Przykłady do zestawów 2 i 3

Spis treści

[Pętle 1](#_Toc151272065)

[For 1](#_Toc151272066)

[While i do-while 2](#_Toc151272067)

[Tablice 3](#_Toc151272068)

[Tablice dwu i więcej wymiarowe 3](#_Toc151272069)

[Funkcje 5](#_Toc151272070)

[cmath 7](#_Toc151272071)

# Pętle

Pętle umożliwiają wielokrotne wykonanie bloku kodu. Istnieją trzy główne rodzaje pętli: for, while i do-while.

## For

Pętla for jest używana do wykonywania określonej liczby iteracji. Może być używana do iteracji przez kontenery, tablice lub wykonania określonego bloku kodu określoną liczbę razy.

#include <iostream>

#include <vector> // potrzebne tylko jeśli używamy std::vector

int main() {

// Pętla for używana do wyświetlenia liczb od 1 do 10.

// int i = 1 - Inicjalizacja zmiennej 'i' i przypisanie jej wartości początkowej równą 1.

// i <= 5 - Warunek kontynuacji pętli: pętla będzie wykonywana dopóki 'i' jest mniejsze lub równe 5.

// ++i - Inkrementacja 'i' o 1 po zakończeniu każdej iteracji.

// std::cout << "Liczba: " << i << std::endl; - Wyświetlenie wartości 'i' wraz z tekstem "Liczba: ".

for (int i = 1; i <= 10; ++i) {

std::cout << "Liczba: " << i << std::endl;

}

for (int i = 10; i > 0; --i) {

std::cout << "Liczba: " << i << std::endl;

}

// Przykład pętli for iterującej przez tablicę

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

// Początkowa deklaracja i inicjalizacja zmiennej sterującej pętlą i

// Wykonuje się raz przed rozpoczęciem pętli

// Warunek sprawdzający czy i jest mniejsze niż 5

// Sprawdza czy warunek jest spełniony przed każdą iteracją

// Inkrementacja zmiennej i po każdej iteracji

for (int i = 0; i < 5; ++i) {

// Wyświetlenie wartości elementów tablicy

std::cout << tablica[i] << " ";

}

std::cout << std::endl;

// WYświetlanie co drugiego elementu

for (int i = 0; i < 5; i=i+2) {

// Wyświetlenie wartości elementów tablicy

std::cout << tablica[i] << " ";

}

std::cout << std::endl;

// Przechodzenie po elementach tablicy

for (int i : tablica) {

// Wyświetlenie wartości elementów tablicy

std::cout << i << " ";

}

std::cout << std::endl;

// Przykład pętli for używającej iteratora do iteracji przez wektor

std::vector<int> wektor = { 10, 20, 30, 40, 50 };

// Iterator it rozpoczyna od początku wektora

// Warunek sprawdzający czy iterator doszedł do końca wektora

// Inkrementacja iteratora po każdej iteracji

for (auto it = wektor.begin(); it != wektor.end(); ++it) {

// Wyświetlenie wartości elementów wektora

std::cout << \*it << " ";

}

std::cout << std::endl;

for (auto it : wektor) {

// Wyświetlenie wartości elementów wektora

std::cout << it << " ";

}

std::cout << std::endl;

return 0;

}

## While i do-while

Pętla while wykonuje blok kodu tak długo, jak długo warunek jest spełniony. Pętla do-while jest podobna do pętli while, ale wykonuje się przynajmniej raz, ponieważ warunek jest sprawdzany na końcu.

#include <iostream>

int main() {

// while

int i = 0; // Inicjalizacja zmiennej i na 0

// Sprawdzenie warunku przed każdą iteracją

// Wykona się, jeśli i jest mniejsze niż 5

while (i < 5) {

// Wyświetlenie wartości i oraz inkrementacja zmiennej i

std::cout << i << " ";

++i;

}

std::cout << std::endl;

// do-while

int j = 0; // Inicjalizacja zmiennej i na 0

// Wykonuje się co najmniej raz

do {

// Wyświetlenie wartości i oraz inkrementacja zmiennej i

std::cout << j << " ";

++j;

} while (j < 0); // Sprawdza warunek na końcu pętli

std::cout << std::endl;

return 0;

