**Ръководство**

„C++ за начинаещи: *Стъпка по стъпка към успеха*“

автор: Стилиян Илиев

фн: 27

използван LLM: ChatGPT

Съдържание

[Тема 1 – Въведение в програмирането с C++ 3](#_Toc178518546)

[Тема 1.1 – Какво е C++ и защо да го учим? 3](#_Toc178518547)

[Тема 1.2 – Инсталиране и настройка на среда за разработка 3](#_Toc178518548)

[Тема 2 – Основи на синтаксиса в C++ 4](#_Toc178518549)

[Тема 2.1 - Променливи и типове данни 4](#_Toc178518550)

[Тема 2.2 – Оператори и изрази 6](#_Toc178518550)

[Тема 2.3 – Вход и изход в C++ 7](#_Toc178518550)

[Тема 3 – Управляващи конструкции 8](#_Toc178518549)

[Тема 3.1 – Условни оператори 8](#_Toc178518550)

[Тема 3.2 – Цикли 11](#_Toc178518550)

[Тема 4 – Функции в C++ 13](#_Toc178518549)

[Тема 4.1 – Дефиниране и извикване на функции 13](#_Toc178518550)

[Тема 4.2 – Рекурсия ……………………………………………………………………...15](#_Toc178518550)

[Тема 5 – Масиви и низове 16](#_Toc178518549)

[Тема 5.1 – Масиви 16](#_Toc178518550)

[Тема 5.2 – Работа с низове 18](#_Toc178518550)

[Тема 6 – Указатели и динамична памет 21](#_Toc178518549)

[Тема 6.1 – Указатели 21](#_Toc178518550)

[Тема 6.2 – Динамично управление на паметта 22](#_Toc178518550)

[Тема 7 – Обектно-ориентирано програмиране (ООП) 25](#_Toc178518549)

[Тема 7.1 – Основи на ООП 25](#_Toc178518550)

[Тема 7.2 – Наследяване и полиморфизъм 2](#_Toc178518550)6

[Тема 7.3 – Инкапсулация 2](#_Toc178518550)7

[Тема 7.4 – Абстракция 2](#_Toc178518550)8

[Тема 8 – Стандартната библиотека на C++ (STL) 28](#_Toc178518549)

[Тема 8.1 – Контейнери в STL 2](#_Toc178518550)8

[Тема 8.2 – Алгоритми в STL 30](#_Toc178518550)

[Тема 8.3 – Итератори и тяхното приложение 31](#_Toc178518550)

[Тема 9 – Файлове и работа с тях 33](#_Toc178518549)

[Тема 9.1 – Четене и писане във файлове 33](#_Toc178518550)

[Тема 9.2 – Работа с текстови и двоични файлове 35](#_Toc178518550)

[Тема 10 – Теми за напреднали 37](#_Toc178518549)

[Тема 10.1 – Шаблони в C++ и тяхното приложение 37](#_Toc178518550)

[Тема 10.2 – Обработка на изключения и управление на грешки 39](#_Toc178518550)

Тема 10.3 – Паралелно програмиране и многопоточност ……………………..……………………………….41

[Тема 11 – Практическо приложение – Финален проект 45](#_Toc178518549)

[Тема 11.1 – Описание на проекта 45](#_Toc178518550)

[Тема 11.2 – Основни функции и код 45](#_Toc178518550)

Тема 11.3 – Екрани от изпълнението ……………………………………………………………………………………….45

**Тема 1:** Въведение в програмирането с C++

**1.1** Какво е C++ и защо да го учим?

C++ е един от най-популярните и мощни програмни езици, използван за разработка на софтуер, игри, операционни системи и други. Езикът е създаден като разширение на езика C от Бярне Строуструп през 1979 година и добавя обектно-ориентирано програмиране към вече съществуващите функции на C. Изучаването на C++ осигурява солидна основа за разбиране на програмирането като цяло, както и важни концепции като управление на паметта и ефективност на кода.

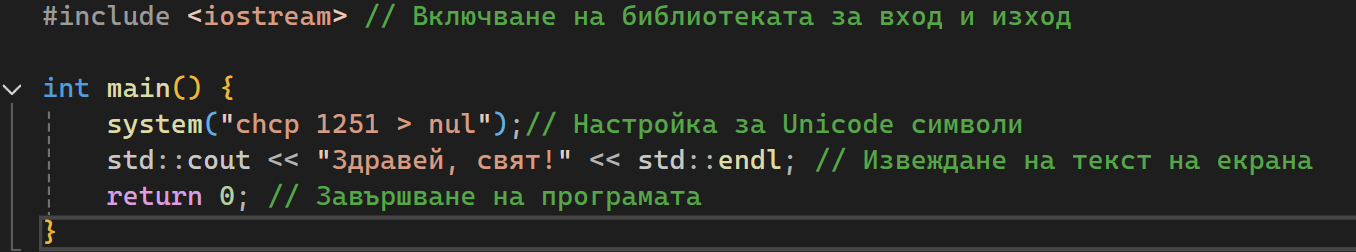
**Основни предимства на C++:**

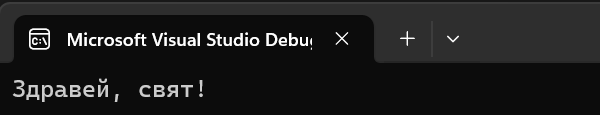
* **Висока производителност**: Подходящ за приложения, които изискват бърза и оптимизирана работа.
* **Гъвкавост и мащабируемост**: Може да се използва за малки програми и големи софтуерни системи.
* **Широко приложение**: От системно програмиране до разработка на игри и графични приложения.

**1.2** Инсталиране и настройка на среда за разработка

За да започнеш с програмирането на C++, ще ти трябва компилатор и среда за разработка (IDE). Някои от популярните опции включват:

* **Visual Studio**: Изключително мощна IDE, особено за Windows.
* **Code::Blocks**: Лека и лесна за настройка среда.
* **GCC**: Популярен компилатор, част от GNU проектите, който може да се използва с текстов редактор.

***Примерен код за първата програма на C++:***

***Изход на конзолата:***

***Обяснение:***

* **#include <iostream>: Инструктира компилатора да включи библиотеката за вход и изход.**
* **int main(): Главната функция, която се изпълнява при стартиране на програмата.**
* **std::cout << "Здравей, свят!" << std::endl;: Команда за извеждане на текст.**
* **return 0;: Завършва изпълнението на програмата и връща стойност към операционната система.**

**-----------------------------------------------------------------------------------------------------------------**

**Тема 2:** Основи на синтаксиса в C++

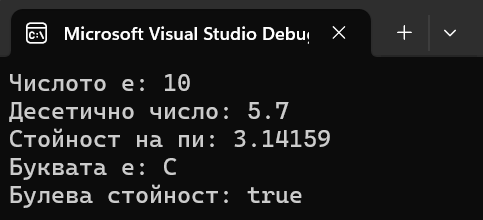
**2.1** Променливи и типове данни

В C++ променливите се използват за съхраняване на данни, които могат да бъдат манипулирани по време на изпълнение на програмата. Всяка променлива има тип, който определя какви стойности може да приема и какви операции могат да се извършват с нея.

**Примери за основни типове данни:**

* **int – цели числа (напр. int a = 5;)**
* **float – дробни числа с плаваща запетая (напр. float b = 5.3;)**
* **double – числа с двойна точност (напр. double c = 3.14159;)**
* **char – единичен символ (напр. char letter = 'A';)**
* **bool – булеви стойности (true или false)**

***Примерен код с различни променливи:***

***Изход на конзолата:***

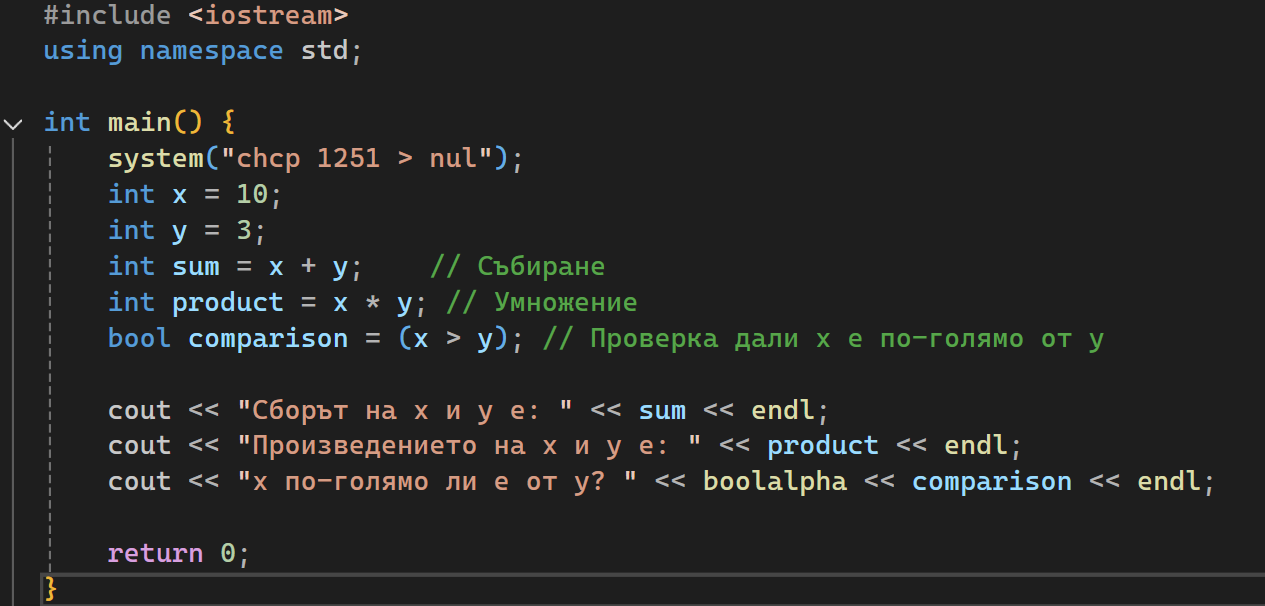
***Обяснение:***

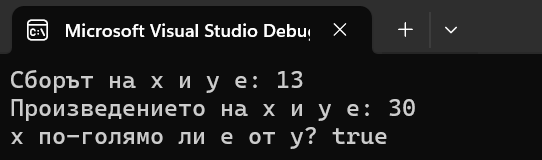
* **Програмата създава променливи от различни типове и ги извежда с помощта на std::cout.**
* **Ключовата дума std::boolalpha форматира булевите стойности като true или false.**

**2.2** Оператори и изрази

C++ поддържа различни видове оператори, които позволяват извършването на изчисления и сравнения. Някои от най-използваните са:

* **Аритметични оператори: +, -, \*, /, %**
* **Логически оператори: && (логическо И), || (логическо ИЛИ), ! (отрицание)**
* **Релационни оператори: ==, !=, <, >, <=, >=**
* **Оператори за присвояване: =, +=, -=, \*=, /=**

***Пример за използване на оператори:***

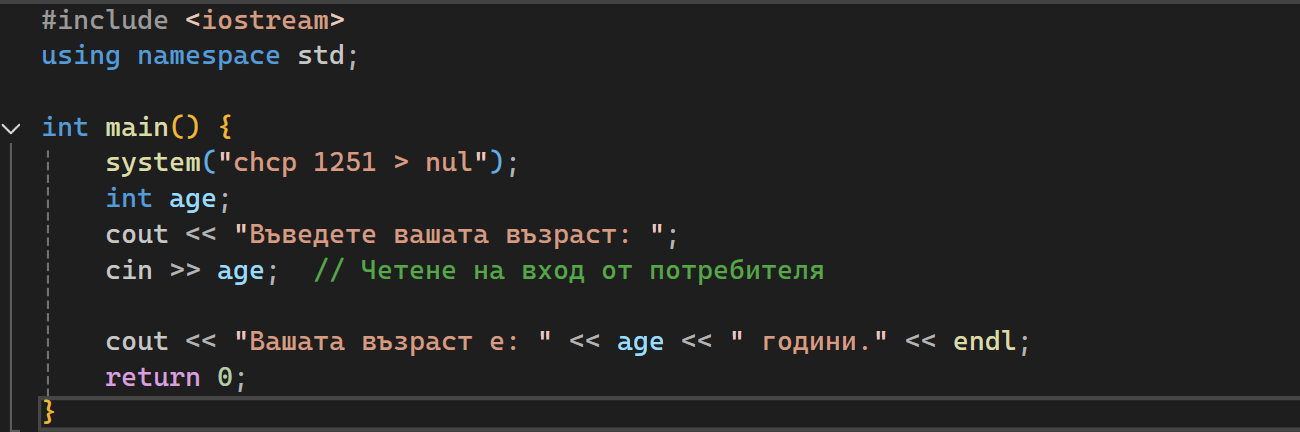
***Изход на конзолата:***

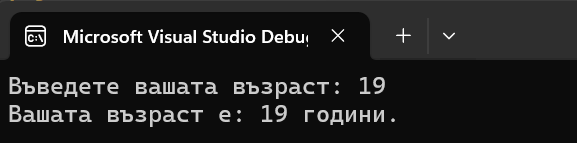
***Обяснение:***

* **Използваме аритметични оператори за извършване на изчисления и оператор за сравнение, за да проверим условие.**
* **Програмата извежда резултата от операциите върху екрана.**
  1. Вход и изход в C++

В C++ стандартната библиотека <iostream> осигурява функциите за вход и изход. Най-често използваните са:

* std::cout – за извеждане на данни на екрана.
* std::cin – за въвеждане на данни от потребителя.

