

Строки

Михаил Смирнов
Разработчик C++



Михаил Смирнов

О спикере:

- В C++ разработке с 2010 года
- С 2002 года работаю в Муромском Институте Владимирского Государственного Университета
- Цифровая обработка сигналов в радиолокации и гидролокации
- Траекторная обработка для радиолокаторов ближней зоны
- Создание автоматизированного рабочего места для управления гидролокатором



Вспоминаем прошрое занятие

Вопрос: как выделить кусок памяти
заданной длины в байтах?



Вспоминаем прошрое занятие

Вопрос: как выделить кусок памяти заданной длины в байтах?

Ответ: использовать функцию malloc



Вспоминаем прошрое занятие

Вопрос: как освободить выделенную память?



Вспоминаем прошное занятие

Вопрос: как освободить выделенную память?

Ответ: с помощью функции free



Вспоминаем прошрое занятие

Вопрос: как создать динамический массив
типа `int` на 20 элементов?



Вспоминаем прошрое занятие

Вопрос: как создать динамический массив
типа `int` на 20 элементов?

Ответ: `int* arr = new int[20];`



Вспоминаем прошрое занятие

Вопрос: как очистить созданный
динамический массив arr?



Вспоминаем прошрое занятие

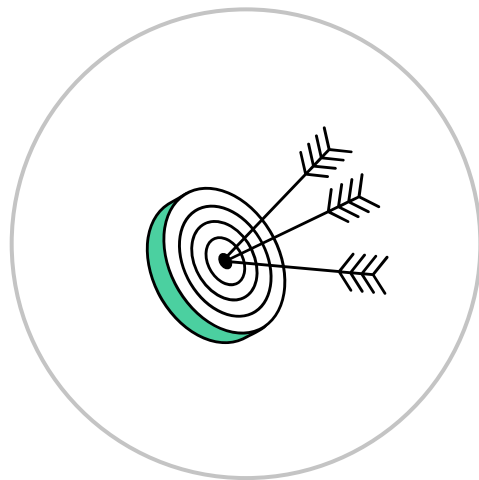
Вопрос: как очистить созданный
динамический массив arr?

Ответ: `delete[] arr;`



Цели занятия

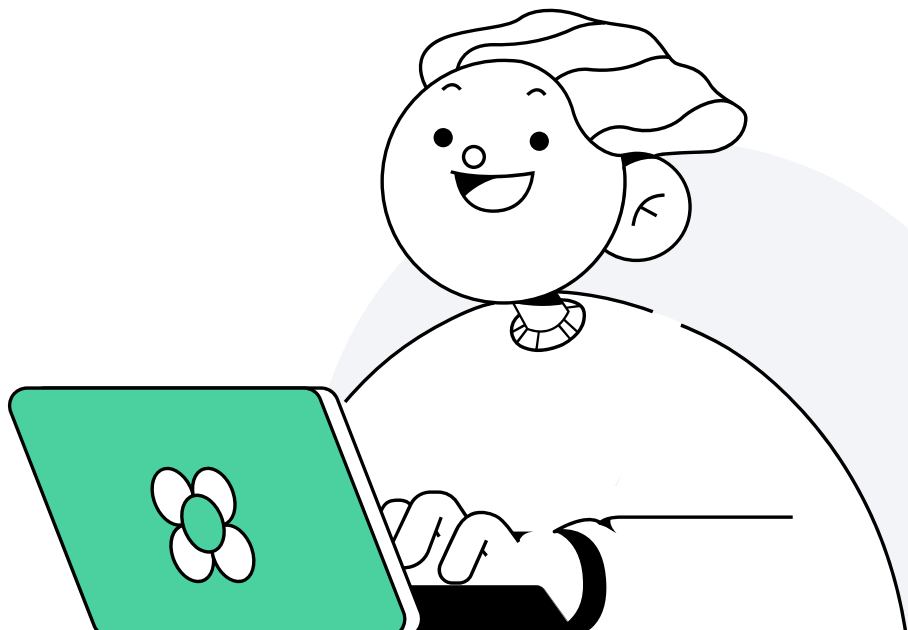
- Разберёмся, как устроены строки в C++
- Познакомимся с таблицей ASCII
- Узнаем, как работать со строками с помощью строковых функций
- Выясним, что такое тип `std::string` и как с ним работать



План занятия

- 1 Устройство строк
- 2 Функции для работы со строками
- 3 Тип `std::string`
- 4 Итоги
- 5 Домашнее задание

*Нажми на нужный раздел для перехода



Устройство строк



1

Что такое строка

Мы с вами уже пользовались строками — на консоль выводятся именно они.