}

# Tablice

Tablice jednowymiarowe przechowują elementy tego samego typu w ciągłej pamięci. Przykład:

// Przykład pętli for iterującej przez tablicę

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

// Początkowa deklaracja i inicjalizacja zmiennej sterującej pętlą i

// Wykonuje się raz przed rozpoczęciem pętli

// Warunek sprawdzający czy i jest mniejsze niż 5

// Sprawdza czy warunek jest spełniony przed każdą iteracją

// Inkrementacja zmiennej i po każdej iteracji

for (int i = 0; i < 5; ++i) {

// Wyświetlenie wartości elementów tablicy

std::cout << i << " ";

}

std::cout << std::endl;

## Tablice dwu i więcej wymiarowe

Tablice dwu i trójwymiarowe to tablice wielowymiarowe, podobne do tablic jednowymiarowych, ale z możliwością przechowywania danych w dwóch lub więcej wymiarach.

#include <iostream>

int main() {

int tablica[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

std::cout << tablica[i][j] << " ";

}

std::cout << std::endl;

}

return 0;

}

#include <iostream>

using namespace std;

int main() {

// Przykład tablicy trójwymiarowej - inicjalizacja i wyświetlanie elementów

int cube[2][2][2] = {

{{1, 2}, {3, 4}},

{{5, 6}, {7, 8}}

};

// Wyświetlanie elementów tablicy trójwymiarowej

for (int i = 0; i < 2; ++i) {

for (int j = 0; j < 2; ++j) {

for (int k = 0; k < 2; ++k) {

cout << "cube[" << i << "][" << j << "][" << k << "] = " << cube[i][j][k] << endl;

}

}

}

return 0;

}

Oczywiście wymiarów może być więcej niż trzy, choć nieczęsto ma to zastosowanie:

#include <iostream>

int main() {

const int SIZE1 = 2;

const int SIZE2 = 3;

const int SIZE3 = 4;

const int SIZE4 = 5;

const int SIZE5 = 6;

int tablica[SIZE1][SIZE2][SIZE3][SIZE4][SIZE5];

// Przykładowe przypisanie wartości do tablicy

for (int i = 0; i < SIZE1; ++i) {

for (int j = 0; j < SIZE2; ++j) {

for (int k = 0; k < SIZE3; ++k) {

for (int l = 0; l < SIZE4; ++l) {

for (int m = 0; m < SIZE5; ++m) {

tablica[i][j][k][l][m] = i + j + k + l + m;

}

}

}

}

}

// Wyświetlanie wartości tablicy

for (int i = 0; i < SIZE1; ++i) {

for (int j = 0; j < SIZE2; ++j) {

for (int k = 0; k < SIZE3; ++k) {

for (int l = 0; l < SIZE4; ++l) {

for (int m = 0; m < SIZE5; ++m) {

std::cout << "tablica[" << i << "][" << j << "][" << k << "][" << l << "][" << m << "] = " << tablica[i][j][k][l][m] << std::endl;

}

}

}

}

}

return 0;

}

# Funkcje

Funkcje w C++ są blokami kodu, które wykonują określone zadania i mogą być wywoływane z różnych miejsc w programie.

Kluczowe elementy funkcji:

* Typ zwracany: Określa typ danych, który funkcja może zwracać po zakończeniu działania.
* Nazwa funkcji: Jest to identyfikator, za pomocą którego funkcję można wywołać w programie.
* Parametry: Opcjonalne wartości, które funkcja przyjmuje jako wejście. Mogą być przekazywane przez wartość, referencję lub wskaźnik.
* Ciało funkcji: Zawiera kod do wykonania określonych zadań. To, co znajduje się wewnątrz nawiasów klamrowych {}, jest zakresem funkcji.
* Instrukcja return: Opcjonalna instrukcja używana do zwracania wartości z funkcji, jeśli funkcja ma określony typ zwracany.