***Пример за въвеждане и извеждане на данни:***

***Изход на конзолата:***

***Обяснение:***

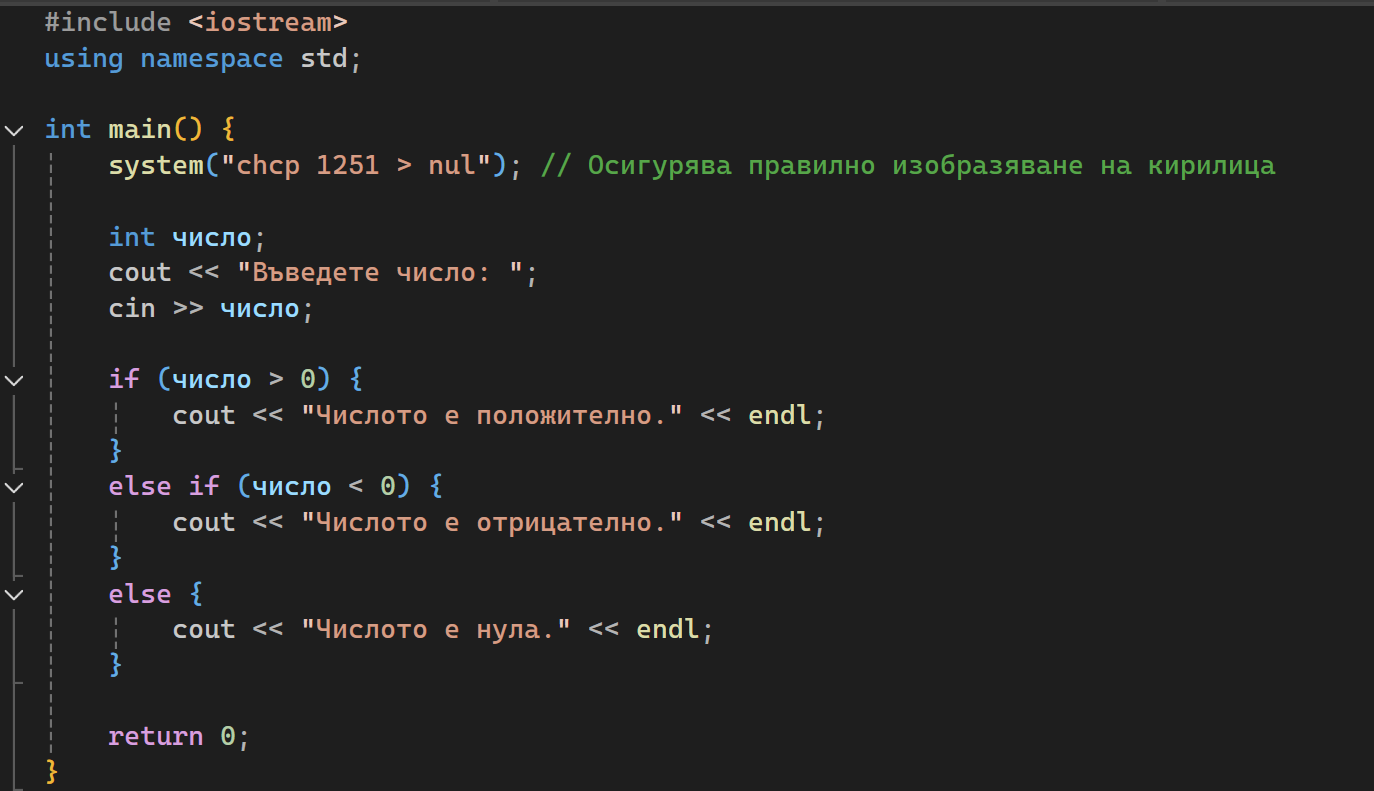
* **Програмата използва std::cin за четене на въведени от потребителя данни и ги извежда с std::cout.**

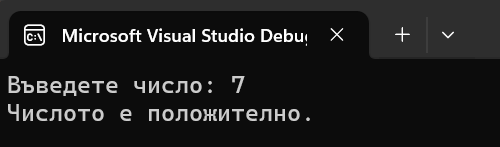
**------------------------------------------------------------------------------------------------------------**

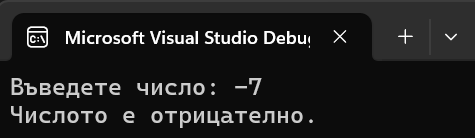
**Тема 3:** Управляващи конструкции

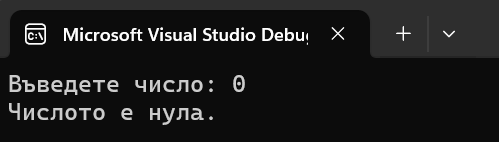
**3.1** Условни оператори

Условните оператори позволяват на програмата да взема решения въз основа на определени условия. Най-често използваните са if, else if, else и switch.

***Пример за използване на if, else if и else:***

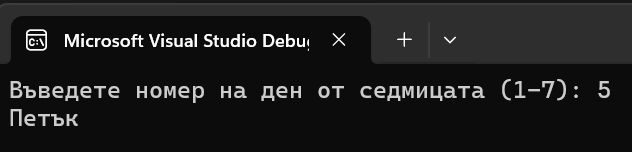
***Изход на конзолата с положително число:***

***Изход на конзолата с отрицателно число:***

***Изход на конзолата с нулева стойност:***

***Обяснение:***

* **Операторът if проверява дали условието е вярно.**
* **Ако първото условие не е изпълнено, програмата преминава към else if.**
* **Ако нито едно от условията не е изпълнено, изпълнява се блокът else.**
* ***Switch***

***Изход на конзолата (switch):***

***Примерен код за switch оператор:***

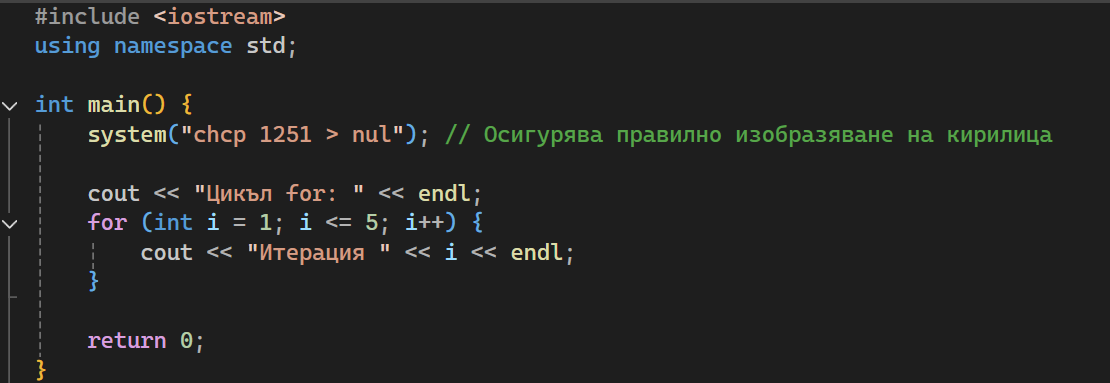
***Обяснение:***

* **Операторът switch проверява стойността на променливата и изпълнява съответния блок case.**
* **Всеки блок завършва с break, за да се предотврати изпълнението на следващите блокове.**
* **Default се изпълнява, ако нито един case не е съвпаднал.**

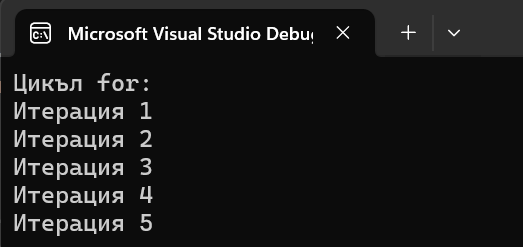
**3.2** Цикли

Циклите се използват за повтаряне на блок от код многократно. В C++ има няколко вида цикли: for, while и do-while.

***Пример за цикъл for:***

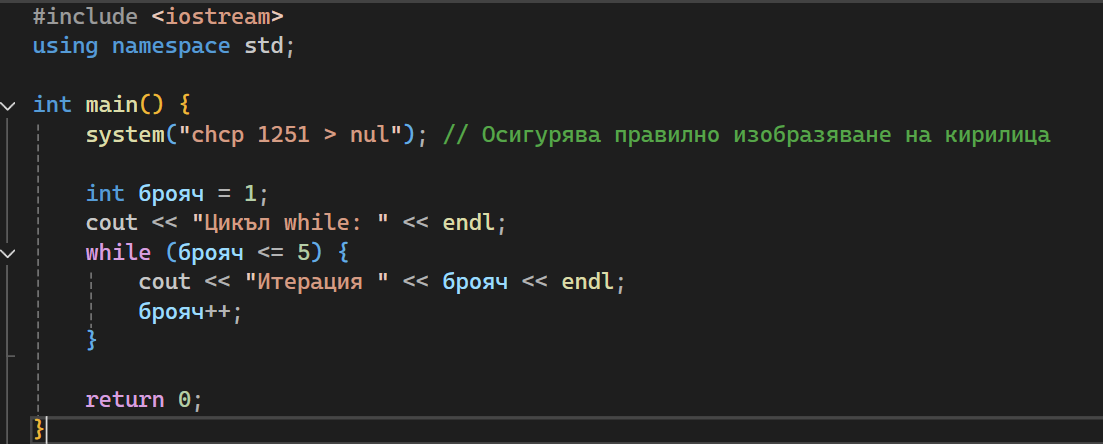
******

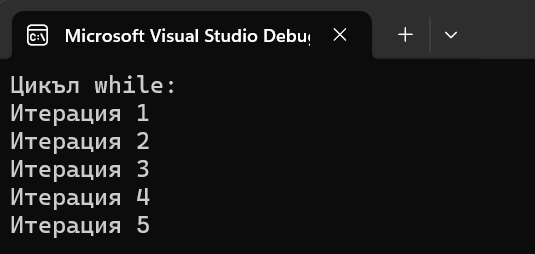
***Изход на конзолата (for цикъл):***

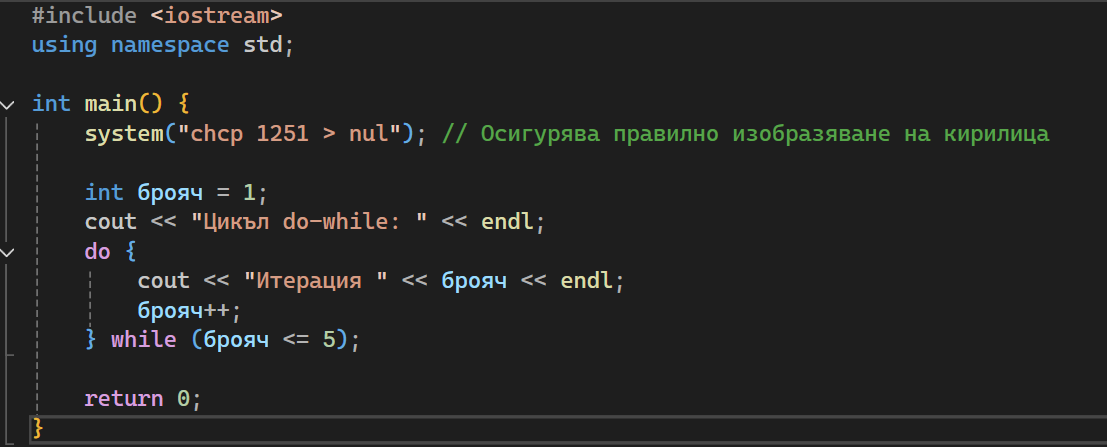
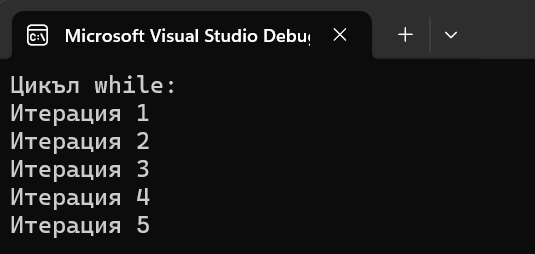
******

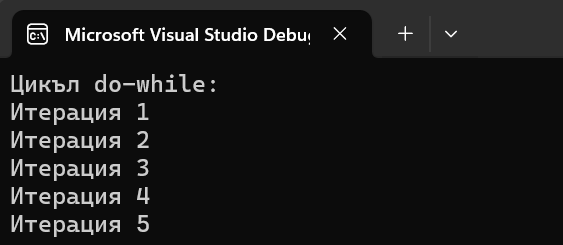
***Обяснение:***

* **Цикълът for включва инициализация на променлива, условие за изпълнение и инкрементиране или декрементиране.**
* **Той изпълнява тялото на цикъла, докато условието е вярно.**

***Примерен код за цикъла while:***

***Изход на конзолата (while):***

***Примерен код за цикъла do-while:***

***Изход на конзолата (do-while):***

***Обяснение:***

* **При цикъла while проверката на условието се извършва преди изпълнението на тялото.**
* **При цикъла do-while тялото се изпълнява поне веднъж, тъй като проверката на условието се извършва след това.**

**------------------------------------------------------------------------------------------------------------**

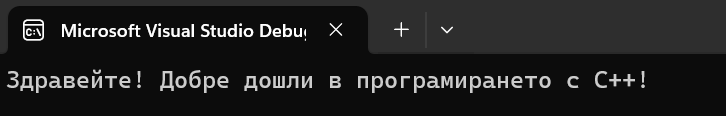
**Тема 4:** Функции в C++

**4.1** Дефиниране и извикване на функции

Функциите са основни блокове в програмирането, които позволяват разделяне на кода на по-малки, лесно управляеми части. Те помагат за повторно използване на код и подобряват четимостта на програмата.

