

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

Шенер Юмер, 2401321044

Използван LLM: Aria

Съдържание

Предговор	3
1 Въведение	4
1.1 Първи стъпки в C++	5
1.1.1 Компиляция и изпълнение	5
1.1.2 Как да компилираме и изпълним програма?	5
1.1.3 Имплементиране на български език в C++	6
1.1.4 Първата ни програма	6
1.2 Ключови елементи на синтаксиса и семантиката на C++	7
1.2.1 Ключови думи	7
1.2.2 Специални символи	8
1.2.3 Идентификатори	10
1.2.4 Литерали	11
1.2.5 Променливи и константи	13
1.2.6 Подпрограми (Функции)	15
1.2.7 Атрибути	18
1.3 Стандартен конзолен I/O в C++	20
1.3.1 Работа със системната конзола	20
1.3.2 Работа със <code>string</code> данни	22
1.4 Структуриране на C++ програма	25
1.4.1 Директиви към предпроцесора	25
1.4.2 Области на имената	28
2 Типове данни	29
2.1 Типове данни за цели и реални числа	29
2.2 Типове данни за символ и низ	29
2.3 Логически данни	29
3 Конструкции за поточен контрол	30
3.1 Конструкции за разклонение	30

<i>СЪДЪРЖАНИЕ</i>	<i>2</i>
3.2 Конструкции за цикъл	30
3.3 Конструкции за прекъсване	30
4 Съставни типове данни	31
4.1 Масиви	31
4.2 Структури	31
4.3 Класове	31
4.4 Обединения	31
4.5 Изброяване	31
4.6 Колекции	31
5 Манипулиране на паметта	32
5.1 Указатели	32
5.2 Референции	32
5.3 Адресна аритметика	32
5.4 Динамично и статично разпределяне на паметта	32
6 Алгоритми	33
Реализация на софтуерно приложение	34

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 Agia от Orega GX, аз се научих да разбирам абстрактни концепции, да решавам сложни проблеми и да създавам код, който работи.

На тези страници ще ти покажа уроците, които научих от Agia, и стъпка по стъпка ще те науча как и ти да овладееш тънкостите на езика C++ . Ще ти докажа, че чрез помощта на изкуственият интелект можете да извлечеш изключително много информация за изучаването на програмни езици и по този начин да се вдъхновиш да се впуснеш в света на програмирането.

Нека това пътуване те вдъхнови да преоткриеш собствения си потенциал и да осъзнаеш, 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 изпълняват 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:

```
1 g++ hello.cpp -o hello
2 ./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:

```
1 #include <locale>
2
3 int main() {
4     setlocale(LC_ALL, "Bulgarian");
5     // Локализация на кирилицата в проекта
6
7     return 0;
8 }
```

`#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:

```
1 #include <iostream>
2 #include <locale>
3
4 int main() {
5     setlocale(LC_ALL, "Bulgarian");
6
7     std::cout << "Hello, world!" << std::endl;
8     return 0;
9 }
```

`#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.

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

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

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):

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

- `/* ... */` (многоредов коментар):

```
1 /*  
2  * This is a multiline comment.  
3  * It can extend on multiple lines.  
4  */
```

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:

```
1 #include <iostream>
2
3 using namespace std; // Дефиниране на областта на имената "std"
4
5 int main() {
6     // Дефиниране на променлива с име "age"
7     int age = 25;
8
9     // Дефиниране на функция с име "printMessage"
10    void printMessage(string message) {
11        cout << message << endl;
12    }
13
14    // Извикване на функцията "printMessage"
15    printMessage("Hello, world!");
16
17    return 0;
18 }
```

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

- Осмойични: 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

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Цялочислени литерали
7     int age = 25;
8     int octalNumber = 012;
9     int hexNumber = 0x1A;
10
11     // Дробни литерали
12     double pi = 3.14;
13     double smallNumber = 1.5e-2;
14
15     // Символни литерали
16     char character = 'A';
17     char newline = '\n';
18
19     // Текстови литерали
20     string message = "Hello, world!";
21     string path = R"(C:\Users\MyUser\Documents)";
22
23     // Булеви литерали
24     bool isTrue = true;
25     bool isFalse = false;
```

```
26
27     // Указателни литерали
28     int* ptr = nullptr;
29
30     return 0;
31 }
```

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:

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

Пример:

```
1 int age; // Дефиниране на променлива от тип "int" с име "age"
2 double price; // Дефиниране на променлива от тип "double" с име "price"
3 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:

```
1 int age = 25; // Инициализиране на променливата "age" със стойност 25
2 // Инициализиране на променливата "price" със стойност 19.99
3 double price = 19.99;
4 // Инициализиране на променливата "name" със стойност "Ivan"
5 string name = "Ivan";
```

3. Използване на променливи

След дефинирането и инициализирането, променливите могат да се използват в програмата.

Пример:

```
1 int age = 25;
2 // Извеждане на стойността на променливата "age"
3 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`:

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

Пример:

```
1 // Дефиниране на константа от тип "int" с име "MAX_AGE" със стойност 120
2 const int MAX_AGE = 120;
3 // Дефиниране на константа от тип "double" с име "PI" със стойност 3.14159
4 const double PI = 3.14159;
5 // Дефиниране на константа от тип "string" с име "GREETING"
6 // със стойност "Greetings!"
7 const string GREETING = "Greetings!";
```

2. Използване на константи

Константите могат да се използват в програмата по същия начин като променливите.

Пример:

```
1 const int MAX_AGE = 120;
2 int age = 25;
3 if (age > MAX_AGE) {
4     cout << "Invalid age!" << endl;
5 }
```

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:

```
1 int sum(int a, int b) {  
2     return a + b;  
3 }
```

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:

```
1 void printHello() {  
2     cout << "Hello, world!" << endl;  
3 }
```

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:

```
1 int sum(int a, int b) {  
2     return a + b;  
3 }  
4  
5 int main() {  
6     int result = sum(10, 20); // Предаване на аргументите 10 и 20  
7     cout << result << endl; // Извеждане на резултата (30)  
8     return 0;  
9 }
```

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:

```
1 void swap(int a, int b) {  
2     int temp = a;  
3     a = b;  
4     b = temp;  
5 }  
6  
7 int main() {  
8     int x = 10;  
9     int y = 20;  
10    swap(x, y); // Предаване по стойност  
11    cout << x << " " << y << endl; // Извеждане на "10 20"  
12    return 0;  
13 }
```

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:

```
1 void swap(int& a, int& b) {  
2     int temp = a;  
3     a = b;  
4     b = temp;  
5 }  
6  
7 int main() {  
8     int x = 10;  
9     int y = 20;  
10    swap(x, y); // Предаване по препратка (директен адрес)  
11    cout << x << " " << y << endl; // Извеждане на "20 10"  
12    return 0;  
13 }
```

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:

```
1 int factorial(int n) {  
2     if (n == 0) {  
3         return 1;  
4     } else {  
5         return n * factorial(n - 1);  
6     }  
7 }
```

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:

```
1 auto sum = [](int a, int b) {  
2     return a + b;  
3 };  
4 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:

```
1 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:

```
1 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 другде.

Пример:

```
1 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:

```
1 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:

```
1 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:

```
1 auto x = 10;
```

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

1.3 Стандартен конзолен I/O в C++

В тази част ще разгледаме как се работи със системната конзола за стандартно въвеждане и извеждане на информация в C++ и как да манипулираме получените данни от конзолата, а именно да обработваме информация от `string` данните.

1.3.1 Работа със системната конзола

В C++ за стандартно въвеждане и извеждане на данни се използва пакетът `<iostream>`. Той предоставя обекти, които улесняват взаимодействието с потребителя и файловете.

- `std::cin` - Стандартен входен поток (конзола).
- `std::cout` - Стандартен изходен поток (конзола).
- `std::cerr` - Стандартен изходен поток за грешки (конзола).
- `std::endl` - `'\n'` (Нов ред).

Операторите `>>` и `<<` са специални оператори, които улесняват взаимодействието с `cin` и `cout`.

- `>>` - Операторът за въвеждане.
- `<<` - Операторът за извеждане.

Пример:

```
1 #include <iostream>
2
3 int main() {
4     int number;
5     std::cout << "Insert a number: "; // Извеждане на текст на конзолата
6     std::cin >> number; // Въвеждане на число от конзолата
7     // Извеждане на числото на конзолата
8     std::cout << "The inserted number is: " << number << std::endl;
9     return 0;
10 }
```

Пакет <iomanip>

Пакетът <iomanip> предоставя методи за манипулиране на входящите и изходящите данни.

Пример:

```
1 #include <iostream>
2 #include <iomanip>
3
4 int main() {
5     double number = 123.456789;
6     std::cout << "The number is: "
7         << std::setprecision(4) << number << std::endl;
8     std::cout << "The number is: "
9         << std::setw(10) << number << std::endl;
10    return 0;
11 }
```

В този пример:

`std::setprecision(4)` - ограничава точните цифри на числото до 4.

`std::setw(10)` - задава ширина на полето за извеждане на числото на 10 символа.

Резултат:

```
1 The number is: 123.5
2 The number is:      123.5
```

Допълнителни методи

Пакетът <iomanip> предлага много други методи за форматиране на входящите и изходящите данни.

