WYDZIAŁ FIZYKI
UNIWERSYTET WARSZAWSKI

Programowanie zaawansowane FM i NI

Wykład 1

Wstęp, rama kodu, warunki, pętle

Krzysztof Piasecki

Semestr letni roku akad. 2024-25

krzysztof.piasecki@fuw.edu.pl

- **Organizacja zajęć [zestawienie pod tym linkiem]**

  Wykład + ćwiczenia (piątki godz. $13^{15} – 16^{00}$ lub 15^{45} , sala 1.28)
  Strona www przedmiotu: www.fuw.edu.pl/~kpias/pzfmni

- **Zasady zaliczenia:**

  Punkty, na które składa się:

[24]	Wejściówki na wstępie ćwiczeń
[24 + 24 + 28]	2 kolokwia + Egzamin pisemny
[28]	Sesja poprawkowa: Egzamin pisemny

  Skala ocen (po zaokrągleniu) :

[97 – …]	5!	[75 – 82]	4	[< 50]	2
[91 – 96]	5	[63 – 74]	3+		
[83 – 90]	4+	[50 – 62]	3		

  Nieobecności na ćwiczeniach: do 2 nieusprawiedliwionych – bezstratnie.
  Każde 1 więcej: odejmuje 8 punktów.

  Zaliczenie eksternistyczne: Prośba o zgłoszenie. Na 1. ćwiczeniach – zadanie kwalifikujące.

  Test w terminie zerowym: Prośba o zgłoszenie

- **Niektóre inne wykłady i kursy w sieci www :**

T. Werner	„Programowanie 2”	`www.fuw.edu.pl/~werner/lecture`
	(wersja książkowa)	`www.fuw.edu.pl/~werner/lecture/CPlusPlus_Wyklad.pdf`
G. Łach	„Programowanie 2R”	`glach.wikidot.com/p2r`
K. Grzelak	„Programowanie”	`www.fuw.edu.pl/~kaste/wykladyfizyka_2020.html`
Geeks-for-Geeks	"C++ Tutorial"	`www.geeksforgeeks.org/cpp-tutorial`
CPlusPlus	"C++ Language"	`cplusplus.com/doc/tutorial`
cpp0x	"Kurs C++"	`cpp0x.pl/kursy/Kurs-C++/1`

- **Strony WWW z dokumentacją:**

Łatwiejsza, niekoniecznie kompletna:	`cplusplus.com/reference`
Trudniejsza, bardziej kompletna:	`cppreference.com`

⊕ **zapytaj prowadzącego**. W domu **zapytaj w przeglądarce**: wiele forów (m.in. `stackoverflow`)

Narzędzia AI

- Potrafią:
  - wprowadzać w zagadnienie
  - poprawić błędy w kodzie
  - wytworzyć kawałki kodu

- Intencją przedmiotu jest:
  - wyszkolenie **Was** w języku C++

- Możecie używać AI:
  - w domu jako korepetytor (myślcie nad odpowiedziami)
  - w domu jako wzorzec składni itd. (myślcie nad odpowiedziami)

- Nie możecie używać AI:
  - na wszelkich testach, w żaden sposób...

- Na ćwiczeniach: *odradzam*.
  Samodzielne główkowanie jest bardzo cenne dla Was!

- **Niektóre środowiska do edycji i kompilacji kodu w sieci:**

  Rozbudowane: `replit` , `Online GDB` ⇦ z przestrzenią dyskową na pliki

  Proste: `cpp.sh` , `TutorialsPoint` , `Programiz`

  (na komórkę): `CxxDroid`

- **Polecane środowisko stacjonarne:**

  Visual Studio Code: `(instrukcja instalacji)` autorstwa K. Grzelak

  Linux: dowolny edytor tekstowy + kompilator w terminalu

- **Dlaczego *warto* chodzić na wykład?**

⇨ Wykład jest mocno spięty z ćwiczeniami, tzn.
na przykładach demonstracyjnych wyjaśnimy konstrukcje,
których użyjecie na ćwiczeniach.
Na ćwiczeniach założymy, że z tymi konstrukcjami się zapoznaliście ☺

⇨ Wejściówki i kolokwia będą miały też pytania z wykładu.