***Примерен код за проста функция: ***

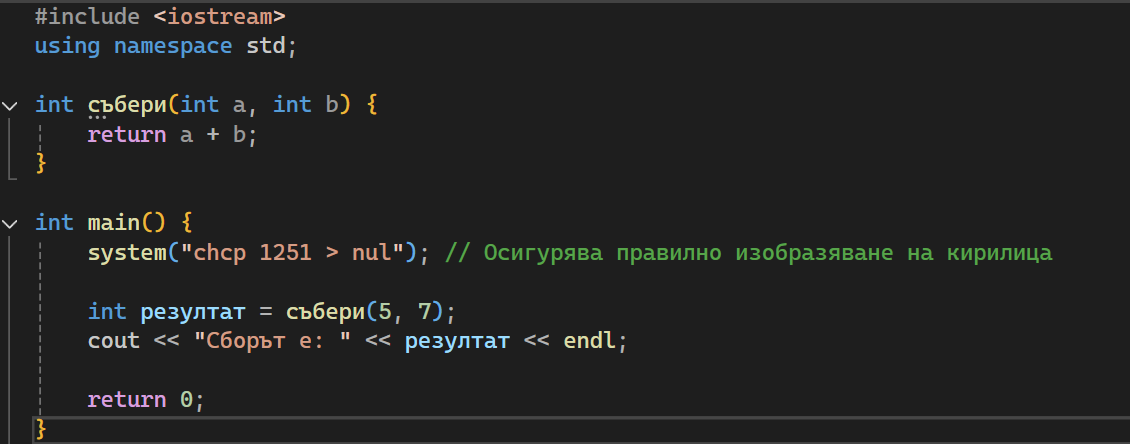
***Изход на конзолата:***

******

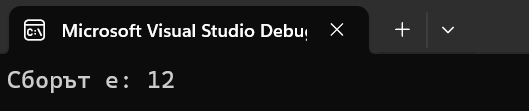
***Обяснение:***

* **Функцията поздрави() е с тип void, което означава, че не връща стойност.**
* **Функцията се извиква в main(), за да се изпълни нейното тяло.**

***Примерен код за функция с параметри:***

******

***Изход на конзолата:***

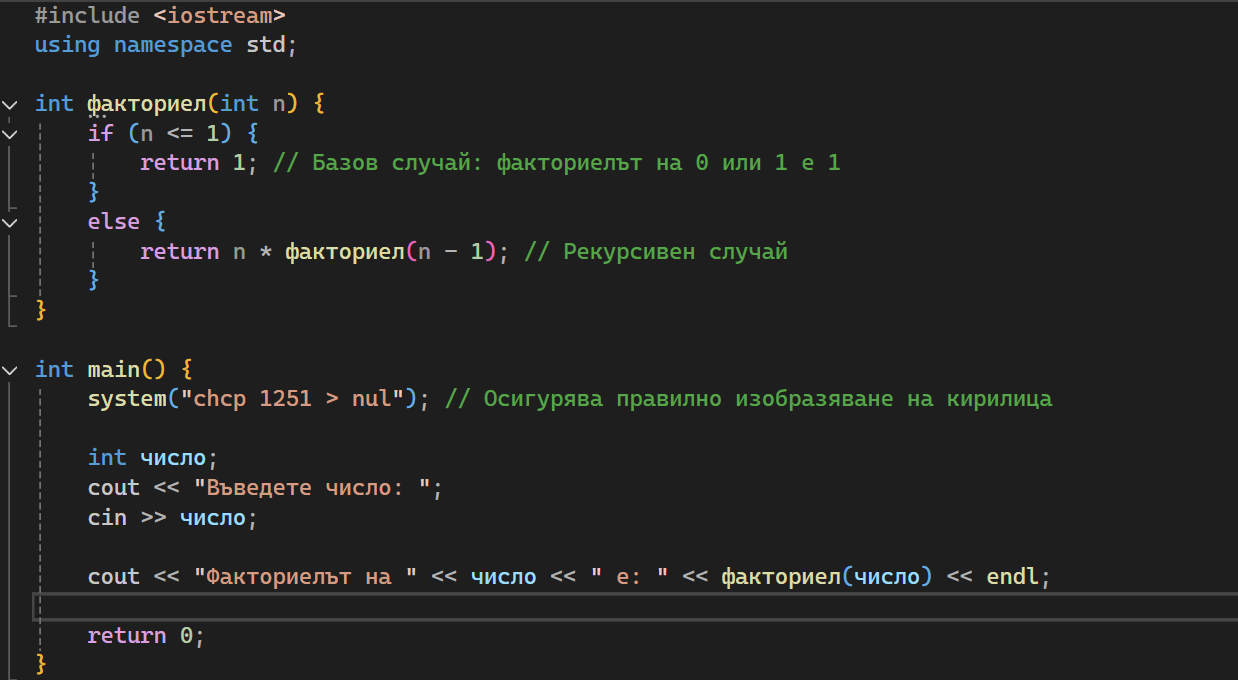
******

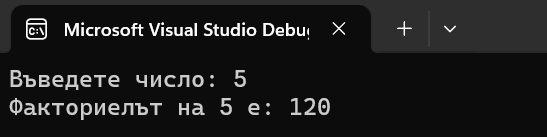
***Обяснение:***

* **Функцията събери() приема два параметъра a и b и връща тяхната сума.**
* **В main() извикваме функцията с конкретни стойности и извеждаме резултата.**

**4.2** Рекурсия

Рекурсията е техника, при която функцията извиква сама себе си, за да реши даден проблем. Това е полезно за задачи, които могат да бъдат разделени на по-малки подзадачи със същата структура.

***Примерен код за рекурсивна функция - Факториел:***

***Изход на конзолата:***

***Обяснение:***

* **Рекурсивната функция факториел() извиква сама себе си с аргумент n - 1, докато достигне базовия случай, при който връща 1.**
* **В main() потребителят въвежда число, за което програмата изчислява факториела.**

***Предимства и недостатъци:***

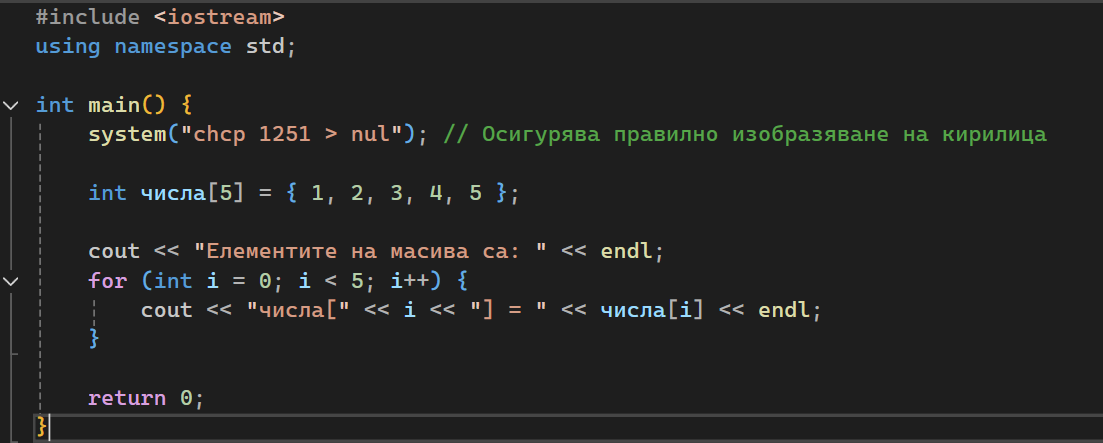
* **Предимства: Ясен и кратък код за проблеми като обхождане на дървета, пресмятане на факториели и Фибоначи.**
* **Недостатъци: Може да води до висока употреба на памет и стек при дълбока рекурсия.**

**------------------------------------------------------------------------------------------------------------**

**Тема 5:** Масиви и низове

**5.1** Масиви

Масивите представляват последователност от елементи от един и същи тип, които са достъпни чрез индекс. Те са полезни за съхраняване на група от данни и лесно манипулиране на тези данни.

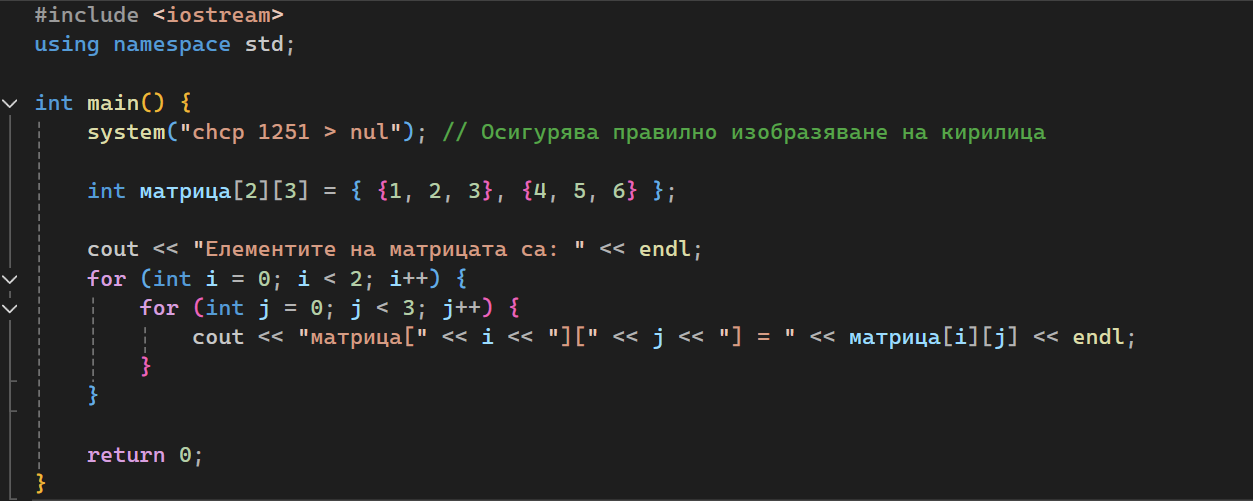
***Примерен код за едномерен масив:***

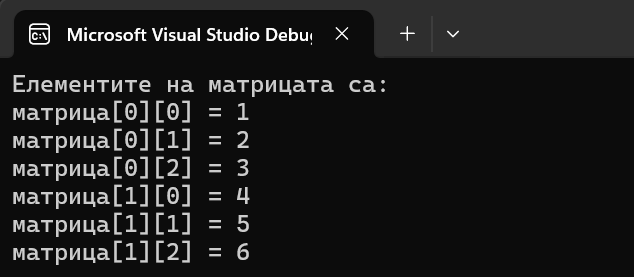
***Изход на конзолата:***

******

***Обяснение:***

* **Масивът числа съдържа 5 елемента, които се достъпват чрез индекси от 0 до 4.**
* **for цикълът преминава през всеки елемент и го извежда на екрана.**

***Примерен код за двумерен масив:***

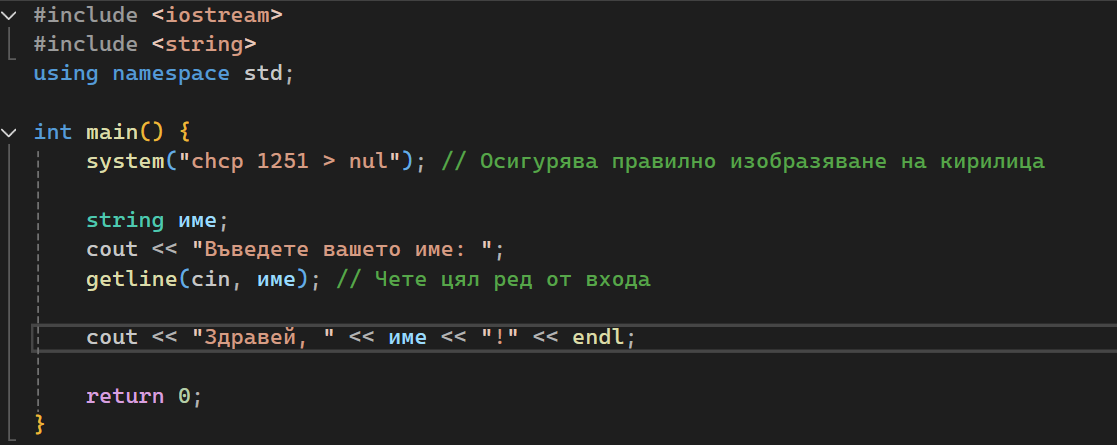
***Изход на конзолата:***

***Обяснение:***

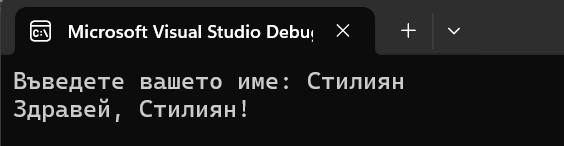
* **Многомерният масив матрица съдържа 2 реда и 3 колони.**
* **Два вложени цикъла for се използват за обхождане на елементите в матрицата.**

**5.2** Работа с низове

В C++ низовете могат да се представят като масиви от символи или чрез класа string от библиотеката <string>.

