ (nadzbiór C)
- Java
- C# (C-sharp)
- JavaScript
- Perl
- Php
- Python

Dyskusja na quora.com

C programmers are the Amish of the programming world:
living anachronisms

Tak, ale…

Podręcznikowy przykład

```
/* hello.c */

#include <stdio.h>

int main   (   )
{
       printf(   ”Hello world\n”   ) ;
       return 0;
}
```

```
> gcc hello.c
> ./a.out
Hello world
>
```

Proces translacji języka interpretacja vs. kompilacja

Interpreter

- Analizuje kolejne instrukcje programu
- Tłumaczy na kod wykonywalny.
- Wykonuje go

- **Wady**
  - Wymagany jest osobny program (interpreter). Interpreter zużywa dostępne zasoby (pamięć).
  - Wykonanie jest znacznie wolniejsze. W przypadku nawrotów ta sama instrukcja jest analizowana i tłumaczona wielokrotnie.
  - Kod programu zazwyczaj musi być w całości załadowany, co ogranicza jego rozmiary.
  - Interpretery dla różnych platform mogą różnić się miedzy sobą, co ogranicza przenośność.
- **Zalety**
  - Łatwa identyfikacja miejsca wystąpienia błędów.
  - Szybkie tworzenie i uruchamianie oprogramowania.

Proces translacji języka interpretacja vs. kompilacja

- **Kompilator**
  - Kod programu jest tłumaczony do postaci kodu maszynowego.
  - Fragmenty programu mogą być umieszczone w odrębnych plikach i kompilowane osobno.
  - Wynikowy kod jest następnie łączony w program wykonywalny przez *konsolidator* (ang.: linker).
- **Zalety**
  - Kod stworzony przez kompilator jest zazwyczaj mniejszy i wymaga mniejszej ilości zasobów (np.: pamięci) w trakcie wykonania.
  - Kod może być wykonywany znacznie szybciej.
  - Możliwe jest tworzenie znacznie większych programów, złożonych z wielu plików źródłowych.
  - Możliwa jest kompilacja skrośna.

Proces translacji języka interpretacja vs. kompilacja

- **Kompilator - wady**
  - Wymagane są odrębne programy (kompilator, linker).
  - Proces kompilacji zajmuje czas i może wymagać środowiska o dużych zasobach (pamięć, prędkość procesora).
  - Proces śledzenia i usuwania błędów jest bardziej skomplikowany.
    Część błędów jest identyfikowana przez kompilator i linker.
    Usuwanie błędów wykonania wymaga użycia odrębnego programu: *debuggera*.
  - Kompilatory zazwyczaj narzucają silne ograniczenia (typizacja zmiennych, konieczność deklaracji zmiennych i funkcji).

Proces translacji języka interpretacja vs. kompilacja

Rozwiązania mieszane

- Współczesne interpretery umożliwiają wstępną kompilację kodu do postaci pośredniej, co znacznie przyspiesza wykonanie.
- Współczesne kompilatory pozwalają na osadzenie pełnej informacji o kodzie źródłowym w programie wykonywalnym i krokowe śledzenie wykonania.
- W przypadku kompilacji skrośnej oferowane są symulatory.

Fazy budowy programu

- Preprocesor – włącza pliki nagłówkowe, zamienia symbole stałych na wartości
- Faza I – analiza kodu, podział na podstawowe symbole (tokeny) i budowa drzewa programu
- Opcjonalnie: globalna optymalizacja drzewa (łączenie i usuwanie podobnych fragmentów)
- Faza II – generacja kodu w postaci plików wynikowych (ang.: *object*) *.obj (*.o)
- Opcjonalnie: optymalizacja – łączenie powtarzających się fragmentów kodu maszynowego.
- Konsolidacja plików wynikowych z bibliotekami *.lib i tworzenie kodu wykonywalnego

Analiza kodu

Komentarz

```
/* hello.c */

#include <stdio.h>

int main   (   )
{
        printf(   "Hello world\n"   ) ;
        return 0;
}
```

Dyrektywa preprocesora włączająca plik nagłówkowy do jednostki kompilacji.

Funkcja main()

Instrukcja
Wywołanie bibliotecznej funkcji printf()
Jej skompilowany kod jest umieszczony w pliku *.lib. Podczas konsolidacji jest łączony z wynikiem kompilacji programu…

Wynik wykonania funkcji main (zwracany do powłoki)

Analiza kodu 2

```
#include <stdio.h>

int main(int argc, char** argv) {
    printf("Tydzien ma %d dni\n",7);
    return 0;
}
```

Funkcja printf() zastępuje ciąg znaków **%d** liczbą całkowitą przekazaną, jako argument

Analiza kodu 3

```
#include <stdio.h>

int main(int argc, char** argv) {
    int days;
    days=7;
    printf("Tydzien ma %d dni\n",days);
    return 0;
}
```

Deklaracja zmiennej

Zamiast stałej można przekazać zmienną. Kompilator automatycznie doda kod, który odczyta jej wartość i przekaże do funkcji.

Analiza kodu 4

Deklaracja stałej preprocesora NUMBER_OF_DAYS

```
#include <stdio.h>
#define NUMBER_OF_DAYS 7

int main(int argc, char** argv) {
    int days;
    days=NUMBER_OF_DAYS;
    printf("Tydzien ma %d dni\n",days);
    return 0;
}
```

W wyniku działania preprocesora każde wystąpienie NUMBER_OF_DAYS zostanie zastąpione wartością stałej (7)

Analiza kodu 5

Włączamy plik nagłówkowy math.h, bo tam są informacje o funkcji sin() i definicja stałej M_PI (czyli π)

```
#include <stdio.h>
#include <math.h>

int main(int argc, char** argv) {
    double y;
    y = sin(M_PI/2);
    printf("Sinus 90 stopni = %f\n",y);
    return 0;
}
```

