

БГТУ, ФИТ, ПОИТ, 2 семестр,

Конструирование программного обеспечения

### Структура языка программирования

План лекции:

#### Структура языка программирования:

- ✓ **алфавит языка:** набор разрешенных символов, кодировка символов исходного кода программ; символы времени трансляции, символы времени выполнения;
- ✓ **идентификаторы:** правила образования идентификаторов; зарезервированные идентификаторы;
- ✓ **литералы;**
- ✓ **ключевые слова;**
- ✓ **фундаментальные (встроенные) типы данных:**
  - предопределенные типы данных;
  - указатели;
- ✓ **пользовательские типы данных**
  - типы, которые может создавать пользователь на основе фундаментальных типов (возможно описание их свойств и поведения);
  - массивы фундаментальных типов;
  - `enum`;
  - `struct`;
  - `union`;
  - `typedef`;
  - `class` (спецификатор типа).

## 1. Спецификация системы программирования:

набор требований к системе программирования, достаточный для ее разработки.

**Система программирования** – инструментальное ПО, предназначенное для разработки программного продукта на этапах программирования и отладки.



## 2. Алфавит языка программирования:

набор символов, разрешенных к использованию языком программирования. Основывается на одной из кодировок.

Совокупность символов, допускаемых в языке – **алфавит языка**.

### **Базовый набор символов исходного кода:**

- 1) строчные и прописные буквы латинского и национального алфавитов
- 2) цифры
- 3) знаки операций
- 4) символы подчеркивания \_ и пробела
- 5) ограничители и разделители
- 6) специальные символы

С помощью символов алфавита записываются *служебные слова*, которые составляют *словарь языка*.

Алфавит языка программирования служит для построения слов в языке программирования, которые называют *лексемами*.

Примеры лексем:

<b>Лексемы</b>	<i>идентификаторы;</i> <i>ключевые (зарезервированные) слова;</i> <i>знаки операций;</i> <i>константы;</i> <i>разделители</i> (скобки, знаки операций, точка, запятая, пробельные символы и т.д.).
----------------	--

Границы лексем определяются с помощью других лексем, таких, как *разделители* или *знаки операций*.

### 3. Компилятор:

символы времени трансляции, символы времени выполнения.

#### *Набор символов времени трансляции:*

текст программы на языке программирования хранится в исходных файлах и основан на определенной кодировке символов.

#### *Набор символов времени выполнения:*

символы, интерпретируемые в среде выполнения. Любые дополнительные символы зависят от локализации.

### 4. Компилятор CL:

исходный код C++ , основные кодировки: ASCII, Windows-1251.

**Стандарт C++:** исходной код основывается на множестве символов ASCII:

- *буквы латинского алфавита:* [a...z], [A...Z];
- *цифры* [0...9];
- *спецсимволы:* \_ { } [ ] ( ) # < > : ; % . ? \* + - / ^ & ~ ! = , " ' @ \$
- *пробельные символы:* пробел, символы табуляции, символы перехода на новую строку.

Дополнительные символы *времени выполнения* определяются **setlocale**.

По умолчанию локаль

```
SetLocale (LC_ALL, "C")
```

устанавливает стандартный контекст C.

Во время выполнения можно установить кодовую страницу языкового стандарта, используя вызов `setlocale(LC_CTYPE, "rus")`

или

воспользоваться следующими функциями (необходимо включить заголовочный файл `<windows.h>`):

```
#include <windows.h>    // windows.h содержит прототипы функций
SetConsoleOutputCP(1251); //установить кодовую таблицу, на поток вывода
SetConsoleCP(1251);      //установить кодовую таблицу, на поток ввода
```

Директива `#pragma` указывает целевой языковый стандарт во время компиляции (гарантируется представление строк с расширенными символами в правильном формате).

## 5. Идентификатор:

*имя компонента* программы (объекта или переменной, функции, метки, макроса, типа, ...), составленное программистом по определенным правилам.

