



Санкт-Петербургский
государственный
университет

БАЗОВЫЕ СРЕДСТВА ЯЗЫКА C++?



Санкт-Петербургский
государственный
университет

БАЗОВЫЕ СРЕДСТВА ЯЗЫКА C++

- ➡ 1 Состав языка
- ➡ 2 Типы данных
- ➡ 3 Переменные и выражения
- ➡ 4 Базовые конструкции структурного программирования
- ➡ 5 Указатели и массивы
- 6 Типы данных, определяемые пользователем
- 7 Функции
- ✓ 8 Директивы препроцессора
- ✓ 9 Области видимости и пространства имён



Санкт-Петербургский
государственный
университет

СОСТАВ ЯЗЫКА

- АЛФАВИТ ЯЗЫКА
- ИДЕНТИФИКАТОРЫ
- КЛЮЧЕВЫЕ СЛОВА
- ЗНАКИ ОПЕРАЦИЙ
- КОНСТАНТЫ
- КОММЕНТАРИИ



Санкт-Петербургский
государственный
университет

СОСТАВ ЯЗЫКА

- **Алфавит** — это неделимые знаки, с помощью которых пишутся все тексты на языке

`a..z, A..Z, 0..9`, специальные знаки: `" { } [] + - / % * . \ ' : ? < = > ! & # ~ ; ^`, пробел, символ табуляции, символ перехода на новую строку

- Из алфавита формируются **лексеммы** (элементарные конструкции):

идентификаторы

ключевые (зарезервированные слова) слова

знаки операций

константы

разделители

- **Константы**

целые (5, 01, 0xA), вещественные ([цифры].[цифры]{E|e}{+|-}[цифры]),

символьные — один или два символа ('A', '\n', '\x07'), строковые («И так сойдёт!»)



Санкт-Петербургский
государственный
университет

СОСТАВ ЯЗЫКА

- **Ключевые слова** — это зарезервированные идентификаторы, которые имеют специальное значение для компилятора

asm	do	if	return	typedef
auto	double	inline	short	typeid
bool	dynamic_cast	int	signed	typename
break	else	long	sizeof	union
case	enum	mutable	static	unsigned
catch	explicit	namespace	static_cast	using
char	export	new	struct	virtual
class	extern	operator	switch	void
const	false	private	template	volatile
const_cast	float	protected	this	wchar_t
continue	for	public	throw	while
default	friend	register	true	
delete	goto	reinterpret_cast	try	



Санкт-Петербургский
государственный
университет

Типы данных

- Тип данных определяет:
 - внутреннее представление данных в памяти
 - множество значений, которые могут принимать величины этого типа
 - операции и функции, которые можно применять
- Типы бывают базовые (примитивные) и составные (пользовательские):
 - `bool` — true / false. 1 байт
 - `char` — символьный. 1 байт. Используется для хранения символов таблицы кодировки ASCII (1963 г.)
 - `wchar_t` — расширенный символьный тип. Предназначен для хранения символов из кодировки Unicode.
 - `std::wcout << L"Привет, мир!";`
 - `int` — целочисленный. 4 байта. Но определяется не стандартом, а разрядностью компьютера
 - `float` — вещественный. 4 байта.
 - `double` — вещественный с двойной точностью. 8 байта.
 - `void` — множество значений пусто. Используется для: 1) определения функций, которые не возвращают значения, 2) указания пустого списка аргументов функции, 3) базовый тип указателей, 4) в операциях приведения типа