Wywołanie funkcji sin dla argumentu π/2

Wstawienie %f umożliwia wydruk wartości zmiennoprzecinkowej typu **double**

Analiza kodu 6

```
#include <stdio.h>
#include <math.h>

int main(int argc, char** argv) {
    int x=90;
    double y;
    y = sin(2*M_PI*x/360);
    printf("Sinus %d stopni = %f\n", x ,y);
    return 0;
}
```

Deklaracja wraz z inicjalizacją

Kolejnym znacznikom w tekście muszą odpowiadać odpowiednie typy danych %d – int, %f - double

Analiza kodu 7

Deklaracja wraz z inicjalizacją zmiennej typu napis (string)

```
#include <stdio.h>

int main(int argc, char** argv) {
    int x=20;
    char* osoba="Jan Kowalski";
    printf("%s ma %d lat\n",osoba,x);
    return 0;
}
```

Możemy też wypisać tekst stosując znacznik **%s** (zmienna **osoba**)

Analiza kodu 8

Możemy zadeklarować własną funkcję

```
#include <stdio.h>

double doKwadratu(double x)
{
    return x*x;
}

int main(int argc, char** argv) {
    double x=1.27;
    double y;
    y=doKwadratu(x);
    printf("%f do kwadratu rowna sie %f\n",x, y);
    return 0;
}
```

Oraz ją wywołać…

Analiza kodu 9

```
#include <stdio.h>

double doKwadratu(double x)
{
    return x*x;
}

int main(int argc, char** argv) {
    double x=1.27;
    //double y;
    //y=doKwadratu(x);
    printf("%f do kwadratu rowna sie %f\n",x, doKwadratu(x));
    return 0;
}
```

Nie jest konieczne wywołanie etapami. Jako argument funkcji można przekazać rezultat wywołania innej funkcji…

Analiza kodu 10

Funkcja nie musi zwracać wartości.
Jeśli jej nie zwraca wpisujemy **void**

```
#include <stdio.h>

void wypiszKwadrat(double x)
{
    printf("%f do kwadratu rowna sie %f\n",x,x*x);
}

int main(int argc, char** argv) {
    double x=1.27;
    wypiszKwadrat(x);
    return 0;
}
```

Analiza kodu 11

Deklaracja tablicy i dostęp do jej elementów

```
int main() {
    int tab[10];
    tab[0]=0;
    tab[1]=1;
    for(int i=2;i<10;i++){
        tab[i]= tab[i-2]+tab[i-1];
    }
    for(int i=0;i<10;i++)printf("%d, ",tab[i]);
    return 0;

}
```

Analiza kodu 12

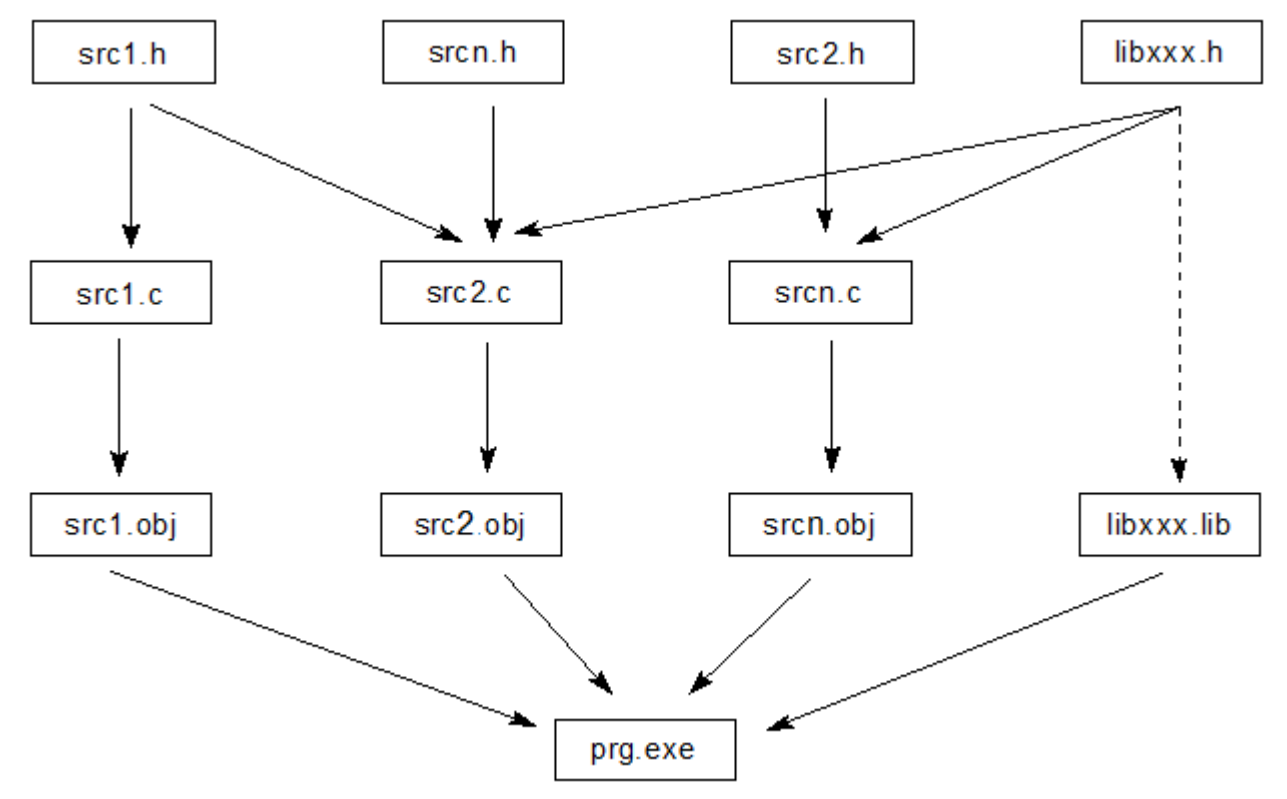
Przekazując tablicę do funkcji na ogół musimy podać jej rozmiar

```
void print_table(double tab[],int size){
    printf("[");
    for(int i=0;i<size;i++){
        printf("%f, ",tab[i]);
    }
    printf("]");
}

int main() {
    double tab[]={1,2,2.5,3};
    print_table(tab,4);
    return 0;

}
```

