Ръководство

за програмиране на езика С+

от нулата

Автор: Мария Павлова

Факултетен номер: 2401321061

#### Съдържание

[Съдържание 1](#_Toc1167582503)

[Увод 2](#_Toc264667147)

[Тема 1. Работа с конзолата, променливи и типове данни 3](#_Toc1682716060)

[1.1 Печатане на конзолата 3](#_Toc2137404739)

[1.2 Основни типове данни в езика С++ 4](#_Toc1728161262)

[1. Цели числа (Integral types) 5](#_Toc907101493)

[2. Числа с плаваща запетая (Floating-point types) 5](#_Toc1162485627)

[3. Символи (Character type) 5](#_Toc1417242341)

[4. Булев тип (Boolean type) 6](#_Toc264533402)

[Таблица на основните типове данни: 6](#_Toc106766811)

[Заключение: 6](#_Toc178552529)

[Тема 1.3 Оператори за аритметични пресмятания 7](#_Toc1066984572)

[Тема 1.4 Четене на данни от конзолата 9](#_Toc2022353164)

[1. Четене на цели числа 9](#_Toc1332110829)

[2. Четене на дробни числа 10](#_Toc1269702637)

[3. Четене на символи 10](#_Toc449176970)

[4. Четене на текстови низове 11](#_Toc43212685)

[5.Четене на цял ред текст: 12](#_Toc1433401596)

[6. Четене на няколко стойности 12](#_Toc1667467581)

[Заключение: 13](#_Toc1979221634)

[Тема 1.5 Прости аритметически пресмятания и работа с текст. Задачи: 13](#_Toc1092572284)

[Задача 1: Пресмятане на лице на правоъгълник 13](#_Toc2142507581)

[2. Пресмятане на средно аритметично 14](#_Toc257909836)

[3. Пресмятане на лице на окръжност 15](#_Toc1287323411)

[4. Преобразуване на температура от Целзий в Фаренхайт 15](#_Toc1440095637)

[Тема 2. Условни конструкции 16](#_Toc158844912)

[Тема 2.1 Оператори за сравнение 16](#_Toc1730096825)

[Тема 2.2 Условен оператор if 17](#_Toc466221986)

[Тема 2.3 Оператор else 18](#_Toc185482537)

[Тема 2.4 Оператор else if 18](#_Toc1596991217)

[Тема 2.5 Условна конструкция switch-case 18](#_Toc685655023)

[Тема 2.6 Оператор ?: (Тернарен оператор) 20](#_Toc1054485099)

[Тема 2.7 Логически оператори 20](#_Toc1926706686)

[Тема 2.8 Задачи 22](#_Toc638705543)

[Задача 1: Проверка на диапазон 22](#_Toc682944828)

[Задача 2: Логическо НЕ 23](#_Toc365546951)

[Задача 3: Валидност на триъгълник 24](#_Toc434991587)

[Задача 4: Проверка за висок успех 25](#_Toc461532551)

[Задача 5: Калкулатор 26](#_Toc484230012)

[Задача 6: Класификация на триъгълник по страни 27](#_Toc571838262)

[Обяснение на логиката: 28](#_Toc114984583)

[Тема 3. Функции 29](#_Toc1117614718)

[Тема 3.1 Структура на функция в езика С++ 30](#_Toc424419847)

[Тема 3.2 Видове функции 30](#_Toc1146757446)

[Тема 3.3 Рекурсивни функции 32](#_Toc1892216802)

[Тема 3.4 Предимства от използването на функции 33](#_Toc2025295888)

[Тема 3.5 Задачи с функции 33](#_Toc411674611)

[Задача 1: Проста функция за сума 33](#_Toc1302054911)

[Задача 2. Функция за намиране на максимална стойност 35](#_Toc844367375)

[Задача 3. Функция за намиране на факториел 36](#_Toc1370767108)

[Задача 4. Функция за повдигане на число на степен 37](#_Toc1973912882)

[Тема 4 Цикли 38](#_Toc162975904)

[Тема 4.1 For loop 38](#_Toc2093422352)

[Тема 4.2 Вложени цикли: 40](#_Toc2057618345)

[Тема 4.3 While loop 40](#_Toc1338099182)

[Потенциални проблеми с while: 42](#_Toc1499829736)

[Пример с потребителски вход: 43](#_Toc790488427)

[Основни разлики между for и while цикли: 44](#_Toc1642669281)

[Тема 4.4 Do –while цикъл 44](#_Toc1255515003)

[Тема 4.5 Разлика между do-while и while 47](#_Toc81094489)

[Заключение 49](#_Toc1231513677)

[Тема 4.6 Задачи с цикли 49](#_Toc188640650)

[1. Отпечатване на числата от 1 до 10 49](#_Toc1932793243)

[2. Сума на числата от 1 до n 50](#_Toc418921084)

[3. Таблица за умножение 51](#_Toc1371953021)

[4. Отпечатване на числов триъгълник 52](#_Toc370466862)

[5. Броене на цифри в число 53](#_Toc1050229898)

[6. Проверка дали число е палиндром 54](#_Toc1631257394)

[7. Принтиране на ромб от звездички 55](#_Toc650700722)

[Тема 5. Статични масиви 57](#_Toc392226600)

[5.1 Едномерен статичен масив 57](#_Toc454628454)

[1. Деклариране на масиви 58](#_Toc135210621)

[2.Инициализиране на масиви 58](#_Toc723355760)

[3. Дължина на статичен масив 58](#_Toc145061614)

[4. Достъп до елементи на масив 59](#_Toc1756476408)

[5. Итерация през масив 60](#_Toc1568475626)

[6. Масиви и функции 60](#_Toc1243230500)

[7. Основни свойства на масивите: 60](#_Toc350083517)

[5.2 Задачи с едномерни статични масиви 61](#_Toc737043826)

[Задача 1: Намиране на максимален и минимален елемент в масив 61](#_Toc1930643118)

[Задача 2: Сумиране на елементите на масив 62](#_Toc1701481311)

[Задача 3: Средно аритметично на елементите в масив 64](#_Toc805797165)

[Задача 4: Обръщане на масив 65](#_Toc1351264966)

[5.3 Многомерни масиви 67](#_Toc373769449)

[1.Деклариране на многомерен масив 67](#_Toc2073282304)

[Двумерен масив (матрица) 67](#_Toc331884274)

[Тримерен масив 68](#_Toc1070337842)

[2.Инициализиране на многомерен масив 68](#_Toc2143301025)

[Инициализиране на двумерен масив: 68](#_Toc485437240)

[Инициализиране на тримерен масив: 69](#_Toc1659068812)

[3.Достъп до елементи на многомерен масив 69](#_Toc1047412929)

[4.Итерация през многомерен масив 69](#_Toc1979165402)

[Итерация през тримерен масив: 70](#_Toc2032018883)

[5.Размер на многомерен масив 71](#_Toc1161008747)

[6.Предаване на многомерен масив на функция 71](#_Toc642618710)

[5.4 Задачи с многомерен масив 72](#_Toc617582414)

[Задача 1: Сума на елементите в двоен масив 72](#_Toc1416339075)

[Задача 2: Транспониране на матрица 73](#_Toc1765555183)

[Задача 3: Намиране на максималния елемент във всяка колона 74](#_Toc1621217858)

[Задача 4: Проверка за симетрична матрица 76](#_Toc1042883224)

[Тема 6. Динамични масиви 80](#_Toc1346003538)

[Реализация на софтуеро приложение. Игра на морски шах 80](#_Toc223266735)

[Основни компоненти на кода 80](#_Toc1132911199)

[Важни моменти в кода 81](#_Toc998031844)

[Код на програмата : 88](#_Toc255689152)

# Увод

C++ е популярен програмен език, създаден през 1983 г. като разширение на езика C. Той комбинира процедурно и обектно-ориентирано програмиране, което го прави подходящ за създаване на големи и сложни приложения. C++ се използва за разработка на софтуер като операционни системи, игри и графични приложения, защото дава добър контрол върху хардуера и е много бърз.

Основни концепции в C++ са класовете и наследяването, които позволяват създаване на по-структуриран код. Въпреки че изисква повече внимание към детайлите, това помага за по-оптимизиран и ефективен код. C++ остава важен език в програмирането с активна общност и много библиотеки.



*Бярне Строуструп, създателят на С++*

# *Тема 1. Работа с конзолата, променливи и типове данни*

## 1.1 Печатане на конзолата

За да отпечатате "Hello, World!" на конзолата в C++, е нужно да използвате стандартния поток за изход std::cout. Ето как изглежда простият код за това:

#include <iostream>

int main()

{

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

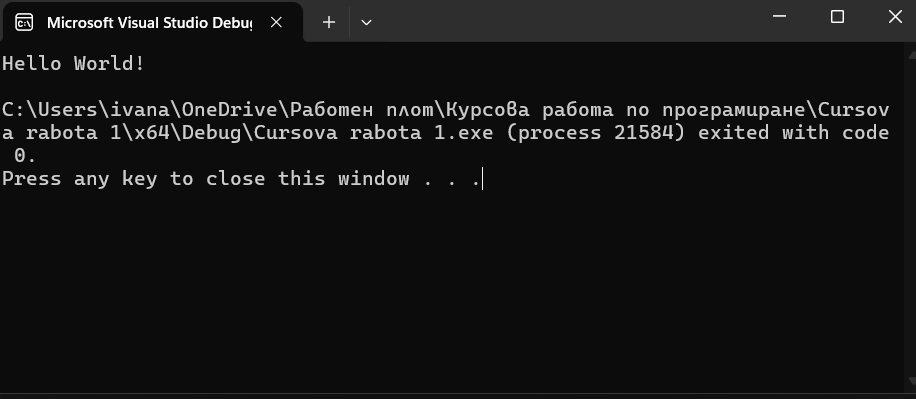
return 0;

}

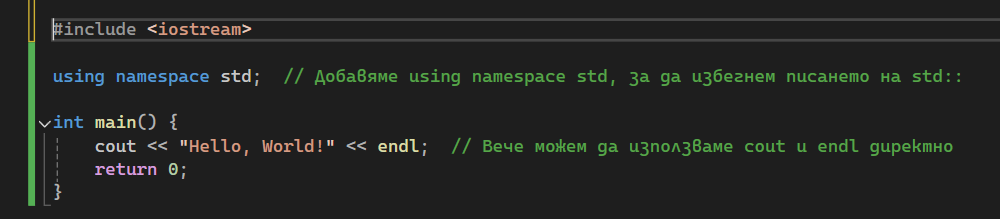
Обяснение:

1. **#include <iostream>**: Това включва стандартната библиотека за вход и изход, която позволява използването на std::cout за печат на текст в конзолата.
2. **int main()**: Това е основната функция на всяка C++ програма. Тя се изпълнява, когато програмата стартира.
3. **std::cout**: Използва се за печат на текст в конзолата. std::endl поставя нов ред след текста.
4. **return 0;**: Този ред показва, че програмата е завършила успешно.

Когато компилирате и изпълните този код, на конзолата ще видите съобщението:

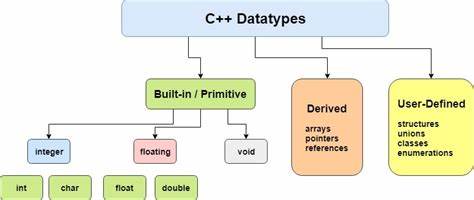


* Ако добавите namespace std, ще можете да използвате елементи от стандартната библиотека без да изписвате std:: пред тях. Ето модифицираната версия на кода с използване на using namespace std:



## 1.2 Основни типове данни в езика С++

Основните (или примитивните)типове данни в C++ се разделят на няколко категории: цели числа, числа с плаваща запетая, символи и булеви стойности. Това са примитивни типове, които предоставят основа за работа с различни стойности и се използват директно в програмите.



1. Цели числа (Integral types)

* int: Представлява цяло число. Обикновено заема 4 байта, но може да варира в зависимост от системата.
  + Пример: int age = 25;
* short: Представлява по-малко цяло число. Обикновено заема 2 байта.
  + Пример: short num = 32000;
* long: Представлява цяло число с по-голям диапазон от int. Обикновено заема 4 байта, но на някои системи може да заема 8 байта.
  + Пример: long distance = 1000000;
* long long: Използва се за много големи цели числа. Обикновено заема 8 байта.
  + Пример: long long bigNumber = 9000000000LL;
* unsigned: Представлява положително цяло число (без знак). Използва се с int, short, long за да удвои диапазона от положителни числа.

Пример: unsigned int positiveNumber = 500;

2. Числа с плаваща запетая (Floating-point types)

* float: Представлява число с плаваща запетая с единична точност. Обикновено заема 4 байта и има по-ниска точност от double.
  + Пример: float pi = 3.14f;
* double: Представлява число с плаваща запетая с двойна точност. Обикновено заема 8 байта и е по-точен от float.
  + Пример: double gravity = 9.80665;
* long double: Още по-точен тип за числа с плаваща запетая. Обикновено заема 12 или 16 байта, в зависимост от системата.
  + Пример: long double preciseValue = 2.718281828459045L;

3. Символи (Character type)

* char: Представлява един символ или малко цяло число. Заема 1 байт и съхранява ASCII кодове на символи.
  + Пример: char letter = 'A';
* wchar\_t: Представлява широк символ и може да съхранява символи от по-голям набор, като Unicode символи. Обикновено заема 2 или 4 байта.

4. Булев тип (Boolean type)

* bool: Представлява булева стойност, която може да бъде само true или false. Заема 1 байт.
  + Пример: bool isActive = true;

#### Таблица на основните типове данни:

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Описание** | **Размер (байта)** | **Диапазон (приблизително)** |
| int | Цяло число | 4 | -2,147,483,648 до 2,147,483,647 |
| short | Малко цяло число | 2 | -32,768 до 32,767 |
| long | Голямо цяло число | 4 или 8 | -2,147,483,648 до 2,147,483,647 |
| long long | Много голямо цяло число | 8 | ±9,223,372,036,854,775,807 |
| float | Число с плаваща запетая (ед. точн.) | 4 | ~7 знака след десетичната запетая |
| double | Число с плаваща запетая (дв. точн.) | 8 | ~15 знака след десетичната запетая |
| char | Символ | 1 | 0 до 255 |
| bool | Булева стойност | 1 | true или false |

Заключение:

Основните типове данни в C++ предоставят базисни средства за съхранение на различни видове стойности – цели числа, числа с плаваща запетая, символи и булеви стойности. Те се използват директно в програмите и играят ключова роля за работата на програмата, като влияят на нейното поведение, ефективност и обхват на приложимост.

## Тема 1.3 Оператори за аритметични пресмятания

В C++ аритметичните оператори се използват за извършване на основни математически операции. Ето основните аритметични оператори:

1. **Събиране (+)**: Добавя две стойности.

int a = 5, b = 3;

int sum =a + b; //sum ще бъде 8

1. **Изваждане (-)**: Изважда втората стойност от първата.

int a = 5, b = 3;

int diff = a – b; //diff ще бъде 2

1. **Умножение (\*)**: Умножава две стойности.

int a = 5, b = 3;

int product = a \* b; //product ще бъде 15

1. **Деление (/)**: Разделя първата стойност на втората. При цели числа резултатът е цяло число (делението отрязва дробната част).

int a = 6, b = 3;

int diff = a / b; //diff ще бъде 2

1. **Остатък при деление (модул) (%)**: Дава остатъка при деление на две цели числа.

int a = 5, b = 3;

int remainder = a % b; //remainder ще бъде 2

Пример:

#include <iostream>

using namespace std;

int main() {

int a = 10, b = 3;

cout << "Събиране: " << a + b << endl; //= 13

cout << "Изваждане: " << a - b << endl; //= 7

cout << "Умножение: " << a \* b << endl; //= 30

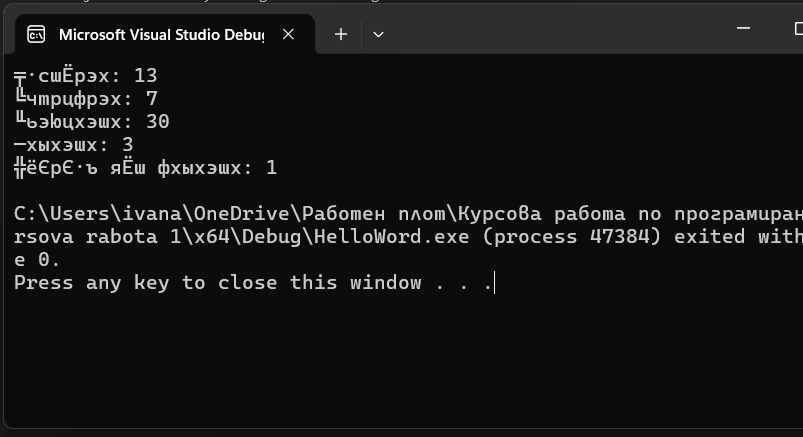
cout << "Деление: " << a / b << endl; // При цели числа делението връща цяло число = 3

cout << "Остатък при деление: " << a % b << endl; //=1

return 0;

}

Забележка: При компилирането на този код ще се получи грешка с текста на български.



