C++

Zalety i mocne strony C++

- Szybki i "lekki" - Programy są szybkie i małe w rozmiarze bo C++ jest językiem **kompilowanym**
- Kompatybilność - Jako że jest językiem kompilowanym, zadziała na każdym urządzeniu
- Możliwości - C++ jest językiem "niskopoziomowym" co daje bezpośredni dostęp do zarządzania pamięcią i czynnościami komputera
- Stary – Dobrze udokumentowany przez środowisko programistów, dla C++ można znaleźć wiele powstałych bibliotek i poradników

Kod Skompilowany – Kod gotowy do wykonania przez komputer w swoim natywnym języku
Kod Interpretowany – Wymagany jest zewnętrzny program który wykonuje ten kod

Wady C++

- Skomplikowany - początkującym jest bardzo łatwo robić błędy
- Stary - współczesny C++ jest "zaśmiecony" starymi rozwiązaniami, oraz tymi z C
- Bardzo manualny – wieloma rzeczami trzeba się zająć ręcznie, gdzie w innych językach (t.j. Python) zostałoby to wykonane automatycznie

Zastosowania C++

- Aplikacje wymagające możliwie najwyższej wydajności (Photoshop, Adobe Premiere Pro)
- Gry (90% gier zostało zrobione w C++ lub w trakcie ich tworzenia wykorzystano C++)
- Systemy Operacyjne (Windows, Linux, itd.)
- Sterowniki (Obsługa myszki, klawiatury, słuchawek itp..)
- Systemy zintegrowane (Telewizor, Samochód, Kalkulator, itp.)

C++ vs Python

Sortowanie 1,000,000 liczb naturalnych
za pomocą wbudowanych funkcji "sort"
(Średnia z 10 testów)

Python: 246.6ms

C++: 68.1ms

Kopiowanie 1,000,000 liczb naturalnych
za pomocą pętli
(Średnia z 10 testów)

Python: 45.3ms

C++: 1.1ms

Wersja Pythona: 3.11.1
Kompilator C++: MSVC C++20 (Release x64)

C++ to język statycznie typowany

Każda zmienna musi mieć określony typ.

C++

```
int mojaZmienna = 123;
```

Typ — Nazwa zmiennej — Wartość początkowa

Python

```
mojaZmienna = 123
```

Nazwa zmiennej — Wartość początkowa

Podstawowe typy w C++

- int – liczba całkowita (np. -41)
- unsigned - liczba naturalna (np. 314)
- float - liczba rzeczywista (np. 3.1415926)
- double - liczba rzeczywista z "podwójną" precyzją
- char – pojedynczy znak (np. 'A')
- bool - wartość prawda/fałsz (true/false)
- std::string - ciąg znaków (np. "Koper")
- void – funkcja niezwracająca nic

```
int          calkowita         = -314;
unsigned int calkowitaDotatnia = 123;

float  piFloat  = 3.141592653589793f;
double piDouble = 3.14159265358979323846;

char znak = 'A';

std::string ciagZnakow = "Koper";
```

Instrukcje warunkowe

C++

```cpp
if (wyrażenie_logiczne1)
{
    // Jeżeli wyrażenie logiczne 1 jest
    // prawdziwe, wykona ten blok kodu.
}
else if (wyrażenie_logiczne2)
{
    // Jeżeli wyrażenie logiczne 1 NIE
    // jest prawdziwe a wyrażenie_logiczne2
    // jest prawdziwe, wykona ten blok kodu.
}
else
{
    // Jeżeli wyrażenie logiczne 1 i
    // wyrażenie logiczne 2 NIE są
    // prawdziwe, wykona ten blok kodu.
}
```

Python

```python
if wyrażenie_logiczne1:

    // Jeżeli wyrażenie logiczne 1 jest
    // prawdziwe, wykona ten blok kodu.

elif wyrażenie_logiczne2:

    // Jeżeli wyrażenie logiczne 1 NIE
    // jest prawdziwe a wyrażenie_logiczne2
    // jest prawdziwe, wykona ten blok kodu.

else:

    // Jeżeli wyrażenie logiczne 1 i
    // wyrażenie logiczne 2 NIE są
    // prawdziwe, wykona ten blok kodu.
```

Wyrażenia logiczne

Python

- && - and
- || - or
- ^ - xor
- ! - not
- > - większe
- >= - większe lub równe
- < - mniejsze
- <= - mniejsze lub równe
- == - równa się
- != - nie równa się

```python
a = 123
b = 642

c = false

if a == b and (c != false or b == 642) and not c:
    // Prawda

else:
    // Fałsz
```

C++

```cpp
int a = 123;
int b = 642;

bool c = false;

if (a == b && (c != false || b == 642) && !c)
{
    // Prawda
}
else
{
    // Fałsz
}
```

Switch

Switch to instrukcja która zależnie od podanej wartości liczbowej przejdzie do danego "przypadku" (case)

W tym przykładzie switch przejdzie do "case 123" i zakończy swoją czynność.