***Примерен код за използване на string:***

***Изход на конзолата:***

******

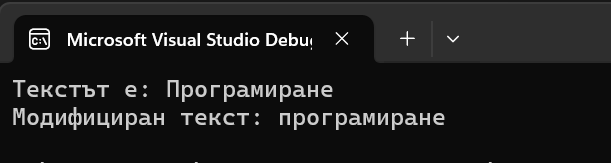
***Обяснение:***

* **Класът string предоставя удобни методи за работа с низове, като getline() за четене на цял ред.**
* **Използваме getline(cin, име) вместо cin, за да четем низове с интервали.**

***Основни операции с низове:***

* **Конкатенация: string пълноИме = име + " " + фамилия;**
* **Дължина на низ: име.length();**
* **Достъп до символи: име[0] за първия символ.**

***Примерен код за манипулация на символи:***

***Изход на конзолата:***

***Обяснение:***

* **Низовете могат да се манипулират подобно на масиви от символи. Промяната на отделни символи става чрез индексиране.**

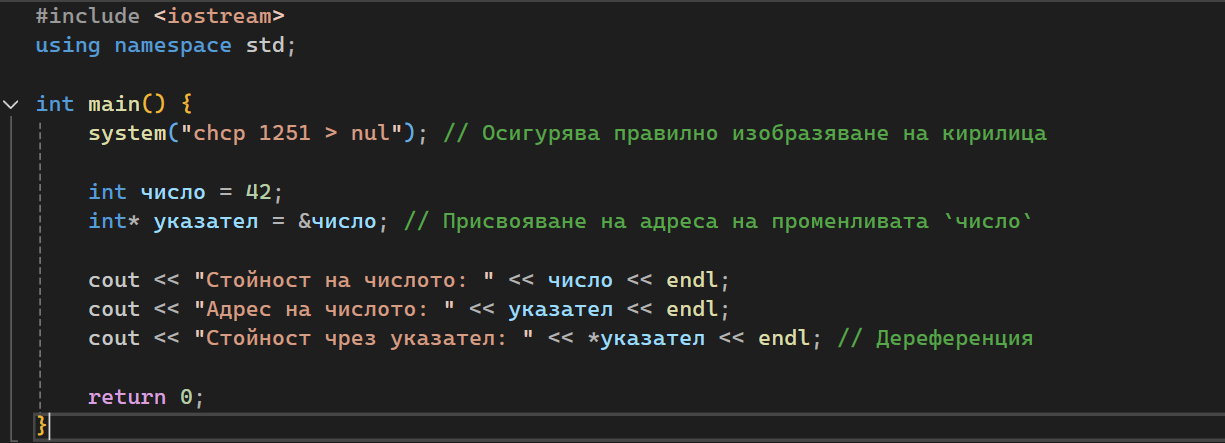
**------------------------------------------------------------------------------------------------------------**

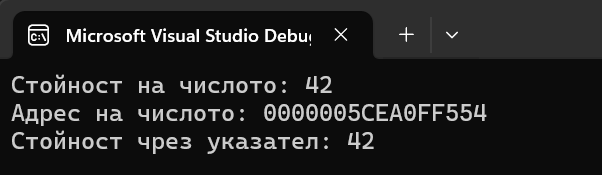
**Тема 6:** Указатели и динамична памет

**6.1** Указатели

Указателите са променливи, които съдържат адреса на друга променлива в паметта. Те са важен инструмент за управление на паметта и предоставят голяма гъвкавост при работа с данни.

***Примерен код за използване на указател:***

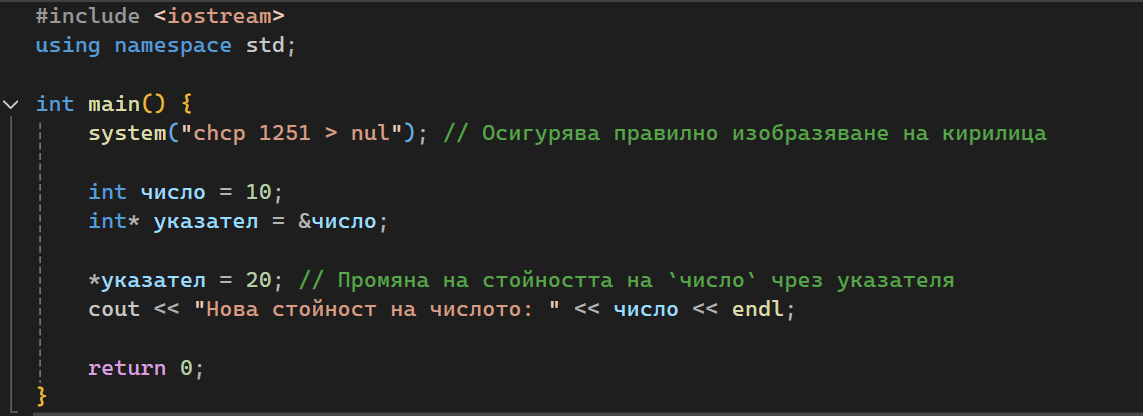
******

***Изход на конзолата:***

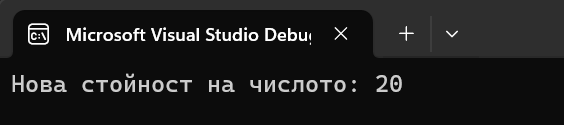
***Обяснение:***

* **int\* указател създава указател към цяло число.**
* **&число дава адреса на променливата число.**
* **\*указател извежда стойността на променливата, към която указателят сочи (дереференция).**

***Примерен код за промяна на стойност чрез указател:***

****

***Изход на конзолата:***

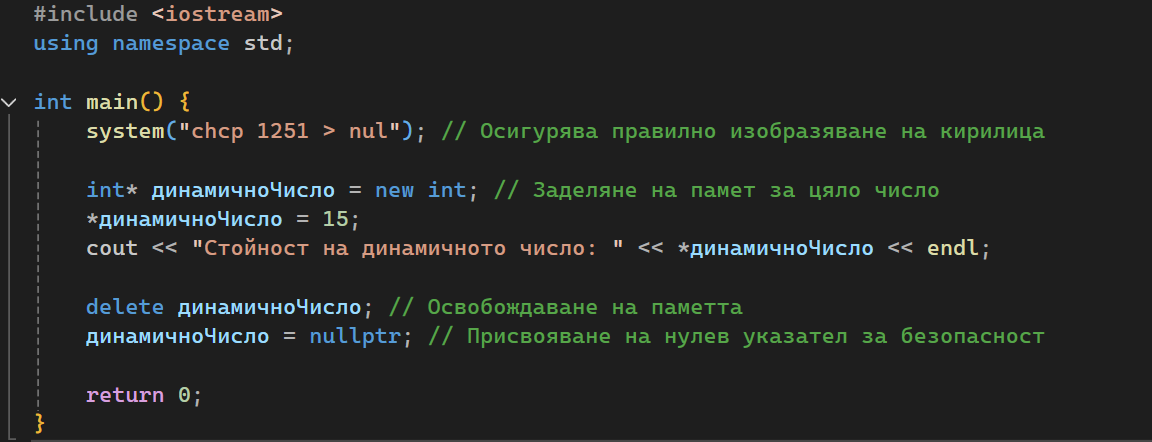
******

***Обяснение:***

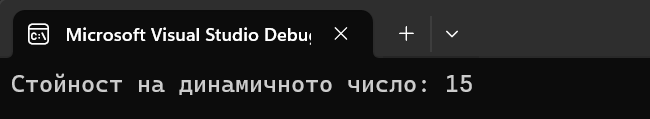
* **Променяме стойността на променливата число, използвайки указателя \*указател.**

**6.2** Динамично управление на паметта

Динамичната памет позволява заделяне на памет по време на изпълнение на програмата, вместо на компилация. Това е особено полезно, когато размерът на масив или структура не е известен предварително.

***Примерен код за използване на new и delete:***

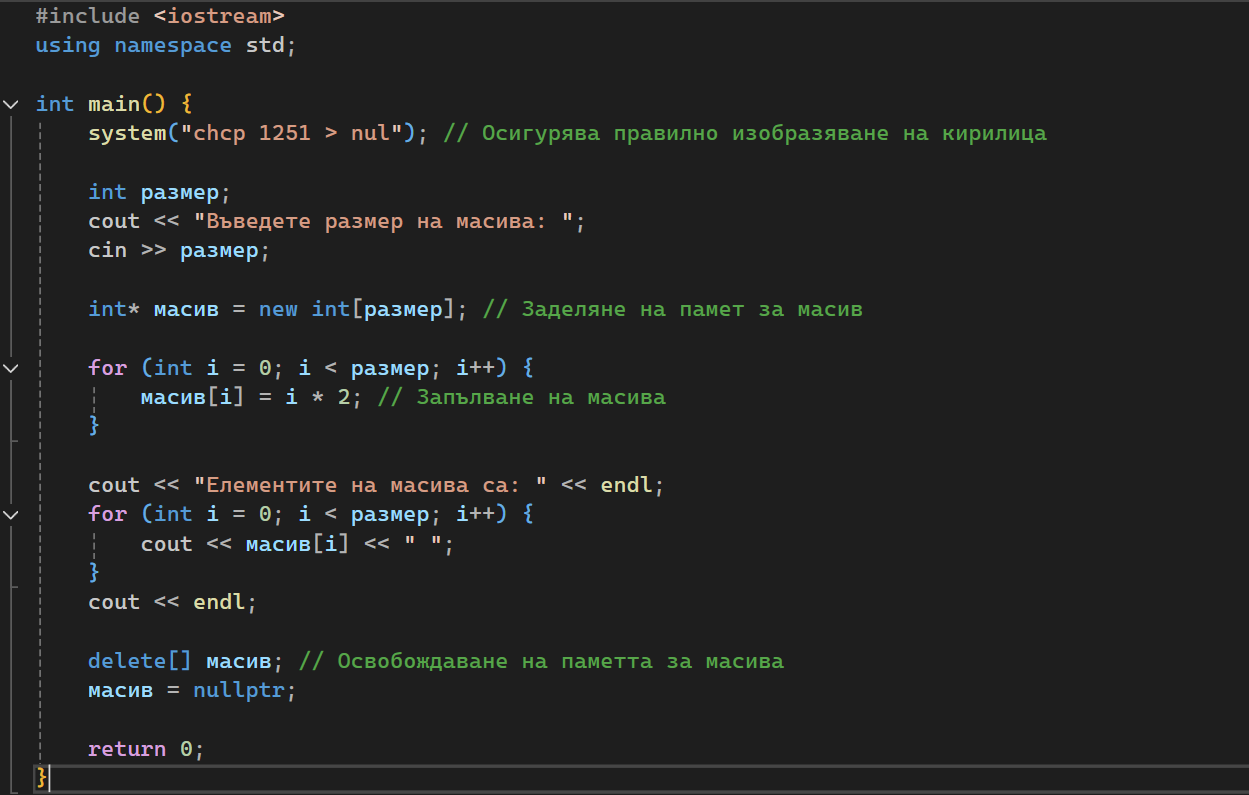
***Изход на конзолата:***

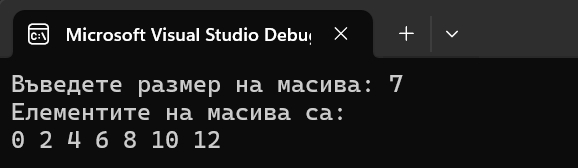
******

***Обяснение:***

* **new int заделя памет за цяло число и връща указател към нея.**
* **delete освобождава заделената памет.**
* **nullptr се използва за указатели, за да се предотвратят случайни операции с невалидни указатели.**

***Примерен код за динамичен масив:***

******

***Изход на конзолата:***

***Обяснение:***

* **new int[размер] заделя памет за динамичен масив с посочения размер.**
* **delete[] масив освобождава паметта, използвана от масива.**

***Практически съвети при работа с динамична памет:***

* **Винаги освобождавайте динамично заделената памет, за да избегнете изтичане на памет.**
* **Проверявайте указателите, за да не извършвате операции с nullptr.**

**---------------------------------------------------------------------------------------------------------------------------**

**Тема 7:** Обектно-ориентирано програмиране (ООП)

**7.1** Основи на ООП

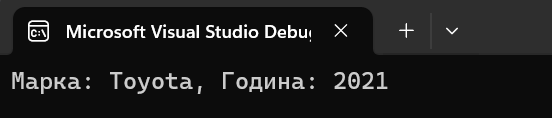
Обектно-ориентираното програмиране е парадигма, която организира кода чрез използване на обекти и класове. Това прави кода по-лесен за поддръжка, повторно използване и разширение.

***Основни понятия:***

* **Клас:** Шаблон или модел, който описва свойства и методи на обект.
* **Обект:** Инстанция на клас, която съдържа специфични стойности на свойствата.
* **Методи:** Функции, които принадлежат на клас и могат да манипулират неговите данни.

***Примерен код за дефиниране на клас и създаване на обект:***

******

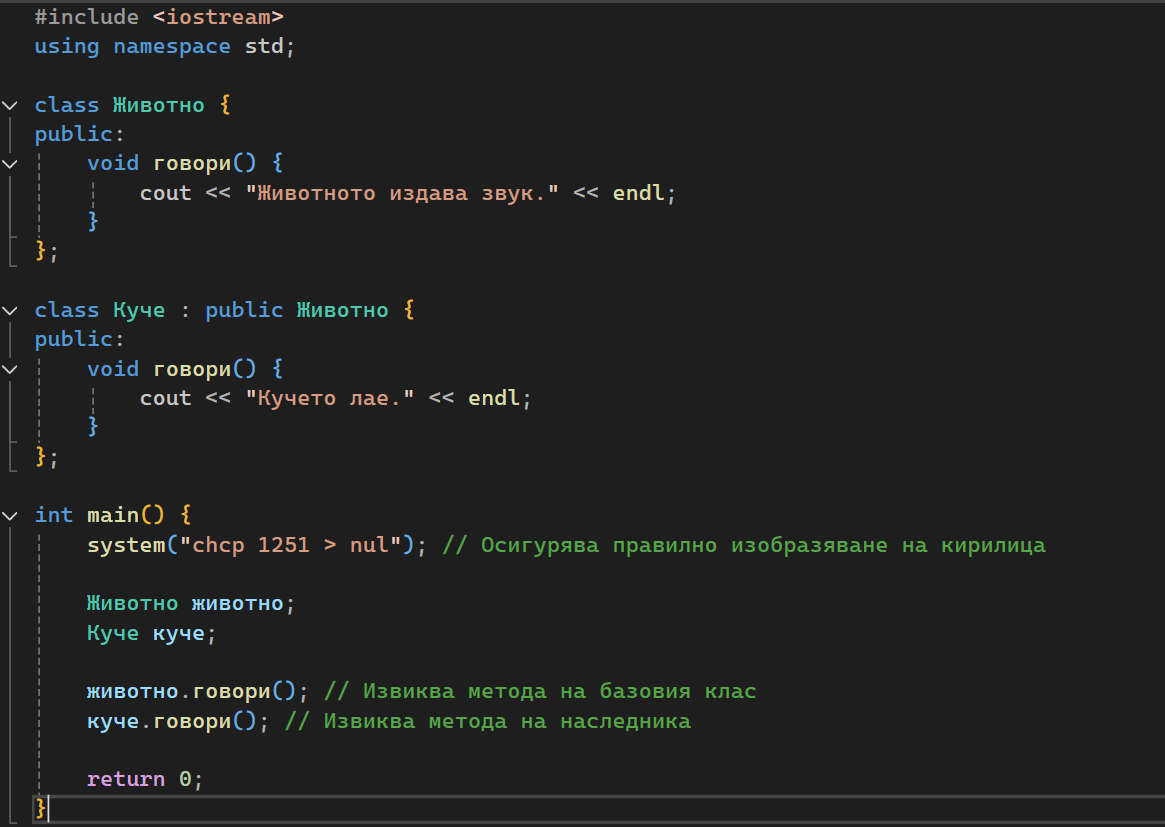
***Изход на конзолата:***

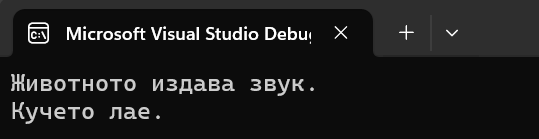
***Обяснение:***

* **Класът Автомобил има два члена: марка и година.**
* **Методът покажиИнформация() извежда стойностите на членовете на обекта.**
* **В main() се създава обектът кола1, задават му се стойности и се извиква методът.**

**7.2** Наследяване и полиморфизъм

Наследяване е концепция, при която един клас може да наследява свойства и методи от друг клас. Това позволява създаване на по-специфични класове въз основа на по-общи класове.

