Наръчник
Как се научих да програмирам на C++ с помощта на изкуствен интелект

Шенер Юмер, 2401321044

Използван LLM: Aria

Съдържание

Предговор

Здравей, скъпи читателю!

Добре дошъл в твоето пътуване през света на програмирането, пътуване, което започна с любопитство и ще завърши с осъзнаването, че дори изкуственият интелект може да бъде учител.

Тази книга е както проектна работа, така и плод на моя опит в изучаването на `C++` , език, който е едновременно мощен и взискателен. С помощта на изкуствения интелект Aria от Opera GX, аз се научих да разбирам абстрактни концепции, да решавам сложни проблеми и да създавам код, който работи.

На тези страници ще ви покажа уроците, които научих от Aria, и стъпка по стъпка ще ви науча как и вие да овладеете тънкостите на езика `C++` . Ще ви докажа, че чрез помощта на изкуственият интелект можете да извлечете изключително много информация за изучаването на програмни езици и по този начин да се вдъхновите да се впуснете в света на програмирането.

Нека това пътуване ви вдъхнови да преоткриете собствения си потенциал и да осъзнаете, че нищо не е невъзможно!

Глава 1

Въведение

Добре дошли в света на C++ ! Този език за програмиране е истински гигант, който стои в основата на безброй приложения, игри и технологии, които използваме всеки ден. C++ е език, който ви дава мощта да създавате сложни и ефективни програми, да контролирате хардуера на компютъра си и да реализирате най-смелите си идеи.

Но C++ не е за начинаещи. Той е мощен и гъвкав, но е и сложен и изисква задълбочено разбиране.

Какво прави C++ толкова специален?

– Обектно-ориентирано програмиране - C++ е език, който ви позволява да структурирате програмите си около обекти, които съдържат данни и функции. Това е като да създадете симулация на реалния свят в код, където всеки обект е отделен елемент с собствени характеристики и поведение.

– Висока производителност - C++ е известен с ефективността си. Той ви дава пълен контрол над ресурсите на компютъра и ви позволява да създавате приложения, които работят бързо и ефективно.

– Гъвкавост - C++ е гъвкав език, който ви позволява да разработвате разнообразни приложения, от операционни системи и игри до приложения за мобилни устройства.

– Широко разпространен - C++ е широко разпространен език, който се използва от милиони програмисти по целия свят. Това означава, че ще имате лесен достъп до ресурси, общности и поддръжка.

Защо да се учите на C++ ?

– Мощни приложения - C++ ви дава мощта да създавате комплексни и ефективни приложения, които могат да решават трудни задачи.

– Дълбоко разбиране - C++ ви учи да разбирате как работи компютърът и как да управлявате ресурсите му.

– Отворена врата към нови възможности - C++ е отворена врата към широк спектър от професионални възможности.

В тази книга ще ви запознаем с основите на C++ и ще ви покажем как да създавате свои собствени програми. Пригответе се за вълнуващо пътешествие в света на програмирането!

1.1 Първи стъпки в C++

В тази глава ще се запознаем с основите на C++ , като започнем с компилация, компилиране и изпълнение на програми.

1.1.1 Компилация и изпълнение

C++ е компилиран език. Това означава, че кодът, който пишете, трябва да бъде преведен на машинно разбираем език, преди да може да се изпълни.

Компилацията е процес, който превръща изходния код (текстовият файл, който вие пишете) в изпълним файл. Изпълним файл е файл, който може да се изпълни от компютъра.

За да компилирате и изпълните C++ програма, ще ви е необходим компилатор. Компилатор е програма, която превежда изходния код на C++ в изпълним файл.

1.1.2 Как да компилираме и изпълним програма?

Ето стъпките, които трябва да следвате, за да компилирате и изпълните C++ програма:

– Създайте нов текстов файл с разширение .cpp.

– Напишете C++ кода си в този файл.

– Отворете командния ред (или терминал) и отидете до директорията, където е вашият текстов файл.

