

Базовые средства языка С++?



Базовые средства языка С++







2 Типы данных



3 Переменные и выражения



4 Базовые конструкции структурного программирования



5 Указатели и массивы



- 7 Функции
- 1

8 Директивы препроцессора



9 Области видимости и пространства имён



- Алфавит языка
- Идентификаторы
- Ключевые слова
- Знаки операций
- Константы
- Комментарии

Состав языка



Состав языка

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

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

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

ИДЕНТИФИКАТОРЫ

КЛЮЧЕВЫЕ (ЗАРЕЗЕРВИРОВАННЫЕ СЛОВА) СЛОВА

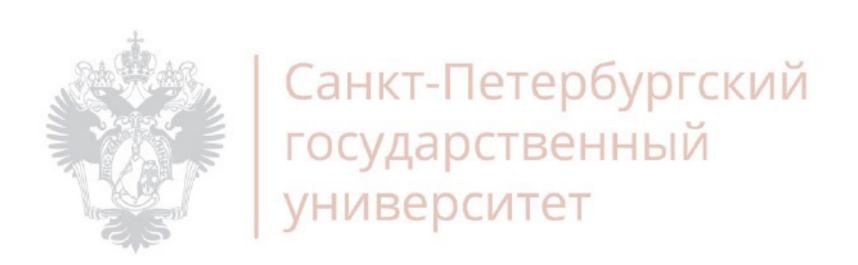
ЗНАКИ ОПЕРАЦИЙ

КОНСТАНТЫ

РАЗДЕЛИТЕЛИ

• Константы

```
целые (5, 01, 0хA), вещественные ( [цифры].[цифры]{E|e}[+|-][цифры] ), символьные — один или два символа ('A', '\n', '\x07'), строковые («И так сойдёт!»)
```



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	



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



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 аритектуре 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:

```
sizeof(char) ≤ sizeof(short) ≤ sizeof(long) ≤ sizeof(long)
sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)
```

• Кроме составления комбинаций типов с помощью спецификаторов, в С++ и стандартной библиотеке есть еще типы:

```
char8_t, char16_t, char32_t, int8_t, uint8_t, ... , int64_t, uint64_t, size_t и другие
```



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

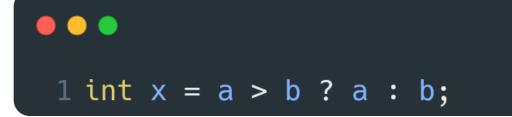


Санкт-Петербургский государственный университет

• Операции по количеству операндов деляться на: унарные, бинарные, тернарные

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

Тернарная операция



Переменные и выражения

*	и тернарная операции	
r.	умножение	
/	деление	
%	остаток от деления	
+	сложение	
÷	вычитание	
<<	сдвиг влево	
>>	сдвиг вправо	
<	меньше	
<=	меньше или равно	
>	больше	
>=	больше или равно	
==0	равно	
!=	не равно	
&	поразрядная конъюнкция (И)	
Λ	поразрядное исключающее ИЛИ	
	поразрядная дизьюнкция (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	



• Операции имеют разный приоритет

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

Переменные и выражения

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 ≥ соответственн	
10	==!= Операторы равенства = и ≠ соответственно	
11	& Побитовое И	
12	^	Побитовый XOR (исключающее или)
13	1	Побитовое ИЛИ (включающее или)
14	8,8	Логическое И
15	П	Логическое ИЛИ
	a?b:c	Тернарный условный оператор
	throw	Оператор throw
16	co_yield	Выражение yield (c++20)
	=	Прямое присваивание
	+= -=	Присваивание с сложением и вычитанием
	*= /= %=	Присваивание с умножением, делением и остатком
	<<= >>=	Присваивание с побитовым сдвигом влево и вправо
	δ= ^= =	Присваивание с побитовым И, XOR и ИЛИ
17	r	Запятая



Санкт-Петербургский государственный университет

Базовые конструкции структурного программирования

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

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

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

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

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

```
1 switch (int или char) {
     case константа_1:
       нет инициализации переменных
       break;
     case константа_2:
     case константа_3:
       нет инициализации переменных
       break;
     case константа_4: {
       есть инициализация переменной
       break;
     default: // необязательный
       . . .
       break;
16 }
```

Range based for **C++11**

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

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



Указатели и массивы

• Указатель — это переменная, которая хранит адрес области памяти.

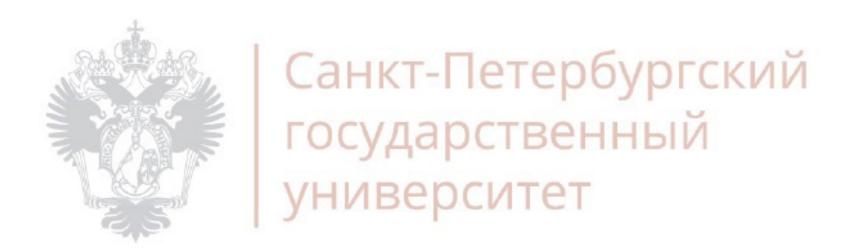
```
тип *имя;
```

```
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</pre>
```

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.



Указатели и массивы

• Особое внимание:

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

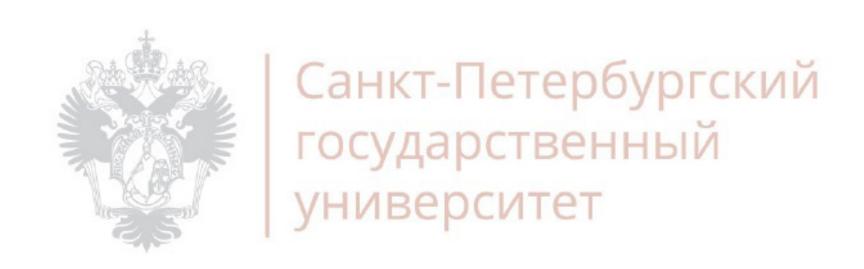
Приведение (явное/неявное) типов (сужение/расширение) при присвоение указателей.

Указатель может 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.



Указатели и массивы

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

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

• а — 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 про-

граммы.

• 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</pre>
```







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