Санкт-Петербургский
государственный
университет

ТИПЫ ДАННЫХ

- 4 СПЕЦИФИКАТОРА ТИПОВ:
 - `short` — короткий. Пример: `short int` занимает 2 байта, а не 4
 - `long` — длинный. Определяется архитектурой: или 4 байта, или 8 байт.
Пример: `long int` – это тоже самое, что и `int` (если `int` в данной архитектуре 4 байта), но `long double` будет занимать 10 байт
 - `signed`
 - `unsigned`
- СТАНДАРТ ANSI УКАЗЫВАЕТ СООТНОШЕНИЯ МЕЖДУ РАЗМЕРАМИ ТИПОВ И ИХ МИНИМАЛЬНЫЕ ЗНАЧЕНИЯ, А НЕ ИХ ДИАПАЗОНЫ:
 - $\text{sizeof(char)} \leq \text{sizeof(short)} \leq \text{sizeof(long)} \leq \text{sizeof(long)}$
 - $\text{sizeof(float)} \leq \text{sizeof(double)} \leq \text{sizeof(long double)}$
- КРОМЕ СОСТАВЛЕНИЯ КОМБИНАЦИЙ ТИПОВ С ПОМОЩЬЮ СПЕЦИФИКАТОРОВ, В C++ И СТАНДАРТНОЙ БИБЛИОТЕКЕ ЕСТЬ ЕЩЕ ТИПЫ:
 - `char8_t`, `char16_t`, `char32_t`, `int8_t`, `uint8_t`, ... , `int64_t`, `uint64_t`, `size_t` и другие



Санкт-Петербургский
государственный
университет

ПЕРЕМЕННЫЕ И ВЫРАЖЕНИЯ

- **ПЕРЕМЕННАЯ** — ИМЕНОВАННАЯ ОБЛАСТЬ ПАМЯТИ, В КОТОРОЙ ХРАНЯТСЯ ДАННЫЕ ОПРЕДЕЛЁННОГО ТИПА
[класс памяти] [const] тип имя [инициализатор];
- Классы памяти: auto, extern, static, register — ОПРЕДЕЛЯЕТ ВРЕМЯ ЖИЗНИ И ОБЛАСТЬ ВИДИМОСТИ. Но в **C++11** auto приобрело другой смысл: ВОЗМОЖНОСТЬ НЕ УКАЗЫВАТЬ ТИП, ЧТОБЫ КОМПИЛЯТОР С ЭТИМ РАЗБИРАЛСЯ САМ.
- const ПОКАЗЫВАЕТ, ЧТО ЗНАЧЕНИЕ ПЕРЕМЕННОЙ НЕЛЬЗЯ МЕНЯТЬ

ПРИМЕЧАНИЕ ПО ИМЕНОВАНИЮ:

#define **MY_VALUE** 1 — МАКРОС

static const int **MY_VALUE** = 1; — СТАТИЧЕСКАЯ ГЛОБАЛЬНАЯ НЕИЗМЕНЯЕМАЯ ПЕРЕМЕННАЯ

const int **my_value** = 1; — ЛОКАЛЬНАЯ НЕИЗМЕНЯЕМАЯ ПЕРЕМЕННАЯ

int **my_value** = 1; — ЛОКАЛЬНАЯ ПЕРЕМЕННАЯ

- **ИНИЦИАЛИЗАЦИЯ** — ЭТО ПРИСВОЕНИЕ ПЕРЕМЕННОЙ НАЧАЛЬНОГО ЗНАЧЕНИЯ ПРИ ОБЪЯВЛЕНИИ. В ОТЛИЧИЕ ОТ ПРИСВАИВАНИЯ, КОТОРОЕ ОСУЩЕСТВЛЯЕТСЯ В ПРОЦЕССЕ ВЫПОЛНЕНИЯ ПРОГРАММЫ, ИНИЦИАЛИЗАЦИЯ ВЫПОЛНЯЕТСЯ ПРИ ВЫДЕЛЕНИИ ДЛЯ ПЕРЕМЕННОЙ УЧАСТКА ПАМЯТИ
- **ВЫРАЖЕНИЕ** — ЭТО ОПЕРАНДЫ, ЗНАКИ ОПЕРАЦИЙ И СКОБОК, РЕЗУЛЬТАТОМ КОТОРЫХ ЯВЛЯЮТСЯ ВЫЧИСЛЕНИЯ ЗНАЧЕНИЙ



Санкт-Петербургский
государственный
университет

- Операции по количеству операндов делятся на:
унарные, бинарные, тернарные

Операция	Краткое описание
Унарные операции	
++	увеличение на 1
--	уменьшение на 1
Sizeof	Размер
~	поразрядное отрицание
!	логическое отрицание
-	арифметическое отрицание (унарный минус)
+	унарный плюс
&	взятие адреса
*	разадресация
new	выделение памяти
delete	освобождение памяти
(type)	преобразование типа