***Примерен код за наследяване:***

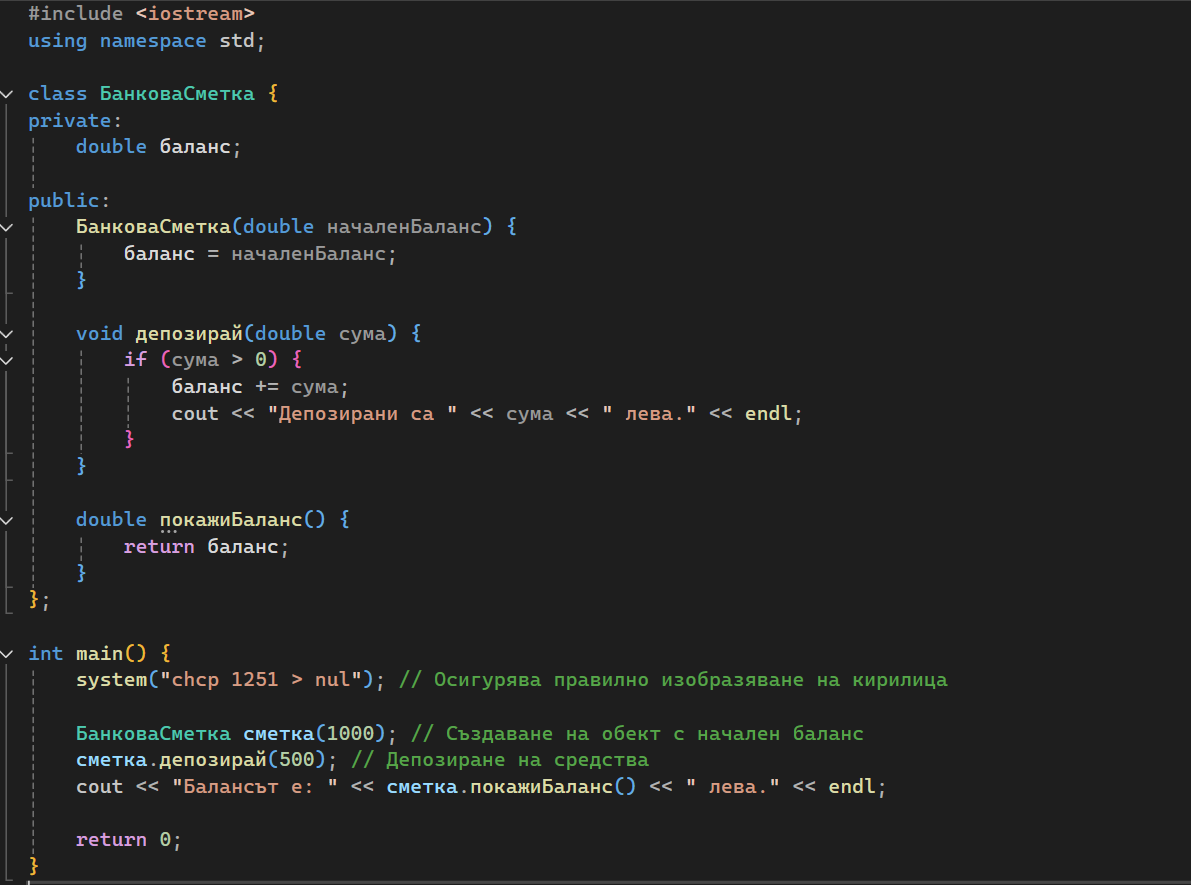
***Изход на конзолата:***

***Обяснение:***

* **Класът Куче наследява класа Животно и пренаписва метода говори().**
* **Полиморфизмът позволява методите с едно и също име да се държат различно в различните класове.**

**7.3**  Инкапсулация

**Инкапсулацията** означава скриване на детайлите за имплементация и предоставяне на интерфейс за взаимодействие с обекта. Това се постига чрез използване на модификатори за достъп като private, protected и public.

***Примерен код за инкапсулация:***

***Изход на конзолата:***

***Обяснение:***

* **баланс е частен член на класа, който е достъпен само чрез публични методи.**
* **Методът депозирай() добавя сума към баланса само ако е положителна.**

**7.4**  Абстракция

**Абстракция** означава фокусиране върху важните характеристики на обекта, като същевременно се скриват ненужните детайли за потребителя. Това се постига чрез интерфейси и абстрактни класове, които могат да бъдат подробно разгледани в по-напредналите теми.

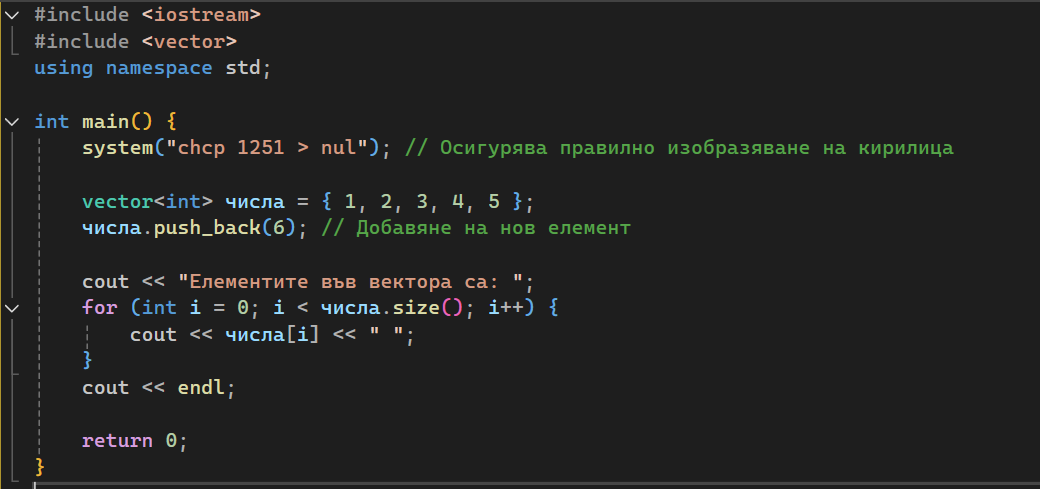
**Тема 8:** Стандартната библиотека на C++ (STL)

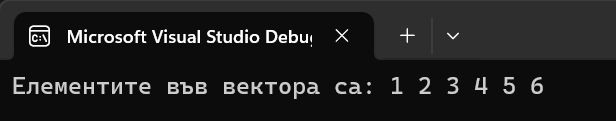
**8.1** Контейнери в STL

STL предоставя набор от готови за употреба контейнери, които улесняват работата с различни структури от данни. Те могат да се разделят на три основни типа:

* **Последователни контейнери**: vector, deque, list
* **Асоциативни контейнери**: set, multiset, map, multimap
* **Контейнери за адаптация**: stack, queue, priority\_queue

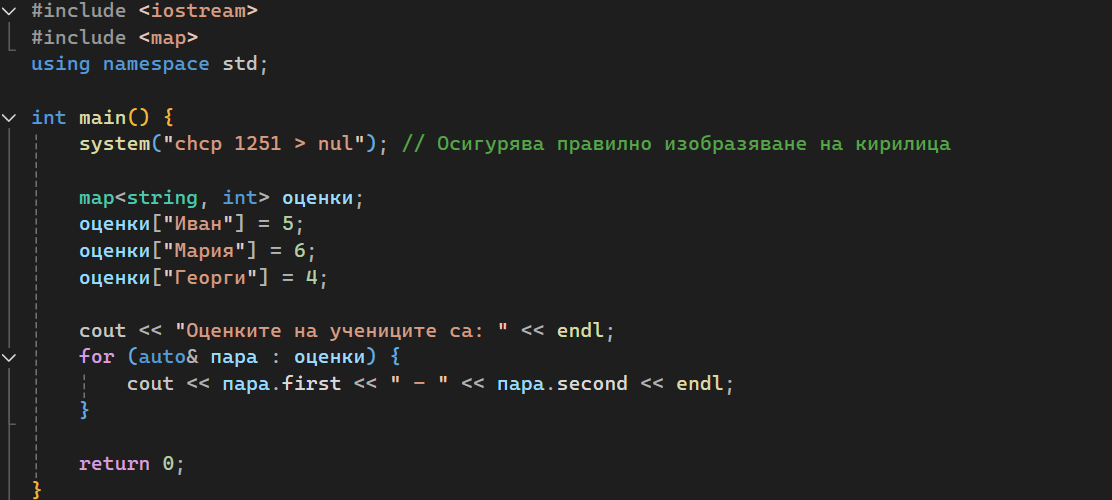
***Примерен код за използване на vector:***

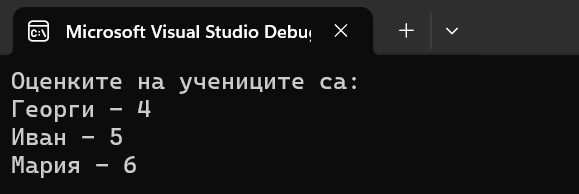
****

***Изход на конзолата:***

***Обяснение:***

* **vector е динамичен масив, който може да променя размера си.**
* **Методът push\_back() добавя нов елемент в края на вектора.**
* **size() връща текущия брой елементи във вектора.**

***Примерен код за използване на map:***

***Изход на конзолата:***

***Обяснение:***

* **map е асоциативен контейнер, който съхранява двойки ключ-стойност.**
* **first и second са методи на итератора, които достъпват ключа и стойността.**

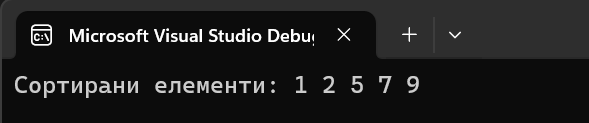
**8.2**  Алгоритми в STL

STL включва множество алгоритми, които работят с контейнерите и улесняват обработката на данни, като сортиране, търсене и модификация.

***Примерен код за сортиране със sort():***

******

***Изход на конзолата:***

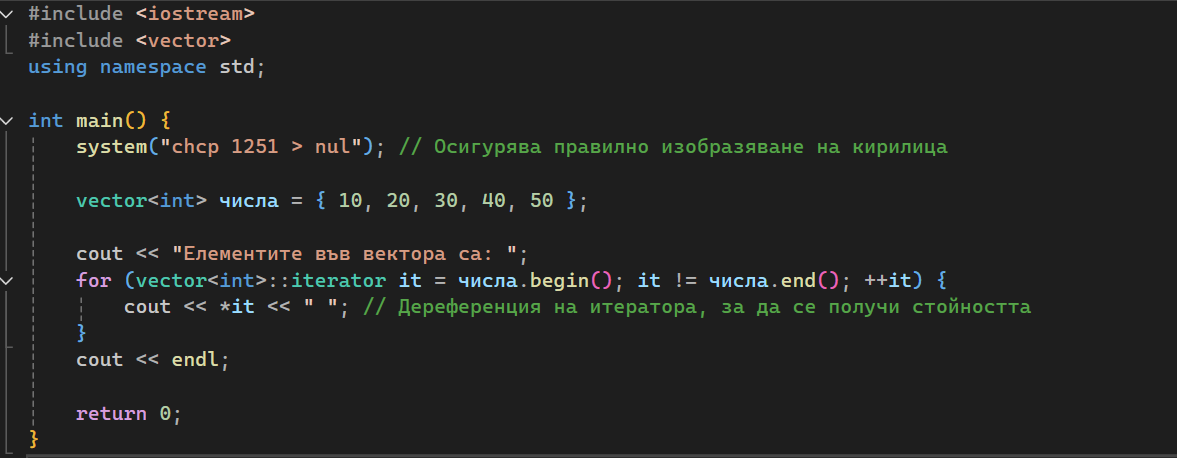
******

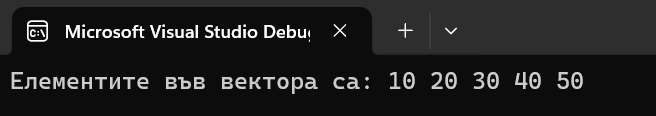
***Обяснение:***

* **sort() приема два параметъра: началото и края на обхвата, който трябва да се сортира.**
* **begin() и end() са методи на контейнерите, които връщат съответно началото и края на контейнера.**

**8.3**  Итератори и тяхното приложение

Итераторите са обекти, които позволяват обхождането на елементите в контейнерите. Те действат като указатели и предоставят стандартен начин за достъп до елементите.