- **Nieco historii**

Dennis Ritchie

Język C : Dennis Ritchie (1972)

Zamierzenia: dostęp niskopoziomowy do pamięci
przełożenie komend na kod maszynowy
mały kod, krótki kod wykonywalny.

Język C++ : Bjarne Stroustrup (1985)

Zamierzenia: rozszerzenie C o klasy
(z dziedziczeniem), szablony i inne

Bjarne Stroustrup

- **Języki o składni bardzo podobnej do C++ :** C , C# , Java

- **Niektóre systemy napisane (głównie) w C / C++ / C#**
  - Jądra systemów operacyjnych: Linux , MS Windows, OS X
  - Bazy danych: Oracle, MySQL
  - Kompilatory/interpretery języków: C/C++ , Python
  - Przeglądarki internetowe: Chrome, Edge, Opera, Safari

- **Popularność języka C++ (i pochodnych)**

Przykład: `TIOBE Index` (za luty 2025) .

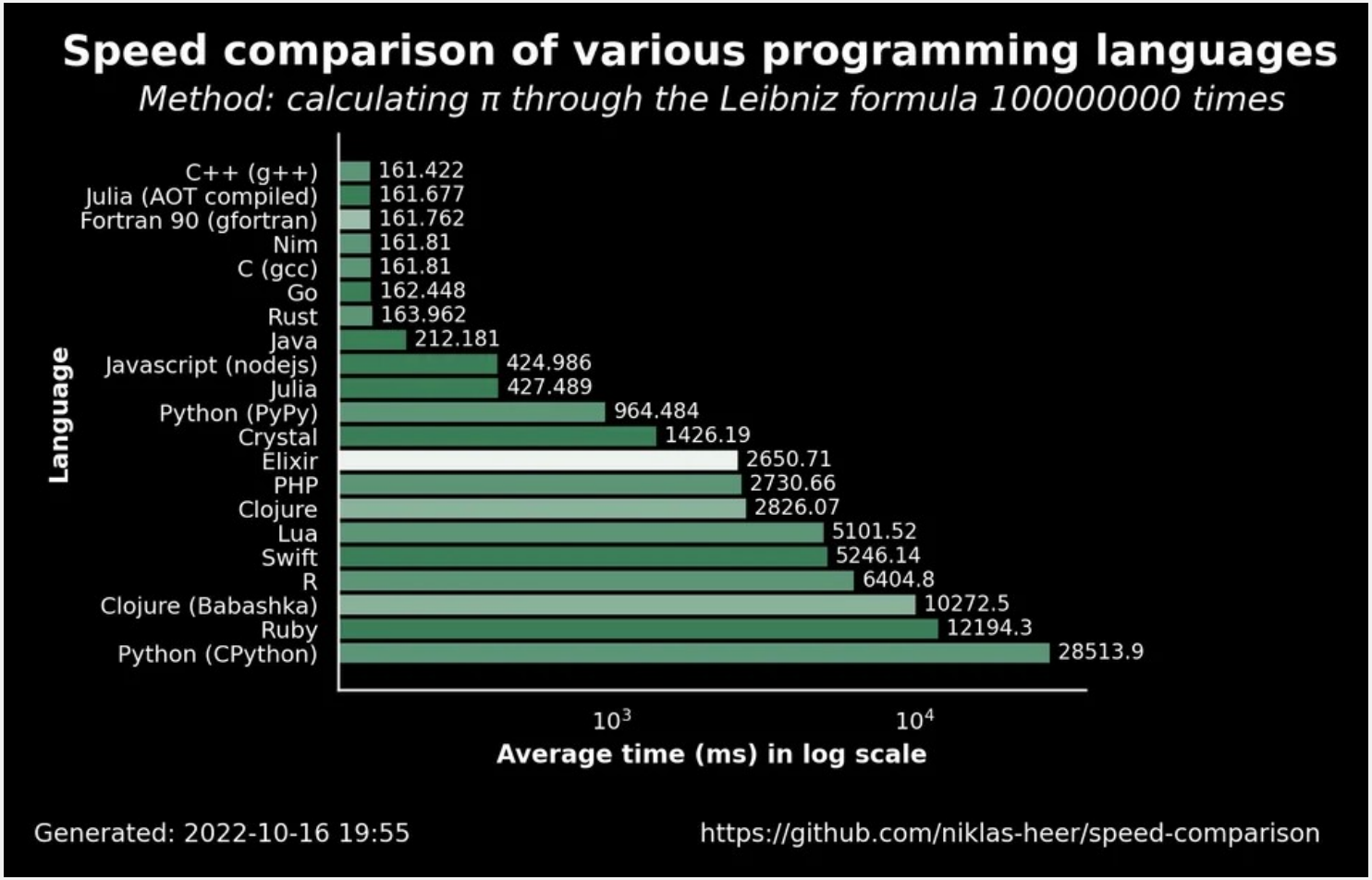
"Calculated from the number of search engine results for queries containing the name of the language. (..) Covers searches in Google, Google Blogs, MSN, Yahoo!, Baidu, Wikipedia and YouTube. "