Switch w porównaniu do if'ów jest zdecydowanie szybszy, gdy jest wiele możliwych ścieżek. (Wiele if'ów i else if'ów)

```
int wartoscLiczbowa = 123;

switch (wartoscLiczbowa) // Wartością liczbową może również być znak (char)
{
case 0:
    std::cout << "To zero!";
    break;
case 44:
    std::cout << "To czterdzieści cztery!";
    break;
case 123:
    std::cout << "To sto dwadzieścia trzy!";
    break;
case 'A':
    std::cout << "To A!";
    break;
default:
    std::cout << "To coś innego.";
    break;
}
```

Podstawowe instrukcje wyjścia

- cout – (console out) – wypisuje zmienną, lub stałą wartość do konsoli

```
std::cout << "Hello, world!" << std::endl; // Wypisze: Hello, world! i zrobi nową linijkę
std::cout << 623462345123 << std::endl;    // Wypisze: 623462345123! i zrobi nową linijkę

int zmienna = 123;
std::string str = "Koper";

std::cout << str;        // Wypisze: Koper
std::cout << zmienna;    // Wypisze: 123 w tej samej linijce
                         // W konsoli pojawi się: Koper123
std::cout << std::endl; // Zrobi nową linijkę

int wiek = 15;
std::string miasto = "Cieszynie";

std::cout << "Mam " << wiek << " lat.\n";     // Wypisze: Mam 15 lat. i zrobi nową linijkę
std::cout << "Mieszkam w " << miasto << '\n'; // Wypisze: Mieszkam w Cieszynie  i zrobi nową linijkę
```

Podstawowe instrukcje wejścia

- cin – (console in) – wczytuje pierwszą wartość podaną przez użytkownika w konsoli

```
int liczba = 0;           // Aplikacja czeka aż użytkownik
std::cin >> liczba;       // poda wartość i kliknie Enter
```

```
std::string tekst = "";
std::cin >> tekst;
```

cin wczytuje **tylko pierwszą** wartość z konsoli,
wszystko po spacji zostanie zignorowane.

```
std::string imie = "";
std::cout << "Jak masz na imie?\n";
std::cin >> imie;
std::cout << "Witaj " << imie << "!\n";
```

Podstawowe instrukcje wejścia

- getline – wczytuje całą linijkę z konsoli jako tekst, bez względu na spacje

```
std::cout << "Co to za miejsce?\n";

std::string linijka = "";
std::getline(std::cin,linijka);

std::cout << "Tak, to " << linijka << std::endl;
```

```
Co to za miejsce?
II Liceum Ogolnoksztalcace im. Mikolaja Kopernika w Cieszynie
Tak, to II Liceum Ogolnoksztalcace im. Mikolaja Kopernika w Cieszynie
```

Struktura programu w C++

```cpp
// Tu są zawarte pliki i biblioteki
#include <iostream> // W tym pliku (bibliotece) znajduje się m.in: cout, oraz cin
#include <string>   // Tutaj anajduje się m.in: typ string i getline

int main() // Punkt wejściowy (w Pythonie program zawsze szedł z góry na dół)
{
    // Tutaj wykonywany jest kod

    int a = 4;
    int b = 41;

    int x = 123 + a + b; // 123 + 4 + 41 = 168

    std::string imie = "Mateusz";
    std::string nazwisko = "Antkiewicz";

    std::string wynik = imie + nazwisko; // "Mateusz" + "Antkiewicz" = "MateuszAntkiewicz"
}
```

Czym jest "std::"?

std ("standard") to **namespace** podstawowo zawarty w bibliotekach C++.

namespace - "szufladka" w której zawarte są funkcje, typy, klasy, oraz inne struktury występujące w C++.