Строка — это **массив символов**.
Символы в языке C++ представлены типом **char**



Что такое char

Char — это целочисленный тип данных, то есть целое число. Размер типа **char** — 1 байт. Он вмещает в себя всего 256 различных значений. Как же число превращается в символы?

Ответ: с помощью таблицы соответствия ASCII

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Таблица ASCII

Таблица ASCII определяет соответствие **кода** символа (его номера, числа) **самому символу**. Например, из таблицы видно, что код маленькой латинской буквы d — это 64, а код большой латинской буквы N — 4E.

Коды представлены шестнадцатеричными числами

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Char

Создадим переменную типа `char`, присвоим ей значение 4E (78 в десятичной системе) и выведем на экран:

```
int main(int argc, char** argv)
{
    char ch = 78;
    std::cout << ch << std::endl; // N
}
```

Проверим себя

```
int main(int argc, char** argv)
{
    char ch = 107;
    std::cout << ch << std::endl; // Что будет выведено? Напишите в чат
}
```

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Проверим себя

```
int main(int argc, char** argv)
{
    char ch = 107;                // 6B в 16-ричной системе
    std::cout << ch << std::endl; // k
}
```

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Посмотреть код символа

Можно проделать и обратную операцию — посмотреть код символа. Для этого в переменную **char** нужно «положить» символ, а при выводе привести переменную **char** к типу **int**:

```
int main(int argc, char** argv)
{
    char ch = 'N';
    std::cout << static_cast<int>(ch) << std::endl; // 78
}
```

Сохраняем строку в переменную

Теперь мы с вами можем сохранить строку в переменную:

```
int main(int argc, char** argv)
{
    char str[] = { "Hello" }; // str - это какой массив? Напишите в чат
}
```

Что такое строка

Теперь мы с вами можем сохранить строку в переменную:

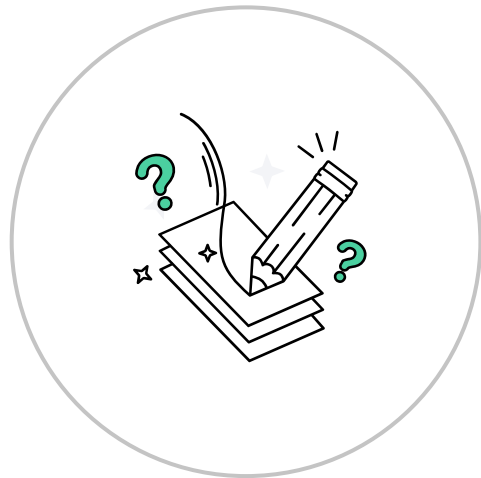
```
int main(int argc, char** argv)
{
    char str[] = { "Hello" }; // str - автоматический массив
}
```

Как устроена строка

Давайте исследуем нашу строку. Например, проверим, сколько в нашем массиве элементов.

Как узнать количество элементов в автоматическом массиве?