За да оправите проблема трябва да използвате функциите SetConsoleCP и SetConsoleOutputCP, за да промените входната и изходната кодова страница на конзолата.

#include <iostream>

#include<windows.h> // ЗА КИРИЛИЗАЦИЯТА

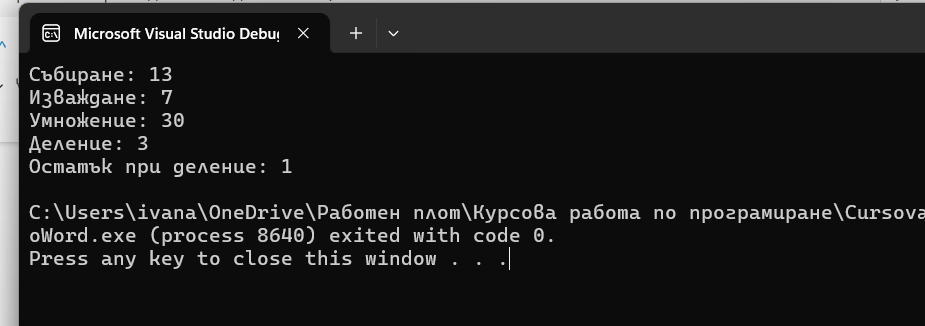
using namespace std;

int main() {

// Настройваме конзолата за работа с Windows 1251

SetConsoleCP(1251); //настройка за входа на конзолата на кирилица

SetConsoleOutputCP(1251); //настройка за изхода на конзолата на кирилица



## Тема 1.4 Четене на данни от конзолата

В C++ четенето на входни данни от конзолата се извършва чрез стандартния поток за вход, който е std::cin. Можете да използвате този поток за въвеждане на различни типове данни като числа, символи, низове и други. По-долу ще разгледаме различни начини за четене на вход от конзолата в зависимост от типа на данните.

### 1. Четене на цели числа

За да четете цели числа, можете да използвате оператора >> с std::cin.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int number;

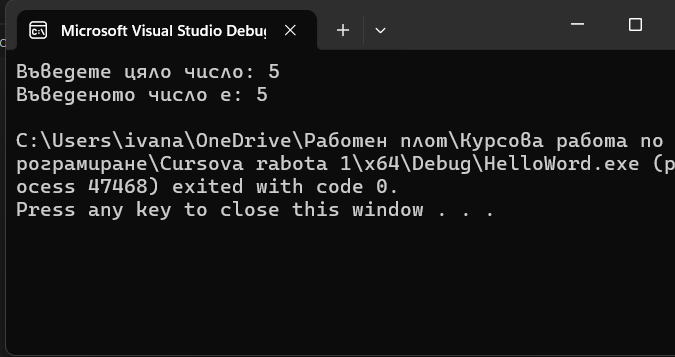
cout << "Въведете цяло число: ";

cin >> number; // Четене на цяло число от конзолата

cout << "Въведеното число е: " << number << endl;

return 0;

}



### 2. Четене на дробни числа

Аналогично на целите числа, за да четете дробни числа като float или double, използвайте cin с оператора >>.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

double decimal;

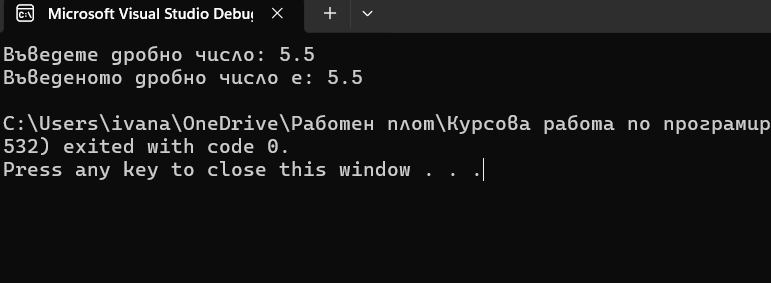
cout << "Въведете дробно число: ";

cin >> decimal; // Четене на дробно число

cout << "Въведеното дробно число е: " << decimal << endl;

return 0;

}



### 3. Четене на символи

Можете да четете по един символ чрез std::cin или като използвате функцията std::cin.get().

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char character;

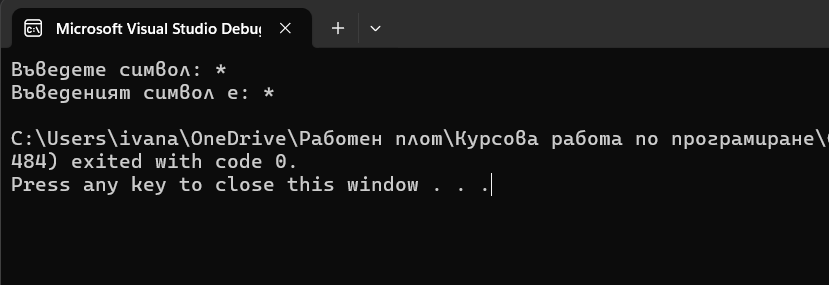
cout << "Въведете символ: ";

cin >> character; // Четене на един символ

cout << "Въведеният символ е: " << character << endl;

return 0;

}



### 4. Четене на текстови низове

Ако искате да четете текстови низове, можете да използвате std::cin за една дума (до първия разделител, като интервал), или std::getline() за цял ред текст.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

string word;

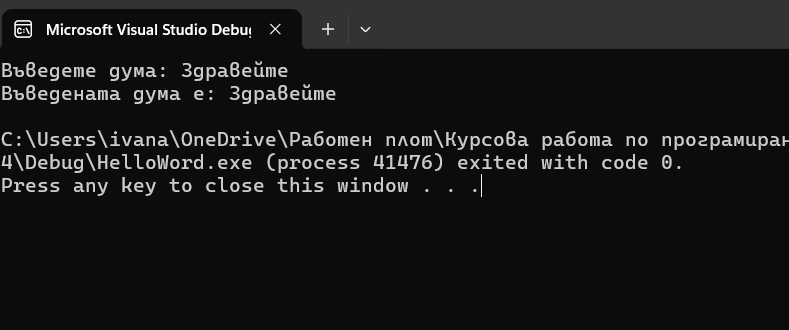
cout << "Въведете дума: ";

cin >> word; // Четене на една дума

cout << "Въведената дума е: " << word << endl;

return 0;

}



### 5.Четене на цял ред текст:

За да четете цял ред текст (включително интервали), използвайте std::getline(): и библиотеката string

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

string line;

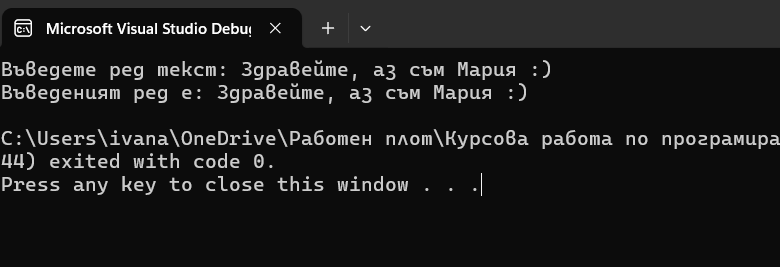
cout << "Въведете ред текст: ";

getline(cin, line); // Четене на цял ред текст

cout << "Въведеният ред е: " << line << endl;

return 0;

}



### 6. Четене на няколко стойности

Можете да четете няколко стойности последователно в един ред:

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int a, b;

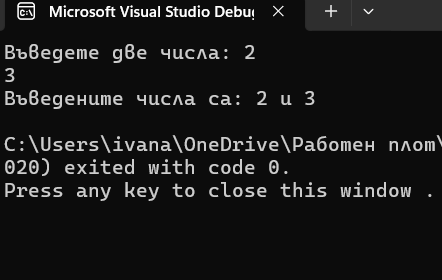
cout << "Въведете две числа: ";

cin >> a >> b; // Четене на две числа

cout << "Въведените числа са: " << a << " и " << b << endl;

return 0;

}



Заключение:

В C++ четенето на данни от конзолата е лесно и гъвкаво благодарение на std::cin и std::getline(). Те поддържат работа с различни типове данни като числа, символи и низове.

## Тема 1.5 Прости аритметически пресмятания и работа с текст. Задачи:

### Задача 1: Пресмятане на лице на правоъгълник

Напишете програма, която пресмята лицето на правоъгълник, като въвеждате дължината на страните му.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

double length, width;

cout << "Въведете дължината на правоъгълника: ";

cin >> length;

cout << "Въведете широчината на правоъгълника: ";

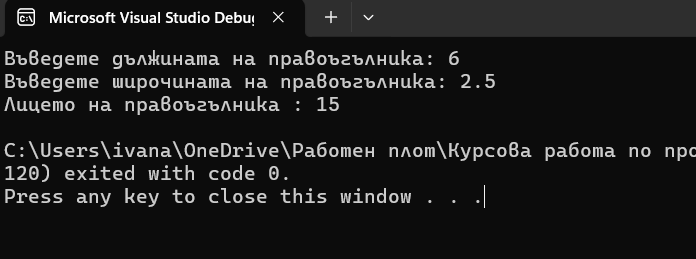
cin>> width;

double area = length \* width;

cout << "Лицето на правоъгълника e : " << area << endl;

return 0;

}



### 2. Пресмятане на средно аритметично

Напишете програма, която чете три числа и намира тяхното средно аритметично.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

double num1, num2, num3;

cout << "Въведете три числа: "<<endl;

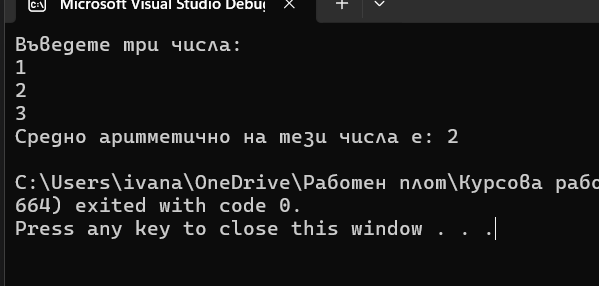
cin >> num1 >> num2 >> num3;

double average = (num1 + num2 + num3) / 3;

cout << "Средно аритметично на тези числа е: " << average << endl;

return 0;

}



### 3. Пресмятане на лице на окръжност

Напишете програма, която пресмята лицето на окръжност по зададен радиус. Резултатът да се закръгли до вторият знак след десетичната запетая.

Формула:

#include <iostream>

#include <cmath> //използваме тази библиотека за повдигането на втора степен

#include<iomanip> //библиотека за форматирането на резултата

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

double radius;

const double PI = 3.14159; //Тъй като числото пи има строга стойност //го записваме като константа

cout << "Въведете радиуса на кръга: ";

cin >> radius;

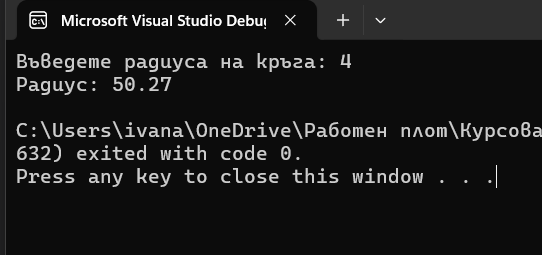
double area = PI \* pow(radius, 2); //функция за повдигане на втора //pow(стойност, степен)

cout << "Радиус: " <<fixed <<setprecision(2)<< area << endl;

// fixed<<setprecision(2)<<стойност<<форматира числото до втория знак след //десетичната запетая

return 0;

}

Проверка: 44= 50,265482457436691815402294132472

### 4. Преобразуване на температура от Целзий в Фаренхайт

Напишете програма, която преобразува температурата от Целзий във Фаренхайт по формулата:

F=C×95+32F = C

#include <iostream>

#include <cmath>

#include<iomanip>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

double celsius;

cout << "Въведете градусите в Целзий: ";

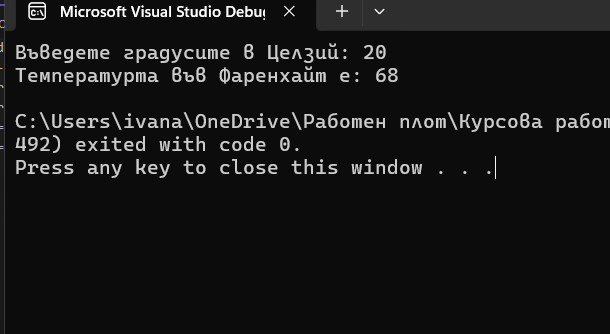
cin >> celsius;

double fahrenheit = celsius \* 9 / 5 + 32;

cout << "Температурта във Фаренхайт е: " << fahrenheit << endl;

return 0;

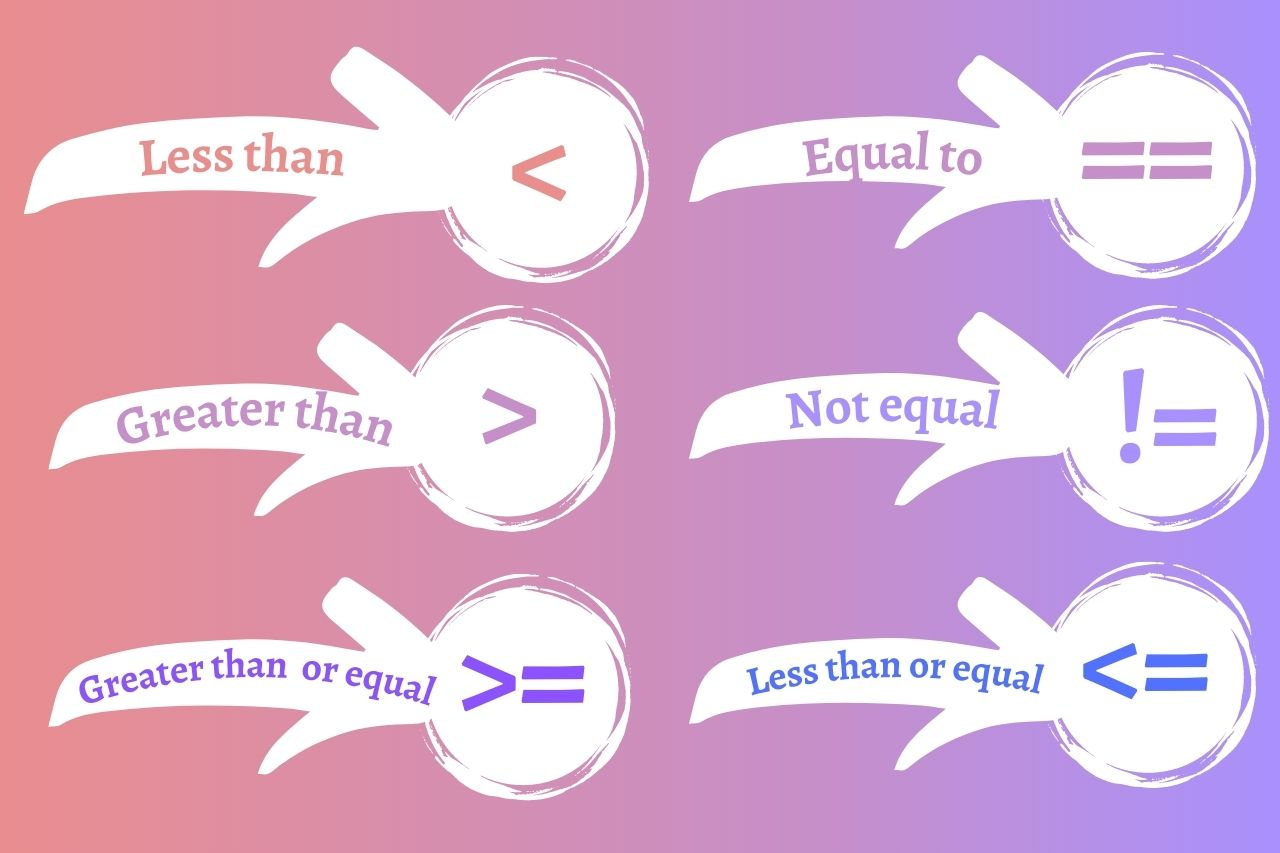
}



# Тема 2. Условни конструкции

В C++ **проверките** (или условните оператори) позволяват на програмата да взема решения на базата на определени условия. Те са основен елемент от логиката на всяка програма, като помагат да се контролира изпълнението на различни блокове от код в зависимост от стойностите на променливи или резултатите от изчисления.

