

Ravesli [Ravesli](#)


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


Урок №35. Символьный тип данных char

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 |

 91849

[↑](#)  8

Хоть тип char и относится к целочисленным типам данных (и, таким образом, следует всем их правилам), работа с char несколько отличается от работы с обычными целочисленными типами.

Оглавление:

1. [Тип данных char](#)
2. [Вывод символов](#)
3. [Оператор static_cast](#)
4. [Ввод символов](#)
5. [Размер, диапазон и знак типа char](#)
6. [Управляющие символы](#)
7. [Что использовать: '\n' или std::endl?](#)
8. [Другие символьные типы: wchar_t, char16_t и char32_t](#)
9. [В чём разница между одинарными и двойными кавычками при использовании с символами?](#)

Тип данных char

Переменная типа char занимает 1 байт. Однако вместо конвертации значения типа char в целое число, оно *интерпретируется* как ASCII-символ.

ASCII (сокр. от «*American Standard Code for Information Interchange*») — это американский стандартный код для обмена информацией, который определяет способ представления символов английского языка (+ несколько других) в виде чисел от 0 до 127. Например: код буквы 'a' — 97, код буквы 'b' — 98. Символы всегда помещаются в одинарные кавычки.

Таблица ASCII-символов:

Код Символ

Код Символ Код Символ Код Символ

0	NUL (null)	32	(space)	64	@	96	`
1	SOH (start of header)	33	!	65	A	97	a
2	STX (start of text)	34	”	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	\$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(72	H	104	h
9	HT (horizontal tab)	41)	73	I	105	i
10	LF (line feed/new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (form feed / new page)	44	,	76	L	108	l
13	CR (carriage return)	45	—	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (data control 1)	49	1	81	Q	113	q
18	DC2 (data control 2)	50	2	82	R	114	r
19	DC3 (data control 3)	51	3	83	S	115	s
20	DC4 (data control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of transmission block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[123	{
28	FS (file separator)	60	<	92	\	124	
29	GS (group separator)	61	=	93]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL (delete)

Символы от 0 до 31 в основном используются для форматирования вывода. Большинство из них уже устарели.

Символы от 32 до 127 используются для вывода. Это буквы, цифры, знаки препинания, которые большинство компьютеров использует для отображения текста (на английском языке).

Следующие два **стейтмента** выполняют одно и то же (присваивают переменным типа char целое число 97):

```
1 char ch1(97); // инициализация переменной типа char целым числом 97
2 char ch2('a'); // инициализация переменной типа char символом 'a' (97)
```

Будьте внимательны при использовании фактических чисел с числами, которые используются для представления символов (из ASCII-таблицы). Следующие два стейтмента выполняют не одно и то же:

```
1 char ch(5); // инициализация переменной типа char целым числом 5
2 char ch('5'); // инициализация переменной типа char символом '5' (53)
```

Вывод символов

При выводе переменных типа char, [объект cout](#) выводит символы вместо цифр:

```
1 #include <iostream>
2
3 int main()
4 {
5     char ch(97); // несмотря на то, что мы инициализируем переменную ch целым числом
6     std::cout << ch << std::endl; // cout выводит символ
7     return 0;
8 }
```

Результат:

a

Также вы можете выводить литералы типа char напрямую:

```
1 std::cout << 'b' << std::endl;
```

Результат:

b

Оператор static_cast

Если вы хотите вывести символы в виде цифр, а не в виде букв, то вам нужно сообщить cout вывести переменные типа char в виде целочисленных значений. Не очень хороший способ это сделать — присвоить переменной типа int переменную типа char и вывести её:

```
1 #include <iostream>
2
3 int main()
4 {
5     char ch(97);
6     int i(ch); // присваиваем значение переменной ch переменной типа int
7     std::cout << i << std::endl; // выводим значение переменной типа int
8     return 0;
9 }
```

Результат:

97

Лучшим способом является конвертация переменной из одного типа данных в другой с помощью оператора **static_cast**.