Напишите в чат



Как устроена строка

Правильно, нужно разделить размер всего массива на размер одного элемента этого массива. Для этого используется оператор **sizeof**:

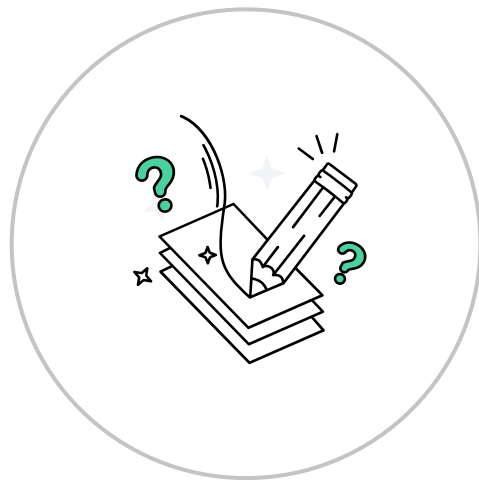
```
int main(int argc, char** argv)
{
    char str[] = { "Hello" };
    std::cout << str << std::endl;           // на консоль будет выведено Hello
    std::cout << sizeof(str) / sizeof(str[0]); // на консоль будет выведено 6
}
```


Как устроена строка

Несмотря на то, что в слове **Hello** всего 5 символов, количество элементов в массиве `str` — **6**. При этом при попытке вывода массива на консоль будет выведено всего 5 символов (наше слово Hello).

Как нам понять, что это за шестой элемент в массиве `str`?

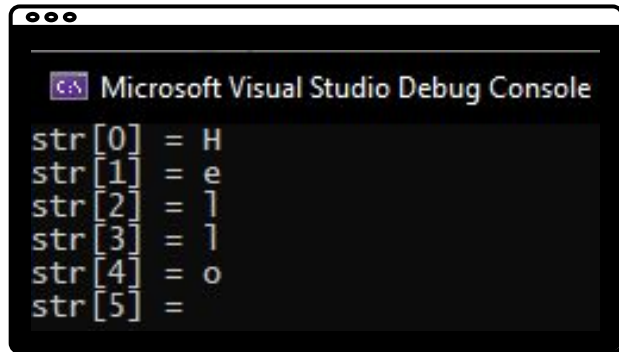
Напишите в чат



Как устроена строка

Ответ — использовать цикл for:

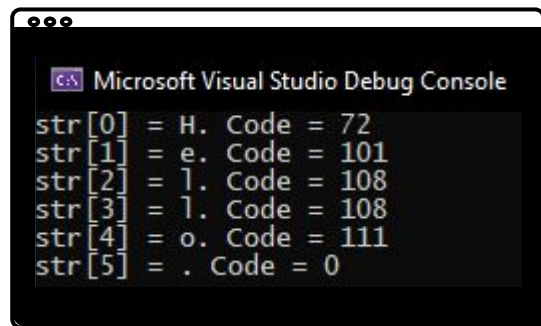
```
int main(int argc, char** argv)
{
    char str[] = { "Hello" };
    int size = sizeof(str) / sizeof(str[0]);
    for (int i = 0; i < size; i++)
    {
        std::cout << "str[" << i << "] = " << str[i] << std::endl;
    }
}
```



Как устроена строка

Как видим, шестой символ — непечатаемый. Давайте выведем на консоль коды символов:

```
int main(int argc, char** argv)
{
    char str[] = { "Hello" };
    int size = sizeof(str) / sizeof(str[0]);
    for (int i = 0; i < size; i++)
    {
        std::cout << "str[" << i << "] = " << str[i] << ". Code = " << static_cast<int>(str[i]) << std::endl;
    }
}
```

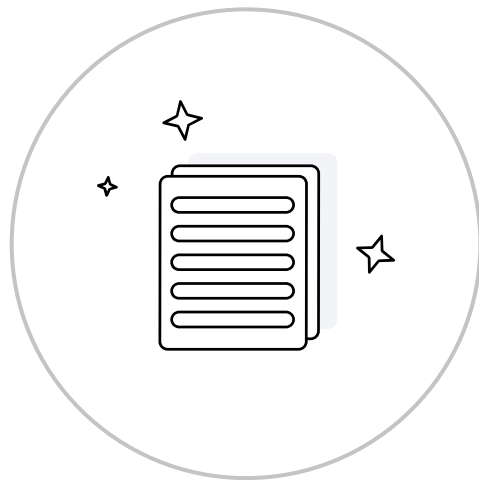


```
Microsoft Visual Studio Debug Console
str[0] = H. Code = 72
str[1] = e. Code = 101
str[2] = l. Code = 108
str[3] = l. Code = 108
str[4] = o. Code = 111
str[5] = . Code = 0
```

Нуль-терминатор

Последний (шестой) символ исследуемой строки имеет код 0.