ТЕРНАРНАЯ ОПЕРАЦИЯ

```
1 int x = a > b ? a : b;
```

ПЕРЕМЕННЫЕ И ВЫРАЖЕНИЯ

Бинарные и тернарная операции	
*	умножение
/	деление
%	остаток от деления
+	сложение
-	вычитание
<<	сдвиг влево
>>	сдвиг вправо
<	меньше
<=	меньше или равно
>	больше
>=	больше или равно
==	равно
!=	не равно
&	поразрядная конъюнкция (И)
^	поразрядное исключающее ИЛИ
	поразрядная дизъюнкция (ИЛИ)
&&	логическое И
	логическое ИЛИ
?:	условная операция (тернарная)
=	присваивание
*=	умножение с присваиванием
/=	деление с присваиванием
%=	остаток от деления с присваиванием
+=	сложение с присваиванием
-=	вычитание с присваиванием
<<=	сдвиг влево с присваиванием
>>=	сдвиг вправо с присваиванием
&=	поразрядное И с присваиванием
=	поразрядное ИЛИ с присваиванием
^=	поразрядное исключающее ИЛИ с присваиванием
,	последовательное вычисление



ПЕРЕМЕННЫЕ И ВЫРАЖЕНИЯ

- ОПЕРАЦИИ ИМЕЮТ РАЗНЫЙ ПРИОРИТЕТ

Приоритет	Оператор	Описание
1	::	Разрешение области видимости
2	a++ a-- тип() тип{} a() a[] . ->	Суффиксный/постфиксный инкремент и декремент Функциональный оператор приведения типов Вызов функции Индексация Доступ к элементу
3	++a --a +a -a ! ~ (тип) *a &a sizeof co_await new new[] delete delete[]	Префиксный инкремент и декремент Унарные плюс и минус Логическое НЕ и побитовое НЕ Приведение типов в стиле C Косвенное обращение (разыменование) Взятие адреса Размер в байтах Выражение await (C++20) Динамическое распределение памяти Динамическое освобождение памяти
4	.* ->*	Указатель на элемент
5	a*b a/b a%b	Умножение, деление и остаток от деления

5	a*b a/b a%b	Умножение, деление и остаток от деления
6	a+b a-b	Сложение и вычитание
7	<< >>	Побитовый сдвиг влево и сдвиг вправо
8	<=>	Оператор трёхстороннего сравнения (C++20)
9	< <= > >=	Для операторов отношения < и ≤ и > и ≥ соответственно
10	== !=	Операторы равенства = и ≠ соответственно
11	&	Побитовое И
12	^	Побитовый XOR (исключающее или)
13		Побитовое ИЛИ (включающее или)
14	&&	Логическое И
15		Логическое ИЛИ
16	a?b:c throw co_yield = += -= *= /= %= <<= >>= &= ^= =	Тернарный условный оператор Оператор throw Выражение yield (C++20) Прямое присваивание Присваивание с сложением и вычитанием Присваивание с умножением, делением и остатком Присваивание с побитовым сдвигом влево и вправо Присваивание с побитовым И, XOR и ИЛИ
17	,	Запятая



Санкт-Петербургский
государственный
университет

БАЗОВЫЕ КОНСТРУКЦИИ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ

- if / else
- switch
- while
- do while
- for

```
1 if (a > b) {
2     ...
3 } else {
4     ...
5 }
```

```
1 if (false) return;
```

```
1 while (выражение) {
2     ...
3 }
```

```
1 do {
2     ...
3 } while (выражение);
```

```
1 for (инициализация int i = 0; выражение i < 5; модификация i++) {
2     ...
3 }
```

```
1 switch (int или char) {
2     case константа_1:
3         нет инициализации переменных
4         break;
5     case константа_2:
6     case константа_3:
7         нет инициализации переменных
8         break;
9     case константа_4: {
10        есть инициализация переменной
11        break;
12    }
13    default: // необязательный
14        ...
15        break;
16 }
```

Range based for **C++11**

```
1 int a[5] = {1, 2, 3, 4, 5};
2 for(int x : a) {
3     ...
4 }
```



Санкт-Петербургский
государственный
университет

УКАЗАТЕЛИ И МАССИВЫ

- **УКАЗАТЕЛЬ** — ЭТО ПЕРЕМЕННАЯ, КОТОРАЯ ХРАНИТ АДРЕС ОБЛАСТИ ПАМЯТИ.