Feb 2025	Feb 2024	Change	Programming Language	Ratings	Change
1	1		Python	23.88%	+8.72%
2	3	▲	C++	11.37%	+0.84%
3	4	▲	Java	10.66%	+1.79%
4	2	▼	C	9.84%	-1.14%
5	5		C#	4.12%	-3.41%
6	6		JavaScript	3.78%	+0.61%
7	7		SQL	2.87%	+1.04%

- **Rankingi szybkości**

Przykład: `algorytm` liczenia π poprzez ciąg Leibniza

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$



- **Kompilator a interpreter**

  Kompilacja: gdy nasz kod w danym języku jest zamieniany na kod maszynowy.
  Proces ten wykonuje **kompilator**.
  Utworzony plik z kodem maszynowym możemy bezpośrednio włączyć.

  Interpretacja: gdy nasz kod w języku jest wykonywany linia po linii przez środowisko.
  To środowisko nazywamy **interpreterem**.

  Kompilacja zabiera czas, ale raz skompilowany kod jest znacznie szybszy.

  Kod w C++ programiści (prawie) zawsze kompilują. Istnieją wyjątki – interpretery C++ .

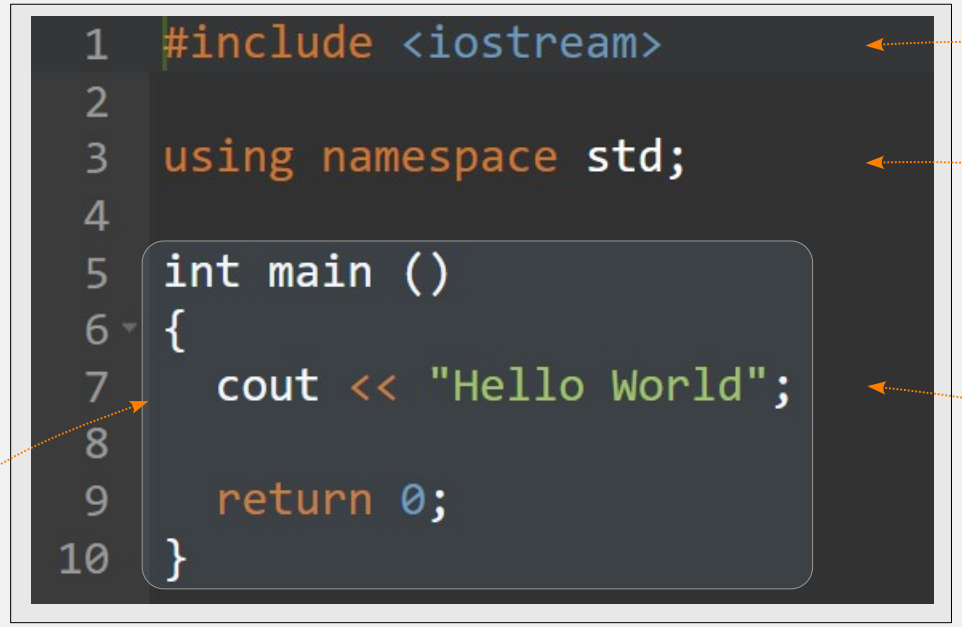
- **Typowe kompilatory C++ :**

  Linux / Mac OS : **g++** , **Clang** (*uwaga*: gcc jest kompilatorem tylko do C)

  Windows : **MinGW**

- **Typowe interpretery C++ :** cint , cling

- **Nasz pierwszy program w C++** [Link]



```
1  #include <iostream>
2
3  using namespace std;
4
5  int main ()
6  {
7    cout << "Hello World";
8
9    return 0;
10 }
```

Załączamy nagłówek biblioteki `iostream`.
to komenda dla preprocesora.
`iostream` : potrzebna do komendy `cout`

Aktywujemy przestrzeń nazw "`std`".
W niej tkwią wszystkie komendy C++ .
Bez tej linii, komendy pisalibyśmy tak:

```
std::cout << "Hello world";
```

Wypisz napis "`Hello World`".
(w uproszczeniu: na ekran).

Funkcja o nazwie `main`. W kodzie musi być przynajmniej jedna funkcja – właśnie `main`.
Program od niej startuje i na niej kończy bieg. O cechach funkcji – wkrótce.

- Wykonajmy kod. W środowisku OnlineGDB, wystarczy ▶ Run .
  W środowiskach terminalowych wpierw kompilujemy: `g++ mycode.C -o mycode.exe`
  a następnie wykonujemy: `$ ./mycode.exe`