Някои от тях са:

- `std::fixed` - Извежда числа с фиксирана точка.
- `std::scientific` - Извежда числа в научен запис.
- `std::left` - Подравнява текста вляво.
- `std::right` - Подравнява текста вдясно.

1.3.2 Работа със string данни

В C++ класът `std::string` предоставя мощен инструмент за работа с текстови низове. Той осигурява множество функции за манипулиране, сравняване, търсене и модифициране на текстови низове.

Инициализиране на string

Има няколко начина за инициализиране на string обект:

Празен string:

```
1 std::string text; // Инициализира празен string
```

С текстова константа:

```
1 // Инициализира string с текстова константа
2 std::string text = "Hello, world!";
```

С друг string:

```
1 std::string text1 = "Hello, ";
2 // Инициализира string с конкатенация на други string-ове
3 std::string text2 = text1 + "world!";
```

С масив от символи:

```
1 char text[] = "Hello, world!"; // string в стил C
2 // Инициализира string в стил C++ с масив от символи
3 std::string textString(text);
```

Основни функции

<code>getline(istream& is, string& str, char delim)</code>	Въвежда редове от стандартен вход (<code>cin</code>) в <code>string</code> обект. <code>delim</code> определя разделителя на редовете. По подразбиране разделителят е <code>'\n'</code> (нов ред).
<code>length()</code>	Връща дължината на текстовия низ.
<code>at(size_t pos)</code>	Връща символа на позиция <code>pos</code> в текстовия низ.
<code>append(const string& str) "+"</code>	Добавя текстов низ <code>str</code> към края на текстовия низ.
<code>compare(const string& str) "=="</code>	Сравнява текстовия низ с <code>str</code> . Връща 0, ако низовете са равни; <0, ако текстовият низ е по-къс от <code>str</code> ; >0, ако текстовият низ е по-дълъг от <code>str</code> .
<code>substr(size_t pos, size_t len)</code>	Връща подниз от текстовия низ, започвайки от позиция <code>pos</code> и с дължина <code>len</code> .
<code>find(const string& str, size_t pos)</code>	Търси подниз <code>str</code> в текстовия низ, започвайки от позиция <code>pos</code> . Връща позицията на първото намиране на <code>str</code> , ако е намерен, иначе връща <code>string::npos</code> .
<code>replace(size_t pos, size_t len, const string& str)</code>	Заменя подниз от текстовия низ, започвайки от позиция <code>pos</code> и с дължина <code>len</code> с <code>str</code> .
<code>insert(size_t pos, const string& str)</code>	Вмъква <code>str</code> в текстовия низ на позиция <code>pos</code> .
<code>erase(size_t pos, size_t len)</code>	Премахва подниз от текстовия низ, започвайки от позиция <code>pos</code> и с дължина <code>len</code> .

Примери:

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main() {
7     string text = "Hello, world!";
8
9     // Въвеждане на текст от конзолата
10    cout << "Insert text: ";
11    getline(cin, text);
12
13    // Извеждане на дължината на текста
14    cout << "The length of the text is: " << text.length() << endl;
15
16    // Извеждане на символа на позиция 5
17    cout << "The character at position 5 is: "
18         << text.at(5) << endl;
19
20    // Добавяне на текст към края
21    text.append(" How are you?");
22    cout << "The text is: " << text << endl;
23
24    // Сравняване на два текста
25    string otherText = "Hello, world!";
26    if (text.compare(otherText) == 0) {
27        cout << "The texts are equal." << endl;
28    } else {
29        cout << "The texts are not equal." << endl;
30    }
31
32    // Извеждане на подниз
33    cout << "The substring from position 7 to the end is: "
34         << text.substr(7) << endl;
35
36    // Търсене на подниз
37    size_t pos = text.find("world");
38    if (pos != string::npos) {
39        cout << "The substring \"world\" was found at position: "
40             << pos << endl;
41    } else {
42        cout << "The substring \"world\" wasn't found." << endl;
43    }
44 }
```

```
44
45 // Замяна на подниз
46 text.replace(7, 5, "universe");
47 cout << "The text is: " << text << endl;
48
49 // Вмъкване на текст
50 text.insert(7, "beautiful ");
51 cout << "The text is: " << text << endl;
52
53 // Премахване на текст
54 text.erase(7, 10);
55 cout << "The text is: " << text << endl;
56
57 return 0;
58 }
```

Резултат: (Примерен входен текст: Hello, world!)