***Примерен код за използване на итератор:***

***Изход на конзолата:***

***Обяснение:***

* **vector<int>::iterator е типът на итератора за вектор от цели числа.**
* **Итераторът се движи от началото (begin()) до края (end()) на вектора.**
* **Дереференцията \*it връща стойността на текущия елемент.**

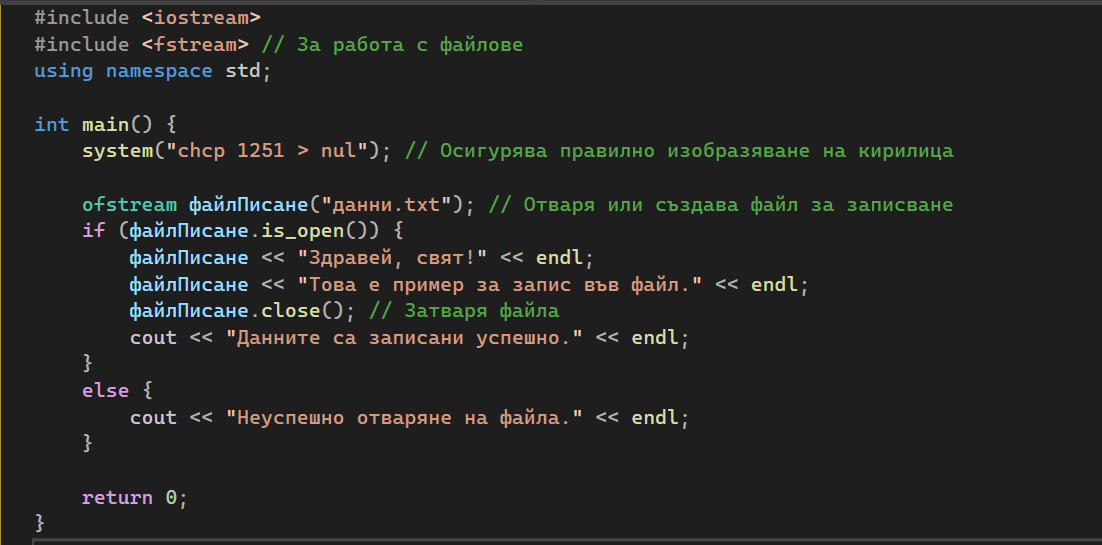
**Тема 9:** Файлове и работа с тях

**9.1** Четене и писане във файлове

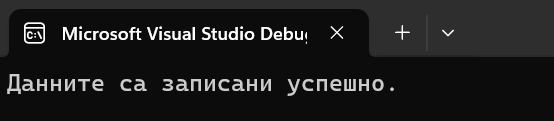
Работата с файлове в C++ се осъществява чрез библиотеката <fstream>, която предоставя три основни класа:

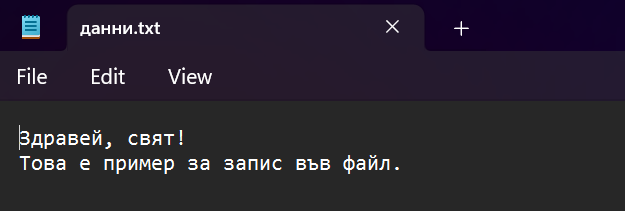
* **ofstream –** за записване във файлове.
* **ifstream –** за четене от файлове.
* **fstream –** за едновременно четене и писане.

***Примерен код за записване във файл:***

******

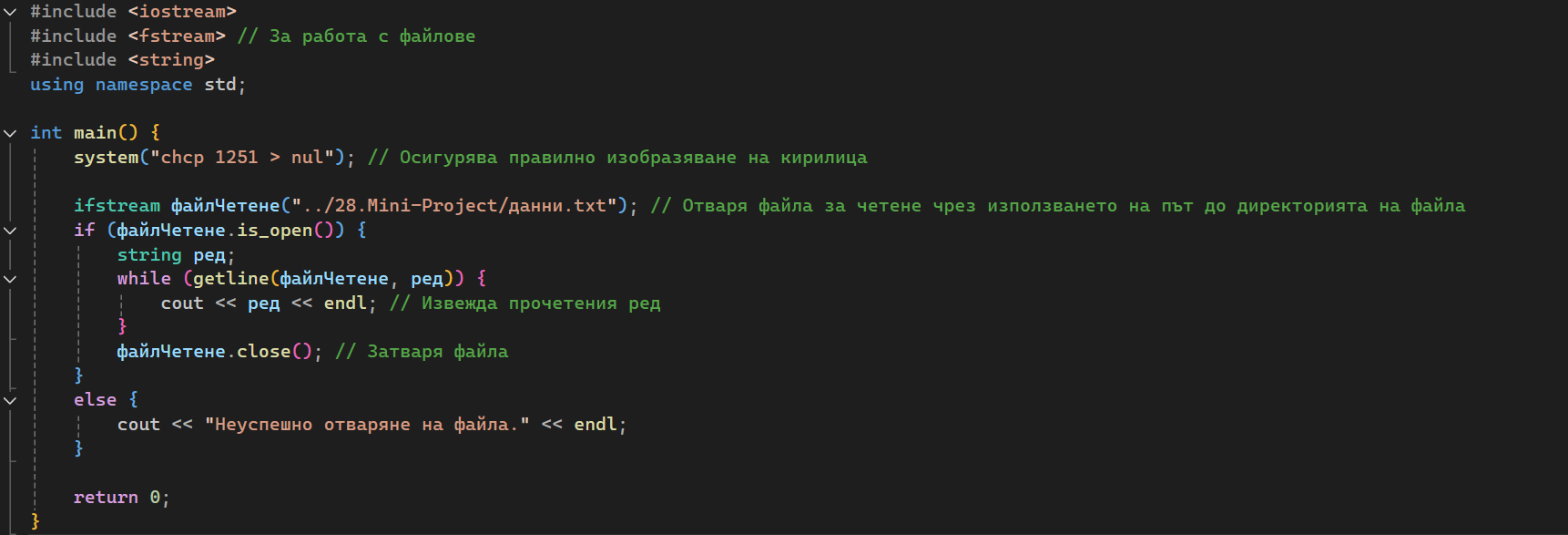
***Изход на конзолата:***

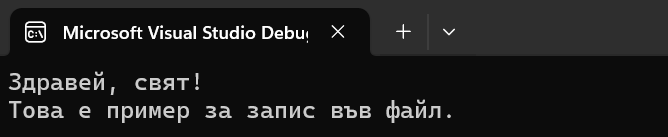
******

***Файл - данни.txt след изпълнение на програмата:***

***Обяснение:***

* **ofstream файлПисане("данни.txt") създава обект за записване и отваря файла данни.txt.**
* **Методът is\_open() проверява дали файлът е успешно отворен.**
* **файлПисане.close() затваря файла след записване.**

***Примерен код за четене от файл:***

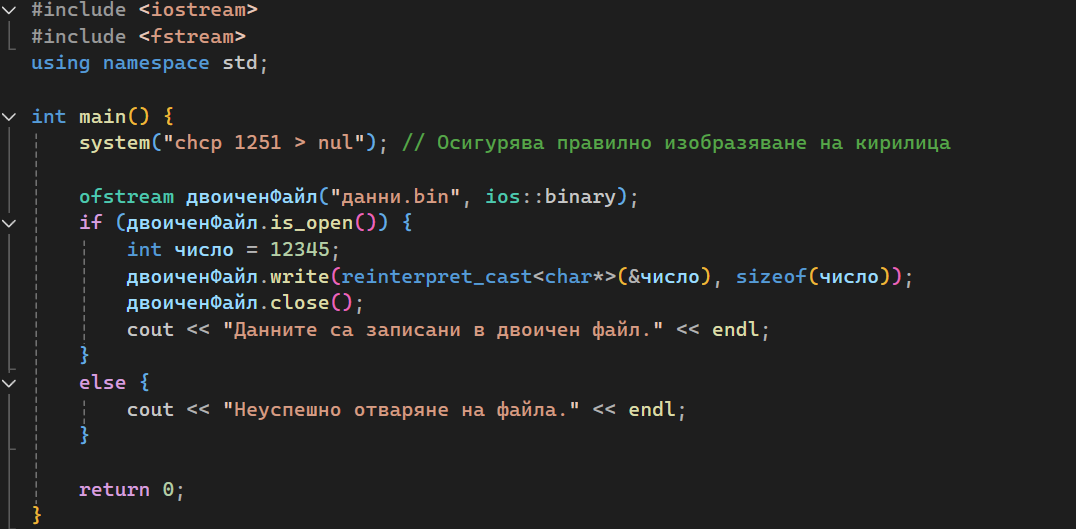
***Изход на конзолата:***

***Обяснение:***

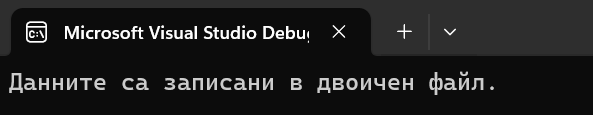
* **ifstream файлЧетене("данни.txt") отваря файла данни.txt за четене.**
* **getline() чете ред по ред от файла и го записва в променливата ред.**
* **Файлът се затваря с файлЧетене.close().**

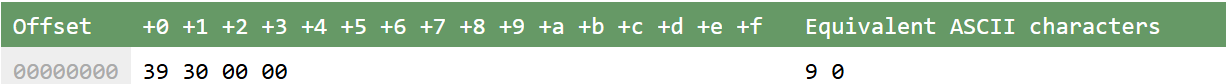
**9.2** Работа с текстови и двоични файлове

Освен текстови файлове, C++ позволява и работа с двоични файлове за по-ефективно съхранение на данни.

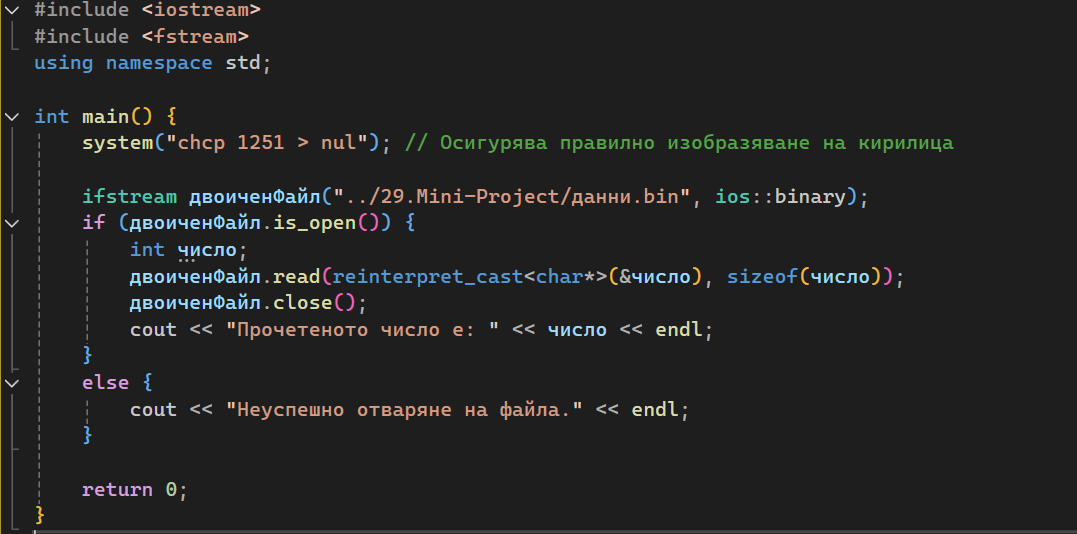
***Примерен код за записване в двоичен файл:***

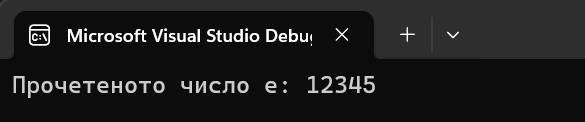
***Изход на конзолата:***

******

***Файл – данни.bin след изпълнението на програмата:***

***Примерен код за четене от двоичен файл:***

****

***Изход на конзолата:***

***Обяснение:***

* **ofstream двоиченФайл("данни.bin", ios::binary) отваря двоичен файл за записване.**
* **write() и read() използват reinterpret\_cast<char\*>, за да преобразуват указателя към тип char\* за работа с двоични данни.**
* **sizeof(число) указва размера на данните за запис или четене.**

***Съвети за работа с файлове:***

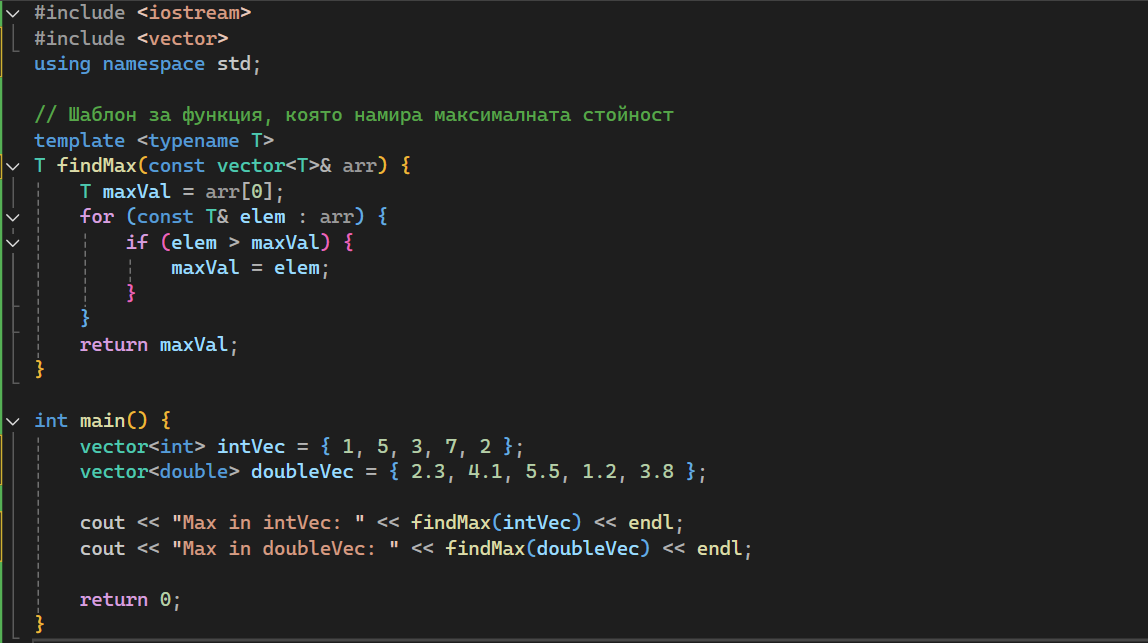
* **Винаги проверявайте дали файловете са успешно отворени преди работа с тях.**
* **Затваряйте файловете с close(), за да освободите ресурсите.**
* **За по-сложни операции може да се използва fstream, който съчетава функционалността на ifstream и ofstream.**

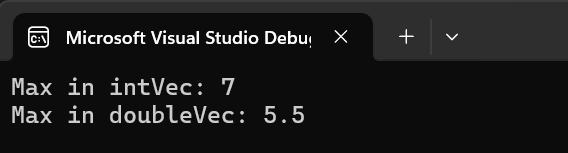
----------------------------------------------------------------------------------------------------------------

**Тема 10:** Теми за напреднали

**10.1** Шаблони в C++ и тяхното приложение

Шаблоните (templates) предоставят мощен начин за създаване на обобщени функции и класове, които могат да работят с различни типове данни. Те са основен инструмент за генериране на гъвкав и многократно използваем код.