**Примеры** правил составления идентификаторов в языках программирования:

### Ruby

Идентификаторы начинаются с буквы или специального модификатора:

- имена локальных переменных начинаются со строчной буквы или знака подчеркивания (`alpha`, `_ident`);
- имена глобальных переменных начинаются со знака доллара (`$beta`);
- имена переменных экземпляра (принадлежащих объекту) начинаются со знака «@» (`@foobar`);
- имена переменных класса (принадлежащих классу) предваряются двумя знаками «@» (`@@not_const`);
- имена констант начинаются с прописной буквы (`K6chip`);
- в именах идентификаторов знак подчеркивания `_` можно использовать наравне со строчными буквами (`$not_const`);
- имена специальных переменных, начинающиеся со знака «\$» (`$beta`).

**MS Transact-SQL** – имена переменных должны начинаться с символа `@`.

### Python

Используются символы Unicode.

Идентификаторы начинаются с латинской буквы в любом регистре или символа подчеркивания, могут содержать цифры, не могут совпадать с ключевыми словами (их список можно узнать по `import keyword; print(keyword.kwlist)`), нежелательно переопределять встроенные имена. Имена, начинающиеся с символа подчеркивания, имеют специальное значение.

В **Python 3** – в идентификаторе допустимы символы любого алфавита в Юникоде, например, кириллицы.

## 6. Идентификатор C++:

- идентификаторы должны начинаться с буквы или нижнего подчеркивания;
- идентификатор не может совпадать с ключевыми словами C++ или с именами библиотечных функций;
- идентификаторы могут состоять из любого количества символов, но компилятор *гарантирует*, что будет считать значащими:
  - 31 первых символов идентификаторов, не имеющих внешней связи;
  - не более 6 значащих символов идентификаторов с внешней связью;
- идентификаторы чувствительны к регистру.

Длина идентификатора по стандарту не ограничена.

Идентификатор создается при объявлении переменной, функции, типа и т. п.

## 7. Зарезервированные идентификаторы:

идентификаторы, которые предварительно определены в системе программирования.

### Python

имеет особое значение идентификатор «\_» – используется для хранения результата последнего вычисления.

## 8. Зарезервированные идентификаторы C++:

- [стандартные ключевые слова C++](#);
- все имена с *двумя подчеркиваниями* считаются зарезервированным;
- кроме того: **isxxxx**, **memxxxx**, **strxxxx**, **toxxxx**, **wcsxxxx**, **Ецифраxxxx**, **LC\_Xxxx**, **SIGXxxx**, **SIG\_Xxxxx**.

## 9. Литерал:

элемент программы, который представляет непосредственное значение.

В C++ существует следующие типы литералов:

- **целочисленный литерал,**
- **вещественный литерал,**
- **логический литерал,**
- **литерал-указатель,**
- **символьный литерал,**
- **строковый литерал.**

Литералы часто используются для инициализации именованных переменных и для передачи аргументов в функции.

Управляющие символные литералы (escape-последовательности):

\0	\x00	null	пустая литера
\a	\x07	bel	сигнал
\b	\x08	bs	возврат на шаг
\f	\x0C	ff	перевод страницы
\n	\x0A	lf	перевод строки
\r	\x0D	cr	возврат каретки
\t	\x09	ht	горизонтальная табуляция
\v	\x0B	vt	вертикальная табуляция
\\	\x5C	\	обратная косая черта
\'	\x27	'	одинарная кавычка
\"	\x22	"	двойная кавычка
\?	\x3F	?	вопросительный знак

**Примеры** литералов в C++:

```
const int answer = 42;           // целочисленный литерал
double d = sin(108.87);         // литералы с плавающей
                                // запятой, передаваемый в функцию sin
bool b = true;                  // логический литерал
MyClass* mc = nullptr;          // литерал-указатель

// Символьные литералы
auto c0 = 'A'; // тип char
auto c1 = u8'A'; // тип char
auto c2 = L'A'; // тип wchar_t
auto c3 = u'A'; // тип char16_t
auto c4 = U'A'; // тип char32_t
// Строковые литералы
auto s0 = "hello"; // const char*
auto s2 = L"hello"; // const wchar_t*
```



## 10. Ключевые слова:

последовательности символов алфавита языка, имеющие специальное назначение.