Если мы заглянем в таблицу ASCII, то увидим, что символ с кодом 0 непечатаемый и обозначается как NUL



Ноль-терминатор

NUL — это ноль-терминатор. В C++ он используется для обозначения конца строки.

Благодаря ноль-терминатору мы можем передавать в функции, работающие со строками (например, в функцию, вычисляющую длину строки), только **указатель на строку**, при этом не передавая её размер. Функция, идя по массиву, сама определит, что строка закончилась, когда встретит ноль-терминатор



Функции для работы со строками



2

Список

Для работы со строками в C++ существует несколько встроенных функций, которые хранятся в библиотеке **<cstring>**. Здесь мы увидим список самых важных из них, а дальше рассмотрим каждую в отдельности:

- `strlen` — определяет длину строки
- `strcpy` — копирует одну строку в другую строку
- `strcat` — объединяет (склеивает) две строки
- `strcmp` — сравнивает две строки
- `strchr` — поиск первого вхождения символа в строке
- `atof` — преобразование строки к типу **double**
- `atoi` — преобразование строки к типу **int**
- `atol` — преобразование строки к типу **long**

Функция strlen

Подсчитывает количество символов в строке без учёта нуль-терминатора.

Сигнатура — **int** strlen(**char*** str). Ожидает на вход указатель на строку, возвращает количество символов в этой строке:

```
#include <iostream>
#include <cstring>
int main(int argc, char** argv)
{
    char str[] = { "Hello" };
    std::cout << strlen(str) << std::endl; // 5
}
```


Функция strcpy

Сигнатура — **char*** strcpy (**char*** destination, **char*** source). Принимает на вход два указателя на строки (destination и source), копирует содержимое строки source в строку destination, включая нуль-терминатор. Возвращает указатель на массив destination.

Для корректной отработки функции необходимо, чтобы массив destination имел достаточный размер, чтобы разместить в себе строку из массива source:

```
int main(int argc, char** argv)
{
    char source[] = { "Hello" };
    char dest[10];
    std::cout << strcpy(dest, source) << std::endl; // Hello
}
```

Функция strcat

Сигнатура — **char*** strcpy (**char*** destination, **char*** source). Принимает на вход два указателя на строки (destination и source), присоединяет содержимое строки source к концу строки destination. Ноль-терминатор строки destination перезаписывается первым символом строки source.

Для корректной отработки функции необходимо, чтобы массив destination имел достаточный размер, чтобы разместить в себе полученную строку:

```
int main(int argc, char** argv)
{
    char source[] = { " world" };
    char dest[30] = { "Hello" };
    std::cout << strcat(dest, source) << std::endl; // Hello world
}
```

Функция strcmp

Сигнатура — **int** strcmp (**char*** str1, **char*** str2). Принимает на вход два указателя на строки (str1 и str2), производит посимвольное сравнение строк. Возвращает:

- Отрицательное число, если str1 меньше, чем str2
- 0, если str1 равно str2
- Положительное число, если str1 больше, чем str2

```
int main(int argc, char** argv)
{
    char str1[] = { "Hello" };
    char str2[] = { "Hi" };
    char str3[] = { "Hello" };
    std::cout << strcmp(str1, str2) << std::endl; // -1
    std::cout << strcmp(str2, str3) << std::endl; // 1
    std::cout << strcmp(str1, str3) << std::endl; // 0
}
```

Функция strchr

Сигнатура — **char*** strchr (**char*** str, **char** character). Принимает на вход указатель на строку (str) и символ (character), выполняет поиск указанного символа в строке. Возвращает **указатель** на найденный в строке символ.

Если символ в строке отсутствует, возвращает **nullptr** — нулевой указатель:

```
int main(int argc, char** argv)
{
    char str[] = { "Hello" };
    char ch = 'e';
    std::cout << strchr(str, ch) << std::endl; // ello
}
```

Функция atoi

Функция выполняет преобразование строки в число типа **int**. В случае неудачи возвращает 0.

