Wprowadzenie do programowania w C++

https://oij.edu.pl/zawodnik/kursy/przewodnik-cpp
wersja 2.1 z dnia 15 października 2023

Olimpiada Informatyczna Juniorów

O tym przewodniku

Olimpiada Informatyczna Juniorów (OIJ) jest wyjątkowym konkursem, który wymaga od uczestników nauki języka programowania. Mamy nadzieję, że wiedza zdobyta w trakcie Olimpiady otworzy przed zawodnikami drzwi do cennej umiejętności, którą z powodzeniem wykorzystają w przyszłości.

Aby pomóc zarówno przyszłym zawodnikom, jak i ich opiekunom, udostępniamy ten przewodnik, który krok po kroku omawia podstawowe konstrukcje języka, które potrzebne są do startu w Olimpiadzie.

Uczestnicy OIJ mogą wysyłać swoje rozwiązania w jednym z dwóch języków programowania: C++ lub Python. Przewodnik ten skupia się na na tym pierwszym. Choć C++ może na początek wydać się trudniejszy, jest on preferowany do używania w konkursach programistycznych, ze względu na szybkość i wydajność. O ile OIJ stara się nie karać zawodników wybierających język Python, to w innych konkursach (w tym Europejskiej Olimpiadzie Informatycznej Juniorów, czy Olimpiadzie Informatycznej dla szkół ponadpodstawowych), wybór mniej wydajnego języka może przekładać się na niższą punktację.

Autor tego przewodnika dołożył wszelkich starań, aby wszystkie koncepty i konstrukcje były wyczerpująco opisane. Jeżeli jednak podczas czytania tego poradnika, znajdziesz jakiś błąd, albo będziesz miał(a) jakieś sugestie jak opisać coś lepiej, lub po prostu coś będzie dla Ciebie niezrozumiałe, autor z chęcią usłyszy Twoje uwagi. Wystarczy że napiszesz maila na adres kostka@oij.edu.pl.

Przewodnik ten będzie w przyszłości ewoluował. Istnieje kilka sekcji, które powinno zostać do niego dopisane, a istniejące sekcje mogą też zostać rozbudowane. Najnowsza wersja będzie zawsze dostępna na stronie `https://oij.edu.pl/zawodnik/kursy/przewodnik-cpp`.

Zmienne

W programowaniu, *zmienna* to miejsce w pamięci komputera, gdzie przechowywane są wartości (np. liczby całkowite). Możemy sobie wyobrazić, że jest to pudełko, w którym komputer będzie przechowywał pewne dane. Oczywiście możemy zawsze otworzyć to pudełko, zobaczyć co w nim jest, albo po pewnym czasie zdecydować, że w tym pudełku powinna znajdować się wartość. W C++, każda zmienna ma określony *typ*, który informuje, jakiego rodzaju dane można w niej przechowywać. Innymi słowy, mamy różne pudełka na różne typy informacji.

Podstawowe typy

Na początku skupimy się na czterech podstawowych typach.

- `int` - służy do przechowywania liczb całkowitych. Na przykład: 5, -3, 42.
- `double` - do przechowywania liczb rzeczywistych (czyli liczb, które mają część ułamkową (po przecinku)). W programowaniu nazywamy je liczbami zmiennoprzecinkowymi i oddzielamy znakiem kropki (`.`). Na przykład: 5.67, -0.01. Nie możemy używać przecinka.
- `char` - do przechowywania pojedynczych znaków. Na przykład: `'A'`, `'z'`, `'1'`, a także `'@'` oraz `'$'`. Pojedyncze znaki są oznaczane w kodzie apostrofem (pojedynczym cudzysłowem): `'`.
- `bool` - do przechowywania wartości logicznych. Jest to zawsze jedna z dwóch wartości: prawda lub fałsz. W języku C++ używamy angielskich słów `true` (prawda) oraz `false` (fałsz).

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".

Deklaracja i nazewnictwo zmiennych

Aby zadeklarować zmienną (stworzyć nowe pudełko) używamy następującej składni:

```
int moja_liczba = 10;
```

Przyjrzyjmy się kolejnym elementom tej deklaracji:

Od lewej do prawej strony mamy:

- typ zmiennej – oznacza typ pudełka, np. `int` lub `char`,
- nazwę zmiennej – jest to nazwę pudełka, która powinna wyjaśniać po co deklarujemy tą zmienną, np. `suma`, `wynik`,
- znak równości – oznacza tutaj przypisanie wartości,
- wartość tej zmiennej – jest to wartość, którą umieszczamy w pudełku. Zwróć uwagę, że typ zmiennej musi być zgodny z zadeklarowanym typem,
- znak średnika (każda instrukcja w C++ kończy się średnikiem).