Wymaganego w wielu miejscach std można się pozbyć w ten sposób:

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    cout << "Witam!" << endl;

    int a = 0;
    cin >> a;

    string text = "ABC";
}
```

Jak sama nazwa wskazuje, C++ domyśla się kiedy chcemy użyć namespace'a "std"

Matematyka

- Podstawowe funkcje matematyczne znajdują się w pliku "math.h"

```
#include <math.h>

int main()
{
    float bok = 5.0f;

    float poleKwadratu = pow(bok,2);
    float objetoscSzescianu = pow(bok,3);

    float przekatnaKwadratu = sqrt(bok);
}
```

- pow(a, b):
  - a - podstawa potęgi
  - b - wykładnik potęgi
- sqrt(a):
  - a - liczba pierwiastkowana

Matematyka

Warto dodać że w C++, przy dzieleniu dwóch liczb całkowitych wynik zostanie zaokrąglony w dół do liczby całkowitej:

```
int i = 40;
std::cout << i / 11 << '\n';
```

```
3
```

Aby wynik był prawdziwy, jedna z liczb musi być liczbą rzeczywistą: (float lub double)

```
float f = 40.0f;
std::cout << f / 11 << '\n';
```

```
3.63636
```

Pętla "for"

```
for(wyrażenie A, wyrażenie B, wyrażenie C)
{
    Blok kodu
}
```

- Wyrażenie A – Zostaje wykonane tylko raz przed rozpoczęciem pętli
- Wyrażenie B - Pętla się wykonuje póki to wyrażenie jest prawdziwe
- Wyrażenie C – Zostaje wykonane zawsze po każdej iteracji ("kroku") pętli
- Blok kodu – Zostaje wykonane przy każdej iteracji ("kroku") pętli

```
for (int i = 0; i < 12; i++)
{
    std::cout << i << '\n'; // Wypisze cyfry od 0 do 11
}
```

Pętla "while"

```
while(wyrażenie A)
{
    Blok kodu
}
```

- Wyrażenie A – Pętla się wykonuje póki to wyrażenie jest prawdziwe
- Blok kodu – Zostaje wykonane przy każdej iteracji ("kroku") pętli

```
int a = 128;
while (a >= 1) // Warunek jest sprawdzany PRZED wykonaniem się wnętrza pętli
{
    std::cout << a << '\n'; // Wypisze 128, 64, 32, 16, 8, 4, 2, 1
    a /= 2;
}
```

Pętla "do...while"

```
do
{
    Blok kodu
} while(wyrażenie A)
```

- Wyrażenie A – Pętla się wykonuje póki to wyrażenie jest prawdziwe
- Blok kodu – Zostaje wykonane przy każdej iteracji ("kroku") pętli

```
int i = 0;
do // Zawsze wykona się co najmniej raz
{
    std::cin >> i; // Użytkownik będzie mógł wpisywać liczby dopóki nie poda wartości 0

} while (i != 0); // Warunek jest sprawdzany PO wykonaniu wnętrza pętli
```

Tablice (ang. array)

Wymagane do przykładu:

```
#include <array>
#include <iostream>

using namespace std;
```

```
// Tablica o długości 4
array<int, 4> liczbyA = { 0, -4, 51, 125125 };

// Tablica o długości 100 wypełniona 0
array<int, 100> liczbyB = { 0 };
```

```
// Edytowanie elementów w tablicy
liczbyB[0] = 123;
liczbyB[1] = 1142;
liczbyB[2] = 51;
liczbyB[60] = -7;
liczbyB[12] = 1;
```

```
// Wypisuje wszystkie elementy z tablicy
for (int i = 0; i < liczbyA.size(); i++) {
    cout << liczbyA[i] << endl;
}

// Wypisuje wszystkie elementy z tablicy
for (int cyfra : liczbyB) {
    cout << cyfra << endl;
}
```

vector (lista)

Wymagane do przykładu:

```
#include <vector>
#include <iostream>

using namespace std;
```

```
// Lista przechowująca liczby całkowite
// Początkowo jest pusta
vector<int> liczby;

// Lista przechowująca liczby całkowite
// Początkowo zawiera 4 elementy
vector<int> liczby = { 123, 4, -512, 55 };
```

```
// Dodawanie elementów do listy
liczby.push_back(123);
liczby.push_back(4);
liczby.push_back(-512);
liczby.push_back(55);
```

```
// Wypisuje wszystkie elementy z listy
for (int i = 0; i < liczby.size(); i++)
    cout << liczby[i] << endl;

// Wypisuje wszystkie elementy z listy
for (auto int liczba : liczby)
    cout << liczba << endl;
```

vector (lista)

Wymagane do przykładu:

```
#include <vector>

using namespace std;
```

.erase(iterator) - usuwa element z listy na danej pozycji

- Iterator – pozycja w liscie na danym indeksie

Wymagany jest zapis "liczby.begin() + 1" zamiast "1" bo ta funkcja przyjmuje iterator a nie indeks w postaci cyfry.

"liczby.begin() + 1" zamienia indeks 1 na iterator elementu 1.

