

PROGRAMOWANIE I

Podstawy programowania w C++

Marek Brancewicz

30 października 2026

Wydział Fizyki
Uniwersytet w Białymstoku

Wykład 2

Plan wykładu

1. Pierwszy program w C++ („Hello, World!”).
2. Struktura programu. Nagłówki i przestrzenie nazw.
3. Jak sprawić, żeby okno programu się nie zamykało.
4. Podstawowe typy zmiennych. Zmienne lokalne i globalne. Operatory arytmetyczne.
5. Wczytywanie danych z klawiatury i wypisywanie ich na ekranie.
6. Instrukcja warunkowa `if`.

Pierwszy program

Pierwszy program

Hello, World!

Plik `hello.cpp`:

```
#include <iostream>                 // biblioteka wejścia-wyjścia
using namespace std;                // przestrzeń nazw std
int main()                          // funkcja główna
{
    cout << "Hello, World!" << endl;
    return 0;                       // 0 = poprawne zakończenie
}
```

`g++ hello.cpp -o hello`	kompilacja
`.\hello`	uruchomienie

Struktura programu

`#include`	dyrektywa preprocesora: wstawia plik nagłówkowy; bez średnika
`using namespace std;`	pozwala pisać `cout` zamiast `std::cout`
`main()`	funkcja główna – od niej zaczyna się program; dokładnie jedna
`{ }`	ograniczają *blok* instrukcji
`;`	kończy *instrukcję* – nie każdą linię (nie ma go po `#include`, `int main()`, `}`)
`return 0;`	kończy program i zwraca systemowi *kod zakończenia*: `0` = sukces (w PowerShellu: `$LASTEXITCODE`)
`//`, `/* */`	komentarz do końca linii lub na wiele linii; kompilator go pomija
spacje, entery	bez znaczenia – z wyjątkiem wnętrza napisów `"..."`

Nagłówki i przestrzenie nazw

Nagłówek plik dołączany przez `#include`; udostępnia gotowe narzędzia biblioteki standardowej

Przestrzeń nazw grupa nazw; pozwala odróżnić elementy o tej samej nazwie z różnych bibliotek

```
#include <iostream>          // cout, cin, end
#include <string>            // string (napisy)
#include <cmath>             // sqrt, pow, sin, ...
```

Nazwy z biblioteki standardowej należą do przestrzeni nazw `std`, np. pełna nazwa obiektu `cout` to `std::cout`. Dzięki `using namespace std;` można pominąć `std::`, ale wtedy własne nazwy mogą kolidować z nazwami biblioteki (np. zmienna `count` i funkcja `std::count`).

Okno programu od razu się zamyka?

Plik `.exe` uruchomiony dwuklikiem ma okno konsoli tylko na czas działania – program się kończy i okno znika. (Z terminala takiego problemu nie ma.)

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, World!" << endl;
    cout << "Nacisnij Enter, aby zakonczyc...";
    cin.get();                                  // czeka na Enter
    return 0;
}
```

Alternatywa tylko dla Windows: `system("pause")`

```
#include <iostream>
#include <cstdlib>                          // funkcje system()
using namespace std;
int main()
{
    cout << "Hello, World!" << endl;
    system("pause");                        // czeka na dowolny klawisz
    return 0;
}
```

- wypisuje „Aby kontynuować, naciśnij dowolny klawisz. . .”,
- tylko Windows: uruchamia polecenie `pause` powłoki `cmd.exe`.

Podstawowe typy zmiennych

Zmienna

Zmienna – nazwany obszar pamięci przechowujący wartość danego **typu**.

```
int wiek;                          // deklaracja
int rok = 2026;                    // deklaracja z inicjalizacją
wiek = 19;                         // przypisanie wartości
double wzrost = 1.82;              // kropka, nie przecinek!
char litera = 'A';                 // jeden znak: apostrofy
bool pelnoletni = true;            // true albo false
const double PI = 3.14159265;      // stała: nie można zmienić
```

Nazwa: litery (bez polskich znaków), cyfry i _; nie zaczyna się cyfrą; wielkość liter ma znaczenie (`wiek` $\neq$ `Wiek`); nie może być słowem kluczowym (`int`, `if`, ...).

Typy wbudowane

Typ	[B]	Zakres / uwagi
`bool`	1	`true` (1) albo `false` (0)
`char`	1	jeden znak, np. `'A'`
`short`	2	$-32\,768 \ldots 32\,767$
`int`	4	$-2\,147\,483\,648 \ldots 2\,147\,483\,647$
`unsigned int`	4	$0 \ldots 4\,294\,967\,295$
`long`	4 / 8	4 w Windows, 8 w Linuksie 64-bit
`long long`	8	ok. $\pm 9{,}2 \cdot 10^{18}$
`float`	4	ok. $\pm 3{,}4 \cdot 10^{38}$, ok. 7 cyfr znaczących
`double`	8	ok. $\pm 1{,}8 \cdot 10^{308}$, ok. 16 cyfr znaczących
`long double`	8–16	zależy od kompilatora (MinGW: 16, MSVC: 8)

Rozmiary zależą od kompilatora i systemu (standard gwarantuje tylko minimalne zakresy). `unsigned` – bez znaku, tylko ≥ 0. `string` to nie typ wbudowany, lecz klasa z `<string>`.

Sprawdzanie rozmiarów: operator `sizeof`