- Rezultat:

```
Hello World
```

- **Przyjrzyjmy się kodowi.**

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main ()
6  {
7    cout << "Hello World";
8
9    return 0;
10 }
```

Sporo **znaków kontrolnych**.
W C++ *nie możemy* ich pomylić.

Każde **polecenie C++** kończy się **średnikiem**.

W linii `#...` nie ma `;`
→ bo to nie jest C++, tylko linia preprocesora.

W nagłówku funkcji `main` też nie ma `;`
To nie *polecenie*, a nagłówek funkcji.

- **Nazwa pliku. Jakie rozszerzenie?**

Dla kodu w C++ plik może mieć jedno z pięciu rozszerzeń: `.C` `.cpp` `.cc` `.cxx` `.c++`
Uwaga: rozszerzenie `.c` jest zarezerwowane dla kodu w C (nie C++).

- **Wcięcia i Enter'y w kodzie**

Kompilator zaakceptuje dowolną liczbę wcięć i Enter'ów.
Człowiek, najsłabsze ogniwo kodowania, zrobi tym więcej błędów, im mniej estetyczny kod !

⇒ Stosujmy jednolitą konwencję. Minimum 2 spacje na każdy poziom zagnieżdżenia kodu, objęty `{ }`.

- Przyjrzyjmy się **kompilacji w terminalu**: `g++ mycode.C -o mycode.exe`

  Opcja `-o` żądamy, aby plik wykonywalny nazywał się `mycode.exe`

  (W systemach Unixowych rozszerzenie w nazwie aplikacji - jest dowolne)

  Uwaga: **nigdy nie zrób tego błędu**: `g++ mycode.C -o mycode.C`
  bo skasujesz swój kod źródłowy...

- Inne proste opcje kompilatora:

  `-Wall` wyświetl wszystkie komunikaty warning (ostrzeżenia)

  `-E` wyświetl kod preprocesowany

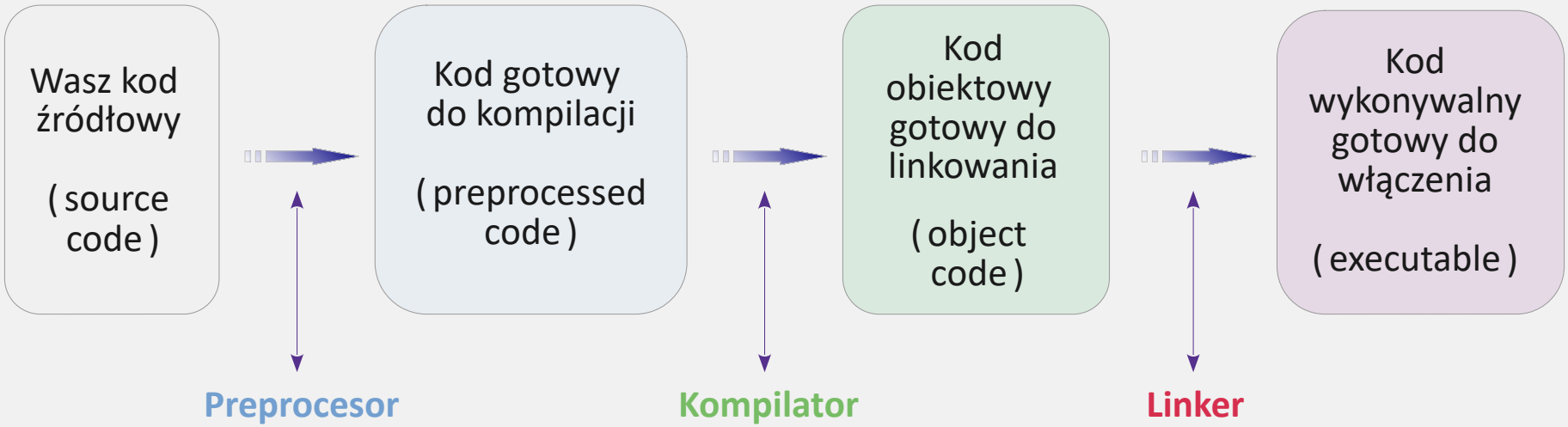