Deklaracja tablicy (połączona z inicjalizacją)

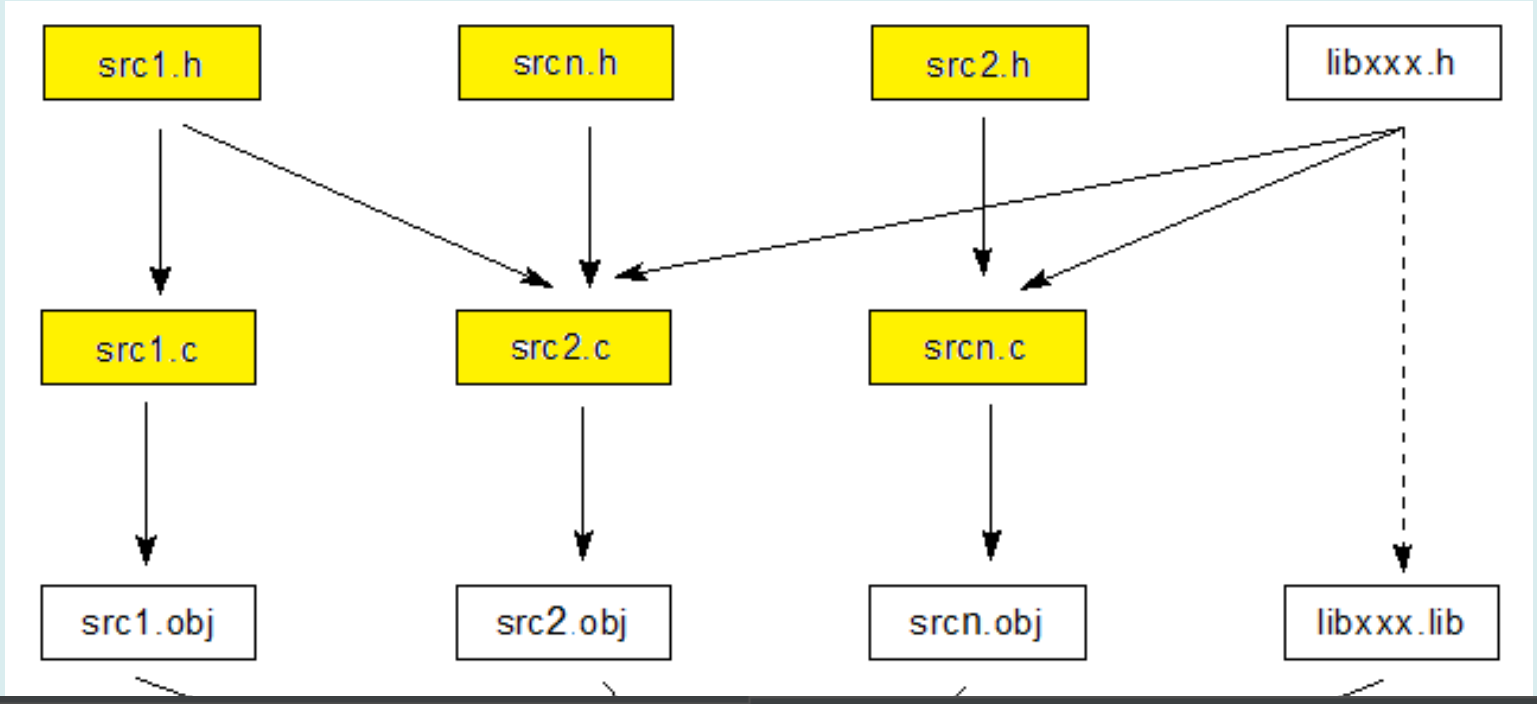
Organizacja kodu 1

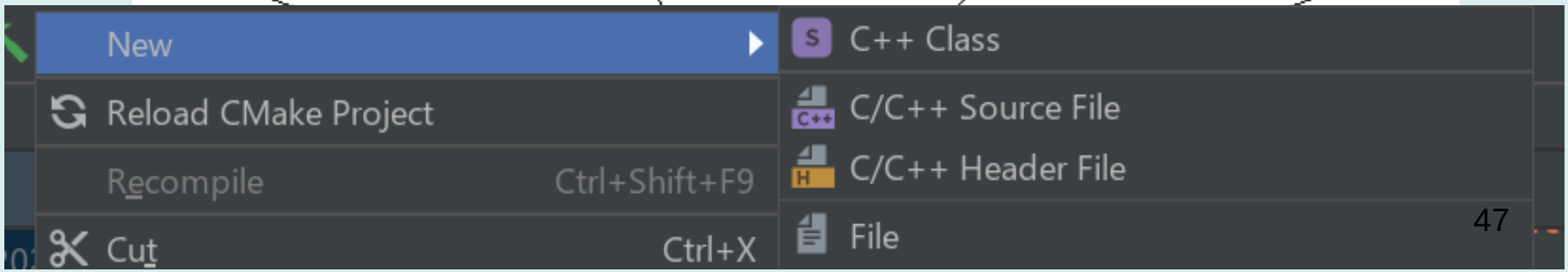
- Elementy programu mogą być umieszczone w jednym lub wielu **plikach źródłowych** oraz **bibliotekach**.