```
int main()
{
    vector<int> liczby = {
        1, 4, 51, 1212
    };

    // Usuwanie elementów z listy
    liczby.erase( _Where: liczby.begin() + 1);
    // liczby to teraz {1, 51, 121}

    liczby.erase( _Where: liczby.begin() + 2);
    // liczby to teraz {1, 51}

    liczby.erase( _Where: liczby.begin() + 0);
    // liczby to teraz {51}
}
```

set (zbiór)

Wymagane do przykładu:

```
#include <set>

using namespace std;
```

Set (zbiór) to swego rodzaju lista która przechowuje unikalne (niepowtarzające się) elementy w kolejności rosnącej, lub alfabetycznej jeśli działamy na znakach.

To oznacza że dana wartość występuje w zbiorze co najwyżej raz.

```
// Pusty zbiór przechowujący liczby całkowite
set<int> liczby;

// Zbiór początkowo przechowujący 1, 5, 12, -2
set<int> liczby = { 1, 5, 12, -2 };


// Dodawanie elementów do zbioru
liczby.insert(_Val: 4);
liczby.insert(_Val: 4525);
liczby.insert(_Val: 123);

// Do zbioru nie da się dodać elementu który już tam występuje
liczby.insert(_Val: 123);
// W zbiorze występuje tylko jeden element o wartości 123

// Usuwanie elementu o podanej wartości (np. liczby 4) ze zbioru
liczby.erase(_Keyval: 4);
```

set (zbiór)

Wymagane do przykładu:

```
#include <set>

using namespace std;
```

Zbiór można utworzyć z listy lub tablicy podając jej początek i koniec:

```
vector<int> liczby = {
    123, 45, 55, 45, 123, 4, 5, 5, 4, 123
};

set<int> unikalne(liczby.begin(), liczby.end());
// unikalne to { 4, 5, 45, 55, 123 }
```

Ilość elementów w zbiorze może się przydać, gdy chcemy określić ile unikalnych (niepowtarzających się) elementów jest w liście lub pliku.

```
// Ilość elementów w zbiorze
int unikalneLiczby = liczby.size();

// Liczba elementów o wartości 123
// W zbiorze funkcja count zwraca tylko wartość 0 lub 1
int ilosc123 = liczby.count( _Keyval: 123);
```

map (słownik)

Wymagane do przykładu:

```cpp
#include <map>
#include <string>

using namespace std;
```

Mapa (słownik) to swego rodzaju lista którą indeksuje się dowolnym typem i przechowuje unikalne (niepowtarzające się) elementy w kolejności rosnącej, lub alfabetycznej jeśli działamy na znakach. Dla kluczów w postaci liczby indeksy nie muszą być ciągłe. (1, 3, 6, 13)

```cpp
// Mapa gdzie klucz to string a wartość to int
map<string, int> mieszkancy;

// Tworzenie nowych elementów w mapie
mieszkancy["Krakow"] = 780000;
mieszkancy["Warszawa"] = 1794000;
mieszkancy["Cieszyn"] = 34000;

// Jeżeli podany klucz istnieje w mapie to edytuje sie jego wartość
// Jeżeli podany klucz nie istnieje to tworzy się nowy element
mieszkancy["Warszawa"] = 2000000;

// Wartości przechowywane w mapie można edytować jak normalne zmienne
mieszkancy["Cieszyn"] += 120;
mieszkancy["Krakow"] -= 1400;

// Usuwanie klucza i jego wartości z mapy
mieszkancy.erase( _Keyval: "Warszawa");
```

map (słownik)

Wymagane do przykładu:

```
#include <map>
#include <string>

using namespace std;
```

```
// Ilość kluczy w mapie
int miasta = mieszkancy.size();
```

```
// Liczba kluczy "Cieszyn"
// W mapie funkcja count zwraca tylko wartość 0 lub 1
int x = mieszkancy.count( _Keyval: "Cieszyn");
```

Ilość elementów w mapie może się przydać, gdy chcemy określić ile unikalnych (niepowtarzających się) elementów jest w liście lub pliku.

Iterowanie przez mapy i zbiory