Istnieją pewne reguły dotyczące nazewnictwa zmiennych:

- Nazwy zmiennych muszą zaczynać się od litery (a-z, A-Z) lub podkreślenia (_). Nie mogą zaczynać się od cyfry.
- Nazwy mogą zawierać litery, cyfry oraz podkreślenia.
- Nazwy zmiennych są czułe na wielkość liter, tzn. `zmienna`, `Zmienna` i `ZMIENNA` to trzy różne nazwy.
- Nie można używać słów kluczowych zarezerwowanych w C++ jako nazw zmiennych, np. `int`, `return`.

Deklaracje zmiennych mogą być bardziej skomplikowane, w szczególności mogą korzystać ze wcześniej zadeklarowanych zmiennych np.:

```
int a = 2 + 2; // Od tego momentu a jest równe 4.
int b = -a;    // Od tego momentu b jest równe -4 (bo a jest równe 4).
int c = a * b; // Od tego momentu c jest równe -16 (bo a jest równe 4, a b = -4).
```

Możemy także stworzyć nową zmienną i nie przypisać do niej żadnej wartości:

```
int nowa_zmienna;
```

Nie możemy jednak nic nie powiedzieć na temat wartości tej zmiennej (jest to tzw. niezadeklarowana zmienna). Wbrew oczekiwaniom, nie musi ona być równa 0, ani żadnej innej stałej wartości. Możemy przypisać jej wartość później:

```
nowa_zmienna = 42;
```

Zauważ, że wtedy nie musimy powtarzać typu zmiennej, jako że już wcześniej zadeklarowaliśmy jakiego typu jest ta zmienna.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

Operacje arytmetyczne

Komputer jest oczywiście dużym kalkulatorem, dlatego może wykonywać proste operacje arytmetyczne.

1. Dodawanie (operator +) - dodaje dwie liczby.

```
int suma = 3 + 4;  // Od tego momentu wartość zmiennej suma to 7.
```

2. Odejmowanie (operator -) - odejmuje jedną liczbę od drugiej.

```
int roznica = 7 - 3;  // Od tego momentu wartość zmiennej różnica to 4.
```

3. Mnożenie (operator *) - mnoży dwie liczby.

```
int iloczyn1 = 4 * (-3);      // Od tego momentu wartość zmiennej iloczyn1 to -12.
double iloczyn2 = 0.5 * 9.0;  // Od tego momentu wartość zmiennej iloczyn2 to 4.5.
```

4. Dzielenie (operator /) - dzieli jedną liczbę przez drugą. Dzielenie przez zero zwraca błąd. Zwróć uwagę, że jeżeli podzielimy dwie liczby całkowite (typu `int`), wynikiem będzie także liczba całkowita. Jeżeli iloraz miałby część ułamkową, to jest ona ignorowana. Nazywamy to "dzieleniem bez reszty".

```
int iloraz1 = 7 / 2;  // Iloraz1 to teraz 3, bo jest to dzielenie bez reszty.
double iloraz2 = 7.0 / 2.0;  // Iloraz2 jest równy 3.5.
```

5. Reszta z dzielenia (operator %) - zwraca resztę z dzielenia jednej liczby przez drugą. Analogicznie próba policzenia reszty dzielenia przez zero zwraca błąd.

```
int reszta = 7 % 3;  // Reszta to 1, bo 7 podzielone przez 3 daje 2 reszta 1.
```

Dodatkowo, w C++ możemy korzystać z nawiasów, by kontrolować kolejność wykonywania operacji arytmetycznych.

```
int wynik1 = 2 + 3 * 4;    // Wynik to 16, bo najpierw wykonujemy operacje mnożenia,
                           // czyli 2 + (3 * 4).
int wynik2 = (2 + 3) * 4;  // Wynik to 5 * 4 = 20.
int wynik3 = 6 / 2 * 3;    // Wynik to 9, bo pierwszeństwo ma operacja dzielenia z lewej
                           // strony czyli (6 / 2) * 3.
int wynik4 = 6 / (2 * 3);  // Wynik to 6 / 6 = 1.
```

Budowa programów

Podstawowy program w C++ składa się z funkcji[1] `main()`, gdzie program rozpoczyna swoje działanie. Przykładowa struktura programu wygląda następująco:

```
#include<iostream>   // Oznacza, że używamy biblioteki umożliwiającej operacje
                     // wejścia/wyjścia.
using namespace std; // Oznacza, że będziemy używać komend ze standardowej biblioteki.

int main() {
  // Kod Twojego programu.
  int a = 2;
  int b = a + a;
```

[1] O funkcjach dowiesz się więcej pod koniec tego poradnika.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

```
  return 0; // Zwraca wartość 0, co oznacza, że program zakończył się poprawnie,
            // bez błędu.
}
```

Zwróć uwagę, że każda instrukcja jest w nowej linii i kończy się średnikiem. Instrukcje wykonywane są od góry do dołu, po kolei. Nie możemy zatem użyć zmiennej, która zostanie zadeklarowana później (poniżej).