Ключевые слова зарезервированы компилятором для обозначения типов переменных, класса хранения, элементов операторов и т.д.

### Ruby:

BEGIN	END	alias	and	begin
break	case	class	def	defined?
do	else	elsif	end	ensure
false	<b>for</b>	<b>if</b>	in	module
next	nil	not	or	redo
rescue	retry	return	self	super
then	true	undef	unless	until
when	while	yield		

### Python (запрещается использовать как обычные идентификаторы)

false	class	finally	is	return
none	continue	<b>for</b>	lambda	try
true	def	from	nonlocal	while
and	del	global	not	with
as	elif	<b>if</b>	or	yield
assert	else	import	pass	print
break	except	in	raise	

### Go

break	case	chan	const	continue	default	
defer	else	fallthrough	<b>for</b>	func	go	
goto	<b>if</b>	import	interface	map	package	
range	return	select	struct	switch	type	var

## 11. Ключевые слова C++:

<https://docs.microsoft.com/ru-ru/cpp/cpp/keywords-cpp?view=vs-2019>

### Примеры ключевых слов C++

break	case	catch	char
char16_t	char32_t	class	const
false	finally	float	for
inline	return	<b>if</b>	struct
__cdecl	__int16 4	__int32 4	__int64



## 12. Типы данных:

- фундаментальные (или встроенные, или примитивные);
- определенные программистом.

### *Тип данных определяет:*

- внутреннее представление данных в памяти компьютера;
- диапазон значений, которые могут принимать величины этого типа;
- операции и функции, которые можно применять к величинам этого типа.

Фундаментальные типы (или встроенные типы) задаются стандартом языка и встроены в компилятор.

### Java 8

примитивные типы: `boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`.

Длины и диапазоны значений примитивных типов определяются **стандартом**, а не реализацией:

Тип	Длина (в байтах)	Диапазон или набор значений
<code>boolean</code>	1 в массивах, 4 в переменных	<code>true</code> , <code>false</code>
<code>byte</code>	1	-128..127
<code>char</code>	2	0.. $2^{16}-1$ , или 0..65535
<code>short</code>	2	$-2^{15}..2^{15}-1$ , или -32768..32767
<code>int</code>	4	$-2^{31}..2^{31}-1$ , или -2147483648..2147483647
<code>long</code>	8	$-2^{63}..2^{63}-1$ , или примерно $-9.2 \cdot 10^{18}..9.2 \cdot 10^{18}$
<code>float</code>	4	$-(2 \cdot 2^{-23}) \cdot 2^{127}..(2 \cdot 2^{-23}) \cdot 2^{127}$ , или примерно $-3.4 \cdot 10^{38}..3.4 \cdot 10^{38}$ , а также NaN
<code>double</code>	8	$-(2 \cdot 2^{-52}) \cdot 2^{1023}..(2 \cdot 2^{-52}) \cdot 2^{1023}$ , или примерно $-1.8 \cdot 10^{308}..1.8 \cdot 10^{308}$ , а также NaN

### Фундаментальные типы C++

определены следующие **ключевые** слова:

- **`int`** (целый);
- **`char`** (символьный);
- **`wchar_t`** (расширенный символьный);
- **`bool`** (логический);
- **`float`** (вещественный);
- **`double`** (вещественный с двойной точностью);
- тип **`void`**.

**Модификаторы основных типов**, уточняющие внутреннее представление и диапазон значений стандартных типов:

- **`short`** (короткий);
- **`long`** (длинный);
- **`signed`** (знаковый);
- **`unsigned`** (беззнаковый).