```
1 Insert text: >>Hello, world!
2 The length of the text is: 13
3 The character at position 5 is: ,
4 The text is: Hello, world! How are you?
5 The texts are not equal.
6 The substring from position 7 to the end is: world! How are you?
7 The substring "world" was found at position: 7
8 The text is: Hello, universe! How are you?
9 The text is: Hello, beautiful universe! How are you?
10 The text is: Hello, universe! How are you?
```

1.4 Структуриране на C++ програма

1.4.1 Директиви към предпроцесора

Предпроцесорът е ключов компонент в C++ компилационния процес, който обработва кода преди да бъде компилиран. Той изпълнява специални инструкции, наречени директиви, които променят структурата на кода преди да бъде предаден на компилатора.

Най-често използваните директиви към предпроцесора в C++ са:

#include

#include е най-често използваната директива. Тя включва съдържанието на друг файл в текущия файл. Това е ключово за организирането на C++ код в

отделни файлове, например за дефиниране на функции или класове в отделни `.h` файлове.

Стандартни файлове: `#include` се използва за включване на стандартни библиотеки, например:

```
1 // Включва стандартния header файл за входящи и изходящи операции
2 #include <iostream>
```

Потребителски файлове: `#include` се използва за включване на файлове, създадени от програмиста, например:

```
1 #include "my_functions.h" // Включва файл с дефиниции на функции
```

`#define`

`#define` се използва за дефиниране на константи и макроси. Константите са променливи, чиято стойност не може да бъде променена след дефинирането им. Макросите са блокове код, които се заменят с дефинираното съдържание по време на предпроцесорната обработка.

Дефиниране на константи:

```
1 #define PI 3.14159 // Дефинира константа PI
2 #define MAX_SIZE 100 // Дефинира константа MAX_SIZE
```

Дефиниране на макроси:

```
1 #define SQUARE(x) (x * x) // Дефинира макрос за изчисляване на квадрат
2 // Дефинира макрос за извеждане на съобщение
3 #define PRINT_MESSAGE(msg) std::cout << msg << std::endl;
```

`#undef`

`#undef` се използва за премахване на предишно определение на константа или макрос.

```
1 #undef PI // Премахва определението на PI
```

`#ifdef`, `#ifndef`, `#else`, `#endif`

Тези директиви се използват за условно компилиране на код. Тоест, определени части от кода се компилират само ако е изпълнено определено условие.

- `#if` - Позволява условно компилиране на код въз основа на резултата от препроцесорно условие. Това условие може да бъде дефинирана константа (`#if defined(MY_CONSTANT)`), макрос (`#if SQUARE(5) == 25`), оператори за сравнение (`#if 10 > 5`), логически оператори (`#if defined(DEBUG) && defined(RELEASE)`).
- `#ifdef` - Проверява дали константа или макрос е дефиниран.
- `#ifndef` - Проверява дали константа или макрос не е дефиниран.
- `#else` - Изпълнява се, ако условието в `#if`, `#ifdef` или `#ifndef` не е изпълнено.
- `#endif` - Завършва блока, дефиниран от `#if`, `#ifdef`, `#ifndef` или `#else`.

Пример:

```
1 #define SIX 6
2
3 #if defined(SIX)
4     #pragma message("The constant SIX is defined.")
5 #endif
6
7 #if (SIX > 5)
8     #pragma message("6 is greater than 5.")
9 #endif
10
11 #undef SIX
12
13 #ifdef SIX
14     // Този ред ще се компилира само ако е дефинирана константата SIX
15     #pragma message("The constant SIX is defined.")
16 #else
17     // Този ред ще се компилира само ако константата SIX не е дефинирана
18     #pragma message("The constant SIX is not defined.")
19 #endif
```

Резултат:

```
1 >| The constant SIX is defined.
2 >| 6 is greater than 5.
3 >| The constant SIX is not defined.
```

#pragma

#pragma е директива, която предоставя инструкции на компилатора. Тези инструкции са специфични за компилатора и могат да се различават между различните компилатори.

#pragma message(String) - Извежда съобщения в конзолата по време на компилирането. Това е полезно при дебъгване на компилаторни проблеми.

```
1 // Изписва съобщението "Hello, world!" редом с другите съобщения от компилатора
2 #pragma message("Hello, world!");
```

#pragma once - Предотвратява многократно включване на header файл. Това е полезно за предотвратяване на грешки при компилация, когато един и същ header файл се включва многократно.

```
1 #pragma once // Предотвратява многократно включване на header файл
```

#error

#error е директива, която генерира грешка по време на компилация. Това е полезно за сигнализиране на грешки, които не могат да бъдат открити от компилатора.

Пример:

```
1 #if !defined(MY_CONSTANT)
2     #error "The constant MY_CONSTANT is not defined!"
3 #endif
```

В този пример, ако константата `MY_CONSTANT` не е дефинирана, компилаторът ще генерира грешка с текст `"The constant MY_CONSTANT is not defined!"`.

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

За повече информация за работа със `string` обекти, виж (1.3.2)

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

Глава 6

Алгоритми

Реализация на софтуерно
приложение "... "