Dodatkowo w programie możemy używać komentarzy, które rozpoczynami dwoma znakami `//`. Wszystko po tych znakach nie jest już brane pod uwagę przez komputer. Komentarze służą do dodawania czytelności naszego kodu i powinniśmy ich używać, szczególnie w miejscach, gdzie kod może być nie do końca jasny.

Wczytywanie i wypisywanie

Wypisywanie

Wypisywanie zmiennych odbywa się za pomocą `cout` (czytane jako "si aut"), który jest skrótem od angielskiego zwrotu *character output*. Aby wypisać wartość zmiennej, musimy ją poprzedzić operatorem <<:

```
  int liczba = 5;
  cout << liczba << "\n";
```

Tutaj `"\n"` oznacza napis oznaczający "enter" (a dokładniej: koniec wiersza) i jest używane do przejścia do nowego wiersza.

Możemy także wypisywać napisy. Napisy muszą znajdować się w cudzysłowie, np. `"banan"`. Żeby wypisać ten napis, używamy następującej instrukcji:

```
  cout << "banan" << "\n";
```

Możemy wypisać kilka liczb / zmiennych / napisów jednocześnie. Wszystkie napisy powinny zawsze znajdować się w cudzysłowie. Zauważ, że `"\n"` jest w istocie napisem.

Poniższy program:

```
  int dwa = 2;
  cout << dwa << "+" << dwa << "\n";
```

wypisze 2+2 (i znak nowego wiersza). Zwróć uwagę, że każdy wypisywany fragment jest ze sobą sklejony. Jeżeli chcemy je rozdzielić, możemy wypisać napis zawierający pojedynczą spację.

Na przykład następujący program:

```
  int trzy = 3;
  cout << trzy << trzy << "\n"
  cout << trzy << " " << trzy << "\n";
  cout << trzy << "\n" << trzy << "\n";
```

wypisze

```
33
3 3
3
3
```

Wczytywanie zmiennych

Wczytywanie zmiennych odbywa się za pomocą `cin` (czytane jako "si yn"), który jest skrótem od *character input*. Aby wczytać wartość do zmiennej, używamy operatora >>:

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

```
int moja_liczba;
cin >> moja_liczba;
```

Gdy użytkownik uruchamia ten program, program czeka aż wpisze on jakąś wartość i naciska Enter, wartość ta jest przypisywana do zmiennej `moja_liczba`. Zwróć uwagę, że wartość ta powinna być liczbą całkowitą.

Rozważmy przykładowy program, który wczytuje dwie liczby, sumuje je, a następnie wypisuje wynik. Zwróć uwagę, że możemy w jednej instrukcji zadeklarować dwie zmienne lub wczytać dwie zmienne.

```
#include <iostream>

using namespace std;

int main() {

  int liczba1, liczba2;

  cin >> liczba1 >> liczba2;

  int suma = liczba1 + liczba2;

  cout << suma << "\n";

  return 0;

}
```

Instrukcje warunkowe

Często w programie chcemy aby część instrukcji była wykonywana tylko w niektórych przypadkach, np. może chcemy uruchomić program, tylko kiedy nasz użytkownik zna tajne hasło, albo chcemy uruchomić różne narzędzia w zależności od tego jakie wejście otrzymamy od użytkownika. Poniżej przedstawiamy instrukcje warunkowe, które są podstawowymi narzędziami kontrolnymi w językach programowania.

Instrukcja `if`

Instrukcje warunkowe pozwalają wykonywać różne fragmenty kodu w zależności od spełnienia określonego warunku. Instrukcja `if` pozwala na wykonanie pewnego fragmentu kodu tylko wtedy, gdy dany warunek jest spełniony.