– Въведете следната команда, за да компилирате програмата:
`g++ име_на_файла.cpp -o име_на_изпълним_файл`

– Въведете следната команда, за да изпълните програмата:
`./име_на_изпълним_файл`

Пример:

Ако вашият файл се казва `hello.cpp` и искате да създадете изпълним файл `hello`, тогава трябва да въведете следните команди:

```
g++ hello.cpp -o hello
./hello
```

1.1.3 Имплементиране на български език в C++

За да използваме български език в кода на C++ и в конзолата, трябва първо да зададем локализацията на проекта:

```
#include <locale>

int main() {
    setlocale(LC_ALL, "Bulgarian");
        // Локализация на кирилицата в проекта

    return 0;
}
```

`#include <locale>` - Тази линия включва библиотеката `locale`, която ни дава достъп до функции за локализация на езици.

`setlocale(LC_ALL, "Bulgarian");` - Стандартна функция за локализация на български език.

1.1.4 Първата ни програма

Ето пример за проста C++ програма, която извежда текст на екрана:

```
#include <iostream>
#include <locale>

int main() {
    setlocale(LC_ALL, "Bulgarian");

    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

`#include <iostream>` - Тази линия включва библиотеката `iostream`, която ни дава достъп до функции за вход и изход.

`int main()` - Тази линия дефинира главната функция на програмата. Всички C++ програми трябва да имат главна функция.

`std::cout << "Hello, world!" << std::endl;` - Тази линия извежда текста `"Hello, world!"` на екрана.

`return 0;` - Тази линия завършва изпълнението на програмата.

1.2 Ключови елементи на синтаксиса и семантиката на C++

1.2.1 Ключови думи

Ключовите думи в C++ са резервирани думи, които имат специално значение за компилатора. Те не могат да се използват като имена на променливи, функции или други идентификатори.

Ето някои от най-важните ключови думи в C++ :

Ключова дума	Описание
`int, float, double, char, bool, void, auto, const, constexpr, decltype`	Типове данни
`if, else, else if, switch, case, default, break, continue, goto`	Условни оператори
`for, while, do while, break, continue`	Цикли
`return, sizeof, new, delete, nullptr, this`	Оператори
`namespace, using, struct, class, enum, union, template, typename, friend, operator`	Организъм на кода
`public, private, protected, static, virtual, override, final, explicit`	Модификатори за достъп
`inline, extern, volatile, mutable, register`	Модификатори за компилация
`try, catch, throw, noexcept`	Обработка на изключения

Ключовите думи се използват в C++ код, за да се определят типове данни, структури, класове, функции и други елементи на програмата.

Пример:

```
int main() {
    int a = 10; // int е ключова дума за дефиниране на целочислена променлива
    if (a > 5) { // if е ключова дума за условен оператор
        std::cout << "a is greater than 5" << std::endl;
    }
```

```
6       return 0; // return е ключова дума за връщане на статус
7   }
```

1.2.2 Специални символи

В C++ езикът, освен букви, цифри и ключови думи, има и специални символи, които имат специално значение за компилатора. Тези символи се използват за определяне на оператори, разделителни символи, коментари и други елементи на кода.

Оператори

Операторите са специални символи, които извършват операции върху операнди.

Пример:

Аритметика

`+` (събиране): `a + b`

`-` (изваждане): `a - b`

`*` (умножение): `a * b`

`/` (деление): `a / b`

`%` (остатък от деление): `a % b`

`++` (увеличаване): `a++`

`--` (намаляване): `a--`

Условни

`==` (равенство): `a == b`

`!=` (неравенство): `a != b`

`>` (по-голямо): `a > b`

`<` (по-малко): `a < b`

`>=` (по-голямо или равно): `a >= b`

`<=` (по-малко или равно): `a <= b`

`&&` (логическо И): `a && b`

`||` (логическо ИЛИ): `a || b`

`!` (логическо НЕ): `!a`

`?:` (тернарен условен оператор): `a ? b : c`

Битова аритметика

`&` (битово И): `a & b`

`|` (битово ИЛИ): `a | b`

`^` (битово ИЗКЛ. ИЛИ): `a ^ b`

`~` (битово НЕ): `~a`

`<<` (ляво битово изместване): `a << b`

`>>` (дясно битово изместване): `a >> b`

Разделителни символи

Разделителните символи се използват за разделяне на различни части на `C++` кода.

Пример:

`;` (точка и запетая): `int a = 10;`

`,` (запетая): `int a = 10, b = 20;`

`:` (двоеточие): `switch (a) { case 1: ...; }`

`::` (обхват на име): `std::cout`

`.` (член на клас): `a.b`

`->` (член на указател): `a->b`

`[]` (индексиране): `a[b]`

`()` (извикване на функция): `a()`

`{}` (блокове код): `{ ... }`

Коментари

Коментарите са текст, който се игнорира от компилатора. Те се използват за обяснение на кода, добавяне на документация или деактивиране на част от кода.

Пример:

- `//` (едноредов коментар):