Organizacja kodu 2

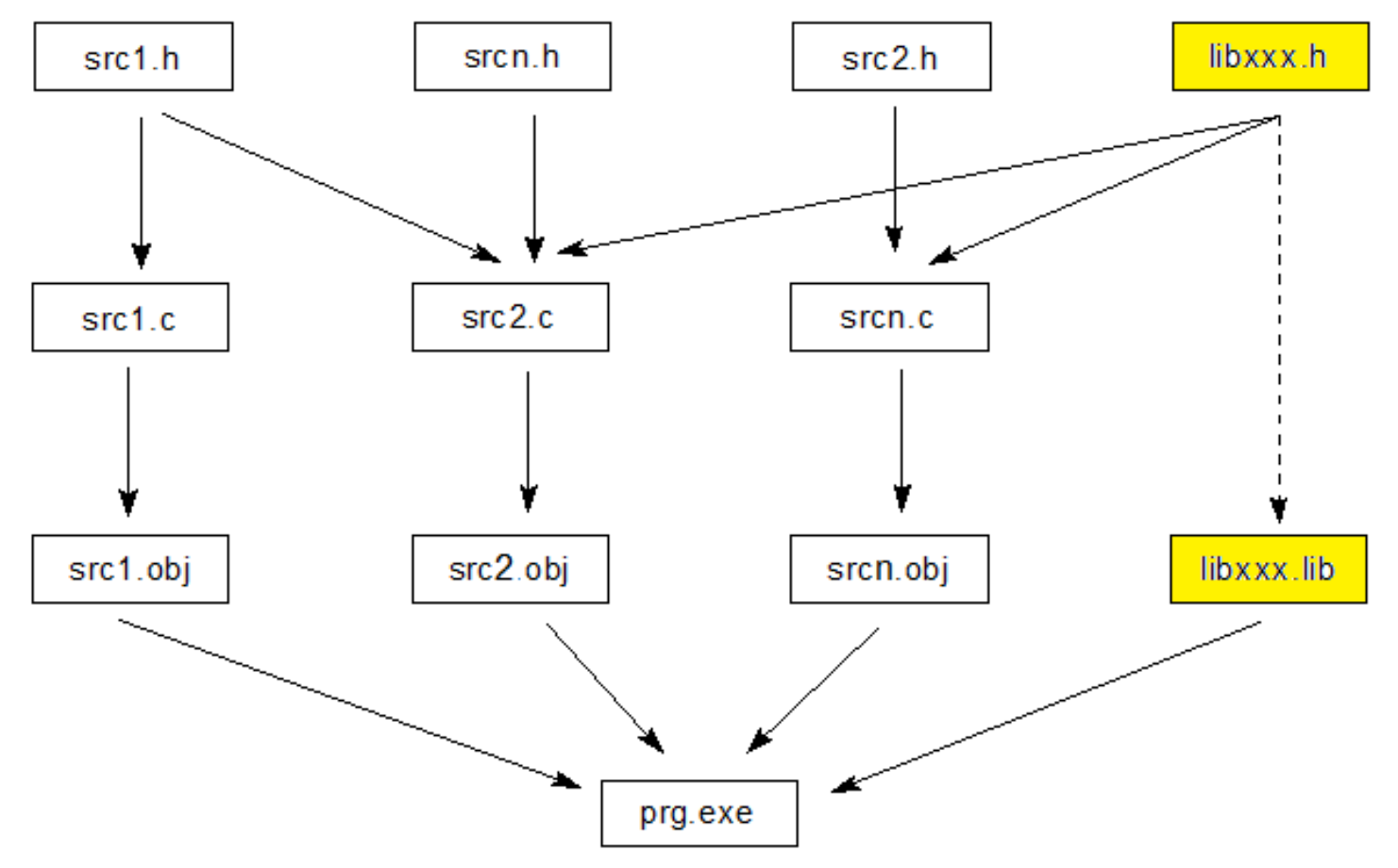
- Pliki źródłowe (*.c, *.cpp, *.h) tworzone są przez programistę aplikacji





Organizacja kodu 3

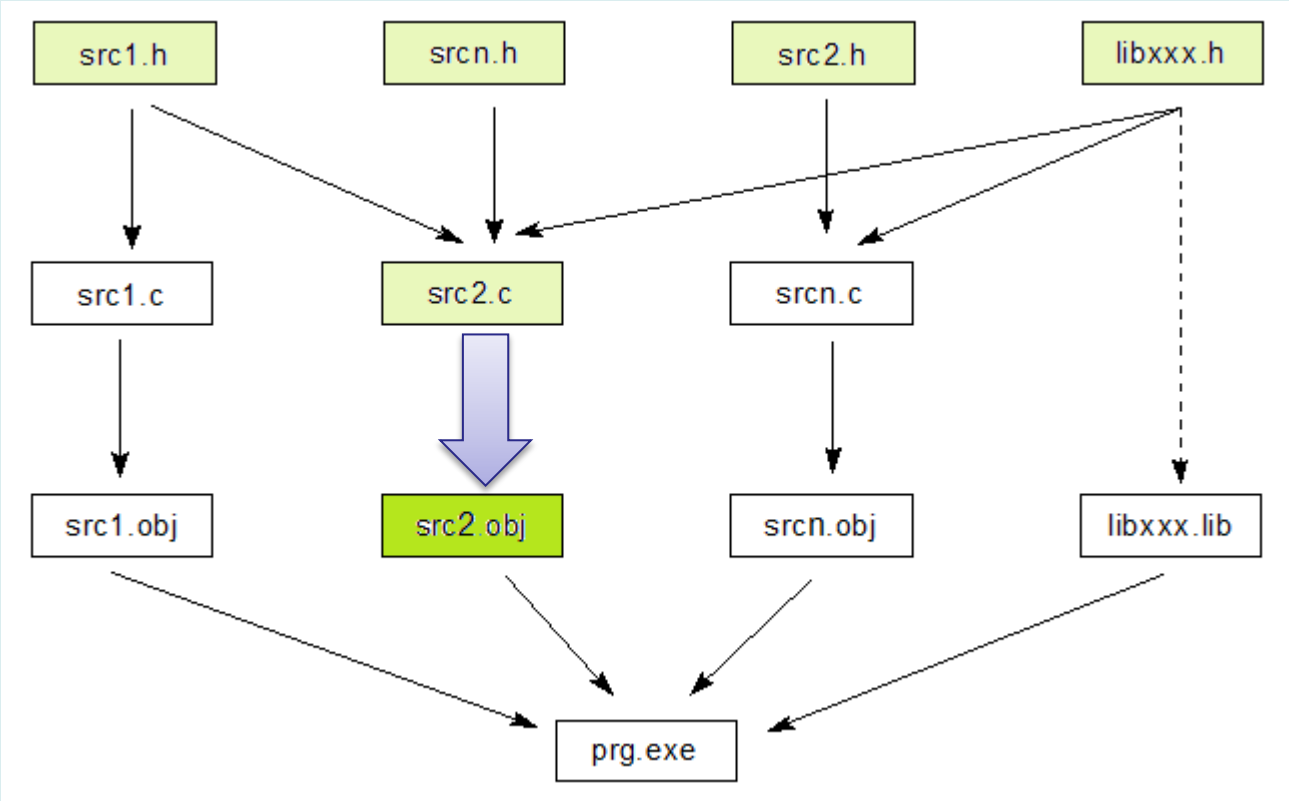
- Pliki biblioteczne wraz z nagłówkami (libxxx.h) najczęściej dostarczane są przez autorów kompilatora.