## Тема 2.1 Оператори за сравнение



* ==: Равно на
* !=: Различно от
* >: По-голямо от
* <: По-малко от
* >=: По-голямо или равно на
* <=: По-малко или равно на

## Тема 2.2 Условен оператор if

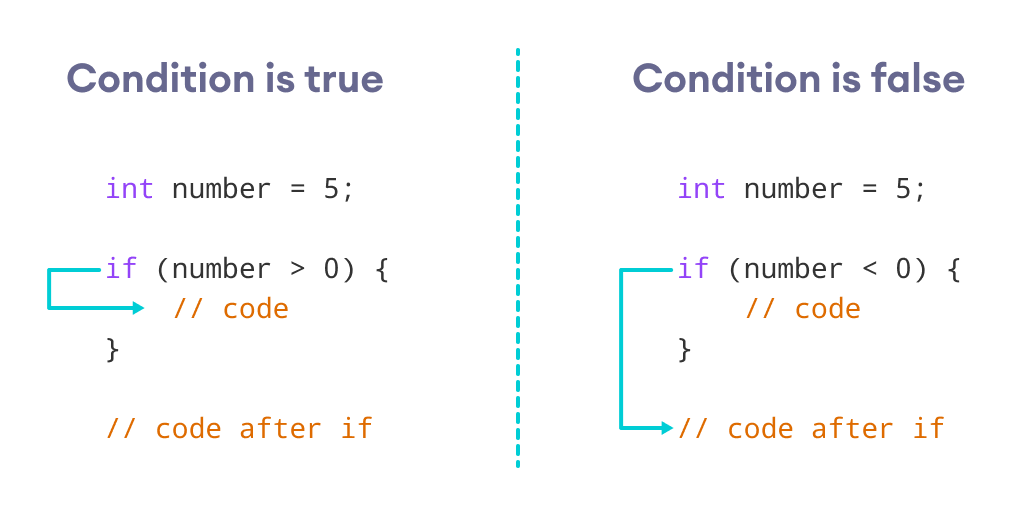
Операторът *if* проверява дали дадено условие е **истинно (true)**. Ако е, изпълнява даден блок от код.

int a = 5;

if (a > 0) {

cout << "a е положително число" << endl;

}



## Тема 2.3 Оператор else

else се използва в комбинация с if за изпълнение на код, когато условието на if не е вярно.

int a = -3;

if (a > 0) {

cout << "a е положително число" << endl;

} else {

cout << "a е отрицателно число" << endl;

}

## Тема 2.4 Оператор else if

else if позволява проверката на няколко различни условия едно след друго.

int a = 0;

if (a > 0) {

cout << "a е положително число" << endl;

} else if (a < 0) {

cout << "a е отрицателно число" << endl;

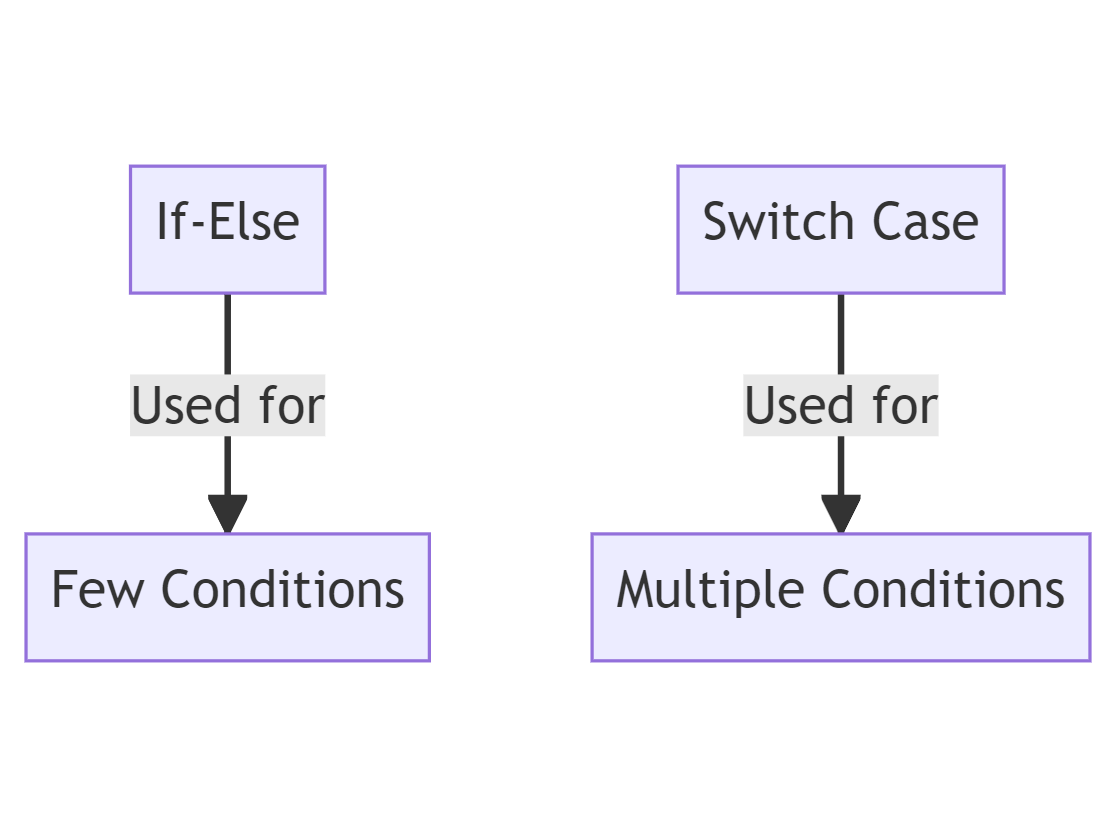
} else {

cout << "a е нула" << endl;

}

Тук се проверяват три различни състояния: дали числото е положително, отрицателно или нула.

## Тема 2.5 Условна конструкция switch-case

switch е друг тип проверка, който е полезен, когато имате множество възможни стойности на дадена променлива (обикновено цели числа или символи). Той може да замени дълги редици от if-else if.

int day ;

cin>>day;

switch(day) {

case 1:

cout << "Понеделник" << endl;

break;

case 2:

cout << "Вторник" << endl;

break;

case 3:

cout << "Сряда" << endl;

break;

...........

default:

cout << "Невалиден ден" << endl;

break;

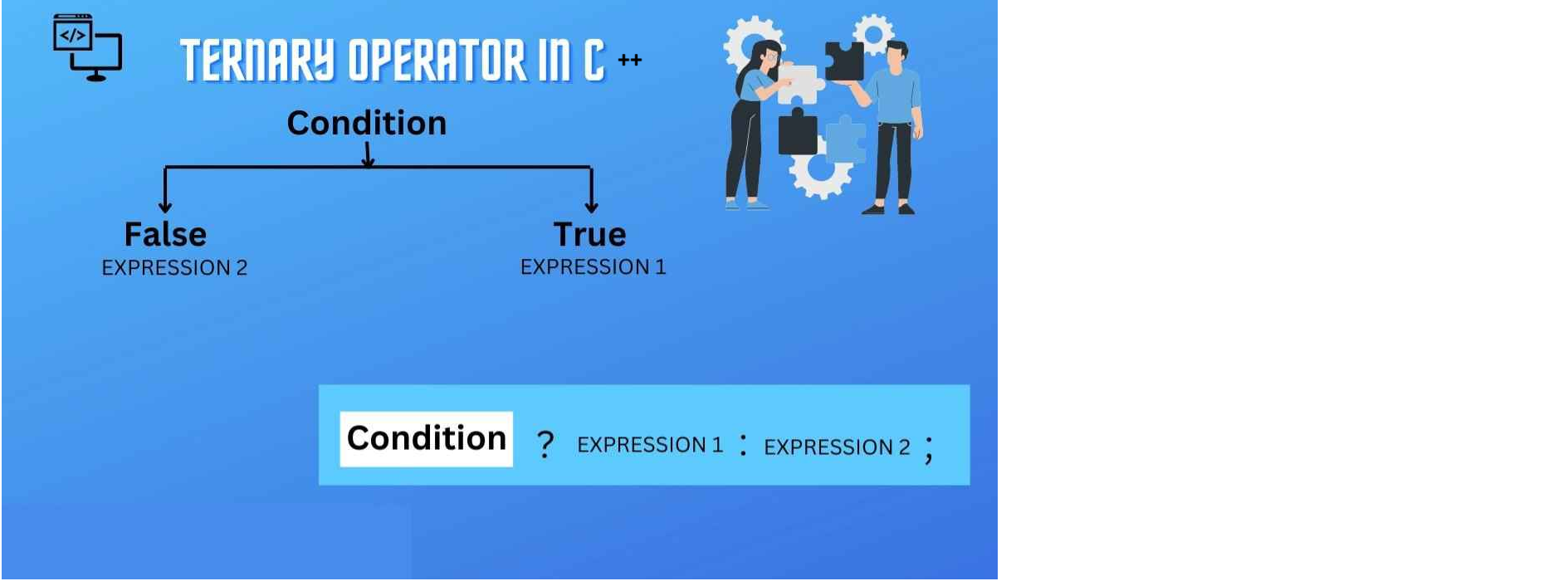
}

В тази задача потребителя въвежда число и програмата отпечатва деня от седмицата, съответстващ на него. При невалидно число се отпечатва "Невалиден ден".

## Тема 2.6 Оператор ?: (Тернарен оператор)

Тернарният оператор е кратка форма на if-else. Той има следния синтаксис:

условие ? стойност\_ако\_е\_true : стойност\_ако\_е\_false;



Пример:

int a = 10;

int b = (a > 0) ? 1 : -1; // Ако a > 0, b ще е 1, иначе ще е -1

cout << b << endl;

## Тема 2.7 Логически оператори

Често е необходимо да се проверят повече от едно условие едновременно. За целта се използват логическите оператори:

* && (логическо "и"): Връща true, ако и двете условия са верни.
* || (логическо "или"): Връща true, ако поне едно от условията е вярно.
* ! (логическо "не"): Обръща стойността на условието. Ако е true, го прави false и обратно.

Пример:

int a = 5, b = 10;

if (a > 0 && b > 0) {

cout << "И двете числа са положителни" << endl;

}

Пример с всички проверки:

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int number;

cout << "Въведете число: ";

cin >> number;

// Пример за if-else if (number > 0) {

cout << "Числото е положително" << endl;

}

else if (number < 0) {

cout << "Числото е отрицателно" << endl;

}

else {

cout << "Числото е нула" << endl;

}

// Пример за switch

switch (number) {

case 0:

cout << "Числото е точно нула" << endl;

break;

case 1:

cout << "Числото е едно" << endl;

break;

default:

cout << "Числото не е нула или едно" << endl;

break;

}

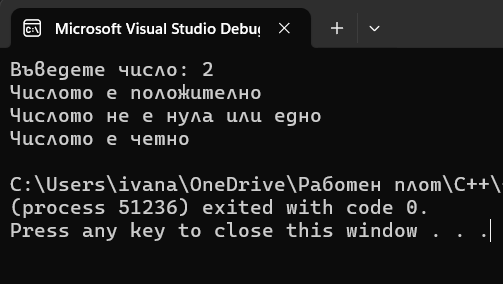
// Пример за тернарен оператор

string result = (number % 2 == 0) ? "четно" : "нечетно";

cout << "Числото е " << result << endl;

return 0;

}



*Обяснение:*

1. **if-else if-else**: Използваме тези оператори за проверка дали числото е положително, отрицателно или нула.
2. **switch**: Тук проверяваме специфични стойности (0 и 1).
3. **Тернарен оператор**: Проверяваме дали числото е четно или нечетно с по-компактен синтаксис.

В while цикъла условието се проверява преди изпълнението на тялото, което означава, че ако условието е неистинно от самото начало, тялото може изобщо да не се изпълни:

## Тема 2.8 Задачи

### Задача 1: Проверка на диапазон

Напишете програма, която приема цяло число от потребителя и проверява дали числото е в диапазона [10, 20] включително.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int number;

cout << "Въведете число: ";

cin >> number;

if (number >= 10 && number <= 20) {//проверяваме числото

cout << "Числото е в диапазона [10, 20]." << endl;

}

else {

cout << "Числото е извън рамките на диапазона." << endl;

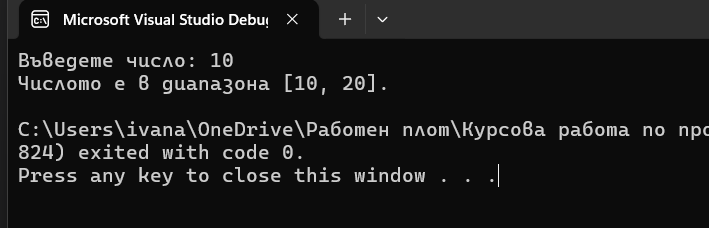
}

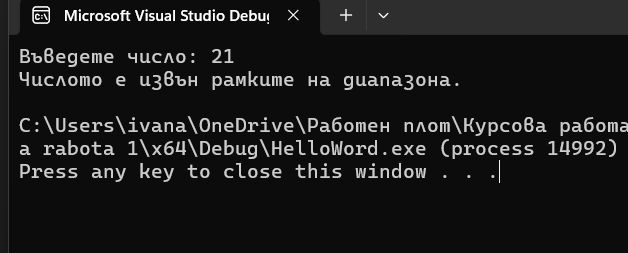
return 0;

}

Ояснение:

1. Прочитаме потребителския вход
2. Създаваме if-else проверка
3. Първо провераваме дали числото е в диапазона, използваме логическия оператор && (и), при съответствие програмата изписва съобщение и приключва
4. След това задаваме съобщението, което ще се изпэлни при несъответствие на първото условие.
5. Тестване на код:





### **Задача 2: Логическо НЕ**

Напишете програма, която приема две числа от потребителя и проверява дали поне едно от тях не е положително число.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int a, b;

cout << "Въведете две числа: "<<endl;

cin >> a >> b;

if (!(a > 0 && b > 0)) {

cout << "Поне едно от тези числа е отрицателно." << endl;

}

else {

cout << "И двете числа са положителни." << endl;

}

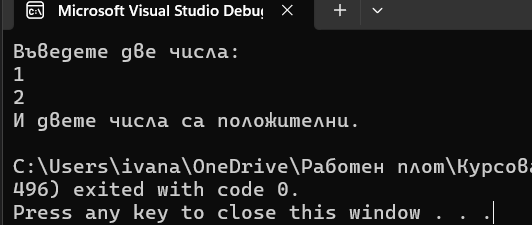
return 0;

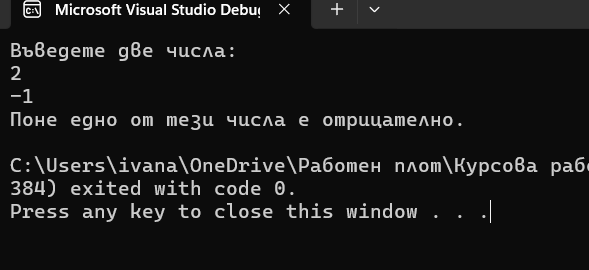
}

Забележка!

Задачата може да се реши и просто с проверка дали и двете числа са отрицателни, целта е упражнение на логическия оператор за отрицание.

Тест на кода:





### **Задача 3: Валидност на триъгълник**

Напишете програма, която приема три страни на триъгълник и проверява дали те образуват валиден триъгълник. Триъгълник е валиден, ако сборът на всяка двойка страни е по-голям от третата.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int a, b, c;

cout << "Въведете трите страни на триъгълника: "<<endl;

cin >> a >> b >> c;

if (a + b > c && a + c > b && b + c > a) {

cout << "Триъгълникът съществува." << endl;

}

else {

cout << "Не съществува триъгълник с тези страни." << endl;

}

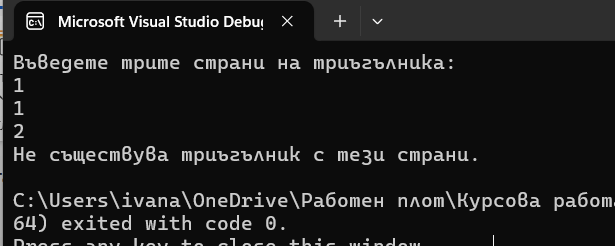
return 0;

}

Проверка на кода : Тест 1

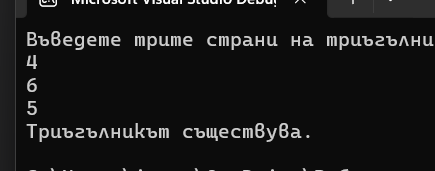
Вход: a=1, b=1, c=2;

! 1+1>2; 1+2>1; 2+1>1



Тест 2:

4+6>5; 5+4>6; 6+5>4



### **Задача 4: Проверка за висок успех**

Напишете програма, която приема оценка от изпит (между 2 и 6) и проверява дали студентът има висок успех (5 или 6), или е скъсан (оценка 2).