Instrukcja `if` ma następującą składnię:

```
if (warunek) {
  // Kod do wykonania, jeśli warunek jest prawdziwy
}
```

Przykład:

```
int moj_wiek = 16;
if (moj_wiek < 18) {
  cout << "Nie jesteś pelnoletni(a)." << "\n";
  cout << "Spróbuj za " << 18-moj_wiek << " lat. " << "\n";
}
```

Zwróć uwagę, że warunek musi być otoczony nawiasami okrągłymi – `( )` . Kod do wykonania w przypadku gdy warunek jest spełniony jest otoczony nawiasami klamrowymi – `{ }` . Po instrukcji `if` nie piszemy średnika.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

Nawiasy klamrowe tworzą w programach „zakres". Jeżeli zadeklarujemy zmienną w zakresie instrukcji `if`, to możemy jej używać tylko do odpowiadającemu jej zamykającemu nawiasowi klamrowemu.

```
int a = 2;
if (a == 2) {
  int b = 4;
  // Jesteśmy w zakresie if'a, i możemy używać zarówno a i b.
  cout << a << " " << b << "\n";
}
// Tutaj już nie możemy użyć b.
// Natomiast dalej możemy używać a.
```

Operatory porównywania

Operatory porównywania służą do porównywania dwóch wartości. Zwracają one wartość logiczną typu `bool` (`true` lub `false`).

- == - równość: zwraca `true` (prawda), jeśli dwie wartości są równe.
- != - nierówność: zwraca `true`, jeśli dwie wartości są różne.
- > - większość: zwraca `true`, jeśli lewa wartość jest większa od prawej.
- < - mniejszość: zwraca `true`, jeśli lewa wartość jest mniejsza od prawej.
- >= - większość lub równość.
- <= - mniejszość lub równość.

Porównywać możemy zarówno liczby, zmienne, jak i proste wyrażenia. Jako ćwiczenie, rozważmy poniższy program. Zastanów się, jakie liczby on wypisze.

```
int siodemka = 7;
int osemka = 8;
char znak_k = 'k';
if (siodemka == osemka) {
  cout << 1 << "\n";
}
if (siodemka != 9) {
  cout << 2 << "\n";
}
if (siodemka < osemka) {
  cout << 3 << "\n";
}
if ('b' > znak_k) {
  cout << 4 << "\n";
}
if (osemka <= osemka * 2) {
  cout << 5 << "\n";
}
if ('C' >= 'B') {
  cout << 6 << "\n";
}
```

Wypisze on w kolejnych wierszach $2, 3, 5, 6$. Zauważ, że możemy porównywać także znaki. Powinniśmy także porównywać tylko znaki podobnego typu (tj. wielkie litery do wielkich liter, liczby do liczb, itd.).[2]

[2]Tak naprawdę wszystkie znaki mają swoją kolejność, która jest ustalona przez kodowanie ASCII.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".

Operatory logiczne

Operatory logiczne mogą służyć do tworzenia bardziej skomplikowanych warunków poprzez łączenie ich.

- `&&` - operator "i" (po angielsku *and*): zwraca `true`, jeśli oba warunki są prawdziwe.
- `||` - operator "lub" (po angielsku *or*): zwraca `true`, jeśli przynajmniej jeden z warunków jest prawdziwy.
- `!` - operator "nie" (po angielsku *not*): odwraca wartość logiczną (z prawdy (`true`) czyni fałsz (`false`) i odwrotnie).

Używamy je w następujący sposób:

```
int temperatura = 20;
bool slonce_swieci = true;
if ((temperatura > 15) && slonce_swieci) {
  cout << "Świetna pogoda na spacer!" << "\n";
}
```

Tutaj łączymy dwa warunki `temperatura > 15` oraz `slonce_swieci`. Jako że użyliśmy operatora `&&` (czyli "i"), to napis `"Świetna pogoda na spacer!"` zostanie wypisany tylko wtedy, gdy oba te warunki są spełnione.

W kolejnym przykładzie sprawdzimy czy litera wprowadzona przez użytkownika jest wielką literą alfabetu:

```
char znak;
cin >> znak;
if (('A' <= znak) && (znak <= 'Z')) {
  cout << "Znak jest wielką literą.";
}
if (!(('A' <= znak) && (znak <= 'Z'))) {
  cout << "Znak nie jest wielką literą.";
}
```

Instrukcje `else` i `else if`

Zauważyłaś/eś pewnie, że w ostatnim przykładzie, druga instrukcja `if` była zaprzeczeniem pierwszej instrukcji. Okazuje się dość często potrzeba nam zdefiniować co się dzieje, w przypadku kiedy warunek w instrukcji `if` nie jest prawdziwy. Mamy do tego instrukcję `else` (z angielskiego "w przeciwnym wypadku") - pozwala na wykonanie kodu, jeśli warunek w instrukcji `if` nie został spełniony. Zwróć uwagę, że instrukcja `else` zawsze musi występować **bezpośrednio** po instrukcji `if` (a konkretniej, po nawiasie klamrowym zamykającym ciało tej instrukcji).