Organizacja kodu 4

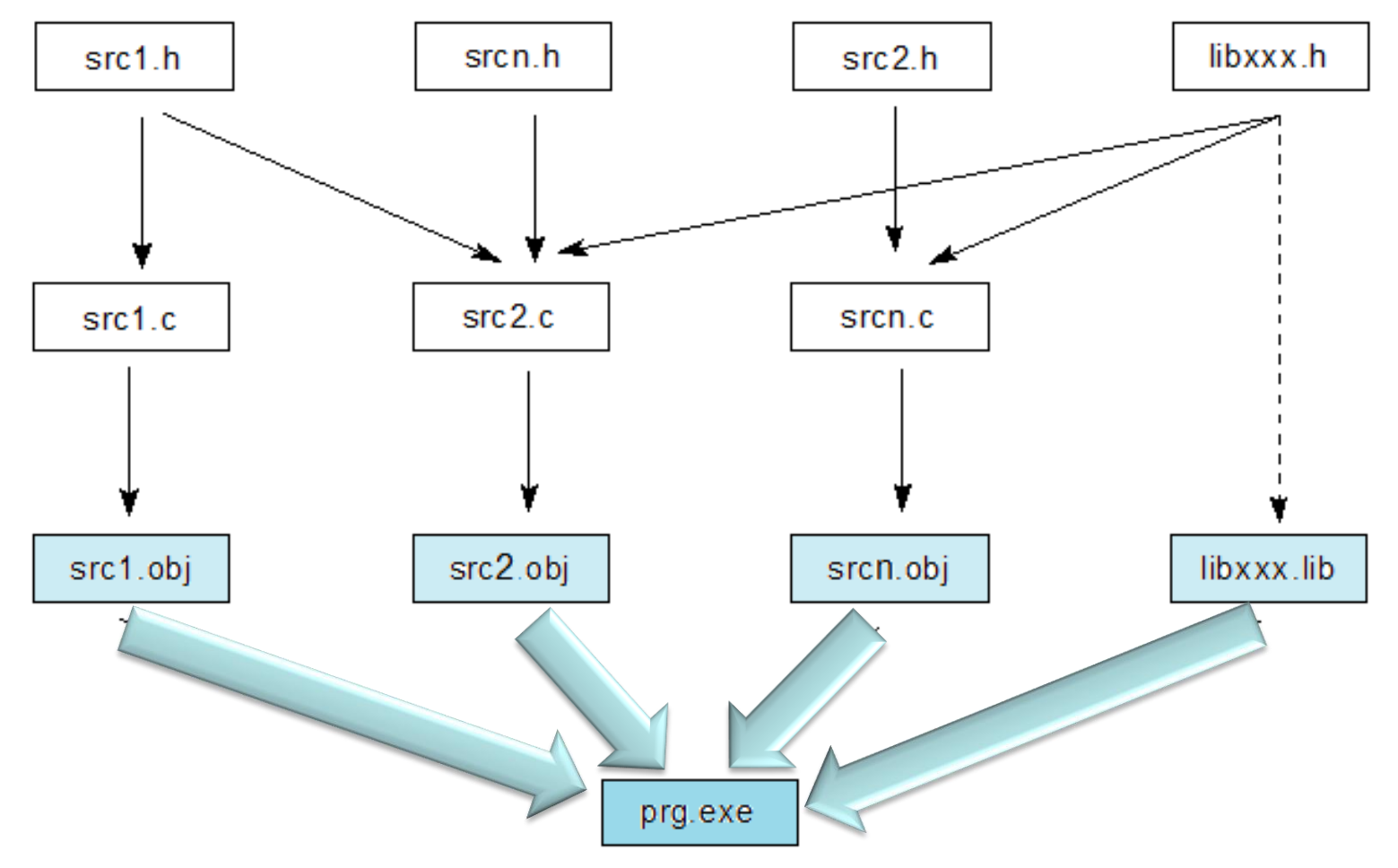
- Podczas kompilacji przetwarzany jest jeden moduł:
  - plik źródłowy *.c
  - wraz z włączonymi plikami nagłówkowymi *.h

Zużycie zasobów jest znacznie mniejsze, niż gdyby poddać kompilacji olbrzymi plik źródłowy złożony ze wszystkich plików składowych.