## Размеры основных встроенных типов в Microsoft C++

(! тип `long` имеет размер 4 байта даже в 64-разрядных операционных системах):

Тип	Размер
<code>bool</code> , <code>char</code> , <code>unsigned char</code> , <code>signed char</code> , <code>__int8</code>	1 байт
<code>__int16</code> , <code>short</code> , <code>unsigned short</code> , <code>wchar_t</code> , <code>__wchar_t</code>	2 байта
<code>__int32</code> , <code>int</code> , <code>unsigned int</code> , <code>long</code> , <code>unsigned long</code> , <code>float</code>	4 байта
<code>__int64</code> , <code>long double</code> , <code>long long</code> , <code>double</code>	8 байт
<code>__int128</code>	16 байт

Диапазоны типов данных в компиляторах Microsoft C++:

<https://docs.microsoft.com/ru-ru/cpp/cpp/data-type-ranges?view=msvc-160>

Большинство встроенных типов имеют размеры, определенные реализацией.

## 13. Фундаментальные типы C++. Примеры.

### 13.1 Тип `bool`

Размер типа `bool` зависит от конкретной реализации.

The screenshot shows a Visual Studio IDE with the following code in `main()`:

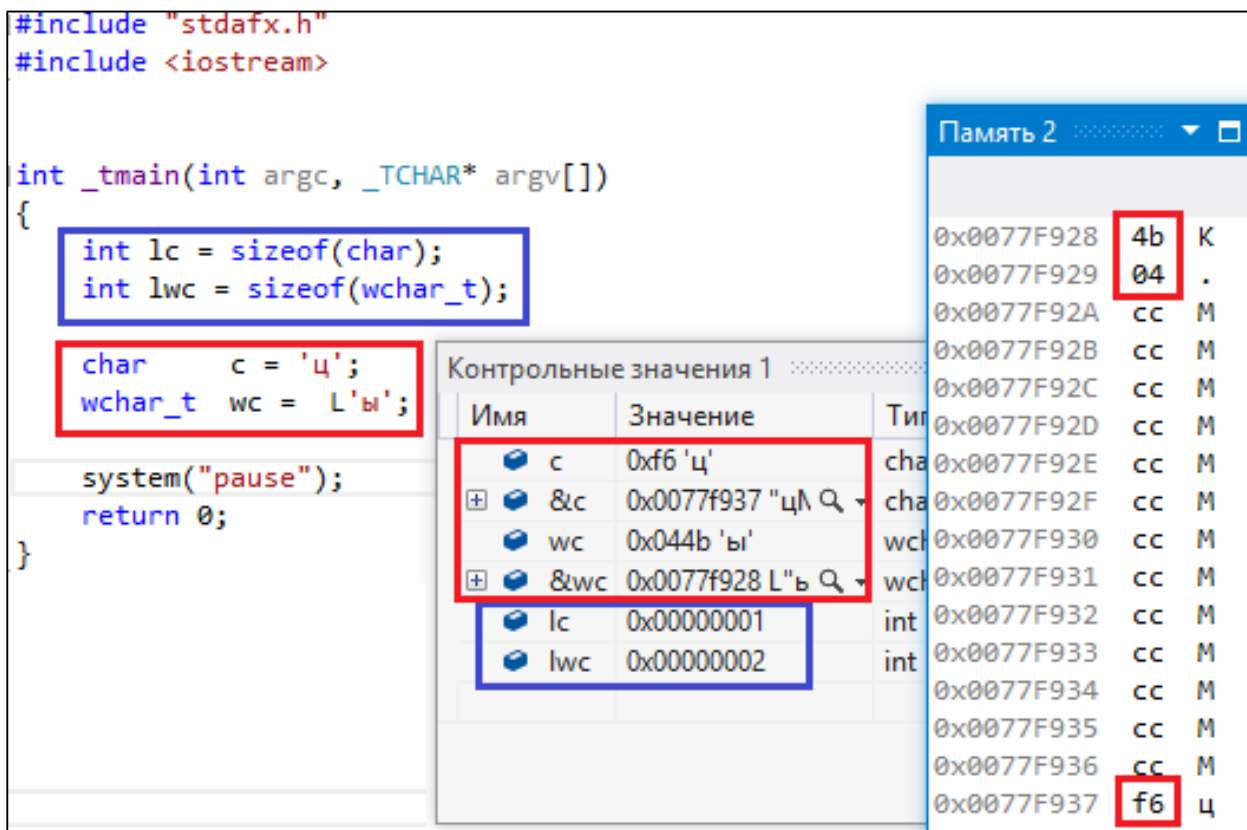
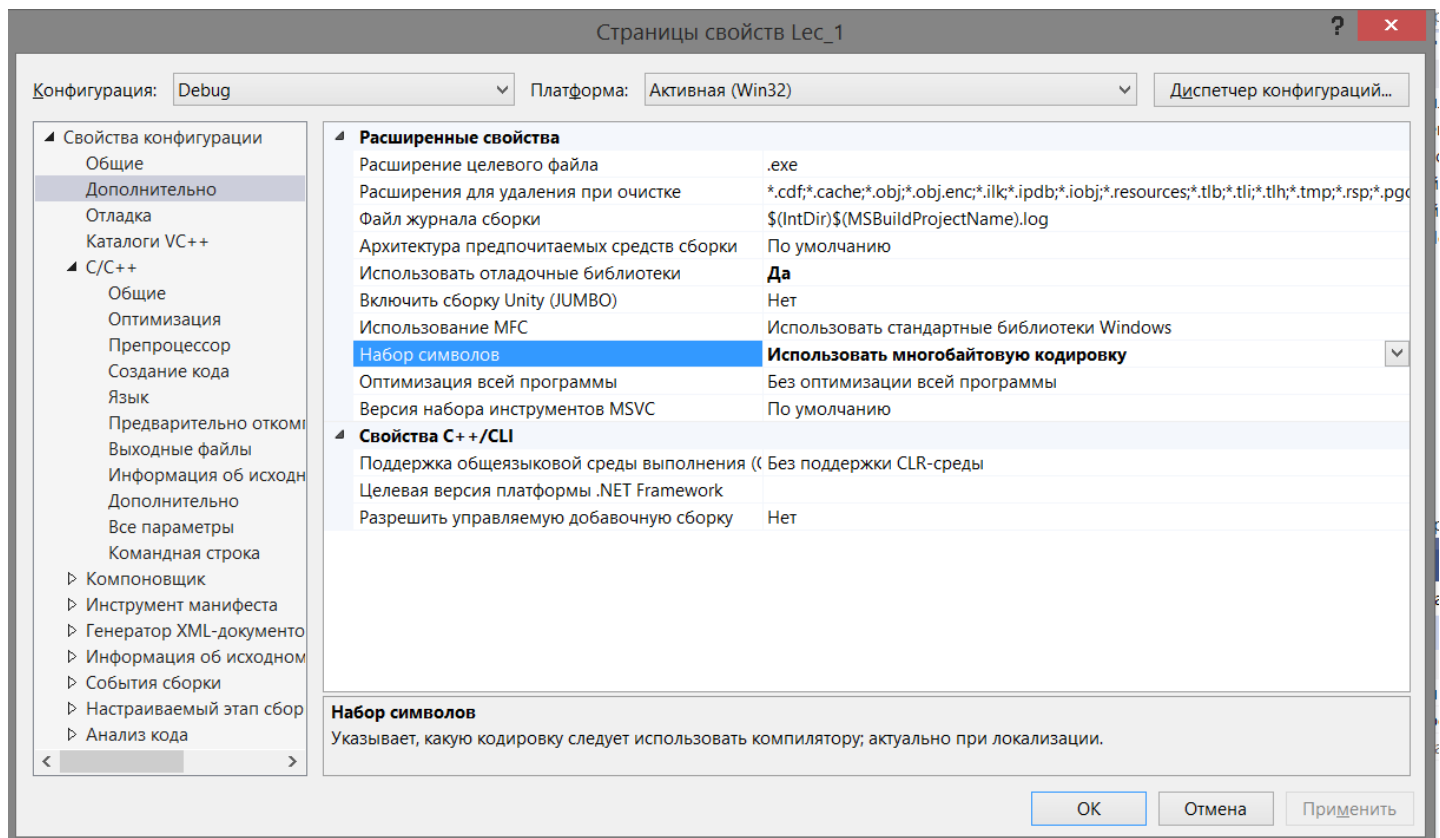
```
1 int main()
2 {
3     bool b1 = true;
4     bool b2 = false;
5     int lb = sizeof(bool);
6 }
7
```