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int grade;

cout << "Въведете оценка: ";

cin >> grade;

if (grade == 5 || grade == 6)//Използваме логическият оператор || //(или) {

cout << "Имате висока оценка." << endl;

}

else if (grade == 2) {

cout << "Скъсан сте на изпита." << endl;

}

else {

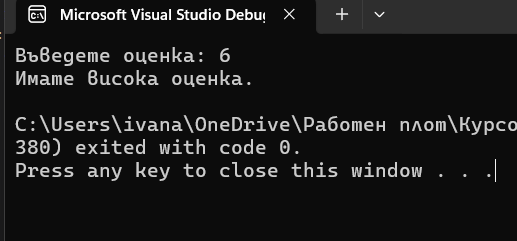
cout << "Взели сте изпита." << endl;

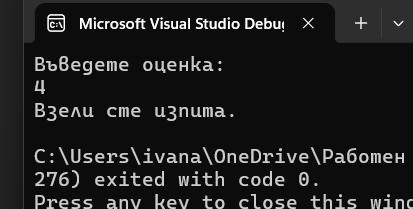
}

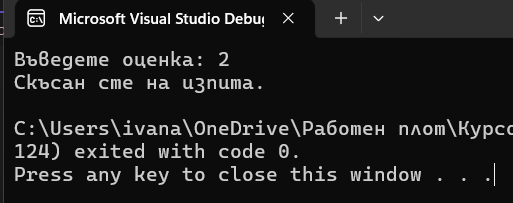
return 0;

}

Проверка на код:







### **Задача 5: Калкулатор**

Напишете програма, която приема две числа и символ за операцията (+, -, \*, /) и изчислява резултата.

**Пример:**

* Вход: 5, 3, +
* Изход: 8

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

double num1, num2;

char operation;

cout << "Въведете първото число: ";

cin >> num1;

cout << "Въведете операция (+, -, \*, /): ";

cin >> operation;

cout << "Въведете второто число: ";

cin >> num2;

switch (operation) {

case '+': cout << "Сбор: " << num1 + num2 << endl; break;

case '-': cout << "Разлика: " << num1 - num2 << endl; break;

case '\*': cout << "Произведение: " << num1 \* num2 << endl; break;

case '/':

if (num2 != 0)

cout << "Частно: " << num1 / num2 << endl;

else

cout << "Грешка, деление на нула." << endl;

break;

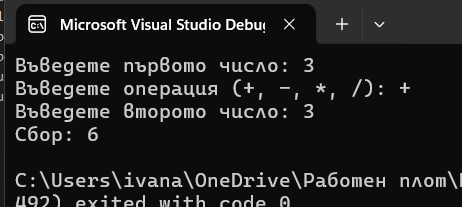
default: cout << "Невалидна операция!" << endl;

}

return 0;

}

Тестване на кода:



### **Задача 6: Класификация на триъгълник по страни**

Напишете програма, която приема три страни на триъгълник и проверява дали триъгълникът е:

* **Равностранен** — ако трите страни са равни.
* **Равнобедрен** — ако две страни са равни.
* **Разностранен** — ако трите страни са различни.
* Ако страните не могат да образуват валиден триъгълник, програмата трябва да изведе съобщение за това.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int a, b, c;

cout << "Въведете трите страни на триъгълника: "<<endl;

cin >> a >> b >> c;

// Проверка дали триъгълникът е валиден

if (a + b > c && a + c > b && b + c > a) {

// Вложена условна конструкция за тип на триъгълника

// Равностранен

if (a == b && b == c) {

cout << "Триъгълникът е равносранен." << endl;

}

// Равнобедрен

else if (a == b || b == c || a == c) {

cout << "Триъгълникът е равнобедрен." << endl; }

// Разностранен

else {

cout << "Триъгърникът е разностранен." << endl; }

}

else {

// Ако страните не образуват валиден триъгълник

cout << "Не съществува триъгълник с тези страни." << endl;

}

return 0;

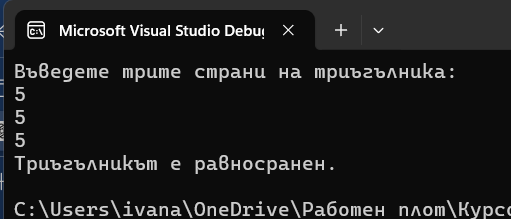
}

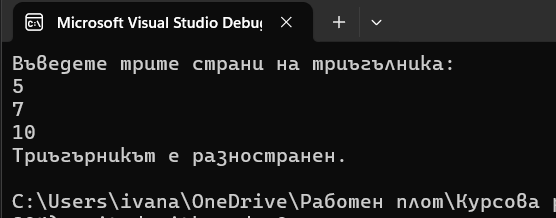
### **Обяснение на логиката:**

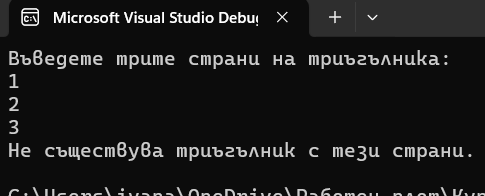
1. Първо се проверява дали триъгълникът е валиден чрез условията за неравенството на триъгълника.
2. Ако триъгълникът е валиден, се използват вложени **if-else** конструкции, за да се определи типа:
   1. Ако всичките страни са равни, той е **равностранен**.
   2. Ако две страни са равни, но третата е различна, той е **равнобедрен**.
   3. Ако всичките страни са различни, триъгълникът е **разностранен**.
3. Ако не е валиден, се извежда съответно съобщение.

**Пример за тест:**

* Вход: 5, 5, 5
* Изход: Триъгълникът е равностранен.
* Вход: 5, 7, 10
* Изход: Триъгълникът е разностранен.
* Вход: 1, 2, 3
* Изход: Триъгълникът не е валиден.







# Тема 3. Функции

Функциите в C++ са основен елемент на структурното и модулното програмиране. Те позволяват разделянето на програмата на по-малки, управляеми части, което прави кода по-четим, по-лесен за поддръжка и повторна употреба.

## Тема 3.1 Структура на функция в езика С++

Функциите в C++ имат следния общ синтаксис:

return\_type function\_name(parameters) {

// Тяло на функцията

// Операции, които функцията изпълнява

return value; // Стойност, която функцията връща (ако има такава)

}

* return\_type: Типът на стойността, която функцията връща (например int, double, void).
* function\_name: Името на функцията, което се използва за извикването й.
* parameters: Списък с параметри (ако има такива), които функцията приема при извикване. Всеки параметър има тип и име.
* return: Операторът return се използва за връщане на стойност от функцията.

## Тема 3.2 Видове функции

* **Функция без параметри и без връщана стойност**: Функцията няма параметри и не връща стойност (типът й е void).

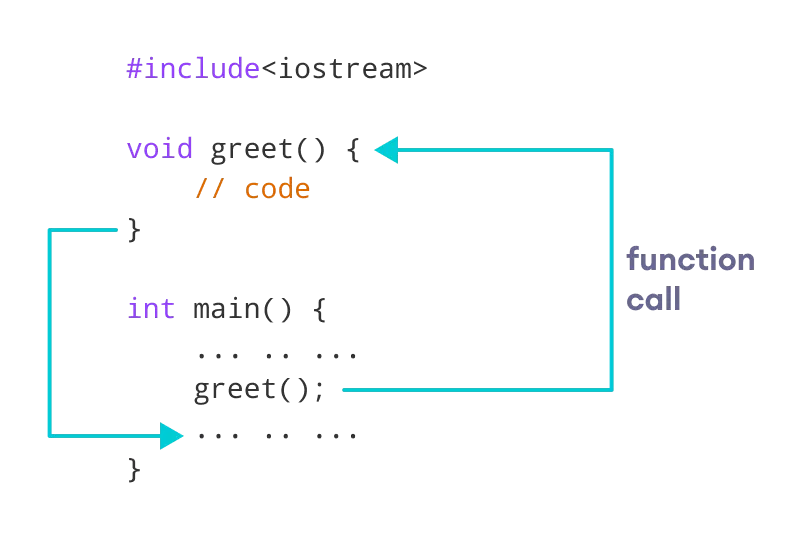
void sayHello() {

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

}

Примерно извикване на функцията:





* Функция с параметри, но без връщана стойност: Функцията приема параметри, но не връща стойност.

void printSum(int a, int b) {

int sum = a + b;

std::cout << "Sum: " << sum << std::endl;

}

Примерно извикване на функцията:

printSum(5, 10);

* Функция с параметри и връщана стойност: Функцията приема параметри и връща резултат от тип, различен от void.

int multiply(int a, int b) {

return a \* b;

}

Примерно извикване на функцията:

int result = multiply(4, 5);

std::cout << "Result: " << result << std::endl;

Функция с подразбиращи се параметри

Можете да зададете стойности по подразбиране за параметрите на функцията. Ако параметрите не бъдат зададени при извикването на функцията, ще се използват стойностите по подразбиране.

void printMessage(std::string message = "Hello!") {

std::cout << message << std::endl;

}

Примерни извиквания:

printMessage(); // Извежда "Hello!"

printMessage("Hi!"); // Извежда "Hi!"

## Тема 3.3 Рекурсивни функции

Функция, която извиква сама себе си, се нарича рекурсивна. Рекурсивните функции често се използват за решаване на проблеми, които могат да бъдат разделени на по-малки подобни задачи.

Ето пример за по-проста рекурсивна функция, която пресмята сумата на числата от 1 до дадено число n. Тази задача може да бъде решена чрез рекурсия по следния начин.

// Рекурсивна функция за намиране на сумата от 1 до n

#include <iostream>

#include<windows.h>

using namespace std;

int sum(int n) {

return n == 0 ? 0 : n + sum(n - 1);

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n;

cout << "Въведете число: ";

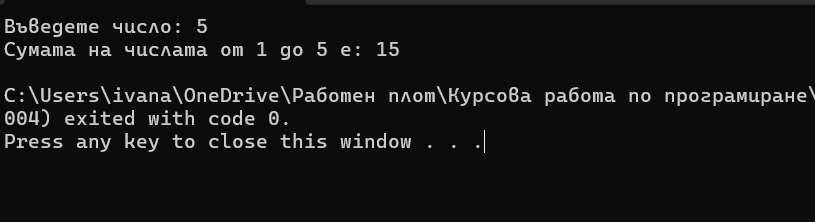
cin >> n;

// Извеждане на резултата

cout << "Сумата на числата от 1 до " << n << " е: " << sum(n) << endl;

return 0;

}



## Тема 3.4 Предимства от използването на функции

* **Повторна употреба на код**: Една и съща логика може да се използва многократно без да се пише наново.
* **Разделяне на логиката**: Кодът става по-четим и структурен, тъй като функциите позволяват логиката да бъде разделена на по-малки блокове.
* **Лесна поддръжка**: Промените в логиката на дадена функция могат да се направят на едно място, без да е нужно да се променя целият код.

## Тема 3.5 Задачи с функции

### Задача 1: Проста функция за сума

Напишете функция, която приема две цели числа като параметри и връща тяхната сума. Използвайте функцията в програмата, за да съберете две числа, въведени от потребителя.

**Пример:**

* Вход: 3, 5
* Изход: Сумата е 8

#include <iostream>

#include<windows.h>

using namespace std;

int sum(int n1, int n2) {

return n1 + n2;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n1, n2;

cout << "Въведете числата: " << endl;

cin >> n1>>n2;

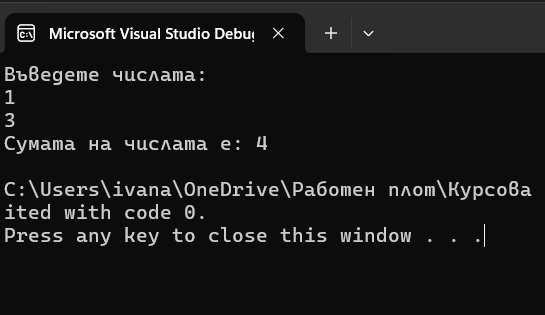
// Извеждане на резултата

cout << "Сумата на числата е: " << sum(n1,n2) << endl;

return 0;

}

Тест на кода:



### Задача 2. Функция за намиране на максимална стойност

Напишете функция max на C++, която приема два аргумента от тип int и връща по-голямото от двете числа.

#include <iostream>

#include<windows.h>

using namespace std;

int maxNumber(int x, int y) {

return (x > y) ? x : y;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int x, y;

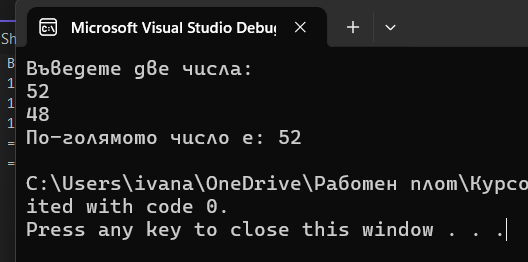
cout << "Въведете две числа: " << endl;

cin >> x >> y;

cout << "По-голямото число е: " << maxNumber(x, y) << endl;

return 0;

}



### Задача 3. Функция за намиране на факториел

**Описание**: Напишете функция factorial, която приема едно число n и връща факториела на това число. Използвайте рекурсия.

#include <iostream>

#include<windows.h>

using namespace std;

int factorial(int n) {

if (n <= 1) return 1;

return n \* factorial(n - 1);

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n;

cout << "Въведете число: ";

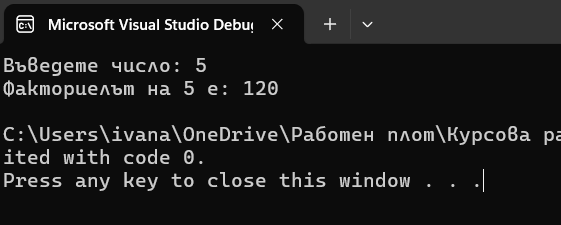
cin >> n;

cout << "Факториелът на " << n << " е: " << factorial(n) << endl;

return 0;

}

Тест на кода:



### Задача 4. Функция за повдигане на число на степен

**Описание**: Напишете функция power, която приема две числа base и exponent и връща стойността на base, повдигнато на степен exponent.

#include <iostream>

#include<windows.h>

using namespace std;

int power(int base, int exponent) {

int result = 1;

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

result \*= base;

}

return result;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int base, exponent;

cout << "Въведете основа и степен: " << endl;

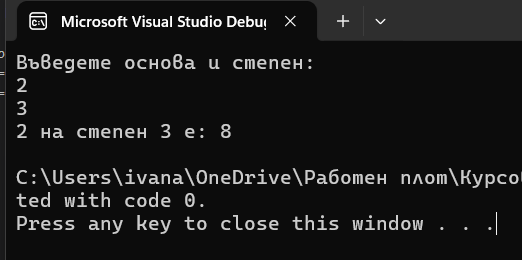
cin >> base >> exponent;

cout << base << " на степен " << exponent << " е: " << power(base, exponent) << endl;

return 0;

}

ТЕСТ НА КОДА:



# Тема 4 Цикли

C++ цикли се използват за многократно изпълнение на блок от код, докато е изпълнено дадено условие. Най-често срещаните видове цикли са:

1. **for цикъл**
2. **while цикъл**
3. **do-while цикъл**

## Тема 4.1 For loop

Цикълът **for** в C++ е полезен, когато знаем предварително колко точно итерации (повторения) трябва да направим. Обикновено се използва, когато искаме да изпълним код многократно с брояч, който се актуализира след всяка итерация.