- Wykonanie kodu w systemach Unixowych: `$ ./mycode.exe`

- **Komentarze w kodzie**

  Gdy w linii umieścimy `//` , to wszystko do końca linii jest komentarzem.

  Gdy fragment ujmiemy w `/* ... */` , to całość (nawet wiele linii) jest komentarzem.

- **Etapy pracy kompilatora od kodu źródłowego do jego działania (wariant prosty)**



- W prostych sytuacjach wszystkie etapy wykonywane są przez jedną komendę, a etapy pośrednie – niewidoczne.

- Ok, ale po co jest preprocesor i linker ?

  Preprocesor: załącza kody bibliotek (z komendami) i inne. Filtruje kod przy zadanych warunkach. Wykonuje tzw. makra i dyrektywy. *Uwaga*: preprocesor nie rozumie C++ ☺

  Linker: po skompilowaniu, nasz kod może się odwoływać do funkcji w zewnętrznych kodach, w tym w bibliotekach. Linker łączy [nasz kod] ↔ [zewnętrzne kody] .

- **Zmienne i typy (wstęp)**

  Zmienna: miejsce w pamięci, mające nazwę i przechowujące wartość (daną).
  W C++ zmienna musi mieć typ. Wskazanie tego typu – to zwykle zadanie dla programisty.

  Typ zmiennej: określa rodzaj danej (np. liczba, napis, ...), ew. zakres liczb i sposób zapisu w pamięci.

- **Podstawowe (arytmetyczne) typy zmiennych**

  Standard C++ dla typów całkowitych definiuje tylko *minimalną* liczbę bajtów.

Typy całkowite

Nazwa	char	short	int	long	long long
Min. rozmiar [B = bajty]	1	2	2	4	8

Typy zmiennoprzecinkowe

Nazwa	float	double	long double
Rozmiar [B]	4	8	16 lub 9

⊕ **Typ logiczny** bool ∈ { true (1) , false (0) }

[Link]