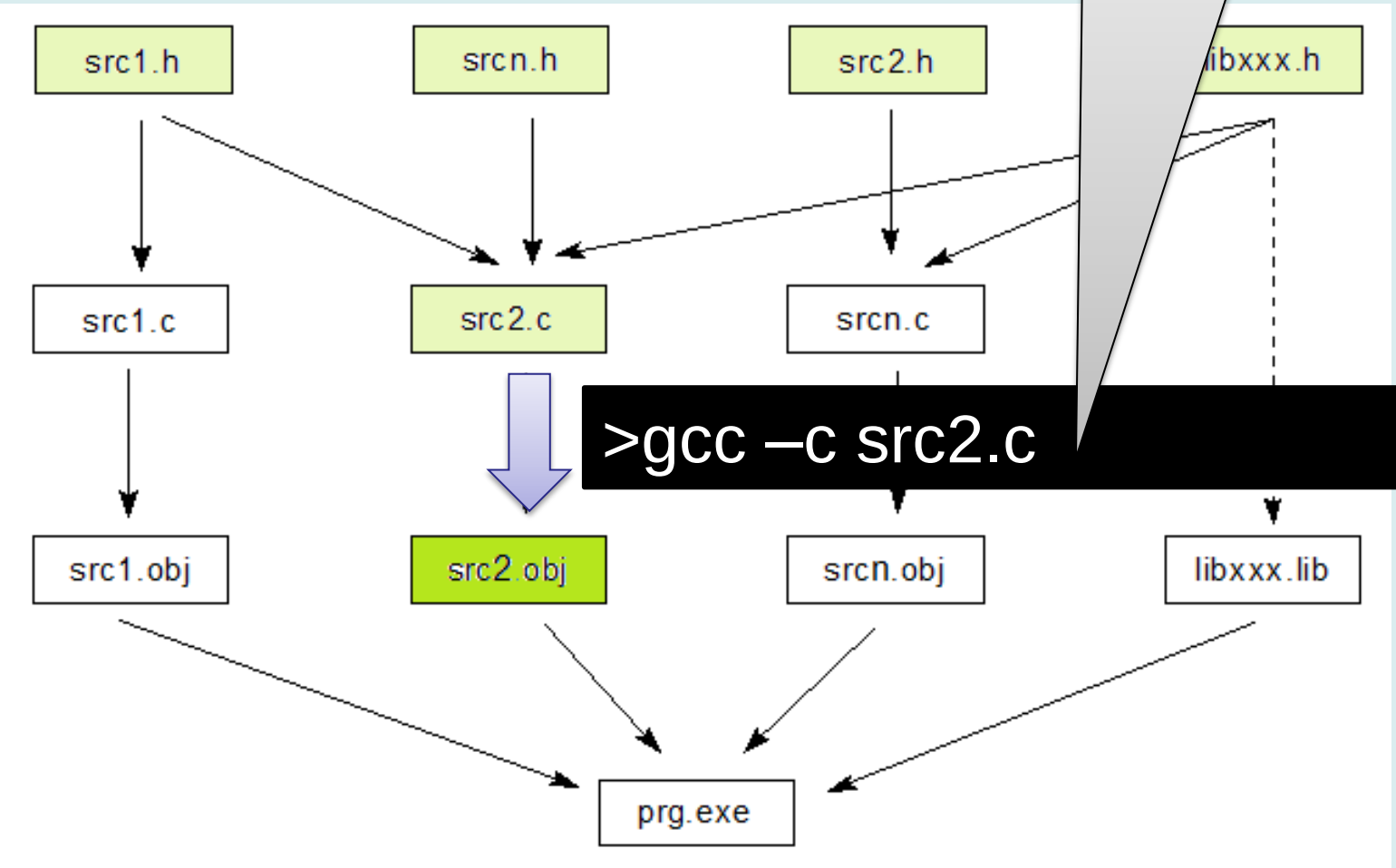
Organizacja kodu 5

- W wyniku konsolidacji (linkowania) plików *.obj i bibliotek *.lib powstaje kod wykonywalny