The **Memory 1** window shows the memory dump starting at address `0x0055FA5B`. The first byte at `0x0055FA5B` contains the value `00` (false). The second byte at `0x0055FA65` contains the value `01` (true). This confirms that `bool` is 1 byte in size.

The **Variable Watch** window shows the following data:

Имя	Значение	Тип
<code>b1</code>	<code>true</code>	<code>bool</code>
<code>&amp;b1</code>	<code>0x0055fab7 (true)</code>	<code>bool *</code>
<code>b2</code>	<code>false</code>	<code>bool</code>

## 13.2 Встроенные типы char, wchar\_t



### 13.2.1 [unsigned] int, [unsigned] short, [unsigned] long

#### Внутреннее представление величины целого типа:

- целое число в двоичном коде.
- **спецификатор signed** – старший разряд (бит) числа интерпретируется как знаковый (0 – положительное число, 1 – отрицательное).
- **спецификатор unsigned**: старший разряд (бит) рассматривается как значащий, позволяет представлять только положительные числа.

По умолчанию все целочисленные типы знаковые, то есть спецификатор **signed** можно опускать. Диапазон значений зависит от реализации.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int lshort = sizeof(short);
    int lint = sizeof(int);
    int llong = sizeof(long);

    short ishort = SHRT_MAX;
    int iint = INT_MAX;
    long ilong = LONG_MAX;
}
```

Память 2

Имя	Значение	Тип
lshort	0x00000002	int
lint	0x00000004	int
llong	0x00000004	int
ishort	0x7fff	short
iint	0x7fffffff	int
ilong	0x7fffffff	long
&ishort	0x0093fe4c	short *
&iint	0x0093fe40	int *
&ilong	0x0093fe34	long *

Контрольные значения 1

Имя	Значение	Тип
ishort	0x7fff	short
iint	0x7fffffff	int
ilong	0x7fffffff	long
&ishort	0x0093fe4c	short *
&iint	0x0093fe40	int *
&ilong	0x0093fe34	long *

```
int _tmain(int argc, _TCHAR* argv[])
{
    short ishort = -11;
    int iint = -22222;
    long ilong = -33333;
}
```

Память 2

Имя	Значение	Тип
&ishort	0x0075fa24	short *
ishort	0xfff5	short
&iint	0x0075fa18	int *
iint	0xffffa932	int
&ilong	0x0075fa0c	long *
ilong	0xffff7dc3	long

Контрольные значения 1

Имя	Значение	Тип
&ishort	0x0075fa24	short *
ishort	0xfff5	short
&iint	0x0075fa18	int *
iint	0xffffa932	int
&ilong	0x0075fa0c	long *
ilong	0xffff7dc3	long

```
int _tmain(int argc, _TCHAR* argv[])
{
    int lushort = sizeof( unsigned short);
    int luint = sizeof(unsigned int);
    int lulong = sizeof(unsigned long);