```
// This is a single line comment.
```

- `/* ... */` (многоредов коментар):

```
/*
 * This is a multiline comment.
 * It can extend on multiple lines.
 */
```

1.2.3 Идентификатори

Идентификаторите в `C++` са имена, които се използват за означаване на променливи, функции, класове, структури, изброявания, области на имена и други елементи на програмата.

Правила за идентификатори

Идентификаторите могат да се състоят от букви, цифри, подчертаване (_). Първият символ на идентификатора не може да бъде цифра. Ключовите думи не могат да се използват като идентификатори. Чувствителност към регистъра: `myVariable` и `MyVariable` са различни идентификатори.

Примери за идентификатори

– Променлива: `age, firstName, totalScore`

– Функция: `calculateArea, printMessage, sortArray`

– Клас: `Person, Car, Database`

– Структура: `Point, Date, Time`

– Изброяване: `Color, Status, Direction`

– Област на имена: `std, myNamespace, utils`

Препоръки за идентификатори

Използвайте описателни имена, които отразяват целта на идентификатора. Използвайте `camelCase` или `snake_case` за по-добра четимост. Избягвайте къси и неясни имена. Не използвайте резервирани думи като идентификатори.

Пример за код с идентификатори:

```cpp
#include <iostream>

using namespace std; // Дефиниране на областта на имената "std"

int main() {
    // Дефиниране на променлива с име "age"
    int age = 25;

    // Дефиниране на функция с име "printMessage"
    void printMessage(string message) {
    cout << message << endl;
    }

    // Извикване на функцията "printMessage"
    printMessage("Hello, world!");

    return 0;
}
```

1.2.4 Литерали

Литералите в C++ са константни стойности, които се използват за представяне на данни в програмата. Те са директни представяния на данни, които се компилират директно в код.

Видове литерали

C++ поддържа различни видове литерали, в зависимост от типа на данните:

- Числови литерали:
  - Цялочислени литерали:

    Десетични: `10, 25, -15`

    Осмоични: `012, 037` (започват с `0`)

    Шестнадесетични: `0x1A, 0x2F` (започват с `0x`)
  - Дробни литерали: `3.14, 1.5e-2` (експоненциална нотация)
- Символни литерали:
  - Обикновени: `'a', 'B', '%'`

  - Escape последователности: `'\n'`, `'\t'`, `'\''`

- Текстови литерали:
  - Обикновени: `"Здравей, свят!` `"Hello, world!"`
  - Raw string литерали: `R"(C:\Users\MyUser\Documents)"` (за запазване на escape последователности)
- Булеви литерали: `true`, `false`
- Указателни литерали: `nullptr` (за празен указател)

Пример за код с литерали

```cpp
#include <iostream>

using namespace std;

int main() {
    // Цялочислени литерали
    int age = 25;
    int octalNumber = 012;
    int hexNumber = 0x1A;

    // Дробни литерали
    double pi = 3.14;
    double smallNumber = 1.5e-2;

    // Символни литерали
    char character = 'A';
    char newline = '\n';

    // Текстови литерали
    string message = "Hello, world!";
    string path = R"(C:\Users\MyUser\Documents)";

    // Булеви литерали
    bool isTrue = true;
    bool isFalse = false;