Синтаксис:

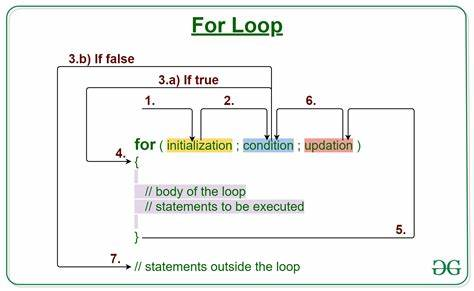
for (инициализация; условие; актуализация) {

// Тяло на цикъла

}

*Компоненти на for цикъла:*

1. **Инициализация:** Това е изразът, който се изпълнява само веднъж, в началото на цикъла. Обикновено се използва за създаване и инициализиране на броячна променлива (напр. int i = 0).
2. **Условие:** Това е логическият израз, който се проверява преди всяка итерация. Ако условието е вярно (true), цикълът се изпълнява; ако е грешно (false), цикълът приключва.
3. **Актуализация:** Това е изразът, който се изпълнява след всяка итерация, обикновено за увеличаване или намаляване на броячната променлива.



#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

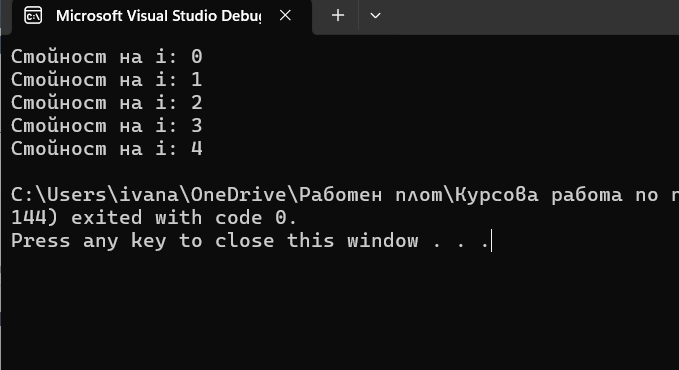
for (int i = 0; i < 5; i++) {

cout << "Стойност на i: " << i << endl;

}

return 0;

}



**Обяснение:**

* **Инициализация**: int i = 0 – Създаваме променлива i и я инициализираме с 0.
* **Условие**: i < 5 – Цикълът ще продължава да се изпълнява, докато i е по-малко от 5.
* **Актуализация**: i++ – След всяка итерация увеличаваме i с 1.
* Цикълът ще отпечата стойностите на i от 0 до 4.

## Тема 4.2 Вложени цикли:

for цикли могат да бъдат вложени един в друг (т.е. един цикъл вътре в друг). Това е полезно при работа с двумерни масиви или при задачи, свързани с матрици. #include <iostream>

using namespace std;

int main() {

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

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

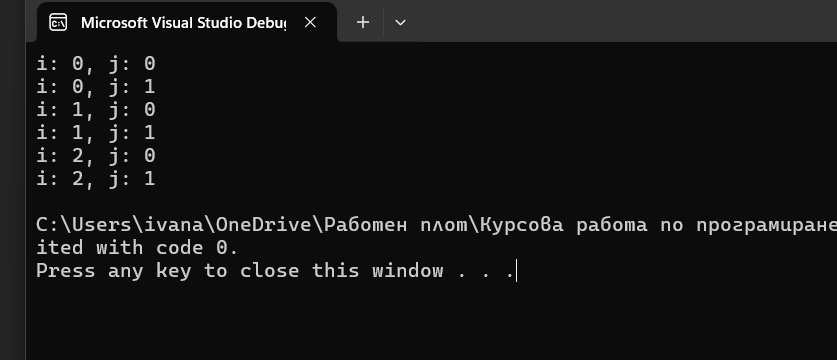
cout << "i: " << i << ", j: " << j << endl;

}

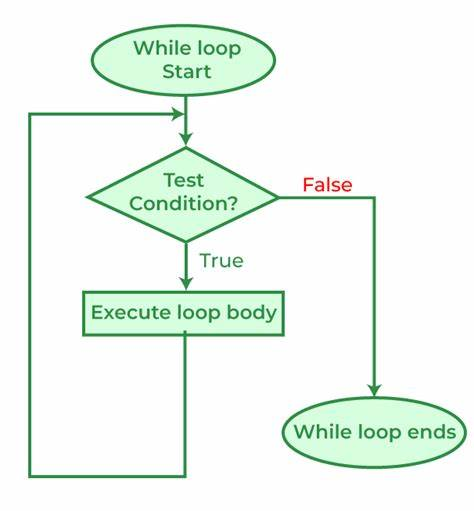
}

return 0;

}



## Тема 4.3 While loop

Използва се, когато не е ясно предварително колко пъти ще се изпълнява цикълът, но условието трябва да бъде проверявано преди всяко изпълнение.

Синтаксис:

while (условие) {

// Тяло на цикъла

}

*Компоненти на while цикъла:*

1. **Условие**: Това е логическият израз, който се проверява преди всяка итерация на цикъла. Ако условието е вярно (true), цикълът се изпълнява; ако е грешно (false), цикълът приключва.
2. **Тяло на цикъла**: Блокът код, който се изпълнява многократно, докато условието е вярно.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int i = 0;

while (i < 5) {

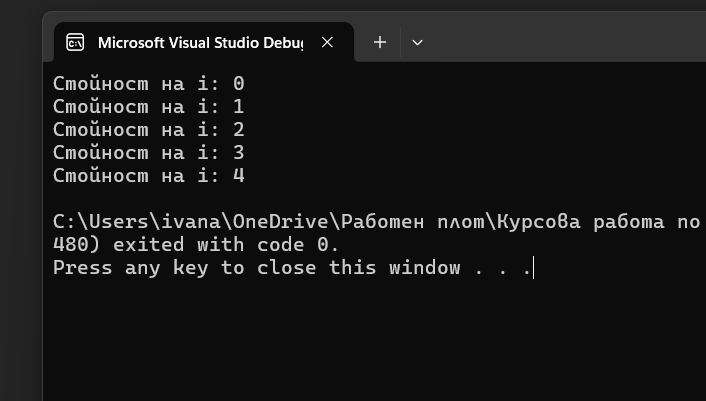
cout << "Стойност на i: " << i << endl;

i++;

}

return 0;

}



**Обяснение:**

* **Условие**: i < 5 – Цикълът ще продължава, докато i е по-малко от 5.
* Всяка итерация отпечатва текущата стойност на i и след това я увеличава с 1.
* Цикълът ще отпечата стойностите от 0 до 4.

#### **Потенциални проблеми с while:**

* Ако забравим да актуализираме променливата в условието (например i++), можем да създадем **безкраен цикъл**, т.е. цикъл, който никога не свършва.
* Ако условието още отначало е невярно, цикълът няма да се изпълни нито веднъж.

Пример за безкраен цикъл:

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int i = 0;

while (i < 5) {

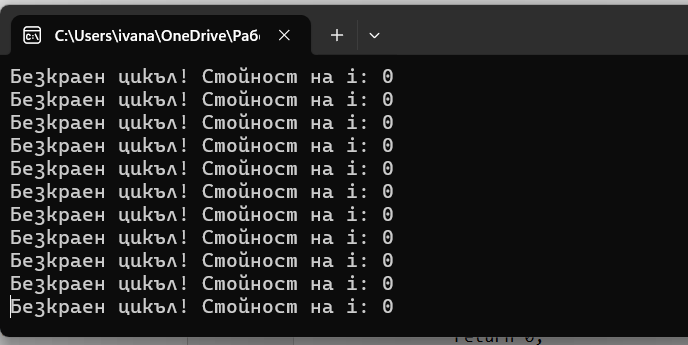
cout << "Безкраен цикъл! Стойност на i: " << i << endl;

// Забравили сме да увеличим i, така че условието i < 5 винаги е вярно

}

return 0;

}



Този код ще продължава да отпечатва "Безкраен цикъл!", защото никога не актуализираме i

###### Пример с потребителски вход:

Цикълът while често се използва, когато трябва да изчакаме действие от потребителя, например въвеждане на правилни данни:

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int number;

cout << "Въведете число, по-голямо от 0: ";

cin >> number;

while (number <= 0) {

cout << "Невалидно число. Въведете отново: ";

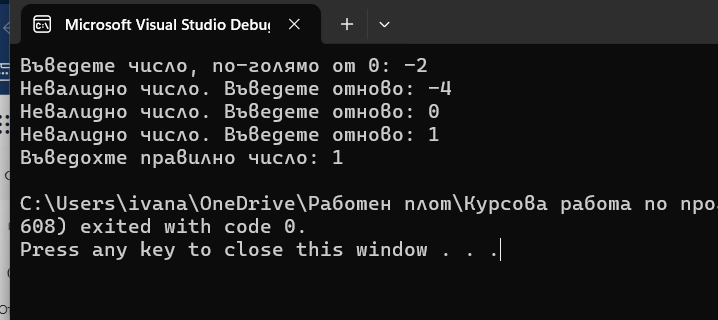
cin >> number;

}

cout << "Въведохте правилно число: " << number << endl;

return 0;

}



### Основни разлики между for и while цикли:

* **for цикъл** е по-подходящ, когато знаем предварително колко пъти ще се изпълнява цикълът.
* **while цикъл** е по-гъвкав, когато не знаем колко итерации ще има, и се базира на условие.
* **for цикъл** съдържа три части (инициализация, условие и актуализация) на едно място, докато при **while цикъл** трябва ръчно да актуализираме променливата вътре в тялото на цикъла.

### Оператори break и continue

В C++ операторите break и continue се използват за контролиране на потока на изпълнение в цикли и оператора switch. Те помагат да се прекъсват или пропускат итерации на цикли в зависимост от определени условия.

### ***Оператор break***

Операторът break прекратява изпълнението на текущия цикъл (for, while или do...while) или излиза от оператора switch. Когато програмата достигне break, цикълът или switch операторът се прекратява, и управлението преминава към следващия оператор след цикъла или switch.

#include <iostream>

using namespace std;

int main() {

for (int i = 0; i < 10; i++) {

if (i == 5) {

break; // Прекратява цикъла, когато i е равно на 5

}

cout << i << " ";

}

// Изход: 0 1 2 3 4

return 0;

}

В този пример, когато i стане равно на 5, break прекратява цикъла for, и програмата продължава със следващия оператор след цикъла.

### ***Оператор continue***

Операторът continue пропуска текущата итерация на цикъла и преминава към следващата. За разлика от break, той не прекратява цикъла, а само прекратява текущата итерация и започва нова.

#include <iostream>

using namespace std;

int main() {

for (int i = 0; i < 10; i++) {

if (i % 2 == 0) {

continue; // Пропуска четните числа

}

cout << i << " ";

}

// Изход: 1 3 5 7 9

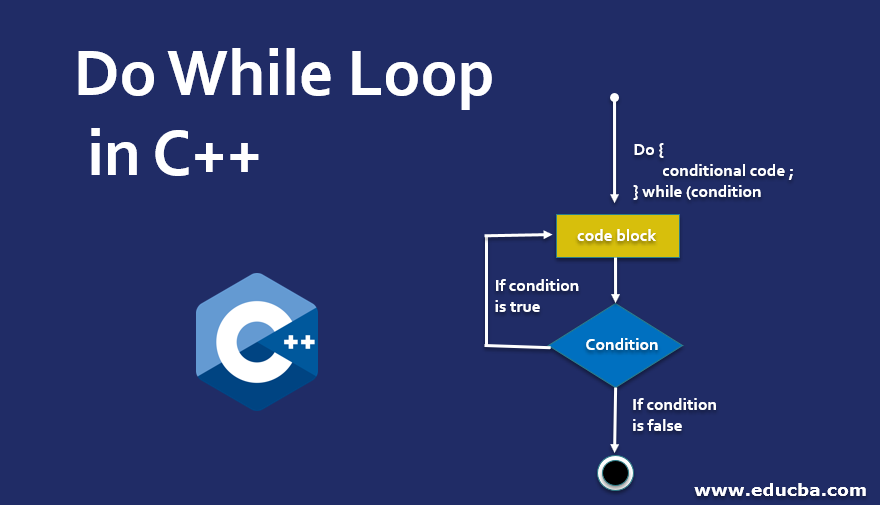
return 0;

}

В този пример, когато i е четно число, continue пропуска текущата итерация и програмата веднага преминава към следващото i.

## Тема 4.4 Do –while цикъл

* Операторът do-while в C++ е вид цикъл, който гарантира, че тялото на цикъла ще се изпълни поне веднъж, независимо от условието. Това го различава от стандартния while цикъл, при който условието се проверява преди изпълнението на тялото.



* Синтаксис на do-while цикъла:

do {

// Тяло на цикъла

// Операции, които се изпълняват

} while (условие);

* Тяло на цикъла: Това е блокът код, който ще се изпълнява многократно.
* Условие: Изразът в скобите на оператора while. Ако е истинен (true), цикълът се повтаря. Ако е неистинен (false), цикълът приключва.
* *Характеристика на do-while цикъла:*
* Тялото на цикъла се изпълнява поне **веднъж**, дори ако условието е неистинно още в началото.
* След всяко изпълнение на тялото, условието се проверява и ако е истина, цикълът се повтаря.

Пример 1: Цикъл за въвеждане на число

Ето пример за цикъл, който кара потребителя да въведе положително число:

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int number;

// Използваме do-while цикъл, за да поискаме положително число

do {

cout << "Въведете положително число: ";

cin >> number;

} while (number <= 0); // Продължаваме, докато числото е отрицателно или нула

cout << "Въведохте: " << number << endl;

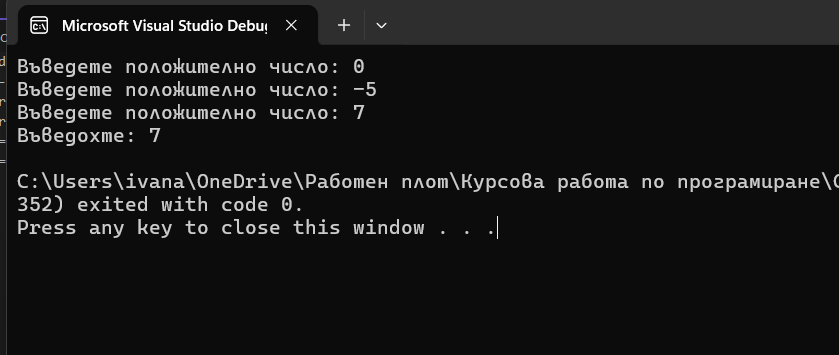
return 0;

}

Обяснение:

В този пример цикълът ще се изпълнява, докато потребителят не въведе положително число. Дори ако потребителят веднага въведе правилно число, тялото на цикъла ще се изпълни поне веднъж.

Ако потребителят въведе отрицателно число или 0, програмата ще продължи да иска ново число.



Пример 2: Сумиране на числа до 100

Следващият пример сумира числа от 1 до 100:

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int sum = 0;

int i = 1;

// Цикъл, който сумира числата от 1 до 100

do {

sum += i;

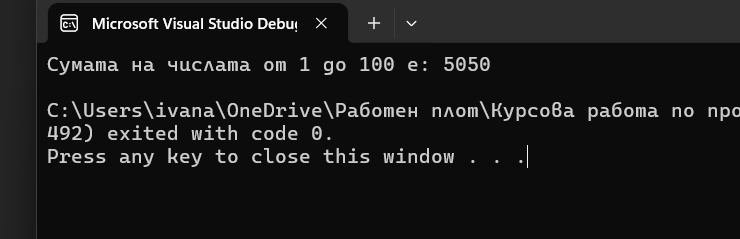
i++;

} while (i <= 100) // Условието за продължаване на цикъл

cout << "Сумата на числата от 1 до 100 е: " << sum << endl;

return 0;

}



Обяснение:

Цикълът започва от числото 1 и продължава до 100.

Всяко число се добавя към променливата sum, докато i не стане 101 (условието в while проверката вече не е вярно).