ТИП `*имя;`

```
1 int x = 1;
2 int *ptr = &x;
3 std::cout << ptr; // Будет выведен адрес x 2374
4 std::cout << *ptr; // Операция разыменования *. Будет выведена 1
```

РАЗМЕР УКАЗАТЕЛЯ В ПАМЯТИ РАВЕН РАЗРЯДНОСТИ МАШИНЫ. В СОВРЕМЕННЫХ МАШИНАХ — 8 БАЙТ.

ЯЧЕЙКА ПАМЯТИ РАВНА 1 БАЙТУ (НО МОЖЕТ БЫТЬ И ДРУГОЙ).

УКАЗАТЕЛЬ УКАЗЫВАЕТ НА ПЕРВЫЙ БАЙТ ЗНАЧЕНИЯ ПЕРЕМЕННОЙ В ПАМЯТИ. ТИП УКАЗАТЕЛЯ УКАЗЫВАЕТ КАКОЕ КОЛИЧЕСТВО БАЙТ НАДО ВЗЯТЬ ЧТОБЫ ПОЛУЧИТЬ ТРЕБУЕМОЕ ЗНАЧЕНИЕ.



Санкт-Петербургский
государственный
университет

УКАЗАТЕЛИ И МАССИВЫ

- ОСОБОЕ ВНИМАНИЕ:

На **один и тот же адрес** в памяти может ссылаться несколько указателей.

Приведение (явное/неявное) **типов** (сужение/расширение) при присвоение указателей.

Указатель может ссылаться на **невыделенную область памяти**

```
1 // Неправильно
2 int *ptr;
3 *ptr = 1; // Ошибка, т.к. ptr ссылается на невыделенную память
4
5 // Правильно
6 int *ptr = NULL;
7 ...
8 if (ptr != NULL) {
9     *ptr = 1;
10 }
```

В программе может динамически выделяться память, но не очищаться (**утечки памяти**).

Арифметика с указателями: **prt++**, ***prt++**, **(*ptr)++** и т.п.



Санкт-Петербургский
государственный
университет

УКАЗАТЕЛИ И МАССИВЫ

- **Массив** в C++ — это не структура данных, а представление последовательной области памяти.

```
int a[] = {1, 5, 2};
```

- `a` — это указатель на массив и он содержит в себе информацию о своём размере:

```
const int size = sizeof(a) / sizeof(a[0]);
```

ПРИМЕЧАНИЕ: `sizeof` — ЭТО ОПЕРАТОР, А НЕ ФУНКЦИЯ

- Операция `sizeof` для указателя на массив возвращает число байт, занимаемых массивом в памяти.

Адрес указателя на массив формируется в момент его объявления и остаётся неизменным на протяжении работы программы.

- **Но внимание:**

```
1 int a[] = {1, 5, 2};
2 int *ptr = a; // Аналогично int *ptr = &a[0];
3
4 std::cout << sizeof(a); // 12
5 std::cout << sizeof(ptr); // 4
6
7 a++; // Ошибка
8 ptr++; // В ptr будет указатель на следующий элемент массива
```




Санкт-Петербургский
государственный
университет





Санкт-Петербургский
государственный
университет

А ВОТ ТАК СОЙДЁТ!

- 1 Код надо писать так, чтобы из того, как он написан, было понятно почему он так написан!
- 2 Код должен быть написан в полной строгости с критериями задачи!
- 3 Не писать ничего лишнего!
- 4 Сила C++ — это работа с памятью!
- 5 Оптимизация достигается в:
 1. Написании правильной логики по выполнению сложных вычислений / логики
 2. В правильном количестве выполненных операций (сложность алгоритма)
 3. Требуемых «движений» памяти для выполнения программы