```cpp
set<string> imiona = {
    "Wojtek",  "Michal",
    "Mariusz", "Tomasz",
    "Adrian",  "Cezary",
};

for (string imie : imiona) {
    cout << imie << '\n';
}
```

```
Adrian
Cezary
Mariusz
Michal
Tomasz
Wojtek
```

```cpp
map<string, int> mieszkancy;
mieszkancy["Krakow"] = 780000;
mieszkancy["Warszawa"] = 1794000;
mieszkancy["Cieszyn"] = 34000;
mieszkancy["Bielsko Biala"] = 166765;
mieszkancy["Gdansk"] = 470907;

for (auto [nazwa, ludnosc] : mieszkancy) {
    cout << nazwa << " ma " << ludnosc << " mieszkancow.\n";
}
```

```
Bielsko Biala ma 166765 mieszkancow.
Cieszyn ma 34000 mieszkancow.
Gdansk ma 470907 mieszkancow.
Krakow ma 780000 mieszkancow.
Warszawa ma 1794000 mieszkancow.
```

Przez mapy można iterować typem: auto [klucz, wartość]

"auto" automatycznie domyśli się typów klucza i wartości w mapie.

Operacje na listach i tablicach

Wymagane do przykładu:

```
#include <vector>
#include <array>
#include <algorithm>

using namespace std;
```

```
// Tablica przechowująca 7 liczb całkowitch
array<int, 7> liczby = { 123, 4, -512, 55, 0, 4142, 5812985 };

// Lista przechowująca takie same liczby jak tablica powyżej
vector<int> liczby = { 123, 4, -512, 55, 0, 4142, 5812985 };
```

Poniższe funkcje działają identycznie dla tablic i list:

```
// Posortuje listę / tablicę
sort( _First: liczby.begin(), _Last: liczby.end());
// liczby to teraz {-512, 0, 4, 55, 123, 4142, 58129895 }


// Odwróci listę / tablicę
reverse( _First: liczby.begin(), _Last: liczby.end());
// liczby to teraz { 58129895, 4142, 123, 55, 4, 0, -512 }
```

Operacje na listach i tablicach

find(początek, koniec, szukana) - znajduje szukaną wartość i zwraca jej iterator w podanym zbiorze

- początek – iterator początkowy zbioru
- koniec – iterator końcowy zbioru
- szukana – szukana wartość

Aby ze zwróconego iteratora zrobić indeks w postaci liczby całkowitej należy odjąć od niego iterator początkowy zbioru.

Można to opisać jako znajdywanie różnicy między początkowym iteratorem a zwróconym iteratorem.

Jeżeli funkcja find nie znajdzie szukanej wartości, zwróci podany iterator końcowy zbioru.

```
int main()
{
    vector<string> imiona = {
        "Mateusz", "Mikolaj",
        "Borys", "Andrzej",
        "Slawek", "Julia"
    };

    // pozycjaBorys = 2
    int pozycjaBorys = find(
        imiona.begin(),
        imiona.end(),
        "Borys"
    ) - imiona.begin();

    // pozycjaIstnieje = false
    bool pozycjaIstnieje = find(
        imiona.begin(),
        imiona.end(),
        "Mariusz"
    ) != imiona.end();
}
```

Operacje na stringach

Wymagane do przykładu:

```
#include <string>

using namespace std;
```

Do istniejącego stringa można dodać tekst używając operatora +=

Z istniejącego stringa można usunąć znak pod danym indeksem używając funkcji erase()

```
int main()
{
    string tekst = "";
    tekst += "Lorem ipsum";
    tekst += " ";
    tekst += "dolor sit amet";

    // dlugoscTekstu = 26
    int dlugoscTekstu = tekst.size();

    tekst.erase(1); // usunie drugi znak 'o'
    tekst.erase(3); // usunie (obecnie) czwarty znak 'm'
}
```

Tekst można konwertować na liczbę na 2 sposoby:

stoi(tekst) - (string to int) - zamienia tekst na liczbę całkowitą

- tekst – string zawierający liczbę całkowitą

stof(tekst) - (string to float) - zamienia tekst na liczbę rzeczywistą

- tekst – string zawierający liczbę rzeczywistą

```
int main()
{
    string liczbaCalkowitaA = "123454";
    string liczbaRzecywistaB = "3.1415926";

    // A = 123454
    int A = stoi(liczbaCalkowitaA);

    // B = 3.1415926
    float B = stof(liczbaRzecywistaB);
}
```

Operacje na stringach

.substr(początek, długość) - zwraca fragment stringa
- początek – indeks pierwszej litery fragmentu
- długość - ilość liter danego fragmentu

.find(szukana) - zwraca indeks szukanego znaku
- szukana – szukany znak w stringu

.find(szukana, początek) - zwraca indeks szukanego znaku który wystąpił po pewnym indeksie
- szukana – szukany znak w stringu
- początek - indeks od którego funkcja szuka wartości

Funkcja .find() ma kilka wariantów lecz ich nazwy są logiczne i mówią same za siebie.