Синтаксис static_cast выглядит следующим образом:

static_cast<новый_тип_данных>(выражение)

Оператор static_cast принимает значение из (выражения) в качестве входных данных и конвертирует его в указанный вами <новый_тип_данных>.

Пример использования оператора static_cast для конвертации типа char в тип int:

```
1 #include <iostream>
2
3 int main()
4 {
5     char ch(97);
6     std::cout << ch << std::endl;
7     std::cout << static_cast<int>(ch) << std::endl;
8     std::cout << ch << std::endl;
9     return 0;
10 }
```

Результат выполнения программы:

```
a
97
a
```

Запомните, static_cast принимает (выражение) в качестве входных данных. Если мы используем переменную в (выражении), то эта переменная изменяет свой тип только в стейтменте с оператором static_cast. Процесс конвертации никак не влияет на исходную переменную с её значением! В вышеприведенном примере, переменная ch остается переменной типа char с прежним значением, чему является подтверждением последний стейтмент с cout.

Также в static_cast нет никакой проверки по диапазону, так что если вы попытаетесь использовать числа, которые будут слишком большие или слишком маленькие для конвертируемого типа, то произойдет [переполнение](#).

Более подробно о static_cast мы еще поговорим на соответствующем уроке.

Ввод символов

Следующая программа просит пользователя ввести символ. Затем она выводит этот символ и его ASCII-код:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Input a keyboard character: ";
6 }
```

```
7 char ch;
8 std::cin >> ch;
9 std::cout << ch << " has ASCII code " << static_cast<int>(ch) << std::endl;
10
11 return 0;
12 }
```

Результат выполнения программы:

```
Input a keyboard character: q
q has ASCII code 113
```

Обратите внимание, даже если `cin` позволит вам ввести несколько символов, переменная `ch` будет хранить только первый символ (именно он и помещается в переменную). Остальная часть пользовательского ввода останется во входном буфере, который использует `cin`, и будет доступна для использования последующим вызовам `cin`.

Рассмотрим это всё на практике:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Input a keyboard character: "; // предположим, что пользователь ввёл 'a'
6
7     char ch;
8     std::cin >> ch; // ch = 'a', "bcd" останется во входном буфере
9     std::cout << ch << " has ASCII code " << static_cast<int>(ch) << std::endl;
10
11     // Обратите внимание, следующий cin не просит пользователя что-либо ввести, дан
12     std::cin >> ch; // ch = 'b', "cd" останется в буфере
13     std::cout << ch << " has ASCII code " << static_cast<int>(ch) << std::endl;
14
15     return 0;
16 }
```

Результат выполнения программы:

```
Input a keyboard character: abcd
a has ASCII code 97
b has ASCII code 98
```

Размер, диапазон и знак типа char

В языке C++ для переменных типа `char` всегда выделяется 1 байт. По умолчанию, `char` может быть как `signed`, так и `unsigned` (хотя обычно `signed`). Если вы используете `char` для хранения ASCII-символов, то вам не нужно указывать знак переменной (поскольку `signed` и `unsigned` могут содержать значения от 0 до 127).

Но если вы используете тип `char` для хранения небольших целых чисел, то тогда следует уточнить знак. Переменная типа `char signed` может хранить числа от -128 до 127. Переменная типа `char unsigned` имеет диапазон от 0 до 255.

Управляющие символы

В языке C++ есть **управляющие символы** (или «*escape-последовательности*»). Они начинаются с бэкслеша (\), а затем следует определенная буква или цифра.