Poniżej zobaczymy jak tej instrukcji się używa, sprawdzając czy liczba z wejścia jest parzysta czy nie:

```
int liczba;
cin >> liczba;
if (liczba % 2 == 0) {  // Sprawdzamy resztę z dzielenia przez 2.
  cout << "Liczba jest parzysta." << "\n";
} else {
  cout << "Liczba jest nieparzysta." << "\n";
}
```

Możemy także łączyć instrukcje `if` i `else`:

```
char znak = "@";
if ('A' <= znak && znak <= 'Z') {
  cout << "Znak jest wielką literą." << "\n";
} else if ('a' <= znak && znak <= 'z') {
  cout << "Znak jest małą literą." << "\n";
} else {
```


Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

```
  cout << "Znak nie jest literą." << "\n";
}
```

Tablice

Bardzo często potrzebujemy kilka lub więcej zmiennych tego samego typu. Zamiast pisać `zmienna_1`, `zmienna_2`, `zmienna_3`, itd., istnieje możliwość zdefiniowania wszystkich zmiennych „na raz". Używamy do tego *tablic*. Tablica to struktura danych, która pozwala przechowywać wiele wartości jednego typu w jednej zmiennej.

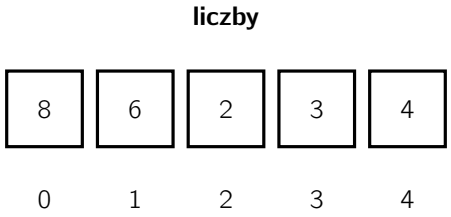
Deklaracja tablicy zaczyna się od określenia typu danych, które będą przechowywane w tablicy (np. `int`, `double`, `char`). Następnie podajemy nazwę tablicy i w nawiasach kwadratowych określamy liczbę elementów, które tablica może przechowywać.

```
int liczby[5];  // Deklarujemy tablicę pięciu liczb całkowitych.
```

Tablicę można również zainicjować podczas deklaracji, przypisując jej konkretne wartości:

```
int liczby[5] = {8, 6, 2, 3, 4};  // Inicjalizacja tablicy wartościami.
```

Aby uzyskać dostęp do konkretnego elementu tablicy, używamy indeksu elementu w nawiasach kwadratowych, przy czym należy pamiętać że indeksujemy od zera, to znaczy, że pierwszy element znajduje się pod indeksem 0, drugi element pod indeksem 1, itd.




```
int liczby[5] = {8, 6, 2, 3, 4};
int a = liczby[0];  // a = 8
int b = liczby[1];  // b = 6
int c = liczby[4];  // c = 4
```

Nie możemy próbować uzyskać dostępu do wartości które nie istnieją (np. `liczby[5]` (bo elementy są indeksowane od 0 do 4), `liczby[7]`, czy `liczby[-2]`).

Napisy

W C++ napisy możemy przechowywać w tablicach znaków (`char`). Standardowa biblioteka C++ dostarcza jednak strukturę danych typu `string`, który ułatwia pracę z napisami.

Aby jej użyć, dołączamy bibliotekę `string` (tą instrukcję umieszczamy na początku kodu):

```
#include <string>
```

Po czym możemy używać tego nowego typu jak pozostałych zmiennych:

```
string napis = "Witaj w OIJ!";
```

Podobnie jak w tablicach, możemy uzyskać dostęp do konkretnego znaku w napisie, używając indeksu:

```
char pierwsza_litera = napis[0];  // pierwsza litera to 'W'
```

Aby sprawdzić długość napisu, możemy użyć funkcji `size` korzystając z następującej składni:

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".

```
int dlugosc_napisu = napis.size();
```

Poniżej pokazujemy cały przykładowy program, który wczytuje napis i wypisuje drugi znak tego napisu, o ile ten znak istnieje.

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

using namespace std;

int main() {
  string napis;
  cin >> napis;
  if (napis.size() >= 2) {
    cout << napis[1] << "\n";
  }

  return 0;
}
```

Pętle

Bardzo często pewne części kodu chcemy wykonywać wielokrotnie. Jeżeli umiemy już tworzyć tablicę na 10 zmiennych, to nie chcemy pisać 10 instrukcji, aby wczytać wartości do tej tablicy. Aby zrobić coś wielokrotnie, używamy do tego *pętli*.

Pętla `for`

Pętla `for` jest głównie używana, gdy znamy dokładną liczbę iteracji (ile razy chcemy powtórzyć wykonywanie fragmentu kodu). Składnia pętli `for` składa się z trzech części: inicjalizacji, warunku i inkrementacji/dekrementacji.