```
// wnętrze funkcji main()
cout << "bool:        " << sizeof(bool) << " B" << endl;
cout << "char:        " << sizeof(char) << " B" << endl;
cout << "short:       " << sizeof(short) << " B" << endl;
cout << "int:         " << sizeof(int) << " B" << endl;
cout << "long:        " << sizeof(long) << " B" << endl;
cout << "long long:   " << sizeof(long long) << " B" << endl;
cout << "float:       " << sizeof(float) << " B" << endl;
cout << "double:      " << sizeof(double) << " B" << endl;
cout << "long double: " << sizeof(long double) << " B" << endl;
```

`sizeof(typ)` – liczba bajtów zajmowanych przez typ lub zmienną.

Zmienne lokalne i globalne

Globalna zadeklarowana poza funkcjami; widoczna w całym pliku od miejsca deklaracji; istnieje przez cały czas działania programu; bez inicjalizacji ma wartość 0

Lokalna zadeklarowana w bloku `{ }`; istnieje i jest widoczna od deklaracji do końca bloku; bez inicjalizacji ma wartość **nieokreśloną**

Przesłanianie zmienna lokalna o tej samej nazwie co globalna zasłania ją w swoim bloku

Zasada: **używaj zmiennych lokalnych** i nadawaj im wartość początkową. Przy zmiennych globalnych trudno śledzić, gdzie zmienia się ich wartość.

Zasięg zmiennych – przykład

```
int x = 10;                    // zmienna globalna
int main()
{
    cout << x << endl;         // 10 - globalna
    {
        int x = 5;             // lokalna: przesłania globalną
        cout << x << endl;     // 5  - lokalna
    }
    cout << x << endl;         // 10 - znów globalna
    return 0;
}
```

Zmienna lokalna `x` istnieje tylko do końca bloku `{ }`. Globalna `x` nie została zmieniona – była tylko **przesłonięta**.

Przypisywanie wartości

```
int liczbaStudentow = 25;
liczbaStudentow = liczbaStudentow + 1;    // 26
liczbaStudentow += 1;                     // to samo krócej: 27
liczbaStudentow++;                        // zwiększenie o 1: 28
```

- = to **przypisanie**, a nie równość z matematyki: najpierw obliczana jest prawa strona, potem wynik trafia do zmiennej po lewej.
- Zapis `x = x + 1` nie ma sensu jako równanie, ale jest poprawną instrukcją.
- Skróty: +=, -=, *=, /=; `x++` zwiększa, a `x--` zmniejsza wartość o 1.

Operatory arytmetyczne

`+`, `-`, `*`, `/` dodawanie, odejmowanie, mnożenie, dzielenie

`%` reszta z dzielenia (tylko liczby całkowite): `7 % 2` $\rightarrow$ `1`

```
int a = 7, b = 2;
double x = 7.0;
cout << a / b << endl;    // 3    int / int: dzielenie całkowite
cout << x / b << endl;    // 3.5  wystarczy jeden double
double d = a / b;         // 3 !  najpierw int / int
```

Typ wyniku zależy od typów **operandów** (wartość, na której działa operator), a nie od typu zmiennej, do której zapisujemy wynik!

Wczytywanie i wypisywanie danych

Strumienie wejścia i wyjścia

`cout <<`	wypisuje dane na ekran (strumień wyjścia)
`cin >>`	wczytuje dane z klawiatury do zmiennej
`endl`	nowa linia i opróżnienie bufora wyjścia
`"\n"`	nowa linia zapisana wewnątrz napisu

Operatory można łączyć w łańcuch:

```
cout << "x = " << x << ", y = " << y << endl;
cout << "2 + 3 = " << 2 + 3 << endl;          // 2 + 3 = 5
cin >> a >> b;     // dwie wartości (spacja lub Enter między nimi)
```

Strzałki pokazują kierunek przepływu danych:

`cout << x` – z `x` na ekran, `cin >> x` – z klawiatury do `x`.

Przykład