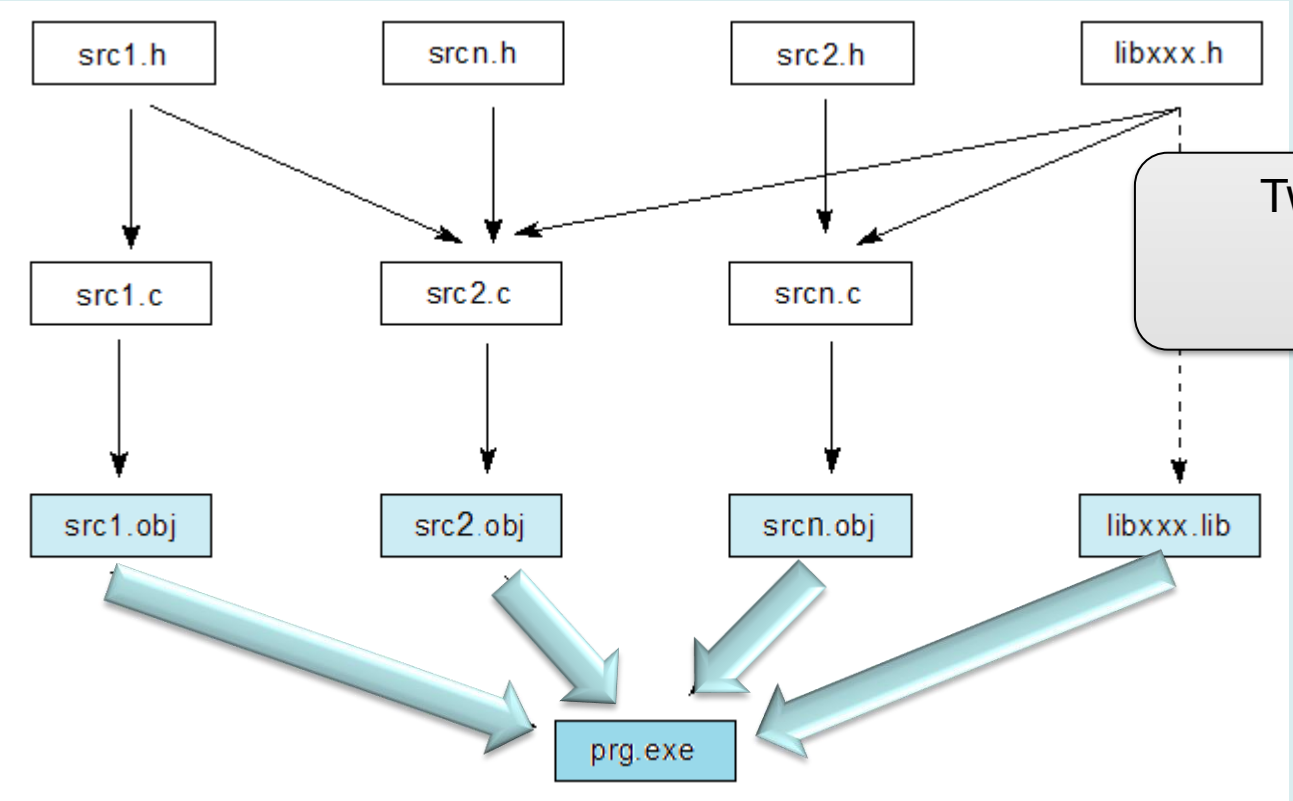


Budowa programu wielomodułowego 1

Tworzony jest plik binarny (*object*) **src2.o**



Budowa programu wielomodułowego 2



Tworzony jest plik z programem (wykonywalnym)

```
>gcc src1.o src2.o srcn.o –o –llibxxx prg.exe
```

Budowa programu wielomodułowego 3

- Operacja może być wykonana w jednym wywołaniu:

  gcc scr1.c src2.c srcn.c –llibxx –o prog.exe

- Ale minimalna zmiana w jednym module wymaga rekompilacji wszystkich…
- Zazwyczaj pomocniczą rolę pełni program **make**
  - porównuje czasy plików,
  - jeśli plik wynikowy jest starszy niż źródłowy, buduje wymagany moduł
- Pisanie plików konfiguracyjnych **makefile** dla programu **make** jest dość trudne…

IDE

IDE (*Integrated Development Environment*) to zintegrowane środowisko budowy aplikacji w C/C++:

- Zarządza zbiorem plików (projektem)
- Pozwala na wybranie bibliotek
- Automatycznie buduje **makefile** (także budując drzewo włączanych plików nagłówkowych: dyrektywy `#include ”srcn.h”`)
- Automatycznie wywołuje program **make** → kompilator i konsolidator
- Dostarcza inteligentnego edytora – podświetlanie składni, automatyczne uzupełnianie nazw, refaktoryzacja kodu…
- Integruje się z debuggerem – programem do śledzenia wykonania i usuwania błędów

IDE – przykłady 1

- NetBeans http://netbeans.org/

HelloWorldExample - NetBeans IDE 7.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services | Classes

- HelloWorldExample
  - Header Files
  - Resource Files
  - Source Files
    - main.c
  - Test Files
  - Important Files
    - Makefile
- ISOA
- IsoaLib
- IsoaTest

main(int argc, char** argv) - Navigator

- main(int argc, char** argv)
- stdio.h

main.c

```
5
6   #include <stdio.h>
7
8   int main(int argc, char** argv) {
9       printf("Hello world\n")
    !   return 0;
11  }
```

Output | Tasks | Search Results | Usages

Project Creation | HelloWorldExample (Build, Run)

```
make[2]: Leaving directory `/cygdrive/c/ps/netbeans/2012/HelloWorldExample'
nbproject/Makefile-Debug.mk:58: recipe for target `.build-conf' failed
make[1]: Leaving directory `/cygdrive/c/ps/netbeans/2012/HelloWorldExample'
nbproject/Makefile-impl.mk:39: recipe for target `.build-impl' failed

BUILD FAILED (exit value 2, total time: 2s)
```

8 | 7 INS

IDE – przykłady 2

- DevCpp http://www.bloodshed.net/dev/devcpp.html

Dev-C++ 4.9.9.2 - [Projekt1] - Projekt1.dev

Plik Edycja Szukaj Widok Projekt Uruchom Odpluskwiacz Narzędzia CVS Okno Pomoc

Nowy Wstaw Przełącz Przejdź Do

Projekt Klasy

Projekt1

main.c

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  printf("Hello world\n");
  system("PAUSE");
  return 0;
}
```

Kompilator Zasoby Log kompilacji Odpluskwiacz Wyniki poszukiwań Zamknij

Informacje:

Liczba błędów: 0

Rozmiar wyjścia: 15839 bajtów (15 KB)

Log kompilacji:

Kompilator: Default compiler
Building Makefile: "C:\lib\Dev-Cpp\Makefile.win"
Wykonywanie make...

6: 27 Wstaw 10 wierszy w pliku

IDE – przykłady 3

- Microsoft Visual Studio (dostępne dla studentów AGH)

HelloWorld - Microsoft Visual Studio

File Edit View Project Build Debug Team Data Tools Test Window Help

Debug | Win32

Solution Explorer

Solution 'HelloWorld' (1 pr
HelloWorld
External Dependencie
Header Files
Resource Files
Source Files
main.cpp

main.cpp | main.c

(Global Scope) | main(int argc, char ** argv)

```
#include <stdio.h>

int main(int argc, char** argv) {
    printf("Hello world\n")
    return 0;
}
```

100 %

Output

Show output from: Build

```
1>  main.cpp
1>c:\ps\cs\helloworld\helloworld\main.cpp(5): error C2143: syntax error : missing ';' bef
1>
1>Build FAILED.
1>
1>Time Elapsed 00:00:01.67
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

S.. C.. P.. T.. Code Definition Window Output Find Results 1

Ready Ln 15 Col 1 Ch 1 INS

Server Explorer

IDE – przykłady 4

- CodeBlocks

main.c [Test] - Code::Blocks 16.01

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Debug

<global>

Management

Projects Symbols Files

Workspace
Test
Sources
main.c

main.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       int i;
7       for(i=0;i<10;i++){
8           j=2*i+1;
9       }
10      return 0;
11  }
12
```

Logs & others

Code::Blocks | Search results | Cccc | Build log | Build messages | CppCheck | CppCheck messages

File	Line	Message
		=== Build: Debug in Test (compiler: GNU GCC Compiler) ===
C:\ps\cpp\Test...		In function 'main':
C:\ps\cpp\Test...	8	error: 'j' undeclared (first use in this function)
C:\ps\cpp\Test...	8	note: each undeclared identifier is reported only once for each function it appears in
		=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===

C:\ps\cpp\Test\main.c | Windows (CR+LF) | WINDOWS-1250 | Line 12, Column 1 | Insert | Read/Write | default

IDE – przykłady 5

- CLion

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help test3 - ...\prime.c

test3 | Debug

test3 prime.c

Project

test3 C:\ps\cpp\2020\test3
- cmake-build-debug
- CMakeLists.txt
- main.cpp
- prime.c

External Libraries

Scratches and Consoles
- Scratches

CMakeLists.txt main.cpp prime.c