```
#include <iostream>
using namespace std;

int main ()
{
  cout << sizeof (int)   << ' '
       << sizeof (long)  << endl;
  return 0;
}
```

- Sprawdźmy realny rozmiar w bajtach: komenda **sizeof(typ)**

→ wynik:

```
4 8
```

- **Wczytywanie z klawiatury do zmiennych**

[Link]

```cpp
#include <iostream>
using namespace std;

int main ()
{
  cout << "Podaj liczbe typu int i double: ";
  int a;
  double b;

  cin >> a >> b ;

  cout << "Wczytane liczby to: "
       << a << ' ' << b << endl;

  return 0;
}
```

Zawsze informujmy użytkownika, o co go prosimy.

Musimy zadeklarować zmienne, do których wejdą wartości podane z klawiatury.

`cin` - to komenda wczytywania z klawiatury do zmiennej (-ych).

W `" "` umieszczamy napis.
W `' '` umieszczamy 1 znak (char).

Drugi przykład ujawnia, jak myśli `cin` :

« *jeśli znak nie pasuje do typu zmiennej, to kończę czytanie do niej.* »

```
Podaj liczbe typu int i double: -12  45.67
Wczytane liczby to: -12 45.67
```

```
Podaj liczbe typu int i double: 45.67 -12
Wczytane liczby to: 45 0.67
```

- **Biblioteka matematyczna `cmath`** (zobacz **spis możliwości**)

[Link]

Działa głównie w przestrzeni `double` .

Zawiera **typowe stałe** , np.:

`M_PI` `M_E` `M_LN2` `M_SQRT2` …

… **działania**, np.:

`fabs(x)`		(moduł)
`min(x,y)`	`max(x,y)`	(wartość min / max)
`fmod(x,y)`		(reszta z dzielenia)
`round(x)`		(zaokrąglij do int)
`ceil(x)`	`floor(x)`	(zaokrąglij ⤒ / ⤓ do int)

… **funkcje matematyczne**, m.in.:

`sqrt(x)`	`cbrt(x)` $\sqrt[3]{}$	`hypot (x,y)`	$\sqrt{x^2+y^2}$
x^y `pow (x,y)`	`exp (x)`		
`sin (x)`	`cos (x)`	`tan (x)`	
`asin (x)`	`acos (x)`		
`atan (x)`	`atan2 (y,x)`		
`sinh (x)`	`tanh (x)`	...	
`asinh (x)`	`acosh (x)`	...	
`log (x)`	`log10 (x)`	...	

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main() {
  cout << "  pi     = " << M_PI    << endl
       << "  e      = " << M_E     << endl
       << "  ln(2)  = " << M_LN2   << endl
       << " sqrt(2) = " << M_SQRT2 << endl;

  double x = 2.5 ;
  cout << sqrt (x)      << endl
       << pow (2, x)    << endl
       << exp (x)       << endl
       << sin (x)       << endl
       << log (x)       << endl
       << log10 (x)     << endl
       << fmod (x, 10)  << endl
       << fabs (x)      << endl
       << hypot (3, 4)  << endl;
  cout << min (-5, 2)  << endl
       << max (-5, 2)  << endl;
  return 0;
}
```

- **Wyrażenia warunkowe (wstęp)**

[Link]

```
1  #include <iostream>
2  using namespace std;
3
4  int main () {
5    int rok = 2025;
6
7    if ( rok == 2025 ) {
8      cout << "You're in present." << endl;
9
10   } else if ( rok < 2025  ) {
11     cout << "Welcome to the past." << endl;
12
13   } else {
14     cout << "You're in future." << endl;
15   }
16 }
```

LHS: **Deklaracja** zmiennej typu `int` o nazwie `rok`

RHS: **Przypisanie** jej wartości.

