

Ravesli [Ravesli](#)


- [Уроки по C++](#)
- [OpenGL](#)
- [SFML](#)
- [Qt5](#)
- [RegExр](#)
- [Ассемблер](#)
- [Купить .PDF](#)


Урок №79. Строки C-style

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 11 Авг 2020 |

 41232

[↑](#)  7

На [уроке №57](#) мы определили термин «строка», как набор последовательных символов (например, Hello, world!). Строки — это основной способ работы с текстом в языке C++, а std::string упрощает этот способ работы.

Современный C++ поддерживает два разных типа строк:

- ➔ **std::string** (как часть Стандартной библиотеки C++);
- ➔ **строки C-style** (изначально унаследованные от языка Си).

std::string реализован с помощью строк C-style.

Оглавление:

1. [Строки C-style](#)
2. [Строки C-style и std::cin](#)
3. [Управление строками C-style](#)
4. [Стоит ли использовать строки C-style?](#)

Строки C-style

Строка C-style — это простой **массив** символов, который использует нуль-терминатор. **Нуль-терминатор** — это специальный символ ([ASCII-код](#) которого равен 0), используемый для обозначения конца строки. Строка C-style еще называется «**нуль-терминированной строкой**».

Для её определения нужно просто объявить массив типа `char` и инициализировать его литералом (например, `string`):

```
1 char mystring[] = "string";
```

Хотя `string` имеет только 6 букв, C++ автоматически добавляет нуль-терминатор в конец строки (нам не нужно добавлять его вручную). Следовательно, длина массива `mystring` на самом деле равна 7!

В качестве примера рассмотрим следующую программу, которая выводит длину строки, а затем ASCII-коды всех символов литерала `string`:

```
1 #include <iostream>
2
3 int main()
4 {
5     char mystring[] = "string";
6     std::cout << mystring << " has " << sizeof(mystring) << " characters.\n";
7     for (int index = 0; index < sizeof(mystring); ++index)
8         std::cout << static_cast<int>(mystring[index]) << " ";
9
10    return 0;
11 }
```

Результат выполнения программы:

```
string has 7 characters.
115 116 114 105 110 103 0
```

Нуль в конце является ASCII-кодом нуль-терминатора, который был добавлен в конец строки.

При таком объявлении строк рекомендуется использовать квадратные скобки `[]`, чтобы позволить компилятору определить длину массива самому. Таким образом, если вы измените строку позже, вам не придется вручную изменять значение длины массива.

Важно отметить, что строки C-style следуют всем тем же правилам, что и массивы. Это означает, что вы можете инициализировать строку при создании, но после этого не сможете присваивать ей значения с помощью оператора присваивания:

```
1 char mystring[] = "string"; // ок
2 mystring = "cat"; // не ок!
```

Это то же самое, как если бы мы сделали следующее:

```
1 int array[] = { 4, 6, 8, 2 }; // ок
2 array = 7; // что это значит?
```

Поскольку строки C-style являются массивами, то вы можете использовать оператор `[]` для изменения отдельных символов в строке:

```
1 #include <iostream>
2
```

```
3 int main()
4 {
5     char mystring[] = "string";
6     mystring[1] = 'p';
7     std::cout << mystring;
8
9     return 0;
10 }
```

Результат выполнения программы:

spring

При выводе строки C-style объект `std::cout` выводит символы до тех пор, пока не встретит нуль-терминатор. Если бы вы случайно перезаписали нуль-терминатор в конце строки (например, присвоив что-либо для `mystring[6]`), то от `std::cout` вы бы получили не только все символы строки, но и всё, что находится в соседних ячейках памяти до тех пор, пока не попался бы `0`!

Обратите внимание, это нормально, если длина массива больше строки, которую он хранит:

```
1 #include <iostream>
2
3 int main()
4 {
5     char name[15] = "Max"; // используется только 4 символа (3 буквы + нуль-терминатор)
6     std::cout << "My name is: " << name << '\n';
7
8     return 0;
9 }
```

В этом случае строка `Max` будет выведена, а `std::cout` остановится на нуль-терминаторе. Остальные символы в массиве будут проигнорированы.

Строки C-style и `std::cin`

Есть много случаев, когда мы не знаем заранее, насколько длинной будет наша строка. Например, рассмотрим проблему написания программы, где мы просим пользователя ввести свое имя. Насколько длинным оно будет? Это неизвестно до тех пор, пока пользователь его не введет!