```
// wnętrze main(); potrzebne #include <iostream> i <string>
    double x;
    string imie;
    cout << "Podaj liczbe: ";
    cin >> x;
    cout << "Podana liczba to: " << x << endl;
    cout << "Podaj imie: ";
    cin >> imie;
    cout << "Czesc, " << imie << "!" << endl;
```

Zmienną trzeba zadeklarować, zanim wczyta się do niej wartość. Typ zmiennej decyduje, jak `cin` interpretuje wpisane znaki.

Wczytywanie napisów: `cin >>` a `getline`

`cin >>` czyta do pierwszego białego znaku (spacji, tabulatora, Entera). Po wpisaniu `Jan Kowalski`:

```
cin >> imie;          // Jan Kowalski  ->  imie = "Jan"
```

Reszta (`Kowalski`) zostaje w strumieniu i trafi do następnego odczytu.

Całą linię wczytuje `getline`:

```
getline(cin, imie);   // Jan Kowalski  ->  imie = "Jan Kowalski"
```

Uwaga: `getline` tuż po `cin >>` wczyta pustą linię – Enter po poprzedniej wartości wciąż jest w strumieniu.

Okno znów się zamyka?

Ten sam problem: po `cin >> n` w strumieniu zostaje Enter (`'\n'`), więc `cin.get()` od razu go odczytuje i program się kończy. Trzeba najpierw usunąć resztę linii:

```
int n;
cout << "Podaj liczbe: ";
cin >> n;
cin.ignore(1000, '\n');      // usuwa resztę linii z Enterem
cout << "Nacisnij Enter, aby zakonczyc...";
cin.get();
```

Tak samo przed `getline` wywołanym po `cin >>`.

Instrukcja warunkowa if

Instrukcja warunkowa if

Czyli jak nauczyć komputer podejmowania decyzji

Przykłady: weryfikacja kodu PIN, logowanie nazwą użytkownika i hasłem.

```
if (warunek)
{
    // wykonywane, gdy warunek jest prawdziwy (true)
}
else
{
    // wykonywane, gdy warunek jest fałszywy (false)
}
```

Warunek zapisujemy w nawiasach okrągłych; część `else` jest opcjonalna.

Przy jednej instrukcji klamry można pominąć, ale bezpieczniej je pisać.

Operatory porównania

`==`	równe	`a == b`
`!=`	różne	`a != b`
`<`	mniejsze	`a < b`
`>`	większe	`a > b`
`<=`	mniejsze lub równe	`a <= b`
`>=`	większe lub równe	`a >= b`

Wynik porównania ma typ `bool`: `true` albo `false` (`cout` wypisuje `1` i `0`).

Uwaga: = to przypisanie, == to porównanie. `if (x = 5)` się skompiluje, ale przypisze `5` do `x` i warunek zawsze będzie prawdziwy; `-Wall` wypisze ostrzeżenie.

Operatory logiczne

`&&`	iloczyn logiczny „i”	`(x > 0) && (x < 10)`
`||`	suma logiczna „lub”	`(x < 0) || (x > 10)`
`!`	negacja „nie”	`!(x == 0)`

Kolejność: najpierw `!`, potem `&&`, na końcu `||`. W razie wątpliwości – nawiasy.

Ćwiczenie. Dla `x = 3`, `a = 7` określ wartość wyrażenia:

```
!((x <= 5) || (x > 12) || (a != 7) && (a > 15))
```

Jak działają operatory logiczne?

A	B	A && B	A || B
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

Negacja: `!true` → `false`, `!false` → `true`.

Obliczanie skrócone: gdy wynik jest znany po lewym operandzie, prawy **nie jest obliczany** (`false && ...` to zawsze `false`, `true || ...` to zawsze `true`).

```
if (n != 0 && 10 / n > 1)    // dla n = 0 dzielenie się nie wykona
```

Wiele warunków: `else if`

```
int punkty = 76;
if (punkty >= 90)
cout << "Ocena: 5" << endl;
else if (punkty >= 70)
cout << "Ocena: 4" << endl;          // wykona się tylko to
else if (punkty >= 50)
cout << "Ocena: 3" << endl;
else                                 // pozostałe przypadki
cout << "Ocena: 2" << endl;
```

Warunki są sprawdzane po kolei; wykonuje się tylko blok **pierwszego** prawdziwego. Kolejność ma znaczenie: gdyby najpierw sprawdzać `punkty >= 50`, każdy wynik od 50 wzwyż dałby ocenę 3.

Przykład: weryfikacja PIN

```
// wnętrze funkcji main();
const int PIN = 1234;          // stała
int pin;
cout << "Podaj PIN: ";
cin >> pin;
if (pin == PIN)
cout << "PIN poprawny." << endl;
else
cout << "Bledny PIN!" << endl;
```

Pytanie: czy `int` to dobry typ do przechowywania PIN-u? Co się stanie dla PIN-u `0123`?

Przykład: logowanie

```
// wnętrze funkcji main(); potrzebne #include <string>
string login, haslo;
cout << "Login: ";
cin >> login;
cout << "Haslo: ";
cin >> haslo;
if (login == "admin" && haslo == "a12b34c")
cout << "Zalogowano." << endl;
else
cout << "Bledny login lub haslo." << endl;
```

Oba warunki muszą być spełnione jednocześnie (&&).

Instrukcja warunkowa if

Ćwiczenie: co zostanie wypisane?