```
for (inicjalizacja; warunek; inkrementacja/dekrementacja) {
  // ciało pętli
}
```

- **Inicjalizacja** - początkowe ustawienie zmiennej sterującej pętlą.
- **Warunek** - jeśli warunek jest prawdziwy, pętla będzie kontynuowana.
- **Inkrementacja/Dekrementacja** - modyfikuje zmienną sterującą po każdej iteracji pętli.

Zwróć uwagę, że podobnie jak w instrukcji `if`, po pętli `for` nie stawiamy średnika.
Na przykład możemy mieć następującą pętle:

```
for (int i = 0; i < 5; i++) {
  cout << i << "\n";
}
```

W powyższym przykładzie zmienna `i` jest inicjalizowana wartością 0, pętla będzie kontynuowana, dopóki `i` jest mniejsze niż 5, a `i` jest zwiększane o 1 po każdej iteracji.

Zwróć uwagę na wyrażenie `i++`, która jest równoważna wyrażeniu `i = i + 1`. Możesz zwrócić uwagę, że nazwa języka (C++) nie jest przypadkowa. Analogicznie definiujemy `i-`.

Większość pętli ma dokładnie taką strukturę i jedyne co zmieniamy, to liczbę powtórzeń kodu w nawiasie klamrowym. Powyższa pętla sprawia, że instrukcja `cout` jest wykonywana pięciokrotnie. Jeżeli zmienimy liczbę 5 na dowolną inną, powiedzmy N, to pętla ta wykona się N razy.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

Poniższy program realizuje wczytanie liczbę elementów tablicy, następnie wczytywania wartości do tablicy, zsumowania wartości liczb z tej tablicy i wypisania tej sumy:

```
#include <iostream>

int main() {
  int liczba_elementow;
  cin >> liczba_elementow;
  int liczby[liczba_elementow];
  for (int i = 0; i < liczba_elementow; i++) {
    cin >> liczby[i];
  }
  int suma = 0;
  for (int i = 0; i < liczba_elementow; i++) {
    suma = suma + liczby[i];
  }
  cout << suma << "\n";

  return 0;
}
```

Możemy sobie wyobrazić także inne, bardziej skomplikowane pętle. Na przykład poniższa pętla inicjalizuje zmienną `i` na `napis.size() - 1` (czyli indeks ostatniego znaku w napisie), i powtarza się dopóki `i >= 0` (czyli do indeksu pierwszego znaku w napisie), za każdym razem zmniejszając `i` o 2.

```
string napis = "OIJ jest super!";
for (int i = napis.size() - 1; i >= 0; i = i - 2) {
  cout << napis[i];
}
cout << "\n";
```

Program ten wypisze zatem co drugi znak od tyłu, czyli: `!eu sjJO`.

Pętla `while`

Pętla `while` jest prostszą funkcją, która jest używana, gdy nie znamy dokładnej liczby iteracji, ale znamy warunek zakończenia. Składnia jest następująca:

```
while (warunek) {
  // ciało pętli
}
```

Jeśli warunek jest prawdziwy na początku lub w trakcie wykonywania pętli, ciało pętli zostanie wykonane. Po każdej iteracji warunek jest ponownie sprawdzany. Jeśli w pewnym momencie warunek stanie się fałszywy, pętla zostanie przerwana.

```
int i = 0;
while (i < 5) {
  cout << i << "\n";
  i++;
}
```

W powyższym przykładzie, dopóki zmienna `i` jest mniejsza niż 5, instrukcje wewnątrz pętli będą wykonywane. Zwróć uwagę, że ta pętla jest równoważna wcześniej już widzianej pętli `for`.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

Tablice wielowymiarowe

Tablice dwuwymiarowe są tablicami tablic. Pozwalają one przechowywać dane w formie tabeli, mając wskazane wiersze i kolumny.

Deklaracja tablicy dwuwymiarowej jest podobna do deklaracji jednowymiarowej, ale podajemy dwa wymiary:

```
typ nazwa_tablicy[liczba_wierszy][liczba_kolumn];
```

Przykład:

```
int tablica[3][4];  // Tablica o 3 wierszach i 4 kolumnach.
```

Można także od razu zainicjalizować tablicę dwuwymiarową:

```
int tablica[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

Aby dostać się do elementu takiej tablicy, bądź przypisać do niego wartość, każdy wymiar podajemy w osobnych nawiasach kwadratowych:

```
int tablica[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
cout << tablica[0][1] << "\n"; // Wypisze 2.
cout << tablica[1][2] << "\n"; // Wypisze 6.
tablica[1][1] = 55;           // Zamieni wartość 5 na 55.
```

Oprócz tablic dwuwymiarowych, C++ umożliwia deklarowanie tablic o większej liczbie wymiarów. Poniżej pokazujemy jak zadeklarować tablicę o trzech wymiarach:

```
int kostka[3][3][3];  // Tablica o 3 wymiarach, każdy o długości 3.
```