Instrukcja warunkowa
`if / else if / else`

Sprawdza prawdziwość warunku (*dokładniej: czy nie-fałsz*)

Jeśli warunek spełniony, to wykonany będzie kod w `{ }` .

Użycie alternatyw jest opcjonalne.

`else` : „jeśli wszystkie próby zawiodą, to wejdź tu”

Uwaga: używamy tu **zawsze ==** . Użycie `=` zmieni sens.

- **Podstawowe operatory logiczne:**

`<` `>` `==` `!=` `<=` `>=` `&&` (i) `||` (lub)
`!` (nieprawda, że)

- Stosowanie nawiasów, np.: `(rok == 2024) && ((miesiac < 4) || (miesiac > 8))`

- **Pętle while i do..while**

[Link]

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6    int i = 5;
7
8    while ( i <= 20 ) {
9      cout << "i = " << i << endl;
10     i += 1;
11   }
12   return 0;
13 }
```

Ta zmienna będzie pełnić rolę indeksu, po którym iterujemy kroki pętli. Przypisujemy wartość startową.

Sprawdza prawdziwość warunku **na początku** każdego kroku (*dokładniej: czy nie-fałsz*)

Pojedynczy krok pętli

[Link]

```
1  #include <iostream>
2  using namespace std;
3
4  int main () {
5    int i = 5;
6    do {
7      cout << i << endl;
8      i += 1;
9    } while ( i <= 10 ) ;
10
11   return 0;
12 }
```

Otwarcie pętli `do...while`

Pojedynczy krok pętli

Sprawdza prawdziwość warunku **na końcu** każdego kroku (*dokładniej: czy nie-fałsz*)

- **Pętla for:** w nagłówku ma zawsze 3 pola, oddzielone średnikami.

[Link]

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6    for ( int i = 1 ; i <= 4 ; i++ )  {
7      cout << i << endl;
8    }
9    return 0;
10 }
```

Pole inicjalizacji

Można tu ustawić zmienną. Można też zadeklarować, ale nie jest to konieczne.

Wykona się zawsze 1×, na początku pętli.

Pole iteracji

Zwykle zmieniamy tu wartość indeksu.

Wykona się, gdy kod dotrze do **końca** kroku.

Pole warunku

Sprawdza prawdziwość warunku (*czy nie-fałsz*) **na początku** każdego kroku

- W polu iteracji użyliśmy `i++` . To **inkrementacja** (zwiększenie wartości zmiennej o 1).
  Można też tak: `i--` . To **dekrementacja** (zmniejszenie wartości zmiennej o 1).

- Można też zmienić wartość zmiennej o dowolną wartość:

```
i += 5 ;    i -= j ;    i *= 5 - k  ;    i /= sqrt(4.)  ;    i %= m ;
```

- Polecenia sterujące **Break** i **Continue**.

[Link]

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6    int i = 0;
7    while ( true )
8    {
9      i++ ;
10     if ( i % 2 == 0 ) {
11       continue ;
12     }
13     if ( i == 9 )
14       break ;
15
16     cout << i << endl;
17   }
18   return 0;
19 }
```

`continue;`

przerywa bieżący krok pętli
(ale nie całą pętlę).
O ile warunek logiczny jest spełniony,
to pętla kontynuuje bieg.

`break;`

przerywa całą pętlę.
Kod przechodzi poniżej końca.

```
1
3
5
7
```

Brak tu nawiasów { } . Tym razem to nie błąd.
C++ pozwala wyjątkowo opuścić { },
jeżeli w środku jest tylko 1 polecenie.

Uwaga: niebezpieczeństwo błędu w działaniu kodu.
Dbaj o czytelność, stosując wcięcie + Enter