## Тема 4.5 Разлика между do-while и while

int x = 5;

while (x < 5) { // Това няма да се изпълни, защото условието е неистинно

cout << "Това няма да се изпълни!" << endl;

}

В **do-while** цикъла тялото винаги се изпълнява поне веднъж, защото условието се проверява след първото изпълнение:

int x = 5;

do {

cout << "Това ще се изпълни веднъж, дори условието да не е вярно!" << endl;

} while (x < 5); // Условието е неистинно, но тялото вече се е изпълнило веднъж

*Пример 3: Меню за избор*

Ето пример за do-while, който се използва за контрол на меню, докато потребителят не избере да излезе:

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int choice;

do {

cout << "Меню:" << endl;

cout << "1. Стартиране на игра" << endl;

cout << "2. Настройки" << endl;

cout << "3. Изход" << endl;

cout << "Изберете опция: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Стартиране на играта..." << endl;

break;

case 2:

cout << "Отваряне на настройките..." << endl;

break;

case 3:

cout << "Изход от програмата..." << endl;

break;

default:

cout << "Невалиден избор. Опитайте отново." << endl;

}

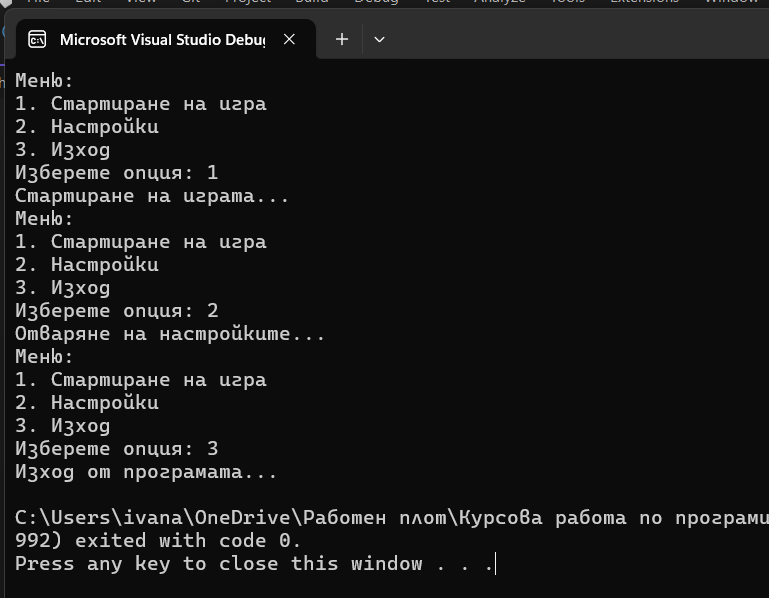
} while (choice != 3); // Цикълът се изпълнява, докато потребителят не избере изход

return 0;

}

Обяснение:

* Менюто се показва многократно, докато потребителят не избере опцията "Изход" (опция 3).
* Тялото на цикъла винаги се изпълнява поне веднъж, което позволява на потребителя да вижда менюто.



### **Заключение**

Цикълът **do-while** е полезен, когато искате тялото на цикъла да се изпълни **поне веднъж**, независимо от условието. Това го прави подходящ за задачи като въвеждане на данни, контрол на меню или изпълнение на действия, които трябва да бъдат извършени поне веднъж преди проверка на условието.

## Тема 4.6 Задачи с цикли

### 1. Отпечатване на числата от 1 до 10

**Описание**: Напишете програма, която използва цикъл, за да отпечата числата от 1 до 10.

#include <iostream>

using namespace std;

int main() {

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

cout << i << " ";

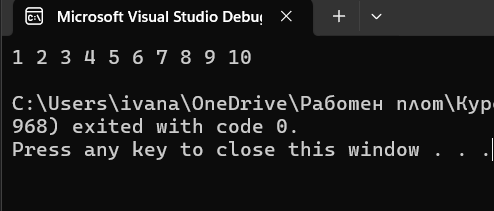
}

cout << endl;

return 0;

}

*Тест на кода:*



### 2. Сума на числата от 1 до n

**Описание**: Напишете програма, която използва цикъл, за да намери сумата на всички числа от 1 до дадено число n, въведено от потребителя.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n, sum = 0;

cout << "Въведете число: ";

cin >> n;

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

sum += i;

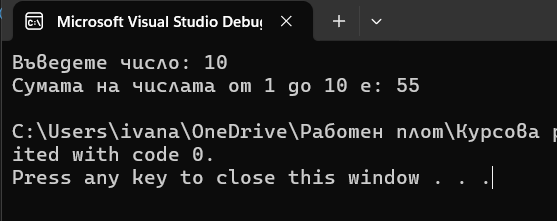
}

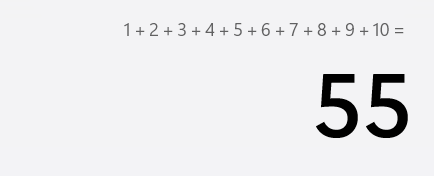
cout << "Сумата на числата от 1 до " << n << " е: " << sum << endl;

return 0;

}

Тест на кода:





### 3. Таблица за умножение

**Описание**: Напишете програма, която използва вложени цикли, за да отпечата таблица за умножение от 1 до 10.

#include <iostream>

using namespace std;

int main() {

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

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

cout << i \* j << "\t";

}

cout << endl;

}

return 0;

}

Тест на кода:

### 

### 4. Отпечатване на числов триъгълник

**Описание**: Напишете програма, която приема цяло число n и отпечатва триъгълник от числа с n реда. Всеки ред трябва да съдържа поредица от числа до номера на реда.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n;

cout << "Въведете брой редове: ";

cin >> n;

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

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

cout << j << " ";

}

cout << endl;

}

return 0;

}

Тест на кода:



### 5. Броене на цифри в число

**Описание**: Напишете програма, която приема цяло число и изчислява броя на цифрите в това число.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n, count = 0;

cout << "Въведете число: ";

cin >> n;

if (n == 0) count = 1;

while (n != 0) {

n /= 10;

count++;

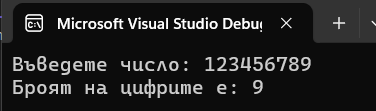
}

cout << "Броят на цифрите е: " << count << endl;

return 0;

}

Тест на кода:



### 6. Проверка дали число е палиндром

**Описание**: Напишете програма, която проверява дали дадено число е палиндром. Палиндром е число, което се чете еднакво отпред назад и обратно.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n, original, reversed = 0;

cout << "Въведете число: ";

cin >> n;

original = n;

while (n != 0) {

int digit = n % 10;

reversed = reversed \* 10 + digit;

n /= 10;

}

if (original == reversed) {

cout << original << " е палиндром." << endl;

}

else {

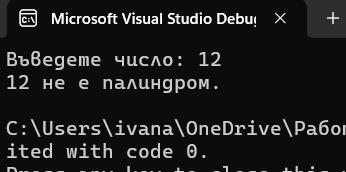
cout << original << " не е палиндром." << endl;

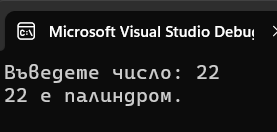
}

return 0;

}

Тест на кода:





### 7. Принтиране на ромб от звездички

**Описание**: Напишете програма, която принтира ромб от звездички с n реда, като n се въвежда от потребителя.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n;

cout << "Въведете брой редове: ";

cin >> n;

// Горната част на ромба

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

for (int j = i; j < n; j++) cout << " ";

for (int k = 1; k <= (2 \* i - 1); k++) cout << "\*";

cout << endl;

}

// Долната част на ромба

for (int i = n - 1; i >= 1; i--) {

for (int j = n; j > i; j--) cout << " ";

for (int k = 1; k <= (2 \* i - 1); k++) cout << "\*";

cout << endl;

}

return 0;

}

***Горната част на ромба***

Тази част използва два вложени цикъла, за да създаде горната половина на ромба.

* **Външен цикъл** for (int i = 1; i <= n; i++):
  + i отговаря за текущия ред. Започва от 1 и се увеличава до n.
* **Първи вътрешен цикъл** for (int j = i; j < n; j++) cout << " ";:
  + Този цикъл отпечатва интервали, за да подравни звездичките в центъра на ромба. Количеството на интервалите намалява с увеличаване на редовете (i), за да се получи формата на ромба.
* **Втори вътрешен цикъл** for (int k = 1; k <= (2 \* i - 1); k++) cout << "\*";:
  + Този цикъл отпечатва звездички, като техният брой се увеличава с всяко преминаване на цикъла. Количеството на звездичките на всеки ред се изчислява с формулата (2 \* i - 1), за да се осигури правилната форма на ромба.

Генериране на долната част на ромба

* Тази част генерира долната половина на ромба, като използва подход, подобен на горната половина.
  + **Външен цикъл** for (int i = n - 1; i >= 1; i--):
    - Започва от n - 1 и се намалява до 1, за да се създаде правилната форма на долната част на ромба.
  + **Първи вътрешен цикъл** for (int j = n; j > i; j--) cout << " ";:
    - Отпечатва интервали, така че звездичките да са правилно подравнени и центрирани. Количеството на интервалите се увеличава с намаляване на i, което придава ромбовидната форма.
  + **Втори вътрешен цикъл** for (int k = 1; k <= (2 \* i - 1); k++) cout << "\*";:
    - Отпечатва звездички, като техният брой също намалява с намаляването на i, което завършва ромбовидната форма.
  + cout << endl; завършва всеки ред за правилно подравняване на следващия ред.

***Пример:***



# Тема 5. Статични масиви

Масивите в езика C++ са структури от данни, които позволяват съхраняване на множество елементи от един и същ тип под едно общо име. Всеки елемент в масива е идентифициран с уникален индекс, който започва от 0. Масивите могат да бъдат от различни типове: цели числа, символи, числа с плаваща запетая и др.

## 5.1 Едномерен статичен масив

### **1. Деклариране на масиви**

Масив в C++ може да бъде деклариран чрез посочване на типа на елементите и размера на масива в квадратни скоби. Например:

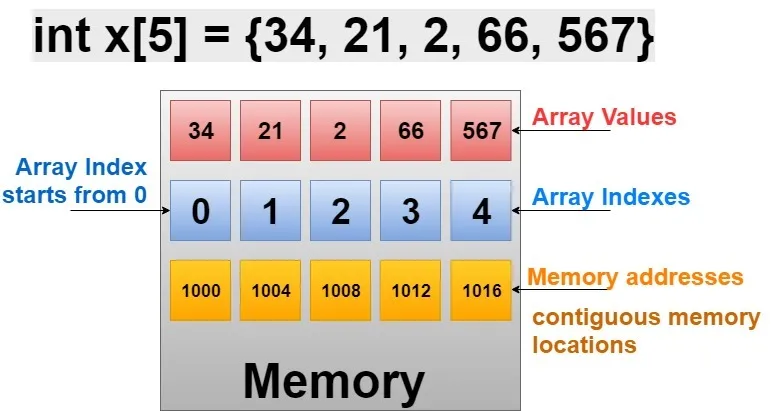
int arr[5]; // Масив от 5 цели числа

Тук arr е масив от 5 цели числа, като всеки елемент е от тип int. Масивът по подразбиране не се инициализира и съдържа произволни стойности.

### **2.Инициализиране на масиви**

Можем да инициализираме масив при неговото създаване:

int arr[5] = {1, 2, 3, 4, 5}; // Масив с 5 елемента, всеки инициализиран със съответната стойност



Може също да пропуснем размера на масива, ако зададем стойностите му при инициализация:

int arr[] = {1, 2, 3}; // Размерът на масива автоматично ще бъде 3

### **3. Дължина на статичен масив**

За да намерим дължината на статичен масив, можем да използваме следния подход:

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

int length = sizeof(arr) / sizeof(arr[0]);

Тук:

* **sizeof(arr)** връща общия размер в байтове на целия масив.
* **sizeof(arr[0])** връща размера на първия елемент (в случая размера на int, което обикновено е 4 байта).
* Делението на тези две стойности дава броя на елементите в масива.

Примерен код:

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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

int length = sizeof(arr) / sizeof(arr[0]);

cout << "Дължината на масива е: " << length << endl;

return 0;

}

**Важно**: Този метод работи **само за статични масиви**, декларирани в рамките на същата функция. Ако предадем масива като аргумент на функция, той ще се третира като указател и този метод няма да работи.

### **4. Достъп до елементи на масив**

Можете да достъпвате елементите на масива чрез индекс. Индексът започва от 0, така че за масив с размер 5, валидните индекси са от 0 до 4.

cout << arr[0]; // Първият елемент

arr[1] = 10; // Променяме стойността на втория елемент

### **5. Итерация през масив**

Можем да използваме цикъл, за да преминем през всички елементи на масива:

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

cout << arr[i] << " ";

}

### **Масиви и функции**

Можем да предаваме масиви като параметри на функции. Важно е да се отбележи, че масивите се предават по указател (адрес), т.е. функцията ще получи достъп до оригиналния масив.

Пример за предаване на масив към функция:

void printArray(int arr[], int size) {

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

cout << arr[i] << " ";

}

}

int main() {

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

printArray(arr, 5);

return 0;

}

### **Основни свойства на масивите:**

1. **Фиксиран размер**: Веднъж деклариран, размерът на масива не може да бъде променян.
2. **Тип на елементите**: Всички елементи в масива трябва да бъдат от един и същ тип.
3. **Низове като масиви от символи**: Низовете в C++ могат да бъдат представени като масиви от символи с добавен \0 терминатор.

char str[] = "Hello";

Това създава масив от символи, където последният символ е \0 (нулевият байт), който маркира края на низа.

## 5.2 Задачи с едномерни статични масиви

### Задача 1: Намиране на максимален и минимален елемент в масив

**Описание:** Напиши програма, която иска от потребителя да въведе N на брой цели числа и ги съхранява в статичен масив. След това програмата намира и извежда максималния и минималния елемент от масива.

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

const int N = 5; // Размер на масива

int arr[N];

// Въвеждане на елементите

cout << "Въведете " << N << " елемента: " << endl;

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

cin >> arr[i];

}

int max = arr[0];

int min = arr[0];

// Намиране на максимален и минимален елемент

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

if (arr[i] > max) {

max = arr[i];

}

if (arr[i] < min) {

min = arr[i];

}

}

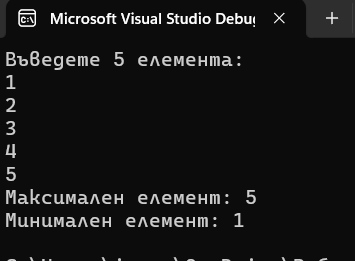
cout << "Максимален елемент: " << max << endl;

cout << "Минимален елемент: " << min << endl;

return 0;

}

Тест на кода:



### Задача 2: Сумиране на елементите на масив

**Описание:** Напиши програма, която събира всички елементи на статичен масив от N цели числа и изкарва сумата им .

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

const int N = 5; // Размер на масива

int arr[N];

// Въвеждане на елементите

cout << "Въведете " << N << " елемента: " << endl;

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

cin >> arr[i];

}

int sum = 0;

// Изчисляване на сумата

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

sum += arr[i];

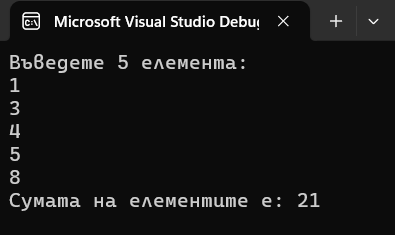
}

cout << "Сумата на елементите е: " << sum << endl;

return 0;

}

Тест на кода:



### Задача 3: Средно аритметично на елементите в масив

Описание: Напиши програма, която изчислява средното аритметично на всички елементи в масива.

#include <iostream>

#include<windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

const int N = 5; // Размер на масива

int arr[N];

// Въвеждане на елементите

cout << "Въведете " << N << " елемента: " << endl;

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

cin >> arr[i];

}

int sum = 0;

// Изчисляване на сумата

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

sum += arr[i];

}

double average = static\_cast<double>(sum) / N;

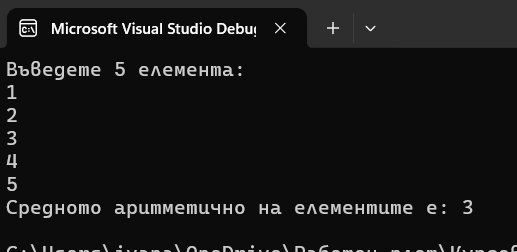
cout << "Средното аритметично на елементите е: " << average << endl;

return 0;

}

Тук изчисляваме средното аритметично, като делим sum на N. Използваме ***static\_cast<double>(sum)*** за да се уверим, че делението е извършено с плаваща запетая, а не с цели числа, което би могло да доведе до загуба на точност.

Тест на кода:



### Задача 4: Обръщане на масив

**Описание:** Напиши програма, която обръща реда на елементите в статичен масив.

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

const int N = 5; // Размер на масива

int arr[N];

// Въвеждане на елементите

cout << "Въведете " << N << " елемента: " << endl;

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

cin >> arr[i];

}

// Обръщане на масива

int reversed[N]; // Използваме същия размер като arr

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

reversed[i] = arr[N - 1 - i];

}

// Извеждане на обърнатия масив

cout << "Обърнат масив: ";

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

cout << reversed[k] << " ";

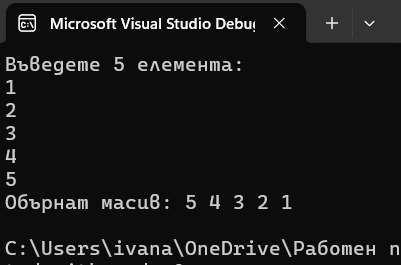
}

cout << endl;

return 0;

}

Тест на кода:



## 5.3 Многомерни масиви

Многомерните масиви в C++ представляват масиви, които съдържат други масиви като елементи. Най-често използваният тип многомерни масиви е двумерен масив, който можем да разглеждаме като таблица или матрица. Въпреки това можем да имаме и тримерни или дори повече измерения, в зависимост от нуждите.