    unsigned short iushort = USHRT_MAX;
    unsigned int iuint = UINT_MAX;
    unsigned long iulong = ULONG_MAX;
}
```

Контрольные значения 1

Имя	Значение	Тип
lushort	0x00000002	int
luint	0x00000004	int
lulong	0x00000004	int
&iushort	0x00ebfe54	unsigned short
&iuint	0x00ebfe48	unsigned int *
&iulong	0x00ebfe3c	unsigned long *

Память 2

Адрес	Данные (HEX)	Данные (ASCII)
0x00EBFE3C	ff ff ff ff	яяяя
0x00EBFE40	cc cc cc cc	ММММ
0x00EBFE44	cc cc cc cc	ММММ
0x00EBFE48	ff ff ff ff	яяяя
0x00EBFE4C	cc cc cc cc	ММММ
0x00EBFE50	cc cc cc cc	ММММ
0x00EBFE54	ff ff	яяММ
0x00EBFE58	cc cc cc cc	ММММ
0x00EBFE5C	cc cc cc cc	ММММ
0x00EBFE60	04 00 00 00	....
0x00EBFE64	cc cc cc cc	ММММ
0x00EBFE68	cc cc cc cc	ММММ
0x00EBFE6C	04 00 00 00	....
0x00EBFE70	cc cc cc cc	ММММ
0x00EBFE74	cc cc cc cc	ММММ
0x00EBFE78	02 00 00 00	....
0x00EBFE7C	cc cc cc cc	ММММ

```
int _tmain(int argc, _TCHAR* argv[])
{
    int iint1 = -22222;
    int iint2 = 22222;
    int iint3 = (unsigned int) 0xffffffff - iint2+1;
    int iint4 = ((unsigned int) 0xffffffff ^ (unsigned int) iint2)+1;
}
```

Контрольные значения 1

Имя	Значение	Тип
&iint1	0x00f2fe5c {-22222}	int *
&iint2	0x00f2fe50 {22222}	int *
&iint3	0x00f2fe44 {-22222}	int *
&iint4	0x00f2fe38 {-22222}	int *

Память 2

Адрес	Данные (HEX)	Данные (ASCII)
0x00F2FE38	32 a9 ff ff	2@яя
0x00F2FE3C	cc cc cc cc	ММММ
0x00F2FE40	cc cc cc cc	ММММ
0x00F2FE44	32 a9 ff ff	2@яя
0x00F2FE48	cc cc cc cc	ММММ
0x00F2FE4C	cc cc cc cc	ММММ
0x00F2FE50	ce 56 00 00	OV..
0x00F2FE54	cc cc cc cc	ММММ
0x00F2FE58	cc cc cc cc	ММММ
0x00F2FE5C	32 a9 ff ff	2@яя
0x00F2FE60	cc cc cc cc	ММММ