```cpp
int main()
{
    string tekst = "Lorem ipsum dolor sit amet"; // ...

    // tekst 0 => 0+11 to "Lorem ipsum"
    string loremIpsum = tekst.substr(0, 11);

    // tekst 12 => 12+0 to "dolor sit"
    string dolorSit = tekst.substr(12, 9);
}
```

```cpp
int main()
{
    string tekst = "Lorem ipsum dolor sit amet"; // ...

    if (tekst.find("sit") != string::npos)
    {
        cout << "W tekscie jest slowo \"sit\"!" << endl;
    }

    // pozycjaDolor = 12
    int pozycjaDolor = tekst.find("dolor");
    int a = tekst.find_first_not_of('a');
    int e = tekst.find_first_of('e', 8);
    int i = tekst.find_last_not_of('i', 5);
    int r = tekst.find_last_of('r');
}
```

Pliki

```
#include <string>
#include <iostream>
#include <fstream>
```

Odczyt słów z pliku

```
int main()
{
    fstream plik("imiona.txt");

    // Wypisuje wszystkie słowa z pliku

    string slowo;

    // Iteruje przez słowa dopóki może
    // (dopóki plik >> slowo jest poprawną operacją)
    while (plik >> slowo)
    {
        cout << slowo;
        // Można też dodawać te słowo np. do listy
    }

    plik.close();
}
```

Taka sama zasada dla cyfr

```
int main()
{
    fstream plik("dane.txt");

    vector<int> liczby;

    int liczba;
    while (plik >> liczba)
    {
        liczby.push_back(_Val: liczba);
    }

    plik.close();
}
```

Zapis słów do pliku (Robi to operator <<)

```
int main()
{
    fstream plik("imiona.txt");

    plik << "Hello World! - To linijka 1\n";
    plik << "Linijka 2.\n";
    plik << "Linijka 3.\n";

    plik.close();
}
```

```
1   Hello World! - To linijka 1
2   Linijka 2.
3   Linijka 3.
```

Funkcje

Funkcje w C++ deklaruje się w następujący sposób:

```
typ nazwa(argumenty, itd..)
```

```
void PrzywitajSie(std::string imie)
{
    std::cout << "Witaj " << imie << "!";
}
```

Funkcje mogą zwracać wartości:

```
float PoleTrojkata(float a, float h)
{
    return a * h / 2.0f;
}
```

Każdy program w C++ musi zawierać funkcję int main().

```
int main()
{
    PrzywitajSie("Mateusz");

    // 12 * 12 = 144
    int kwadrat = PoleKwadratu(12);

    // 3.0 * 7.0 / 2.0f = 10.5f
    float trojkat = PoleTrojkata(3.0f,3.0f);
}
```

C++ automatycznie zwraca 0 na końcu tej funkcji, lecz można samemu zwrócić (inną) wartość wcześniej.

Include

#include <nazwa_pliku>
Lub
#include "nazwa_pliku"

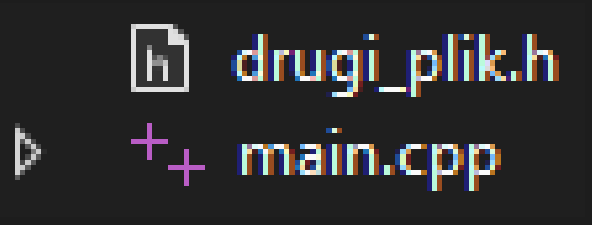
#include importuje bibliotekę lub plik.
Można to porównać do instrukcji "import" w Pythonie.

Importowanie bibliotek z C++

```
#include <vector>
#include <array>
#include <algorithm>
#include <string>
#include <iostream>
#include <thread>
```

Importowanie innych plików utworzonych przez użytkownika

```
#include "drugi_plik.h"
```



Dziękuję za uwagę

- Mateusz Antkiewicz