Składnia funkcji w C++:

typ\_zwracany nazwa\_funkcji(parametry) {

// Ciało funkcji

// Kod wykonujący określone zadania

return wartosc\_zwracana; // Opcjonalne, jeśli funkcja ma typ zwracany

}

Przykłady:

#include <iostream>

void helloWorld() {

std::cout << "Hello, World!" << std::endl;

}

int main() {

// Wywołanie funkcji helloWorld

helloWorld();

return 0;

}

#include <iostream>

// Przeciążenie funkcji z różną liczbą parametrów

void powitanie() {

std::cout << "Witaj, Świecie!" << std::endl;

}

void powitanie(const std::string imie) {

std::cout << "Witaj, " << imie << "!" << std::endl;

}

void powitanie(const std::string imie, const std::string komunikat) {

std::cout << komunikat << ", " << imie << "!" << std::endl;

}

int main() {

powitanie(); // Wywołanie funkcji bez parametrów

powitanie("Jan"); // Wywołanie funkcji z jednym parametrem

powitanie("Anna", "Dzien dobry"); // Wywołanie funkcji z dwoma parametrami

return 0;

}

#include <iostream>

// Uzycie wartości domyslnych

void powitanie(const std::string imie = "Świecie", const std::string komunikat = "Witaj") {

std::cout << komunikat << ", " << imie << "!" << std::endl;

}

int main() {

powitanie(); // Wywołanie funkcji bez parametrów

powitanie("Jan"); // Wywołanie funkcji z jednym parametrem

powitanie("Anna", "Dzien dobry"); // Wywołanie funkcji z dwoma parametrami

return 0;

}

#include <iostream>

// Przeciążenie funkcji na podstawie typów parametrów

int suma(int a, int b) {

return a + b;

}

double suma(double a, double b) {

return a + b;

}

double suma(double a, double b, double c) {

return a + b + c;

}

int main() {

int wynikInt = suma(3, 4); // Wywołanie funkcji dla parametrów typu int

double wynikDouble = suma(2.5, 3.7); // Wywołanie funkcji dla parametrów typu double

double wynikDouble2 = suma(2.5, 3.7, 8.4); // Wywołanie funkcji dla parametrów typu double

std::cout << "Suma int: " << wynikInt << std::endl;

std::cout << "Suma double: " << wynikDouble << std::endl;

std::cout << "Suma double 2: " << wynikDouble2 << std::endl;

return 0;

}

Funkcje mogę wywoływać inne funkcje. Jeżeli funkcja wywołuje samą siebie, to mówimy o niej, iż jest rekurencyjną.

#include <iostream>

int sumaOdJeden(int n) {

if (n == 1) {

return 1;

}

return n + sumaOdJeden(n - 1);

}

int main() {

int wartosc = 5; // Przykładowa wartość, dla której będziemy sumować liczby od 1 do tej wartości

int wynik = sumaOdJeden(wartosc);

std::cout << "Suma liczb od 1 do " << wartosc << " wynosi: " << wynik << std::endl;

return 0;

}

# cmath

cmath to biblioteka w języku C++, która zawiera zestaw funkcji matematycznych do wykonywania operacji na liczbach. Oto kilka przykładów z jej wykorzystaniem:

#include <iostream>

#include <cmath>

#ifndef M\_PI

constexpr double M\_PI = 3.14159265358979323846;

#endif

int main() {

double x = 4.0, y = 2.0, z = -3.5;

// Pierwiastek kwadratowy

std::cout << "Pierwiastek z " << x << " = " << sqrt(x) << std::endl;

// Potęgowanie

std::cout << x << " do potęgi " << y << " = " << pow(x, y) << std::endl;

// Wartość bezwzględna

std::cout << "Wartość bezwzględna z " << z << " = " << abs(z) << std::endl;

// Zaokrąglanie do najbliższej liczby całkowitej

std::cout << "Zaokrąglone z " << z << " = " << round(z) << std::endl;

double angle = 45.0; // W stopniach

// Konwersja stopni na radiany

double radians = angle \* M\_PI / 180.0;

// Sinus, cosinus, tangens

std::cout << "Sinus " << angle << " stopni = " << sin(radians) << std::endl;

std::cout << "Cosinus " << angle << " stopni = " << cos(radians) << std::endl;

std::cout << "Tangens " << angle << " stopni = " << tan(radians) << std::endl;

double value = 0.5;

// Logarytm naturalny

std::cout << "Logarytm z " << value << " = " << log(value) << std::endl;

// Potęga e^x

std::cout << "e do potęgi " << value << " = " << exp(value) << std::endl;

// Moduł liczby

std::cout << "Moduł " << value << " = " << fabs(value) << std::endl;

return 0;

}