Наиболее распространенным управляющим символом в языке C++ является '\n', который обозначает символ новой строки:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "First line\nSecond line" << std::endl;
6     return 0;
7 }
```

Результат:

First line
Second line

Еще одним часто используемым управляющим символом является '\t', который заменяет клавишу TAB, вставляя большой отступ:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "First part\tSecond part";
6     return 0;
7 }
```

Результат:

First part Second part

Таблица всех управляющих символов в языке C++:

Название	Символ	Значение
Предупреждение (alert)	\a	Предупреждение (звуковой сигнал)
Backspace	\b	Перемещение курсора на одну позицию назад
formfeed	\f	Перемещение курсора к следующей логической странице
Символ новой строки (newline)	\n	Перемещение курсора на следующую строку
Возврат каретки (carriage return)	\r	Перемещение курсора в начало строки
Горизонтальный таб (horizontal tab)	\t	Вставка горизонтального TAB-a
Вертикальный таб (vertical tab)	\v	Вставка вертикального TAB-a
Одинарная кавычка	\'	Вставка одинарной кавычки (или апострофа)
Двойная кавычка	\"	Вставка двойной кавычки

Бэкслеш	<code>\\</code>	Вставка обратной косой черты (бэкслэша)
Вопросительный знак	<code>\?</code>	Вставка знака вопроса
Восьмеричное число	<code>\(number)</code>	Перевод числа из восьмеричной системы счисления в тип <code>char</code>
Шестнадцатеричное число	<code>\x(number)</code>	Перевод числа из шестнадцатеричной системы счисления в тип <code>char</code>

Рассмотрим пример в коде:

```

1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "\"This is quoted text\"\n";
6     std::cout << "This string contains a single backslash \" << std::endl;
7     std::cout << "6F in hex is char '\x6F'" << std::endl;
8     return 0;
9 }
```

Результат выполнения программы:

```

"This is quoted text"
This string contains a single backslash \
6F in hex is char 'o'
```

Что использовать: `'\n'` или `std::endl`?

Вы могли заметить, что в последнем примере мы использовали `'\n'` для перемещения курсора на следующую строку. Но мы могли бы использовать и `std::endl`. Какая между ними разница? Сейчас разберемся.

При использовании `std::cout`, данные для вывода могут помещаться в буфер, т.е. `std::cout` может не отправлять данные сразу же на вывод. Вместо этого он может оставить их *при себе* на некоторое время (в целях улучшения производительности).

И `'\n'`, и `std::endl` оба переводят курсор на следующую строку. Только `std::endl` еще гарантирует, что все данные из буфера будут выведены, перед тем, как продолжить.

Так когда же использовать `'\n'`, а когда `std::endl`?

- ➔ Используйте `std::endl`, когда нужно, чтобы ваши данные выводились сразу же (например, при записи в файл или при обновлении индикатора состояния какого-либо процесса). Обратите внимание, это может повлечь за собой незначительное снижение производительности, особенно если запись на устройство происходит медленно (например, запись файла на диск).
- ➔ Используйте `'\n'` во всех остальных случаях.

Другие символьные типы: `wchar_t`, `char16_t` и `char32_t`

Тип `wchar_t` следует избегать практически во всех случаях (кроме тех, когда происходит взаимодействие с Windows API).

Так же, как и стандарт ASCII использует целые числа для представления символов английского языка, так и другие кодировки используют целые числа для представления символов других языков. Наиболее известный стандарт (после ASCII) — **Unicode**, который имеет в запасе более 110 000 целых чисел для представления символов из разных языков.

Существуют следующие кодировки Unicode:

- ➔ **UTF-32** — требует 32 бита для представления символа.
- ➔ **UTF-16** — требует 16 бит для представления символа.
- ➔ **UTF-8** — требует 8 бит для представления символа.

Типы `char16_t` и `char32_t` были добавлены в C++11 для поддержки 16-битных и 32-битных символов Unicode (8-битные символы и так поддерживаются типом `char`).

В чём разница между одинарными и двойными кавычками при использовании с символами?

Как вы уже знаете, символы всегда помещаются в одинарные кавычки (например, `'a'`, `'+'`, `'5'`). Переменная типа `char` представляет только один символ (например, буква `a`, символ `+`, число `5`). Следующий стейтмент не является корректным:

```
1 char ch('56'); // переменная типа char может хранить только один символ
```

Текст, который находится в двойных кавычках, называется строкой (например, `"Hello, world!"`). Строка (тип `string`) — это набор последовательных символов.