```
12      return 0;
13  }
14
15  int main() {
16      printf( "%s\n",is_prime( n: 247) ? "tak": "nie");
17      return 0;
18  }
```

Run: test3

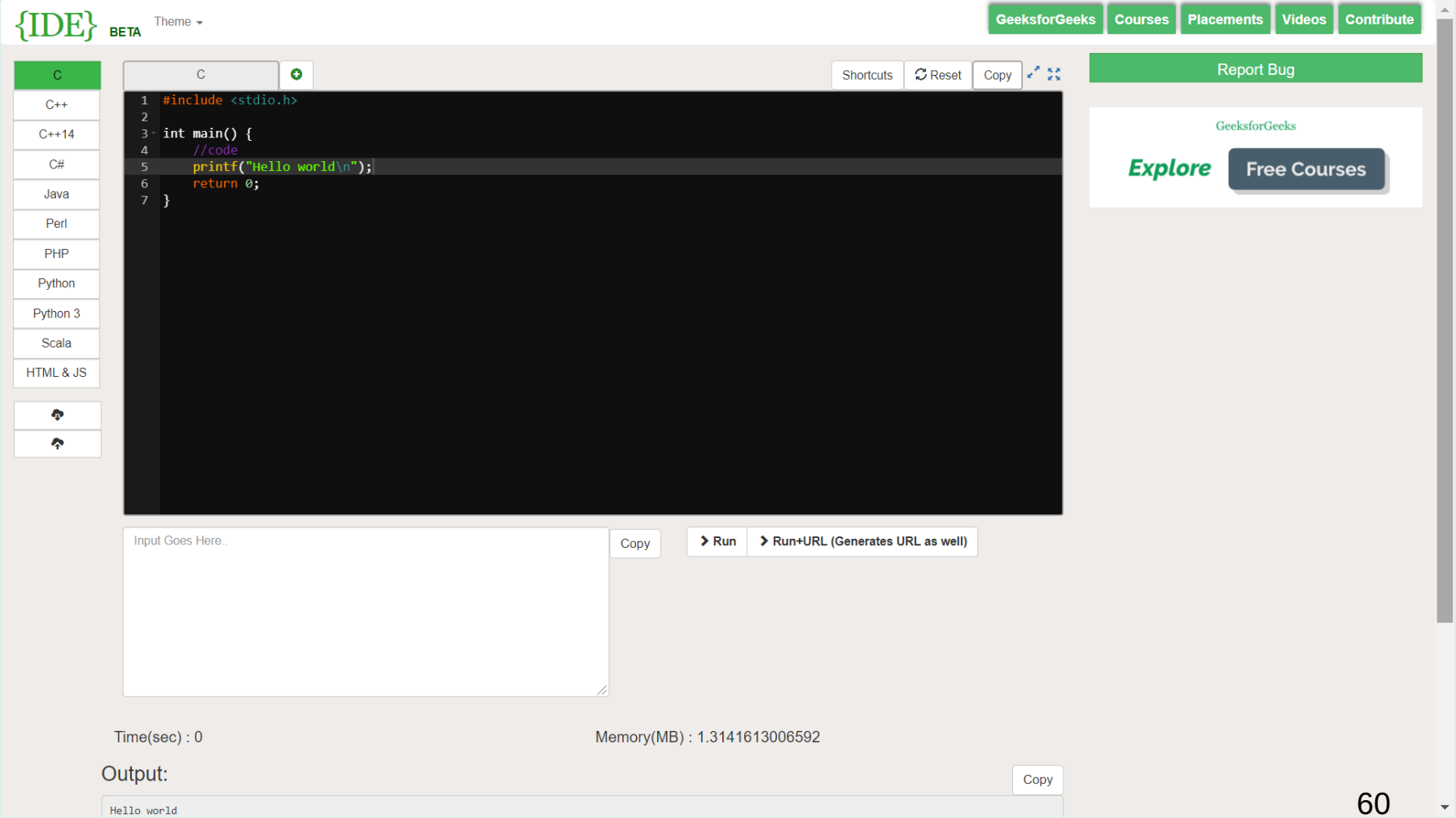
```
C:\ps\cpp\2020\test3\cmake-build-debug\test3.exe
tak

Process finished with exit code 0
```

Terminal CMake 0: Messages 4: Run 5: Debug 6: TODO Event Log

Process finished with exit code 0 14:1 CRLF UTF-8 4 spaces C: test3 | Debug

IDE – przykłady 6

- Interfejs webowy, np. https://ide.geeksforgeeks.org/



IDE – przykłady 7

I wiele innych:

- Eclipse + CDT (C/C++ Development Tooling) http://www.eclipse.org/cdt/
- Google: best IDE C for linux/windows/mac

Do zapamiętania

- **Preprocesor** – włącza pliki nagłówkowe, zamienia symbole na wartości, także zastępuje fragmenty kodu
- **Kompilator** – analizuje składnię, wykrywa błędy, tworzy pliki wynikowe *object*
- **Konsolidator** (linker) łączy pliki wynikowe z bibliotekami i tworzy plik wykonywalny (*.exe)
- Program **make** organizuje proces budowy programu
- **IDE** – pozwala skupić się na programowaniu, ma przyjazny edytor, automatycznie buduje makefile, uruchamia kompilator i konsolidator, wyświetla błędy, integruje się z debuggerem.