В таком случае мы можем объявить массив размером больше, чем нам нужно:

```
1 #include <iostream>
2
3 int main()
4 {
5     char name[255]; // объявляем достаточно большой массив (для хранения 255 символов)
6     std::cout << "Enter your name: ";
```

```
7 |     std::cin >> name;
8 |     std::cout << "You entered: " << name << '\n';
9 |
10 |     return 0;
11 | }
```

В программе, приведенной выше, мы объявили массив из 255 символов, предполагая, что пользователь не введет имя длиннее 255 символов. Хотя это и распространенная практика, но она не очень эффективна, так как пользователю ничего не мешает ввести имя, содержащее более 255 символов (случайно или намеренно).

Намного лучше сделать следующим образом:

```
1 | #include <iostream>
2 |
3 | int main()
4 | {
5 |     char name[255]; // объявляем достаточно большой массив (для хранения 255 символов)
6 |     std::cout << "Enter your name: ";
7 |     std::cin.getline(name, 255);
8 |     std::cout << "You entered: " << name << '\n';
9 |
10 |     return 0;
11 | }
```

Вызов `cin.getline()` будет принимать до 254 символов в массив `name` (оставляя место для нуль-терминатора!). Любые лишние символы будут проигнорированы. Таким образом, мы можем гарантировать, что массив не будет переполнен!

Управление строками C-style

Язык C++ предоставляет множество функций для управления строками C-style, которые подключаются с помощью [заголовочного файла](#) `cstring`. Вот несколько самых полезных функций:

Функция `strcpy_s()` позволяет копировать содержимое одной строки в другую. Чаще всего это используется для присваивания значений строке:

```
1 | #include <iostream>
2 | #include <cstring>
3 |
4 | int main()
5 | {
6 |     char text[] = "Print this!";
7 |     char dest[50];
8 |     strcpy_s(dest, text);
9 |     std::cout << dest; // выводим "Print this!"
10 |
11 |     return 0;
12 | }
```

Тем не менее, использование функции `strcpy_s()` может легко вызвать [переполнение](#) массива, если не быть осторожным! В следующей программе, длина массива `dest` меньше длины копируемой строки, поэтому в результате мы получим переполнение массива:

```
1 #include <iostream>
2 #include <cstring>
3
4 int main()
5 {
6     char text[] = "Print this!";
7     char dest[5]; // обратите внимание, длина массива dest всего 5 символов!
8     strcpy_s(dest, text); // переполнение!
9     std::cout << dest;
10
11     return 0;
12 }
```

Еще одной полезной функцией управления строками является **функция `strlen()`**, которая возвращает длину строки C-style (без учета нуль-терминатора):

```
1 #include <iostream>
2 #include <cstring>
3
4 int main()
5 {
6     char name[15] = "Max"; // используется только 4 символа (3 буквы + нуль-терминатор)
7     std::cout << "My name is " << name << '\n';
8     std::cout << name << " has " << strlen(name) << " letters.\n";
9     std::cout << name << " has " << sizeof(name) << " characters in the array.\n";
10
11     return 0;
12 }
```

Результат выполнения программы:

```
My name is Max
Max has 3 letters.
Max has 15 characters in the array.
```

Обратите внимание на разницу между функцией `strlen()` и оператором `sizeof`. `strlen()` выводит количество символов до нуль-терминатора, тогда как [оператор `sizeof`](#) возвращает размер целого массива, независимо от того, что в нем находится.

Вот еще **полезные функции для управления строками C-style**:

- ➔ **функция `strcat()`** — добавляет одну строку к другой (опасно);
- ➔ **функция `strncat()`** — добавляет одну строку к другой (с проверкой размера места назначения);

→ функция `strcmp()` — сравнивает две строки (возвращает 0, если они равны);

→ функция `strncmp()` — сравнивает две строки до определенного количества символов (возвращает 0, если до указанного символа не было различий).

Например:

```
1 #include <iostream>
2 #include <cstring>
3
4 int main()
5 {
6     // Просим пользователя ввести строку
7     char buffer[255];
8     std::cout << "Enter a string: ";
9     std::cin.getline(buffer, 255);
10
11     int spacesFound = 0;
12     // Перебираем каждый символ, который ввел пользователь
13     for (int index = 0; index < strlen(buffer); ++index)
14     {
15         // Подсчитываем количество пробелов
16         if (buffer[index] == ' ')
17             spacesFound++;
18     }
19
20     std::cout << "You typed " << spacesFound << " spaces!\n";
21
22     return 0;
23 }
```

Стоит ли использовать строки C-style?

Знать о строках C-style стоит, так как они используются не так уж и редко, но использовать их без веской на то причины не рекомендуется. Вместо строк C-style используйте `std::string` (подключая заголовочный файл `string`), так как он проще, безопаснее и гибче.

Правило: Используйте `std::string` вместо строк C-style.