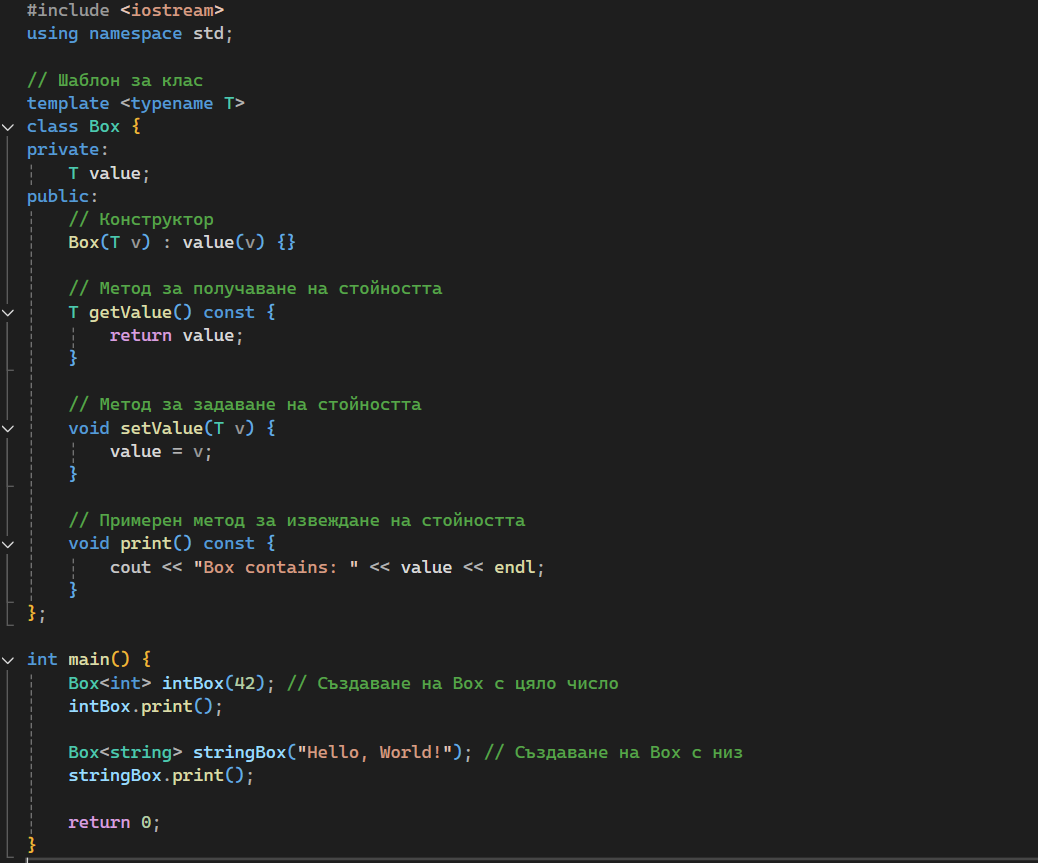
***Примерен код за шаблон на функция в C++:***

***Изход на конзолата:***

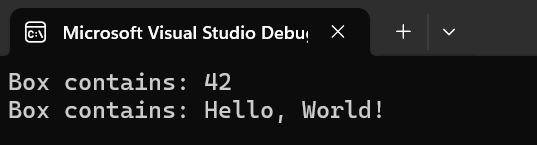
***Обяснение:***

* **Тази функция използва шаблон template <typename T>, който позволява на функцията findMax да работи с всякакъв тип данни, като int, double и други. Функцията приема вектор и връща най-голямата стойност в него.**

***Примерен код за шаблон на клас в C++:***

******

***Изход на конзолата:***



***Обяснение:***

* **Шаблонът за клас template <typename T> позволява създаване на клас Box, който може да съдържа данни от различни типове (int, string и др.). Конструкторът и методите работят с типа T, което прави класа гъвкав и многократно използваем.**

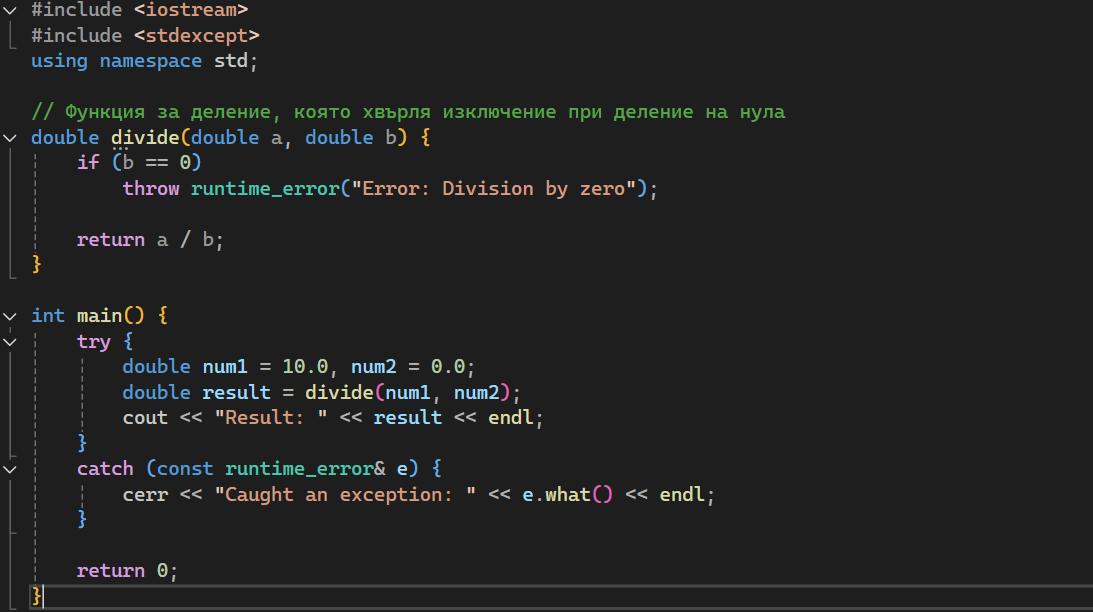
**10.2** Обработка на изключения и управление на грешки

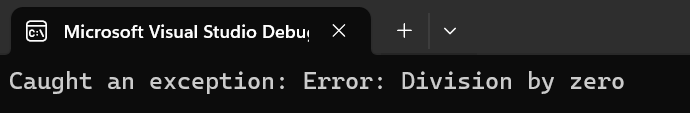
Обработката на изключения е важна част от съвременното програмиране. Тя позволява на програмите да се справят с неочаквани събития и грешки по елегантен начин, без да прекъсват изпълнението си.

***Основни концепции:***

* Използването на блокове try, catch и throw за обработка на грешки.
* Разглеждане на йерархията на изключенията и създаване на персонализирани класове за изключения.

***Примерен код за обработка на изключения и управление на грешки:***



***Изход на конзолата:***

***Обяснение:***

* **Функцията divide хвърля изключение (throw runtime\_error), ако вторият аргумент е нула.**
* **В main() блокът try извиква функцията и хваща изключението с catch, извеждайки съобщение за грешка. Това предотвратява срив на програмата.**

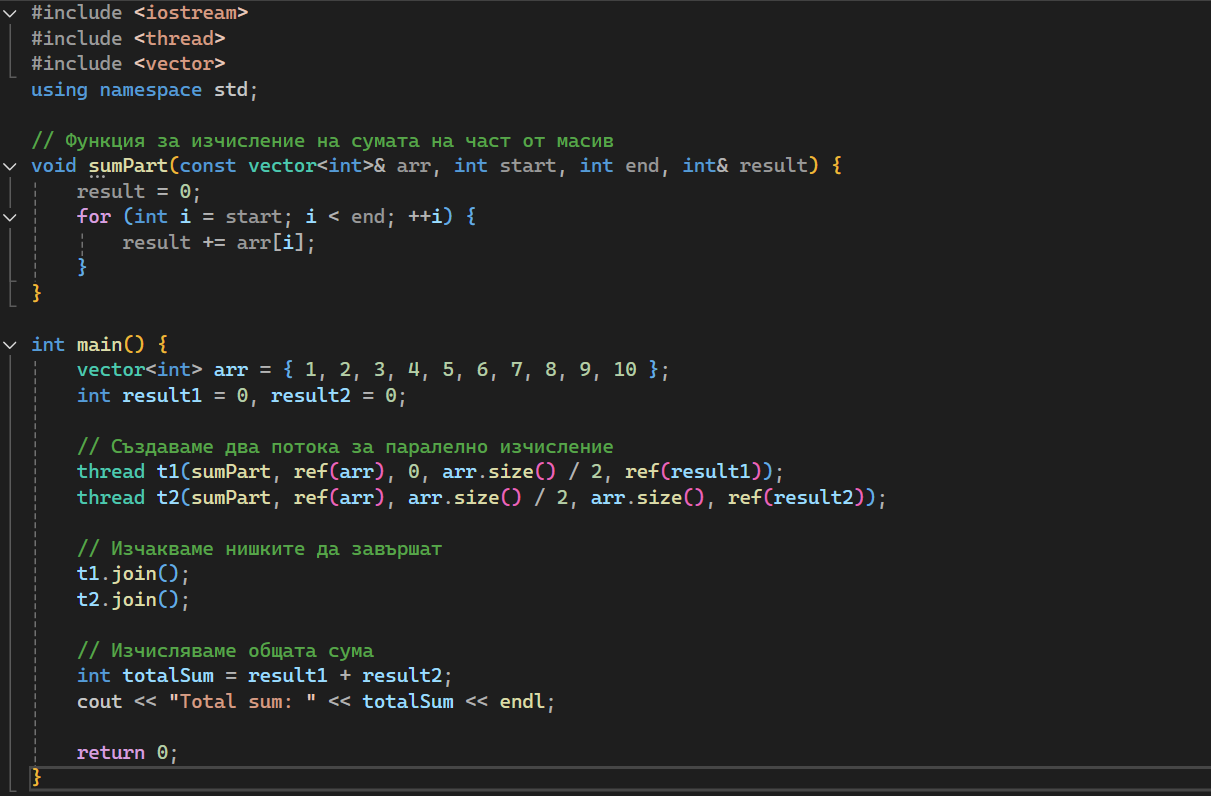
**10.3** Паралелно програмиране и многопоточност

Паралелното програмиране позволява на програмите да изпълняват множество операции едновременно, което повишава ефективността и производителността на софтуера. Многопоточността е важна част от това.

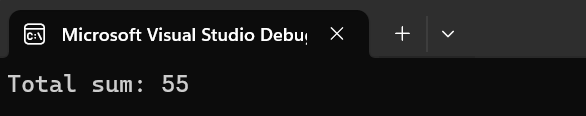
***Концепции:***

* Използване на библиотеката <thread в C++ за създаване и управление на потоци.
* Синхронизация на данните с помощта на mutex и lock\_guard за избягване на състезателни условия (race conditions).

***Примерен код за паралелно изчисление на сумата на елементи в масив:***

******

***Изход на конзолата:***



***Обяснение:***

* Функцията sumPart изчислява сумата на част от масив.
* Два потока се създават, за да изчислят сумата на две различни части на масива.
* Резултатите се събират в main() и се извежда общата сума.

***Примерен код за паралелно сортиране с функцията std::sort:***

***Изход на конзолата:***



***Обяснение:***

* Масивът се разделя на две части и всяка част се сортира паралелно с помощта на sort в отделни потоци.
* След това двете части се обединяват с помощта на inplace\_merge, за да се получи окончателно сортираният масив.

***Защо да използваме паралелния подход?***

1. ***Паралелна обработка***

* **Скорост:** Когато имате голям набор от данни, можете да използвате паралелизъм, за да ускорите обработката. Например, ако разпределите сортирането на две части на масива в два отделни потока (нишки), всяка нишка може да работи независимо, като по този начин намалява общото време за сортиране. При големи обеми данни, паралелното сортиране може да доведе до значително подобрение в производителността.