Zagnieżdżone pętle

Aby przejść przez każdy element tablicy dwuwymiarowej, często używa się dwóch zagnieżdżonych pętli – jednej pętli dla wierszy i jednej dla kolumn. Zwróć uwagę, że używamy innych zmiennych dla zagnieżdżonych pętli – nie możemy użyć zmiennej `i` do obu pętli, jako że zmienna ta już jest zadeklarowana w pierwszej (zewnętrznej) pętli, nie możemy zatem jej użyć ponownie w drugiej (wewnętrznej) pętli. Dlatego używamy zmiennej `j`.

```
for (int i = 0; i < 2; i++) {
  for (int j = 0; j < 3; j++) {
    cout << macierz[i][j] << " ";
  }
  cout << "\n";
}
```

W powyższym przykładzie zewnętrzna pętla przechodzi przez wiersze, a wewnętrzna pętla przechodzi przez kolumny, umożliwiając dostęp do każdego elementu macierzy.

Zagnieżdżone pętle nie są ograniczone tylko do tablic dwuwymiarowych. Mogą być używane w wielu innych przypadkach, kiedy potrzebujemy zagnieżdżonej iteracji.

Załóżmy, że chcemy wydrukować wszystkie możliwe kombinacje trzech kolorów: czerwonego, zielonego i niebieskiego:

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

```
string kolory[3] = {"czerwony", "zielony", "niebieski"};

for (int i = 0; i < 3; i++) {
  for (int j = 0; j < 3; j++) {
    for (int k = 0; k < 3; k++) {
      cout << kolory[i] << ", " << kolory[j] << ", " << kolory[k] << "\n";
    }
  }
}
```

W powyższym przykładzie trzy zagnieżdżone pętle tworzą wszystkie możliwe kombinacje trzech kolorów.

Funkcje i rekurencja

Funkcje

Funkcje umożliwiają grupowanie instrukcji, które razem wykonują konkretną czynność. Używanie funkcji sprawia, że kod staje się bardziej przejrzysty, łatwiejszy do zarządzania i wielokrotnego użycia.

Składnia funkcji

Przykładowa funkcja wygląda następująco:

```
int Maksimum(int a, int b) {
  if (a > b) {
    return a;
  }
  else {
    return b;
  }
}
```

Ogólna składnia funkcji w C++ jest następująca:

```
typ_zwracany nazwa_funkcji(typ_argumentu1 nazwa_argumentu1, typ_argumentu2 nazwa_argumentu2,
    ...) {
    // Ciało funkcji.
    return wartość;  // Jeżeli funkcja coś zwraca.
}
```

gdzie:

- **typ_zwracany** - Określa rodzaj wartości, jaką funkcja ma zwrócić. Jeśli funkcja niczego nie zwraca, używamy typu `void`.
- **nazwa_funkcji** - To, jak funkcja będzie nazywana w kodzie. Powinna być unikalna w obrębie programu.
- **lista argumentów** - Określa, jakie wartości można przekazać do funkcji. Argumenty są umieszczone w nawiasach i oddzielone przecinkami. Składają się z typu argumentu (`int`, `string`, itd.) oraz nazwy tego argumentu.
- **ciało funkcji** - Zawiera instrukcje, które są wykonywane, gdy funkcja jest wywoływana.

Zwróćmy uwagę na słowo `return`. Oznacza to, że funkcja zwraca wynik, tzn. że zakończyła już odpowiednie kalkulacje i można zakończyć bieg tej funkcji. Oznacza to, że jeżeli użyjemy instrukcji `return`, to instrukcje po niej zostaną zignorowane.

Zastanów się teraz, jaki to ma związek z funkcją `int main()`, która jest szkieletem programu oraz instrukcją `return 0;` na jej końcu.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

Wywoływanie funkcji

Aby skorzystać z funkcji, trzeba ją wywołać w programie, używając jej nazwy i dostarczając odpowiednie argumenty. Na przykład rozważmy następujący program:

```
#include <iostream>

int KolejnaLiczba(int x) {
  return x + 1;
}

int main() {
  int a = 5;
  int b = KolejnaLiczba(a);          // Wywołujemy funkcję z argumentem a = 5.
                                     // Zatem b = 6 ( = 5 + 1).
  cout << KolejnaLiczba(b) << "\n"; // Wywołujemy funkcję z argumentem b = 6.
                                     // Zatem wypisujemy 7 ( = 6 + 1).
}
```

Kolejna sprawa, którą musimy poruszyć, to co się dzieje ze zmiennymi, które przekazywane są jako argumenty funkcji. Zastanów się, co się stanie w poniższym przypadku:

```
int a = 5;
int b = KolejnaLiczba(a);
cout << a;
```

Mamy tak naprawdę dwie możliwości: a może być równe albo 6, albo 5. Okazuje się, że odpowiedzią jest 5. Funkcje domyślnie tworzą „kopię" wartości, którą przekazaliśmy jako argument i nie nadpisują tej zmiennej, nawet jak wykonujemy na niej operacje. W powyższym przypadku niezależnie od tego, co zrobimy z a w funkcji `KolejnaLiczba`, wartość a pozostanie taka sama.