### **1.Деклариране на многомерен масив**

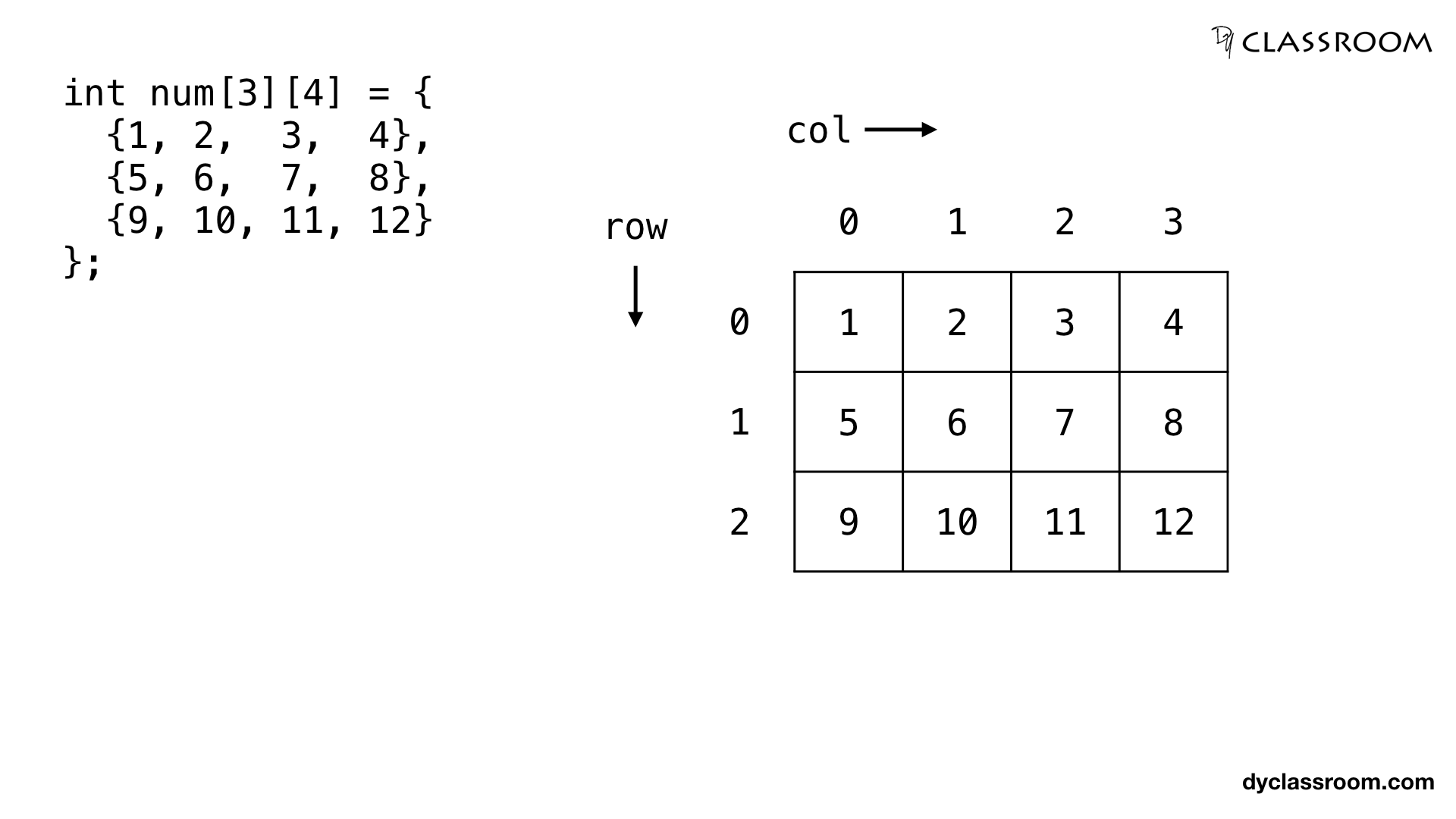
За да декларираме многомерен масив, трябва да посочим броя на измеренията и размера на всяко измерение в квадратни скоби.

#### **Двумерен масив (матрица)**

Двумерният масив може да се разглежда като масив от редове, всеки от които съдържа няколко колони. Например:

int matrix[3][4]; // Декларира двумерен масив с 3 реда и 4 колони

Този масив съдържа 3 реда и 4 колони, общо 12 елемента.



#### **Тримерен масив**

Тримерният масив може да се разглежда като масив от "таблици", всяка от които съдържа няколко реда и колони:

int cube[2][3][4]; // Декларира тримерен масив с 2 "таблици", всяка съдържаща 3 реда и 4 колони

### **2.Инициализиране на многомерен масив**

Можем да инициализираме многомерен масив с конкретни стойности по време на неговото създаване. При инициализацията всеки ред (или всяко подниво) се затваря в отделни фигурни скоби.

#### **Инициализиране на двумерен масив:**

int matrix[3][4] = {

{1, 2, 3, 4}, // Първи ред

{5, 6, 7, 8}, // Втори ред

{9, 10, 11, 12} // Трети ред

};

В този пример масивът съдържа 3 реда и 4 колони, като всеки елемент е инициализиран със стойност.

#### **Инициализиране на тримерен масив:**

int cube[2][2][3] = {

{{1, 2, 3}, {4, 5, 6}}, // Първа таблица

{{7, 8, 9}, {10, 11, 12}} // Втора таблица

};

Тук имаме тримерен масив с 2 таблици, всяка съдържаща 2 реда и 3 колони.

### **3.Достъп до елементи на многомерен масив**

За да достъпим конкретен елемент от многомерен масив, трябва да използваме индексите за всяко от измеренията.

Достъп до елемент в двумерен масив:

int value = matrix[1][2]; // Достъп до елемента от втория ред и третата колона (стойност 7)

Достъп до елемент в тримерен масив:

int value = cube[1][0][2]; // Достъп до елемента в таблица 2, ред 1, колона 3 (стойност 9)

### **4.Итерация през многомерен масив**

Можем да използваме **вложени цикли**, за да итерираме през елементите на многомерен масив. За двумерен масив, ще ни трябват два цикъла — един за редовете и един за колоните.

#include <iostream>

using namespace std;

int main() {

int matrix[3][4] = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12}

};

for (int i = 0; i < 3; i++) { // Цикъл за редовете

for (int j = 0; j < 4; j++) { // Цикъл за колоните

cout << matrix[i][j] << " ";

}

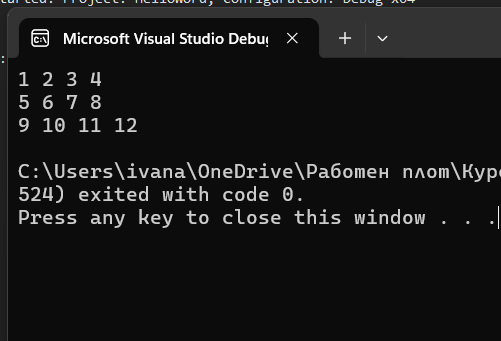
cout << endl;

}

return 0;

}

Тук използваме два вложени цикъла: единият итерира през редовете, а другият през колоните.



#### **Итерация през тримерен масив:**

За тримерен масив ще ни трябват три вложени цикъла:

#include <iostream>

using namespace std;

int main() {

int cube[2][2][3] = {

{{1, 2, 3}, {4, 5, 6}},

{{7, 8, 9}, {10, 11, 12}}

};

for (int i = 0; i < 2; i++) { // Цикъл за "таблиците"

for (int j = 0; j < 2; j++) { // Цикъл за редовете

for (int k = 0; k < 3; k++) { // Цикъл за колоните

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

}

cout << endl;

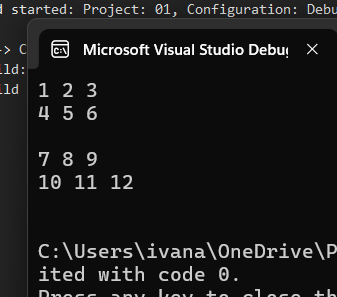
}

cout << endl;

}

return 0;

}



Многомерните масиви в C++ представляват масиви, които съдържат други масиви като елементи. Най-често използваният тип многомерни масиви е двумерен масив, който можем да разглеждаме като таблица или матрица. Въпреки това можем да имаме и тримерни или дори повече измерения, в зависимост от нуждите.

### **5.Размер на многомерен масив**

Както при едномерните масиви, можем да намерим размера на всяко измерение с помощта на **sizeof**.

Пример за двумерен масив:

int matrix[3][4];

int totalSize = sizeof(matrix); // Общ размер на масива в байтове

int rowSize = sizeof(matrix[0]); // Размер на един ред

int elementSize = sizeof(matrix[0][0]); // Размер на един елемент

int numRows = sizeof(matrix) / sizeof(matrix[0]); // Брой редове

int numCols = sizeof(matrix[0]) / sizeof(matrix[0][0]); // Брой колони

### **6.Предаване на многомерен масив на функция**

При предаване на многомерен масив като аргумент на функция трябва да посочим размера на всички измерения (с изключение на първото). Например, за двумерен масив:

void printMatrix(int matrix[3][4]) {

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

for (int j = 0; j < 4; j++) {

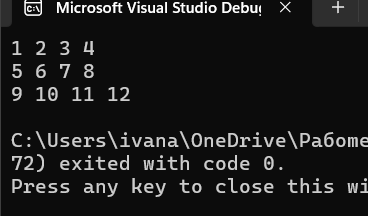
cout << matrix[i][j] << " ";

}

cout << endl;

}

}



### 5.4 Задачи с многомерен масив

### Задача 1: Сума на елементите в двоен масив

**Условие:** Даден е двумерен масив int arr[3][3]. Напишете програма, която намира сумата на всички елементи в масива.

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int arr[3][3] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int sum = 0;

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

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

sum += arr[i][j];

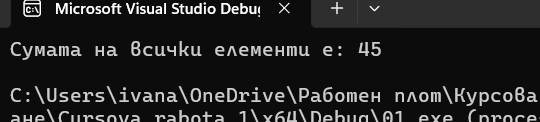
}

}

cout << "Сумата на всички елементи е: " << sum << endl;

return 0;

}



Разяснение:

Използваме два вложени цикъла for. Външният цикъл (с i) обхожда редовете, а вътрешният (с j) обхожда колоните в текущия ред. Всеки елемент се добавя към променливата sum, която в крайна сметка ще съдържа общата сума на всички елементи.

### Задача 2: Транспониране на матрица

**Условие:** Дадена е квадратна матрица n x n. Транспонирайте матрицата (разменете редовете с колоните) и я изведете.

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

const int n = 3;

int arr[n][n] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int transposed[n][n];

// Транспониране

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

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

transposed[j][i] = arr[i][j];

}

}

// Извеждане на транспонираната матрица

cout << "Транспонирана матрица:\n";

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

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

cout << transposed[i][j] << " ";

}

cout << endl;

}

return 0;

}

Разяснение:

При транспониране редовете стават колони и обратното. Създаваме нова матрица transposed, в която поставяме елементите от оригиналната матрица на разменени позиции. По този начин arr[i][j] от оригинала ще стане transposed[j][i]. След като обходим всички елементи, новата матрица transposed съдържа транспонираната версия.

### Задача 3: Намиране на максималния елемент във всяка колона

**Условие:** Дадена е двумерна матрица int arr[3][4]. Намерете и изведете максималния елемент за всяка колона.

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int arr[3][4] = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12}

};

const int rows = 3, cols = 4;

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

int maxInColumn = arr[0][j];

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

if (arr[i][j] > maxInColumn) {

maxInColumn = arr[i][j];

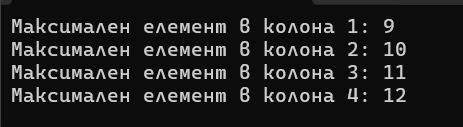
}

}

cout << "Максимален елемент в колона " << j + 1 << ": " << maxInColumn << endl;

}

return 0;}



Разяснение:

В тази задача трябва да обходим всеки елемент в една колона и да намерим най-големия от тях. Задаваме начална стойност на maxInColumn като първия елемент в колоната, след което сравняваме всички останали елементи в колоната с него и актуализираме максимума, ако намерим по-голяма стойност.

Външният цикъл с j обхожда колоните, а вътрешният цикъл с i обхожда елементите във всяка колона. Така за всяка колона изчисляваме максимума и го извеждаме.

### Задача 4: Проверка за симетрична матрица

**Условие:** Дадена е квадратна матрица n x n. Напишете програма, която проверява дали матрицата е симетрична спрямо главния диагонал.

#include <iostream>

#include <windows.h>

using namespace std;

bool isSymmetric(int arr[][3], int n) {

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

for (int j = i + 1; j < n; j++) {

if (arr[i][j] != arr[j][i]) {

return false;

}

}

}

return true;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int arr[3][3] = {

{1, 2, 3},

{2, 5, 6},

{3, 6, 9}

};

if (isSymmetric(arr, 3)) {

cout << "Матрицата е симетрична." << endl;

} else {

cout << "Матрицата не е симетрична." << endl;

}

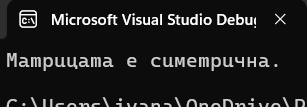
return 0;

}

Разяснение:

Трябва да проверим дали дадена квадратна матрица е симетрична спрямо главния диагонал, т.е. arr[i][j] == arr[j][i].

Симетрична матрица има еднакви стойности от двете страни на главния диагонал (който преминава от горния ляв до долния десен ъгъл). За да проверим това, обхождаме елементите над главния диагонал (затова започваме с j = i + 1) и проверяваме дали всеки arr[i][j] е равен на съответния arr[j][i]. Ако намерим разлика, матрицата не е симетрична и програмата връща false.



Задача 5: Умножение на две матрици

Условие: Дадени са две матрици A[2][3] и B[3][2]. Напишете програма, която намира произведението на тези две матрици.

#include <iostream>

#include <windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int A[2][3] = {

{1, 2, 3},

{4, 5, 6}

};

int B[3][2] = {

{7, 8},

{9, 10},

{11, 12}

};

int result[2][2] = { 0 };

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

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

result[i][j] = 0;

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

result[i][j] += A[i][k] \* B[k][j];

}

}

}

cout << "Произведение на матриците A и B:\n";

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

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

cout << result[i][j] << " ";

}

cout << endl;

}

return 0;

}

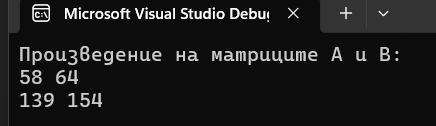
Разяснение:

Искаме да намерим произведението на две матрици A[2][3] и B[3][2]. Произведението на матрица се получава чрез умножение на всеки елемент от реда на първата с елементите от колоната на втората.

При умножение на две матрици A[m][p] и B[p][n], резултатната матрица ще има размер m x n. За да намерим елемента на позиция result[i][j], умножаваме елементите на ред i от A с елементите на колона j от B и ги сумираме.

Това изисква три вложени цикъла:

1. Външният цикъл i обхожда редовете на резултата.
2. Средният цикъл j обхожда колоните на резултата.
3. Вътрешният цикъл k умножава съответните елементи и ги добавя към result[i][j].



# Реализация на софтуеро приложение. Игра на морски шах

## Основни компоненти на кода

1. **Включване на библиотеки**:

#include <iostream>  
#include <ctime>  
#include <windows.h>

* + **#include <iostream>:** Позволява използването на вход/изход (cout, cin).
  + **#include <ctime>**: Използва се за генериране на случайни числа.
  + **#include <windows.h>**: Позволява използването на Windows API за промяна на цветовете на текста в конзолата и кирилизация на текста.

1. **Функции**: Кодът съдържа няколко функции, всяка от които играе важна роля в логиката на играта.

* **void drawBoard(char\* spaces):**

Отговорна за рисуването на текущото състояние на игралното поле. Полето се изобразява като матрица 3x3 и цветът на символите (X или O) се променя в зависимост от играча.