Вы можете использовать литералы типа `string` в коде:

```
1 std::cout << "Hello, world!"; // "Hello, world!" - это литерал типа string
```

Более подробно о типе `string` мы поговорим на соответствующем уроке.

Оценить статью:

★★★★★ (347 оценок, среднее: 4,88 из 5)



← [Урок №34. Логический тип данных bool](#)



[Урок №36. Литералы и магические числа](#) →

Комментариев: 8

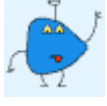


1. *Александр:*

[5 сентября 2020 в 14:22](#)

странно, но в c++ `sizeof('a') == sizeof(char)`, в то время как в си — `sizeof('a') == sizeof(int)`.
неожиданно ...

[Ответить](#)



2. *Alex:*

[26 февраля 2019 в 00:56](#)

Привет!

Вот с этим не понятно ничего:

```
1 | std::cout << "6F in hex is char \'\\x6F\\\'" << std::endl;
2 | 6F in hex is char 'o'
```

Каим образом x6F конвертируется в o?

[Ответить](#)



1. *Константин:*

[24 марта 2019 в 13:49](#)

У ЭВМ нет букв — только цифры. И то только две — 0 или 1. Поэтому вкладывая ему в оперативку значение 6F, сопровождают его ксивой `char`, так мол и так — эта комбинация нулей и единиц должна светиться символом 'o'...

[Ответить](#)



2. *Максим:*

[25 февраля 2020 в 17:33](#)

Выражение в кавычках с помощью управляющих символов превращается в другие символы. См. таблицу управляющих символов.

"6F in hex is char '\\x6F\\'" :

6F in hex is char — просто текст;

' — одинарная кавычка;

\\x6F — перевод числа из 16-ной системы в тип `char`.

Число 6F(шестнадцатеричное) = 111(десятичное).

Код 111 = символ "o" в кодировке ASCII;

' — еще одна одинарная кавычка;

Получается:

\\x6F' => 'o'

Итого:

6F in hex is char '\\x6F' => 6F in hex is char 'o'

[Ответить](#)



3. Константин:

[6 августа 2018 в 18:09](#)

Юрий, помогите "озадачить персонажей", например, конечный пользователь запускает файл.exe и читает: "Введите название материала". Он пишет: "песок мокрый". Программа отвечает "Ведро с мокрым песком имеет массу 20 кг". Или он пишет: "песок сухой". Тогда программа отвечает: "Ведро с сухим песком имеет массу 15 кг". Также он может написать: "глина мокрая". А программа в ответ: "Ведро с глиной мокрой 23 кг". Ну и "глина сухая". Ответ: "Ведро с глиной сухой имеет массу 16 кг". Как закодить такую задачку?

[Ответить](#)



1. Константин:

[10 сентября 2018 в 18:24](#)

Все, разобрался! ПК сам-то не думает ничего! Это я должен ему всунуть заранее написанные ответы, а он их будет по запросу показывать.

Например, `std::cout<< "спрашивал? Получи ответ!"<<std::endl;`. А вот применение отдельных 'б' 'у' 'к' 'в' пока не могу понять — куда их вписывать в код, какую смысловую нагрузку они несут?

[Ответить](#)



4. Степан:

[16 апреля 2018 в 16:11](#)

Привет, Юр.

Вроде неплохо работает и такой вид представления символьной переменной в целочисленном виде:

```
1 char a{98};  
2 ...  
3 ...  
4 std::cout<<(int)a<<std::endl;
```

И вообще, с помощью скобок с типом данных перед переменной можно и десятичные десятичные части от деления двух целочисленных переменных ловить.

[Ответить](#)



1. Юрий:

[17 апреля 2018 в 23:14](#)

Привет, вы здесь используете C-style cast (явное преобразование типов данных). C-style cast не рекомендуется использовать, предпочтение отдается `static_cast`. Об этом подробнее в [уроке 56](#).

Но для простых примеров — да, можно использовать, всё работает.

[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 «Same Game»](#)
- [⚙️ Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

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