```
int x = 12;
int y = 7;
if (x > 10 && y < 5)
cout << "A" << endl;
else
cout << "B" << endl;
```

- Który z warunków jest prawdziwy?
- Jaka jest wartość całego wyrażenia połączonego przez &&?
- Co się zmieni, jeśli zamienimy && na ||?

Sprawdzanie poprawności wczytanych danych

```
int n;
cin >> n;
if (cin.fail())                    // odczyt się nie udał
{
    cout << "To nie jest liczba!" << endl;
    cin.clear();                   // kasuje stan błędu
    cin.ignore(1000, '\n');        // usuwa resztę linii
}
```

`cin.fail()`	`true`, gdy odczyt się nie udał (np. litery)
`cin.clear()`	kasuje stan błędu (inaczej `cin >>` nie działa)
`cin.ignore()`	usuwa ze strumienia jeden znak
`cin.ignore(1000, '\n')`	usuwa resztę linii (najwyżej 1000 znaków)

Ćwiczenie: znak liczby

Napisz program, który:

1. wczyta liczbę całkowitą,
2. sprawdzi, czy jest dodatnia, ujemna czy równa zero,
3. wypisze odpowiedni komunikat.

Podpowiedź: `cin`, `if`, `else if`, `else`.

```
int liczba;
cin >> liczba;
// uzupełnij warunki
```

Najczęstsze błędy

- brak średnika na końcu instrukcji: `int x = 3` zamiast `int x = 3;`
- = zamiast == w warunku: `if (x = 5)`
- średnik po warunku: `if (x > 0);` – warunek steruje wtedy pustą instrukcją, a blok w klamrach wykona się zawsze
- brak klamr przy kilku instrukcjach: do `if` należy tylko pierwsza z nich
- mylenie napisu `"123"` z liczbą `123`
- oczekiwanie wyniku `3.5` dla `7 / 2`
- odczyt zmiennej, której nie nadano wartości

Część z nich wykryje kompilator – czytaj ostrzeżenia (`-Wall -Wextra`).

Podsumowanie

- Wykonanie programu C++ zaczyna się w funkcji `main()`.
- Typ zmiennej określa, jakie wartości może przechowywać i ile zajmuje pamięci.
- Wynik działania arytmetycznego zależy od typów operandów (`7 / 2` to 3).
- `cout <<` wypisuje dane, `cin >>` je wczytuje; Enter zostaje w strumieniu.
- Operatory porównania i logiczne (`&&`, `||`, `!`) tworzą warunki.
- `if`, `else if` i `else` pozwalają wybrać ścieżkę wykonania.

Pytania kontrolne

1. Czym różni się = od ==?
2. Jaki typ wybierzesz do przechowywania wartości `true`/`false`?
3. Ile wynosi `7 / 2`, a ile `7 % 2`, jeśli obie liczby są typu `int`?
4. Kiedy warunek `A && B` jest prawdziwy, a kiedy `A || B`?
5. Czym różni się `if` od `if–else`?
6. Dlaczego `cin.get()` wywołane po `cin >> n` nie zatrzymuje programu?

Zadania

Zadanie 1: wczytywanie i wypisywanie danych

Napisz program, który wczytuje z klawiatury liczby (całkowite lub rzeczywiste), wykonuje na nich proste obliczenia (np. dodawanie) i wypisuje wynik z odpowiednim opisem. Wykorzystaj zmienne liczbowe i tekstowe.

Sprawdź:

- co się stanie, gdy zamiast liczby wpiszesz tekst,
- wyniki działań na liczbach tego samego typu i różnych typów (`int`, `double`) – zwróć uwagę na dzielenie, np. `7 / 2`,
- co się stanie przy dzieleniu przez `0` – osobno dla `int` i `double`.

Zadanie 2: sprawdzanie wieku

Program pyta o rok urodzenia i oblicza wiek. Sprawdza, czy użytkownik jest pełnoletni (18+) i czy może kandydować na urząd Prezydenta RP (35+), i wypisuje odpowiedni komunikat. Program można rozbudować o inne progi wieku i związane z nimi odpowiednie informacje zwrotne.

Zadanie 3: równanie kwadratowe

Program wczytuje współczynniki a, b, c i rozwiązuje równanie $ax^2 + bx + c = 0$. Za pomocą instrukcji `if` wykonaj obliczenia i wypisz komunikat zależny od znaku Δ. // wnętrze funkcji main(); Uwzględnij przypadek $a = 0$. Pierwiastek kwadratowy: `sqrt()` z biblioteki `<cmath>`.