* **void playerMove(char\* spaces, char player):**

Позволява на играча да направи ход. Играча

въвежда число от 1 до 9, което определя позицията на хода. Функцията проверява валидността на входа и актуализира полето.

* **void computerMove(char\* spaces, char computer, char player):**

Компютърът прави ход, опитвайки се първо да спечели, след това да блокира играча и накрая да избере случайна свободна позиция.

* **bool checkWinner(char\* spaces, char player):**

Проверява дали текущият играч е спечелил. Използва набор от предварително определени комбинации за печалба.

* **bool checkTie(char\* spaces):**

Проверява дали е настъпило равенство, т.е. дали няма повече свободни полета за игра.

* **void setColor(int color):**

Променя цвета на текста в конзолата с помощта на Windows API. Използва се за различаване между ходовете на играча и компютъра.

* **void drawTitle():**

Рисува заглавието на играта.

* **bool playAgain():**

Запитва играча дали желае да играе отново и връща булева стойност (true/false).

* **void displayScores(int playerWins, int computerWins):**

Показва текущия резултат на играча и компютъра.

1. **Основна функция (int main())**:

* Основната функция стартира играта. В нея се инициализират променливите, които ще се използват за следене на хода на играта и резултатите.
* **Цикъл do-while**: Позволява играта да се играе многократно, докато играчът желае.
* В цикъла се рисува дъската, получават се ходове от играча и компютъра, проверяват се условията за победа или равенство и накрая се показват резултатите.

1. **Изпълнение на логиката**:

* Всяка итерация от основния цикъл започва с изчистване на игралното поле.
* Играчът прави ход, следван от проверка за победа или равенство.
* Компютърът следва, а след всяко ходене се проверяват условията отново.

#### Важни моменти в кода

* **Цветове**: Играта използва различни цветови настройки, за да предостави визуална обратна връзка. Ходовете на играча са в жълт цвят, а печелившите съобщения са в зелено за играча или червено за компютъра.
* **Случайност**: За компютъра се използва **srand(time(0))**, за да се инициализира генератора на случайни числа, което прави ходовете на компютъра непредсказуеми.
* **Обработка на грешки**: Има проверка за невалиден вход, която осигурява, че играчът въвежда валидни числа.

***Ще обясня подробно всяка функция в кода и как точно работи, за да бъде по-лесно разбрана логиката на играта.***

* Функция setColor(int color)

void setColor(int color) {  
 SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), color);  
}

* **Цел**: Променя цвета на текста в конзолата.
* Функция drawTitle()

void drawTitle() {  
 setColor(14);  
 cout << "===========================" << endl;  
 cout << " Морски шах " << endl;  
 cout << "===========================" << endl;  
 setColor(7);  
}

* **Цел**: Показва заглавието на играта в конзолата.
* **Как работи**:
  + Задава жълт цвят за текста чрез setColor(14).
  + Извежда заглавието на играта.
  + Връща цвета на нормалния (бял) със setColor(7).



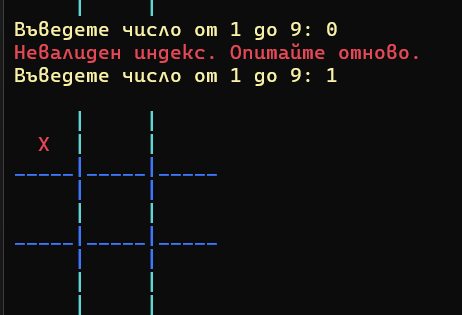
* Функция drawBoard(char\* spaces)
* Цел: Изобразява текущото състояние на игралното поле.
* Как работи:
* Преминава през всички позиции на полето и извежда съдържанието на всяка клетка от масива spaces.
* spaces[i] е 'X' или 'O' или празно пространство (' ').
* Задава различен цвят на символите в зависимост от това дали е ход на играча (X) или на компютъра (O).
* Всеки ред на полето се изобразява с вертикални и хоризонтални линии.



* Функция playerMove(char\* spaces, char player)

void playerMove(char\* spaces, char player) {  
 int number;  
 do {  
 setColor(14);   
 cout << "Въведете число от 1 до 9: ";  
 cin >> number;  
 number--;  
 if (number >= 0 && number <= 8 && spaces[number] == ' ') {  
 spaces[number] = player;  
 break;  
 }  
 else {  
 setColor(12);  
 cout << "Невалиден индекс. Опитайте отново." << endl;  
 setColor(7);  
 }  
 } while (true);  
}

* **Цел**: Позволява на играча да направи ход.
* **Как работи**:
  + Подканя играча да въведе число от 1 до 9, което представлява позиция на полето.
  + Намалява number с 1, за да съвпадне с индексите на масива (0 до 8).
  + Проверява дали индексът е валиден (числото е между 0 и 8 и клетката е празна).
  + Ако клетката е свободна, задава символа на играча ('X') на съответната позиция.
  + Ако клетката не е свободна или е въведено невалидно число, извежда съобщение за грешка и играчът трябва да въведе отново.

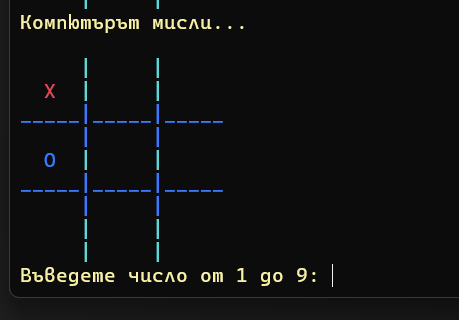


* Функция computerMove(char\* spaces, char computer, char player)

void computerMove(char\* spaces, char computer, char player) {  
 int number;  
 srand(time(0));  
 setColor(14);  
 cout << "Компютърът мисли..." << endl;  
 Sleep(1000);  
 //Проверява за печеливш ход  
 for (int i = 0; i < 9; i++) {  
 if (spaces[i] == ' ') {  
 spaces[i] = computer;  
 if (checkWinner(spaces, computer)) {  
 setColor(7);  
 return;  
 }  
 spaces[i] = ' ';  
 }  
 }  
 //Блокира печелившия ход на играча  
 for (int i = 0; i < 9; i++) {  
 if (spaces[i] == ' ') {  
 spaces[i] = player;  
 if (checkWinner(spaces, player)) {  
 spaces[i] = computer;  
 setColor(7);  
 return;  
 }  
 spaces[i] = ' ';  
 }  
 }  
 while (true) {  
 number = rand() % 9;  
 if (spaces[number] == ' ') {  
 spaces[number] = computer;  
 break;  
 }  
}

setColor(7);  
}

* **Цел**: Прави ход за компютъра.
* **Как работи**:
  + Опитва се първо да направи печеливш ход (ако има такъв) чрез проверка на всички празни позиции.
  + Ако не може да спечели, проверява дали играчът има печеливш ход и ако е така, го блокира.
  + Ако няма нужда от блокиране, избира случайна свободна клетка, където да постави своя символ ('O').



* Функция bool checkWinner(char\* spaces, char player)

bool checkWinner(char\* spaces, char player) {  
 char winCombos[8][3] = { {0, 1, 2}, {3, 4, 5}, {6, 7, 8}, {0, 3, 6}, {1, 4, 7}, {2, 5, 8}, {0, 4, 8}, {2, 4, 6} };  
 for (auto& comb : winCombos) {

if (spaces[comb[0]] != ' ' && spaces[comb[0]] == spaces[comb[1]] && spaces[comb[1]] == spaces[comb[2]]) {  
 return true;  
 }  
 }  
 return false;  
}

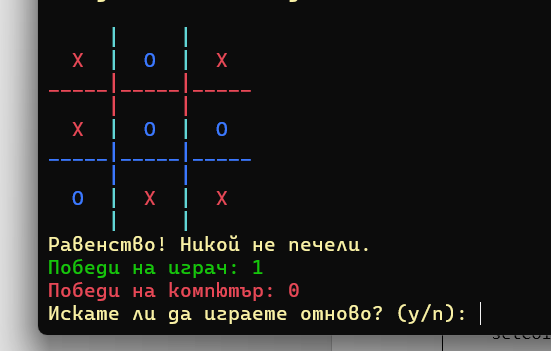
* **Цел**: Проверява дали даден играч е спечелил играта.
* **Как работи**:
  + Преглежда всички възможни печеливши комбинации (редове, колони и диагонали).
  + За всяка комбинация проверява дали същият символ се намира и в трите клетки.
  + Ако има съвпадение, връща true, в противен случай false.
* Функция bool checkTie(char\* spaces)

bool checkTie(char\* spaces) {  
 for (int i = 0; i < 9; i++) {  
 if (spaces[i] == ' ') {  
 return false;  
 }  
 }  
 setColor(14);  
 cout << "Равенство! Никой не печели." << endl;

setColor(7);

return true;  
}

* **Цел**: Проверява дали играта е завършила с равенство.
* **Как работи**:
  + Проверява всички клетки на полето.
  + Ако има поне една празна клетка (' '), играта не е завършила и връща false.
  + Ако всички клетки са заети и няма победител, връща true.

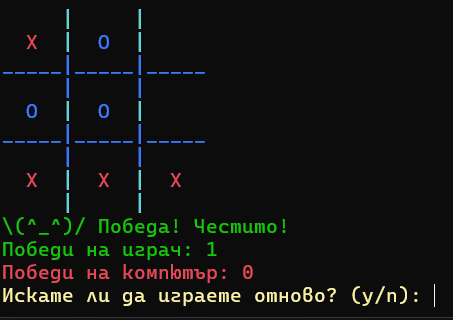


* Функция bool playAgain()

bool playAgain() {  
 char choice;  
 setColor(14);  
 cout << "Искате ли да играете отново? (y/n): ";  
 cin >> choice;  
 setColor(7);  
 return choice == 'y' || choice == 'Y';  
}

* **Цел**: Запитва играча дали иска да играе отново.
* **Как работи**:
  + Изчаква отговор от потребителя.
  + Ако отговорът е y или Y, връща true, в противен случай връща false.
* Функция void displayScores(int playerWins, int computerWins)

void displayScores(int playerWins, int computerWins) {  
 setColor(10);  
 cout << "Победи на играч: " << playerWins << endl;  
 setColor(12);  
 cout << "Победи на компютър: " << computerWins << endl;  
 setColor(7);  
}

* **Цел**: Показва резултатите от игрите на играча и компютъра.
* **Как работи**:
  + Извежда броя на победите на играча и компютъра в различни цветове.
  + Връща цветовете към нормален след извеждане.
  + 

## Код на програмата :

#include <iostream>

#include <ctime>

#include <windows.h>

using namespace std;

void drawBoard(char\* spaces);//Функция, която ще рисува текущото положение на дъската

void playerMove(char\* spaces, char player);//Функция, която позволава на играча да направи ход

void computerMove(char\* spaces, char computer, char player);//избор на ход на компютъра

bool checkWinner(char\* spaces, char player);//Проверява има ли победител

bool checkTie(char\* spaces);//Проверява за равинство

void setColor(int color);//Променя цвета

void drawTitle();//Рисува заглавието, извиква се само веднъж

bool playAgain();//Дава опция за още една игра

void displayScores(int playerWins, int computerWins);//Показва резултатите от игрите

void setColor(int color) {

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), color);

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char player = 'X';

char computer = 'O';

bool running = true;

int playerWins = 0;

int computerWins = 0;

do {

char spaces[9] = { ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' };

drawTitle();

drawBoard(spaces);

while (running) {

playerMove(spaces, player);

drawBoard(spaces);

if (checkWinner(spaces, player)) {

setColor(10);

cout << "\\(^\_^)/ Победа! Честито!" << endl;

setColor(7);

playerWins++;

running = false;

break;

}

else if (checkTie(spaces)) {

running = false;

break;

}

computerMove(spaces, computer, player);

drawBoard(spaces);

if (checkWinner(spaces, computer)) {

setColor(12);

cout << "(T\_T) Компютърът печели! Опитайте отново." << endl;

setColor(7);

computerWins++;

running = false;

break;

}

else if (checkTie(spaces)) {

running = false;

break;

}

}

displayScores(playerWins, computerWins);

running = playAgain();

} while (running);

setColor(10);

cout << "Благодаря, че играхте :) ";

setColor(7);

return 0;

}

void drawTitle() {

setColor(14);

cout << "===========================" << endl;

cout << " Морски шах " << endl;

cout << "===========================" << endl;

setColor(7);

}

void drawBoard(char\* spaces) {

cout << endl;

setColor(11);

cout << " | | " << endl;

cout << " ";

setColor(spaces[0] == 'X' ? 12 : 9);

cout << spaces[0];

setColor(11);

cout << " | ";

setColor(spaces[1] == 'X' ? 12 : 9);

cout << spaces[1];

setColor(11);

cout << " | ";

setColor(spaces[2] == 'X' ? 12 : 9);

cout << spaces[2] << " " << endl;

cout << "\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_" << endl;

cout << " | | " << endl;

cout << " ";

setColor(spaces[3] == 'X' ? 12 : 9);

cout << spaces[3];

setColor(11);

cout << " | ";

setColor(spaces[4] == 'X' ? 12 : 9);

cout << spaces[4];

setColor(11);

cout << " | ";

setColor(spaces[5] == 'X' ? 12 : 9);

cout << spaces[5] << " " << endl;

cout << "\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_" << endl;

cout << " | | " << endl;

cout << " ";

setColor(spaces[6] == 'X' ? 12 : 9);

cout << spaces[6];

setColor(11);

cout << " | ";

setColor(spaces[7] == 'X' ? 12 : 9);

cout << spaces[7];

setColor(11);

cout << " | ";

setColor(spaces[8] == 'X' ? 12 : 9);

cout << spaces[8] << " " << endl;

setColor(11);

cout << " | | " << endl;

setColor(7); //Връща нармалния цвят

}

void playerMove(char\* spaces, char player) {

int number;

do {

setColor(14); // Задава цвета да е жълт

cout << "Въведете число от 1 до 9: ";

cin >> number;

number--; // Изваждаме единица от входното число, тъй като масивите започват броенето си от 0

if (number >= 0 && number <= 8 && spaces[number] == ' ') //Ако въведеният индекс е валиден и на него в масива няма нищо, прибавяме хода на играча ицикъла приключва

{

spaces[number] = player;

break;

}

else {

setColor(12); //Съобщението за грешен инпут ще е в червено

cout << "Невалиден индекс. Опитайте отново." << endl;

setColor(7);

}

} while (true);

}

void computerMove(char\* spaces, char computer, char player) {

int number;

srand(time(0)); // избира случайно число

setColor(14);

cout << "Компютърът мисли..." << endl;

Sleep(1000); // Забавя програмата

//Проверява за печеливш ход

for (int i = 0; i < 9; i++) {

if (spaces[i] == ' ') {

spaces[i] = computer;

if (checkWinner(spaces, computer)) {

setColor(7);

return;

}

spaces[i] = ' ';

}

}

//Блокира печелившия ход на играча

for (int i = 0; i < 9; i++) {

if (spaces[i] == ' ') {

spaces[i] = player;

if (checkWinner(spaces, player)) {

spaces[i] = computer;

setColor(7);

return;

}

spaces[i] = ' ';

}

}

//Ако няма печеливш избира случаен под

while (true) {

number = rand() % 9;

if (spaces[number] == ' ') {

spaces[number] = computer;

break;

}

}

setColor(7); //Връща нормалния цвят

}

bool checkWinner(char\* spaces, char player) {

char winCombos[8][3] = { {0, 1, 2}, {3, 4, 5}, {6, 7, 8}, {0, 3, 6}, {1, 4, 7}, {2, 5, 8}, {0, 4, 8}, {2, 4, 6} };

for (auto& comb : winCombos) {

if (spaces[comb[0]] != ' ' && spaces[comb[0]] == spaces[comb[1]] && spaces[comb[1]] == spaces[comb[2]]) {

return true;

}

}

return false;

}

bool checkTie(char\* spaces) {

for (int i = 0; i < 9; i++) {

if (spaces[i] == ' ') {

return false;

}

}

setColor(14); // Жълто за съобщението

cout << "Равенство! Никой не печели." << endl;

setColor(7); // Връщаме цвета

return true;

}

bool playAgain() {

char choice;

setColor(14); // Жълто за съобщението

cout << "Искате ли да играете отново? (y/n): ";

cin >> choice;

setColor(7); // Връщаме цвета

return choice == 'y' || choice == 'Y';

}

void displayScores(int playerWins, int computerWins) {

setColor(10); // Зелено за играча

cout << "Победи на играч: " << playerWins << endl;

setColor(12); // Червено за компютъра

cout << "Победи на компютър: " << computerWins << endl;

setColor(7); // Връщаме цвета

}