    // Указателни литерали
    int* ptr = nullptr;

    return 0;
}
```

1.2.5 Променливи и константи

Променливи

Променливите в C++ са имена, които се използват за съхраняване на данни в паметта. Тези данни могат да бъдат променяни по време на изпълнението на програмата.

1. Дефиниране на променливи:

```
<datatype> <variable_name>;
```

Пример:

```
int age; // Дефиниране на променлива от тип "int" с име "age"
double price; // Дефиниране на променлива от тип "double" с име "price"
string name; // Дефиниране на променлива от тип "string" с име "name"
```

2. Инициализиране на променливи
   Инициализирането на променлива е процесът на присвояване на начална стойност при дефинирането.

Пример:

```
int age = 25; // Инициализиране на променливата "age" със стойност 25
// Инициализиране на променливата "price" със стойност 19.99
double price = 19.99;
// Инициализиране на променливата "name" със стойност "Ivan"
string name = "Ivan";
```

3. Използване на променливи
   След дефинирането и инициализирането, променливите могат да се използват в програмата.

Пример:

```
int age = 25;
// Извеждане на стойността на променливата "age"
cout << "Your age is: " << age << endl;
```

Константи

Константите в C++ са имена, които се използват за съхраняване на данни в паметта, но стойностите им не могат да се променят по време на изпълнението на програмата.

1. Дефиниране на константи
   За да се дефинира константа, се използва ключовата дума `const`:

```
const <datatype> <NAME_OF_CONSTANT> = <value>;
```

Пример:

```
// Дефиниране на константа от тип "int" с име "MAX_AGE" със стойност 120
const int MAX_AGE = 120;
// Дефиниране на константа от тип "double" с име "PI" със стойност 3.14159
const double PI = 3.14159;
// Дефиниране на константа от тип "string" с име "GREETING"
// със стойност "Greetings!"
const string GREETING = "Greetings!";
```

2. Използване на константи
   Константите могат да се използват в програмата по същия начин като променливите.

Пример:

```
const int MAX_AGE = 120;
int age = 25;
if (age > MAX_AGE) {
    cout << "Invalid age!" << endl;
}
```

1.2.6 Подпрограми (Функции)

Функциите в C++ са блокове от код, които изпълняват конкретна задача. Те могат да приемат аргументи и връщат резултат.

Функции с тип

Функциите с тип връщат резултат от конкретен тип.

Пример:

```
int sum(int a, int b) {
    return a + b;
}
```

В този пример функцията `sum` приема два целочислени аргумента (`a` и `b`) и връща цяло число (`int`), което е сумата на двата аргумента.

Void функции

`Void` функциите не връщат резултат. Те се използват за изпълнение на действия, които не връщат стойност.

Пример:

```
void printHello() {
    cout << "Hello, world!" << endl;
}
```

В този пример функцията `printHello` не връща резултат. Тя просто извежда текст на конзолата.

Аргументи на функцията

Аргументите на функцията са стойности, които се предават на функцията при викането ѝ.

Пример:

```
int sum(int a, int b) {
    return a + b;
}

int main() {
    int result = sum(10, 20); // Предаване на аргументите 10 и 20
    cout << result << endl; // Извеждане на резултата (30)
    return 0;
}
```

В този пример функцията `sum` приема два целочислени аргумента (`a` и `b`). При викането ѝ в `main` функцията, се предават стойностите `10` и `20` за `a` и `b` съответно.

Предаване по стойност

При предаване по стойност, на функцията се предава копия на аргументите.
Пример:

```
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 10;
    int y = 20;
    swap(x, y); // Предаване по стойност
    cout << x << " " << y << endl; // Извеждане на "10 20"
    return 0;
}
```

В този пример, функцията `swap` не модифицира оригиналните стойности на `x` и `y`, защото работи с копия.

Предаване по препратка

При предаване по препратка, на функцията се предава адреса на аргументите.
Пример:

```
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 10;
    int y = 20;
    swap(x, y); // Предаване по препратка (директен адрес)
    cout << x << " " << y << endl; // Извеждане на "20 10"
    return 0;
}
```

В този пример, функцията swap модифицира оригиналните стойности на `x` и `y`, защото работи с адресите им.

Рекурсия

Рекурсията е техника, при която функция се вика сама себе си.

Пример:

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

В този пример функцията `factorial` изчислява факториела на число чрез рекурсивната формула $n! = n \cdot (n-1)!$.

Важно: Рекурсията трябва да има базов случай, който прекратява рекурсивните викания. В противен случай програмата изпада в безкраен цикъл, докато не се стигне до момента, в който паметта, нужна за запазване на променливите и информацията, надвиши разпределената за процеса памет в стека (Stack overflow).

Lambda функции

Lambda функциите са анонимни функции, които могат да се дефинират и използват в една линия код.

Пример:

```
auto sum = [](int a, int b) {
    return a + b;
};
int result = sum(10, 20); // Извикване на lambda функцията
```

В този пример lambda функцията `sum` приема два целочислени аргумента (`a` и `b`) и връща цяло число (`int`), което е сумата на двата аргумента.

1.2.7 Атрибути

Атрибутите в `C++` са специални ключови думи, които модифицират поведението на променливи, функции, класове и други елементи на кода. Те определят важни характеристики, като обхват, вид, жизнен цикъл, достъп и други.

Const

Атрибутът `const` определя, че стойността на променливата не може да се променя след инициализирането ѝ. `Const` гарантира, че стойността няма да се промени неволно, подобрявайки безопасността на кода.

Пример:

```
const int PI = 3.14159;
```

В този пример `PI` е константа с стойност `3.14159`.

Static

Атрибутът `static` определя, че променливата е статична. Статичните променливи съществуват само в рамките на файла, в който са дефинирани. Инициализират се само веднъж при първото викане на файла и живеят цял живот на програмата. Споделят се между всички функции в файла.

Пример:

```
static int count = 0;
```

В този пример `count` е статична променлива.

Extern

Атрибутът `extern` определя, че променливата е дефинирана в друг файл. Атрибутът `extern` не инициализира променливата, само указва, че тя съществува някъде другe.

Пример:

```
extern int count;
```

В този пример `count` е променлива, която е дефинирана в друг файл.

Volatile

Атрибутът `volatile` указва, че стойността на променливата може да се променя външно, без да се вижда от компилатора. Атрибутът `volatile` предотвратява компилатора да оптимизира кода, който работи с променливата. Използва се за променливи, чиято стойност може да се промени от външни фактори, като прекъсвания, таймери или други процеси.

Пример:

```
volatile int counter;
```

В този пример `counter` е променлива, чиято стойност може да се променя от външен код.

Register

Атрибутът `register` препоръчва на компилатора да съхранява променливата в регистър на процесора. Атрибутът `register` е само препоръка. Не гарантира, че компилаторът ще съхрани променливата в регистър. Използва се за често използвани променливи, за да се подобри ефективността.

Пример:

```
register int i;
```

В този пример `i` е променлива, която компилаторът може да съхрани в регистър.

Auto

Атрибутът `auto` позволява на компилатора да определи типа на променливата автоматично. `Auto` улеснява кода, когато типът на променливата е ясен от инициализацията.

Пример:

```
auto x = 10;
```

В този пример `x` е променлива с тип `int`, определен автоматично от компилатора.

1.3 Стандартен конзолен I/O в C++

1.3.1 Работа с конзолните вход и изход за данни

1.3.2 Работа със `string` данни

1.4 Структуриране на C++ програма

1.4.1 Директиви към предпроцесора

1.4.2 Области на имената

Глава 2

Типове данни

2.1 Типове данни за цели и реални числа

2.2 Типове данни за символ и низ

2.3 Логически данни

Глава 3

Конструкции за поточен контрол

3.1 Конструкции за разклонение

!! Ternary operator

3.2 Конструкции за цикъл

3.3 Конструкции за прекъсване

Глава 4

Съставни типове данни

4.1 Масиви

4.2 Структури

4.3 Класове

4.4 Обединения

4.5 Изброяване

4.6 Колекции

Глава 5

Манипулиране на паметта

5.1 Указатели

5.2 Референции

5.3 Адресна аритметика

5.4 Динамично и статично разпределяне на паметта