Оценить статью:

★★★★★ (242 оценок, среднее: 4,95 из 5)



← [Урок №78. Многомерные массивы](#)

[Введение в класс std::string_view в C++](#)

Комментариев: 7



1. Павел:

[20 мая 2020 в 12:59](#)

Тут нашел задачник с ответами, там была задача:

Составить программу, которая на входе должна получать последовательность цифр, после чего программа показывает цифру, порядковый номер которой ввел пользователь.

Я ее решил и потом решил посмотреть ответ и немного прифигел от решения, так как я думал strlen только считает длину строки складывая, а тут используется для сравнения.

```
1 #include <iostream>
2 #include <cstring> // для функции strlen
3 using namespace std;
4
5 int main()
6 {
7     char string[100]; //символьный массив, для хранения введенной последовательности
8     cout << "Введите последовательность цифр: ";
9     cin >> string;
10
11     int k; // переменная целого типа, для хранения порядкового номера цифры
12     cout << "\nВведите порядковый номер цифры: ";
13     cin >> k;
14     // проверка порядкового номера
15     if ((k - 1) < 0 || k > strlen(string)) // если введенный пользователем порядк
16         cout << "\nНекорректный ввод порядкового номера" << endl << endl; // напеч
17     else
18         cout << "nk-я цифра последовательности: " << string[k - 1] << endl; // вывод к
19     return 0;
20 }
```

[Ответить](#)



2. Alex_1988:

[29 января 2020 в 13:06](#)

Пройдя 79 часть решил объявить строку C-style через структуру:

```
1 struct People
2 {
3     char name[255]{};
4     int age{};
```

```
5     int kurs{};
6 };
7
8 enum Student
9 {
10     JOHN,
11     TONY,
12     ROBIN,
13     ALEX,
14     SIZE
15 };
16
17
18 int main(int argc, char* argv[])
19 {
20     setlocale(LC_ALL, "ru");
21
22     People mass[SIZE]{};
23     mass[JOHN].name = "John";
24
25     system("pause");
26     return 0;
27 }
```

Но не получается присвоить данные сразу же:

```
1 mass[JOHN].name = "John";
```

а только если добавлять посимвольно:

```
1 mass[JOHN].name[0] = 'J';
```

Как правильно присвоить слово целиком через структуру?

[Ответить](#)



1. *Сергей:*

[27 февраля 2020 в 19:41](#)

Присвоить слово целиком через структуру можно:

```
1 mass[JOHN].name[0] = "John";
2 mass[TONY].name[0] = "tony";
```

Дело в том, что `mass[JOHN].name` — это адрес памяти, а адресу нельзя присвоить строковую константу.

Это легко увидеть в следующих выражениях

```
1 std::cout << mass[JOHN].name << "\n";           // #include <iostream>
```



```
2 | std::cout << &mass[JOHN].name [0] << "\n"; // где & - урок №80 (Оператор
```

[Ответить](#)

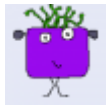

3. *Игорь:*
[2 ноября 2019 в 13:07](#)

Хмм. То есть, я правильно понял — весь текст который я видел выше это набор массивов? Каждое слово это массив? Или здесь какая-то другая технология?

[Ответить](#)


4. *Ad1lMkn:*
[27 марта 2019 в 11:56](#)

Лол, интересно))))00)))))))))

[Ответить](#)


5. *Владимир:*
[3 декабря 2018 в 21:25](#)

При переборе символьной строки безопаснее перебирать её до тех пор, пока не встретится завершающий ноль (если вы его конечно же не перезаписали) вместо использования функции `strlen()`, например:

```
1 char buffer[] {"Put any text here."};
2
3 for (int i{}; buffer[i] != '\0'; ++i)
4 {
5     cout << buffer[i];
6 }
7 cout << '\n';
```

Результат:
 Put any text here.

[Ответить](#)


1. *Юра:*
[18 января 2019 в 21:02](#)

Ну и в чем безопасность? `strlen` возвращает кол-во символов до ТЕРМИНАЛЬНОГО НУЛЯ. Зачем переписывать велосипед?

[Ответить](#)

Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены *

Имя *

Email *

Комментарий






☐ Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию

☐ Получать уведомления о новых комментариях по электронной почте. Вы можете [подписаться](#) без комментирования.

[TELEGRAM](#)  [КАНАЛ](#)

[ПАБЛИК](#) 

ТОП СТАТЬИ

-  [Словарь программиста. Сленг, который должен знать каждый кодер](#)
-  [Урок №1. Введение в программирование](#)
-  [70+ бесплатных ресурсов для изучения программирования](#)
-  [Урок №1: Введение в создание игры «SameGame» на C++/MFC](#)
-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

- [Ravesli](#)
- - [О проекте/Контакты](#) -
- - [Пользовательское Соглашение](#) -
- - [Все статьи](#) -
- Copyright © 2015 - 2020