Сигнатура: **int** atoi (**char*** str)

```
int main(int argc, char** argv)
{
    char correct_str[] = { "123" };
    char incorrect_str[] = { "Hello" };
    std::cout << atoi(correct_str) << std::endl;      // 123
    std::cout << atoi(incorrect_str) << std::endl;    // 0
}
```

Функция atoi

Функция выполняет преобразование строки в число типа **long** . В случае неудачи возвращает 0.

Сигнатура: **long** atoi (**char*** str)

```
int main(int argc, char** argv)
{
    char correct_str[] = { "123" };
    char incorrect_str[] = { "Hello" };
    std::cout << atoi(correct_str) << std::endl;      // 123
    std::cout << atoi(incorrect_str) << std::endl;    // 0
}
```

Функция atof

Функция выполняет преобразование строки в число типа **double**. В случае неудачи возвращает 0.

Сигнатура: **double** atoi (**char*** str)

```
int main(int argc, char** argv)
{
    char correct_str[] = { "123.52" };
    char incorrect_str[] = { "Hello" };
    std::cout << atoi(correct_str) << std::endl;           // 123.52
    std::cout << atoi(incorrect_str) << std::endl;         // 0
}
```

Перерыв



Тип `std::string`



3

Более удобный тип

В C++ был добавлен специальный тип, чтобы было удобнее работать со строками — это тип **std::string**. Он находится в библиотеке **<string>**, в пространстве имён **std**:

```
#include <string>
int main(int argc, char** argv)
{
    std::string str = { "Hello" };
    std::cout << str << std::endl; // Hello
}
```

Инициализация

Переменную типа **std::string** можно инициализировать по-разному. Можно с помощью списка инициализации, через знак «=». Можно создать переменную типа `std::string` с помощью другой переменной типа `std::string` или с помощью обычной строки



Инициализация. Пример

```
#include <string>
int main(int argc, char** argv)
{
    char str0[] = { "Hello0" };
    std::string str1 = { "Hello1" };
    std::string str2 { "Hello2" };
    std::string str3("Hello3");
    std::string str4 = "Hello4";
    std::string str5 = str4;
    std::string str6 = str0;
    std::cout << str5 << std::endl; // Hello4
    std::cout << str6 << std::endl; // Hello0
}
```

Работа с памятью

Особенность типа **std::string** — то, что он сам работает с динамической памятью. То есть нам не нужно вручную выделять память для сохранения в переменную строки произвольной длины. Например, из пользовательского ввода. С обычной строкой мы так сделать не можем, потому что обычная строка — это просто массив символов



Работа с памятью

```
#include <string>
int main(int argc, char** argv)
{
    char* name_char_unalloc;
    char name_char[30];
    std::string name_string;
    std::cout << "Введите имя: ";
    std::cin >> name_char_unalloc; // Так нельзя, память по указателю name_char_unalloc не выделена
    std::cin >> name_char;         // В эту переменную не поместится имя длиннее 29 символов (1 под NUL)
    std::cin >> name_string;       // Сюда поместится имя произвольной длины
}
```

Длина строки

Тип **std::string** самостоятельно предоставляет несколько функций для работы со строками. Чтобы узнать длину строки, нужно вызвать функцию **length** у переменной типа **std::string**. Эта функция как бы принадлежит переменной, поэтому мы вызываем её через точку после имени переменной:

```
#include <string>
int main(int argc, char** argv)
{
    std::string name_string;
    std::cout << "Введите имя: ";
    std::cin >> name_string;           // Ввели Антон
    std::cout << name_string.length(); // 5
}
```

Копирование строки

Чтобы скопировать одну строку в другую, достаточно просто вызвать операцию присвоения с помощью оператора «=»:

```
#include <string>
int main(int argc, char** argv)
{
    std::string name_string;
    std::cout << "Введите имя: ";
    std::cin >> name_string;           // Ввели Антон
    std::string name_copy = name_string;
    name_copy[0] = 'Э';
    std::cout << name_string << std::endl; // Антон
    std::cout << name_copy << std::endl;   // Энтон
}
```