1. ***Разделяне на данните***

* **Специализирани случаи**: В определени ситуации, например, когато имате много предварително сортирани данни или данни, разпределени по определен начин, разделянето на данните на две части и последващото им сливане може да бъде по-ефективно от сортирането на целия масив. Например, ако имате данни, които вече са частично сортирани, алгоритмът за сливане може да бъде много по-бърз, отколкото пълното сортиране.

1. ***Гъвкавост***

* **Масиви с различни размери:** При работа с различни размери на масиви или вектори, подходът с паралелно сортиране може да бъде по-гъвкав. Можете лесно да увеличите броя на нишките и да разширите метода за работа с по-голям брой части, ако имате много данни.

1. ***Използване на памет***

* **Оптимизация на паметта:** При използването на std::inplace\_merge паметта не се разширява, тъй като данните остават в оригиналния контейнер, вместо да се създава нов контейнер за обединение. В ситуации, когато паметта е ограничена, това може да бъде важен фактор.

----------------------------------------------------------------------------------------------------------------

**Тема 11:** Практическо приложение – Финален проект (Библиотека)

**11.1** Описание на проекта

* + Целта на този финален проект е да се създаде програма, която управлява базова система за библиотека.
  + Приложението ще поддържа основни функции като добавяне на книги, преглед на наличните книги и търсене на книга по заглавие.

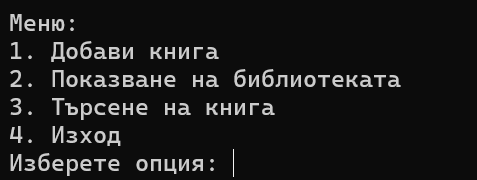
**11.2** Основни функции и код

Програмата ще използва концепции като:

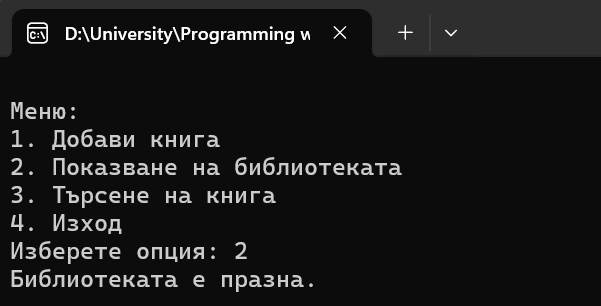
* + - * Класове и обекти за съхранение на информация за книгите.
      * STL контейнери (vector) за съхранение на списъка с книги.
      * Файлове за запазване и зареждане на данните при стартиране на програмата.
      * Указатели и динамична памет за по-сложни операции (ако е необходимо).

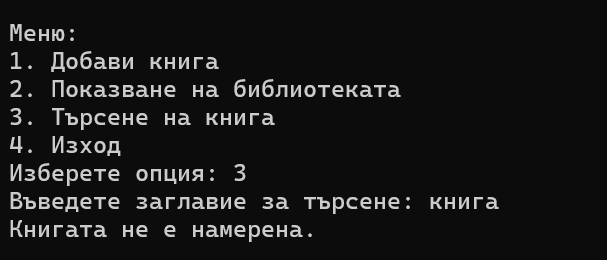
***Екрани от изпълнението:***

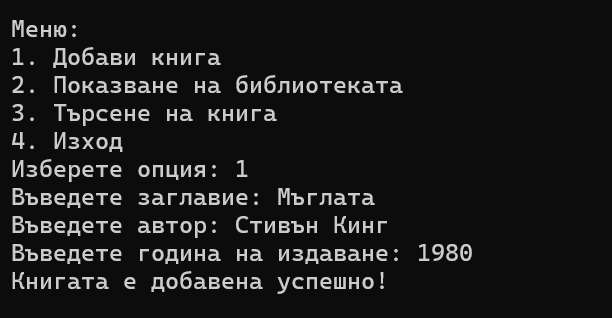
***Начален екран при стартиране на приложението с възможност да се избере опция (1-4)***

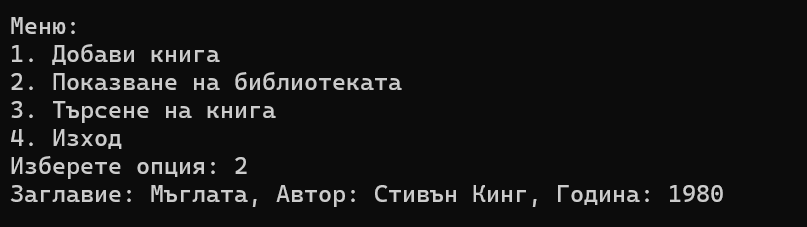


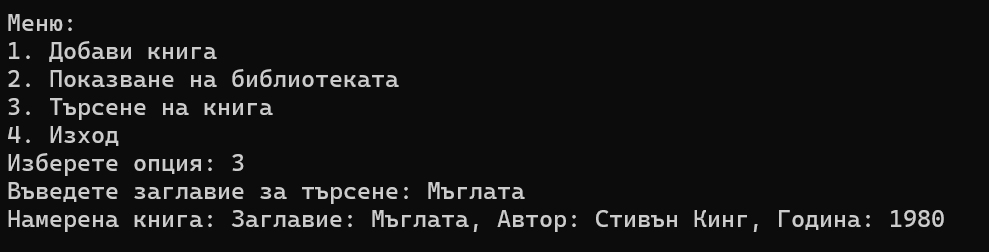
***Изход на конзолата при „Опция 2 – Показване на библиотеката“, в случай, че самата тя е празна***

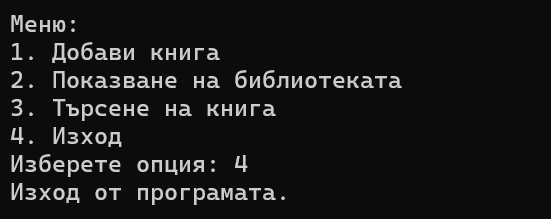


***Изход на конзолата при „Опция 3 – Търсене на книга“, в случай, че такава не е намерена***

***Добавяне на книга с параметри(входни данни) чрез използване на „Опция 1 – Добави книга“***

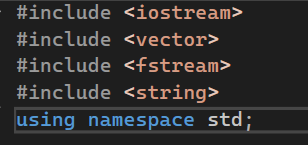
***Изход на конзолата при „Опция 2 – Показване на библиотеката“ след добавяне на нова книга***

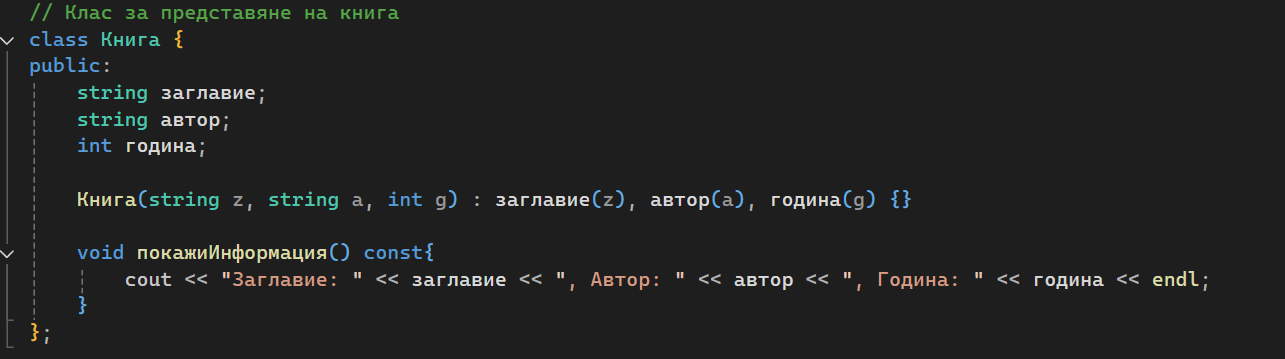
***Изход на конзолата при „Опция 3 – Търсене на книга“ с параметър: заглавие***

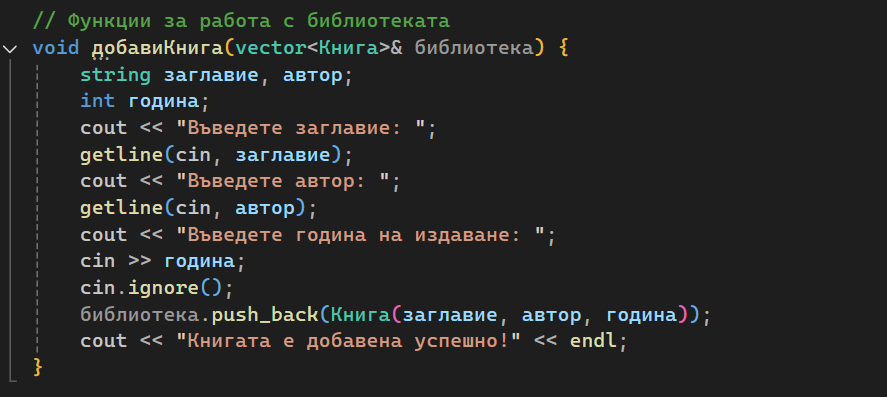
***Изход на конзолата при „Опция 4 – Изход“***

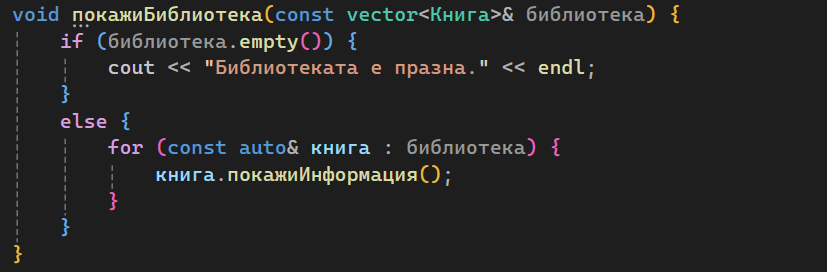
***СОРС КОД на Приложението:***

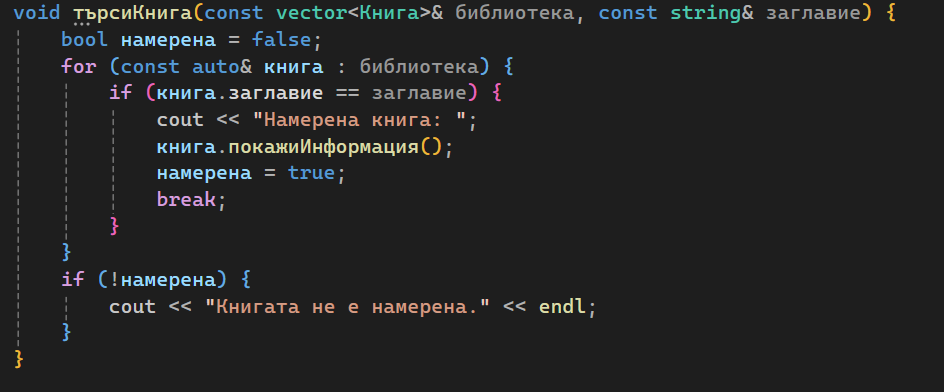
***Използвани библиотеки:***

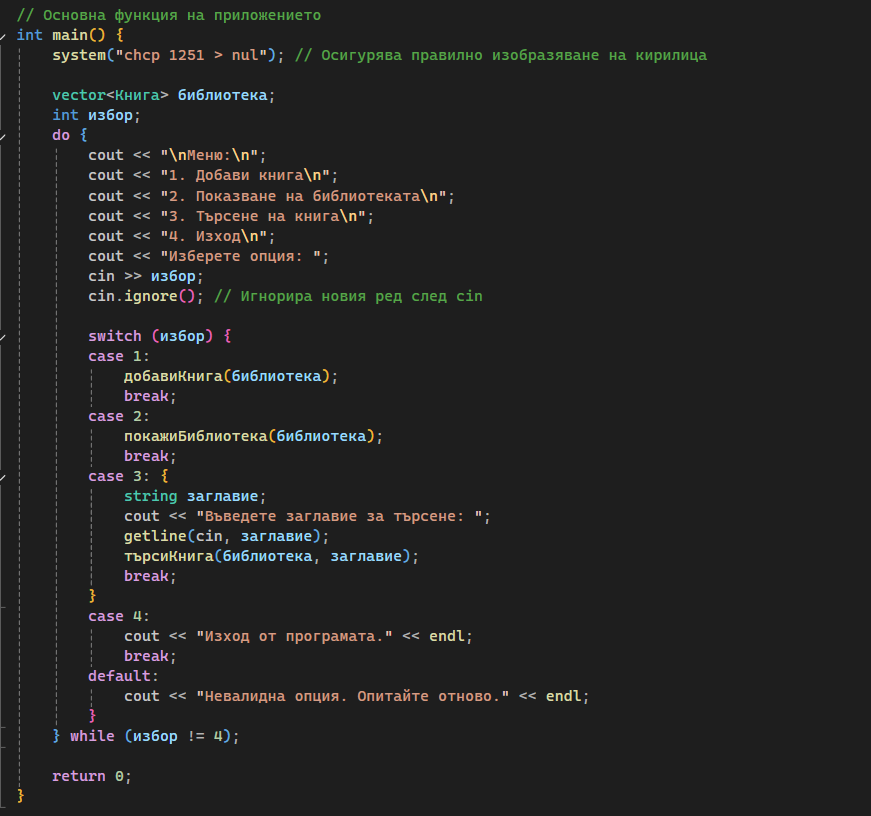
******

***Клас – Книга(заглавие, автор, година):***

***Метод за добавяне на нова книга:***

***Метод за извеждане/показване на всички книги в библиотеката:***

***Метод за търсене на книга:***

***Основен метод main() - бизнес логика***