Rozważmy jeszcze kilka innych przykładów funkcji:

```
// Funkcja obliczająca pole prostokąta.
double PoleProstokata(double dlugosc, double szerokosc) {
  return dlugosc * szerokosc;
}
```

```
bool JestParzysta(int liczba) {
  if (liczba % 2 == 0) {
    return true;
  } else {
    return false;
  }
}
```

Funkcje nie zawsze muszą zwracać wartość. Jeśli funkcja jest używana do wykonania określonej czynności, ale nie potrzebuje zwracać żadnych danych, możemy użyć typu `void` jako typu zwracanego.

Przykład:

```
void WypiszPowitanie() {
    cout << "Witaj w programie!" << "\n";
}
```

W powyższym przykładzie funkcja `WypiszPowitanie` po prostu wypisuje powitanie na ekranie. Aby wywołać tę funkcję, wystarczy użyć jej nazwy w programie:

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".

```
WypiszPowitanie();  // Wywołuje funkcję i wypisuje "Witaj w programie!" na ekranie
```

W funkcjach możemy także wywoływać inne funkcje. W poniższym przykładzie chcemy znaleźć minimum czterech liczb. Możemy napisać funkcję, która zwraca minimum dwóch liczb i użyć jej kilkukrotnie:

```
int Minimum(int a, int b) {
  if (a < b) return a;
  else return b;
}

int MinimumCzterechLiczb(int a, int b, int c, int d) {
  int minimum_a_b = Minimum(a, b);
  int minimum_c_d = Minimum(c, d);
  return Minimum(minimum_a_b, minimum_c_d);
}
```

Albo jeszcze prościej:

```
int MinimumCzterechLiczb(int a, int b, int c, int d) {
  return Minimum(Minimum(a, b), Minimum(c, d));
}
```

Rekurencja

Rekurencja to technika, w której funkcja wywołuje samą siebie. Jest to użyteczne w sytuacjach, gdy problem można podzielić na mniejsze instancje tego samego problemu.

Dla przykładu rozważmy obliczanie sumy cyfr danej liczby. Możemy powiedzieć, że suma cyfr liczby X, to suma dwóch składników: ostatniej cyfry oraz sumy cyfr liczby z uciętą ostatnią cyfrą (czyli $X/10$).

```
int SumaCyfr(int X) {
  if (X == 0) return 0;
  return X % 10 + SumaCyfr(X / 10);
}
```

Przyjrzyjmy się bliżej jak działa ta funkcja:

1. Jeśli liczba X jest równa 0, funkcja zwraca 0, co jest bazowym przypadkiem rekursji.
2. W każdym innym przypadku funkcja oblicza resztę z dzielenia liczby X przez 10 (operacja $X \mod 10$), co daje ostatnią cyfrę liczby.
3. Następnie, funkcja rekurencyjnie wywołuje samą siebie dla liczby X podzielonej przez 10 (operacja $\lfloor \frac{X}{10} \rfloor$), co efektywnie "usuwa" ostatnią cyfrę liczby.
4. Rezultaty tych dwóch operacji (ostatnia cyfra liczby i suma cyfr reszty liczby) są dodawane do siebie.
5. Proces ten kontynuowany jest aż do momentu, gdy osiągnięty zostanie bazowy przypadek (czyli $X = 0$).

Zwróć uwagę, że wszystkie funkcje rekurencyjne potrzebują mieć przypadek końcowy (bazowy), dla którego wynik łatwo policzyć. W naszym przypadku jest to $X = 0$.

Koniec

Gratulujemy! Udało Ci się przejść cały wstęp do programowania w C++. Liczymy, że to początek Twojej wspaniałej przygody z programowaniem. Zachęcamy do zapoznania się z **Samouczkiem**, który opisuje jak działa system oceny programów na OIJ i jak wysyłać swoje rozwiązania. Dostępne są także **Przykładowe zadania**. Kiedy będziesz już gotowa/y, zmierz się z **Archiwalnymi zadaniami** i oczywiście **Wystartuj w OIJ**!.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.