Конкатенация (соединение) строк

Чтобы соединить две строки в одну, достаточно вызвать оператор «+»:

```
#include <string>
int main(int argc, char** argv)
{
    std::string str1 = "Hello";
    std::string str2 = " world";
    std::string str3 = str1 + str2;
    str1 += str2;
    std::cout << str1 << std::endl;           // Hello world
    std::cout << str3 << std::endl;           // Hello world
    std::cout << str3 + " again!" << std::endl; // Hello world again!
}
```

Сравнение строк

Чтобы сравнить две строки, нужно вызвать функцию `compare`, которая тоже принадлежит типу `std::string`, и передать в качестве аргумента другую строку, с которой необходимо произвести сравнение.

Ещё можно использовать оператор «`==`» — он возвращает 1, если строки равны, и 0, если не равны



Сравнение строк

```
int main(int argc, char** argv)
{
    std::string str1 = "Hello";
    std::string str2 = "Hi";
    std::string str3 = "Hello";
    std::cout << str1.compare(str2) << std::endl; // -1
    std::cout << str2.compare(str3) << std::endl; // 1
    std::cout << str1.compare(str3) << std::endl; // 0
    std::cout << (str1 ==str2) << std::endl;      // 0
    std::cout << (str2 ==str3) << std::endl;      // 0
    std::cout << (str1 ==str3) << std::endl;      // 1
}
```

Поиск в строке

Чтобы найти символ в строке, нужно вызвать ещё одну функцию, принадлежащую типу **std::string** — функцию `find`. В качестве аргумента она принимает символ, который надо найти, или целую строку. Возвращает позицию найденного символа или подстроки в исходной строке. Если совпадений не найдено, возвращается специальное значение **std::string::npos**:

```
#include <string>
int main(int argc, char** argv)
{
    std::string str = "Hello";
    std::cout << str.find('e') << std::endl;           // 1
    std::cout << str.find("lo") << std::endl;          // 3
    std::cout << (str.find("Hi") == std::string::npos) << std::endl; // 1(то есть true)
}
```

Преобразование в число

Для преобразования значения типа `std::string` в число используются функции, аналогичные функции `atoi`, `atof`:

- `int std::stoi(std::string str)`
- `long std::stol(std::string str)`
- `float std::stof(std::string str)`
- `double std::stod(std::string str)`

В случае неудачи будет выброшено исключение — пока мы не знаем, как с ним работать:

```
int main(int argc, char** argv)
{
    std::string str = "123";
    std::cout << std::stoi(str) << std::endl; // 123
}
```

Итоги



Итоги занятия

Сегодня мы

- 1 Разобрались, как устроены строки в C++
- 2 Познакомились с таблицей ASCII
- 3 Узнали, как работать со строками с помощью строковых функций
- 4 Выяснили, что такое тип `std::string` и как с ним работать



Дополнительные материалы

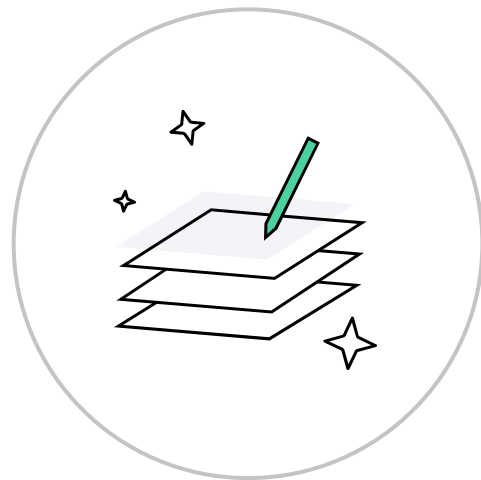
- [Работа со строками](#)



Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 Задачи можно сдавать по частям
- 3 Зачёт по домашней работе ставят после того, как приняты все задачи



Задавайте вопросы и пишите отзыв о лекции

Михаил Смирнов
Разработчик C++