```
//int lb = sizeof(bool);
// b1 = false;
// b2 = true;
```

Два стандартных включаемых заголовочных файла, `<limits.h>` и `<float.h>`, определяют числовые ограничения или минимальное и максимальное значения, которые может хранить переменная данного типа.

**Ограничения** для некоторых целочисленных типов, заданные в стандартном файле заголовка `<limits.h>`, представлены в таблице:

Константа	Значение	Значение
CHAR_BIT	Количество битов в наименьшей переменной, которая не является битовым полем	8
SCHAR_MIN	Минимальное значение для переменной типа <b>signed char</b>	-128
SCHAR_MAX	Максимальное значение для переменной типа <b>signed char</b>	127
UCHAR_MAX	Максимальное значение для переменной типа <b>unsigned char</b>	255 (0xff)

### 13.3 Типы `float`, `double`

Стандарт языка C++ определяет три типа данных для хранения вещественных значений: ***float***, ***double*** и ***long double***.

**Стандарт IEEE 754** описывает формат представления чисел с плавающей точкой. Используется в программных (компиляторы разных языков программирования) и аппаратных (CPU и FPU) реализациях арифметических действий (математических операций).

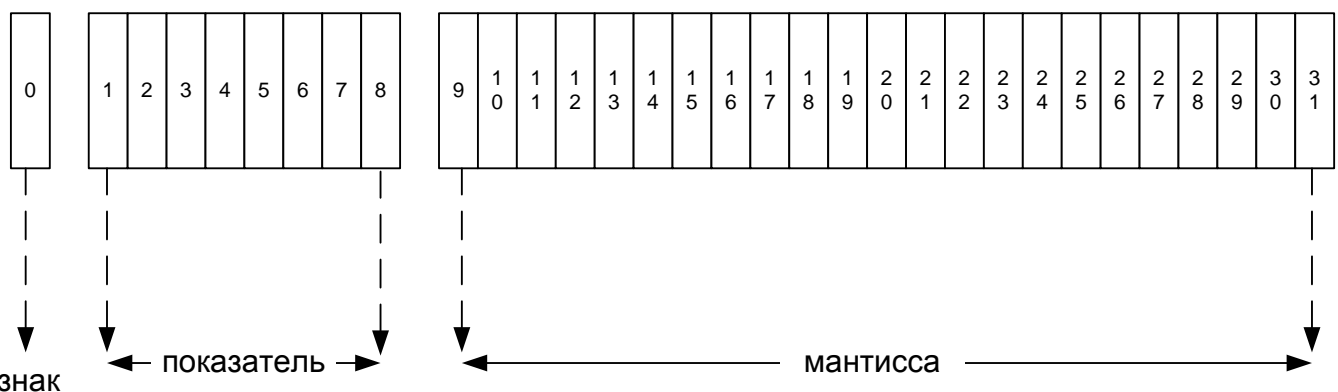
Стандарт описывает:

- формат чисел с плавающей точкой: мантисса, показатель (экспонента), знак числа;
- представление положительного и отрицательного нуля, положительной и отрицательной бесконечностей, а также нечисла (англ. Not-a-Number, NaN);
- методы, используемые для преобразования числа при выполнении математических операций;
- исключительные ситуации: деление на ноль, переполнение, потеря значимости, работа с денормализованными числами и другие;
- операции: арифметические и другие.

Стандарт 2008 года заменяет IEEE 754-1985. В новый стандарт включены двоичные форматы из предыдущего стандарта и три новых формата.

## IEEE 754-1985

### float



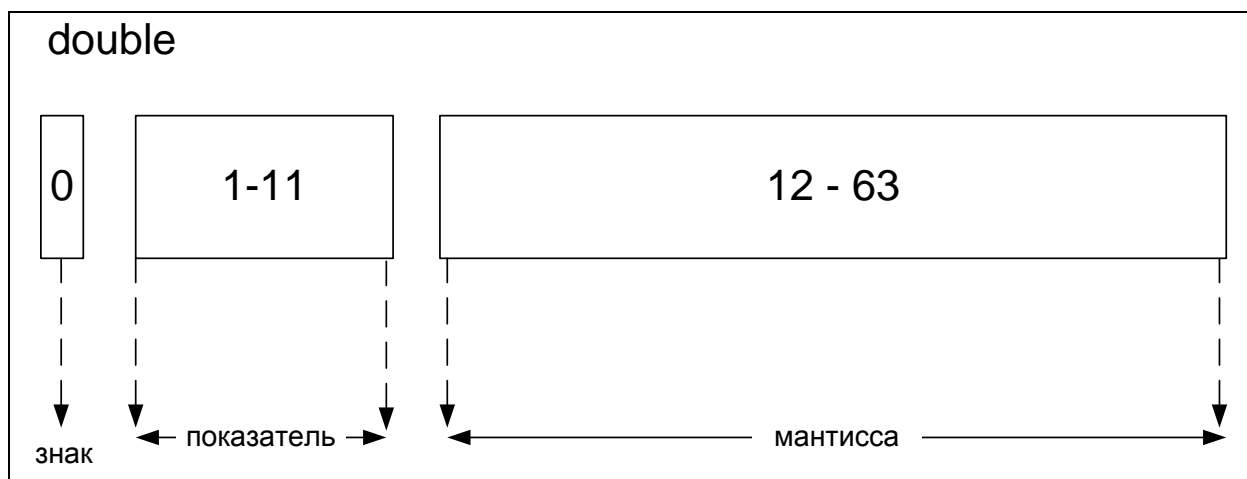
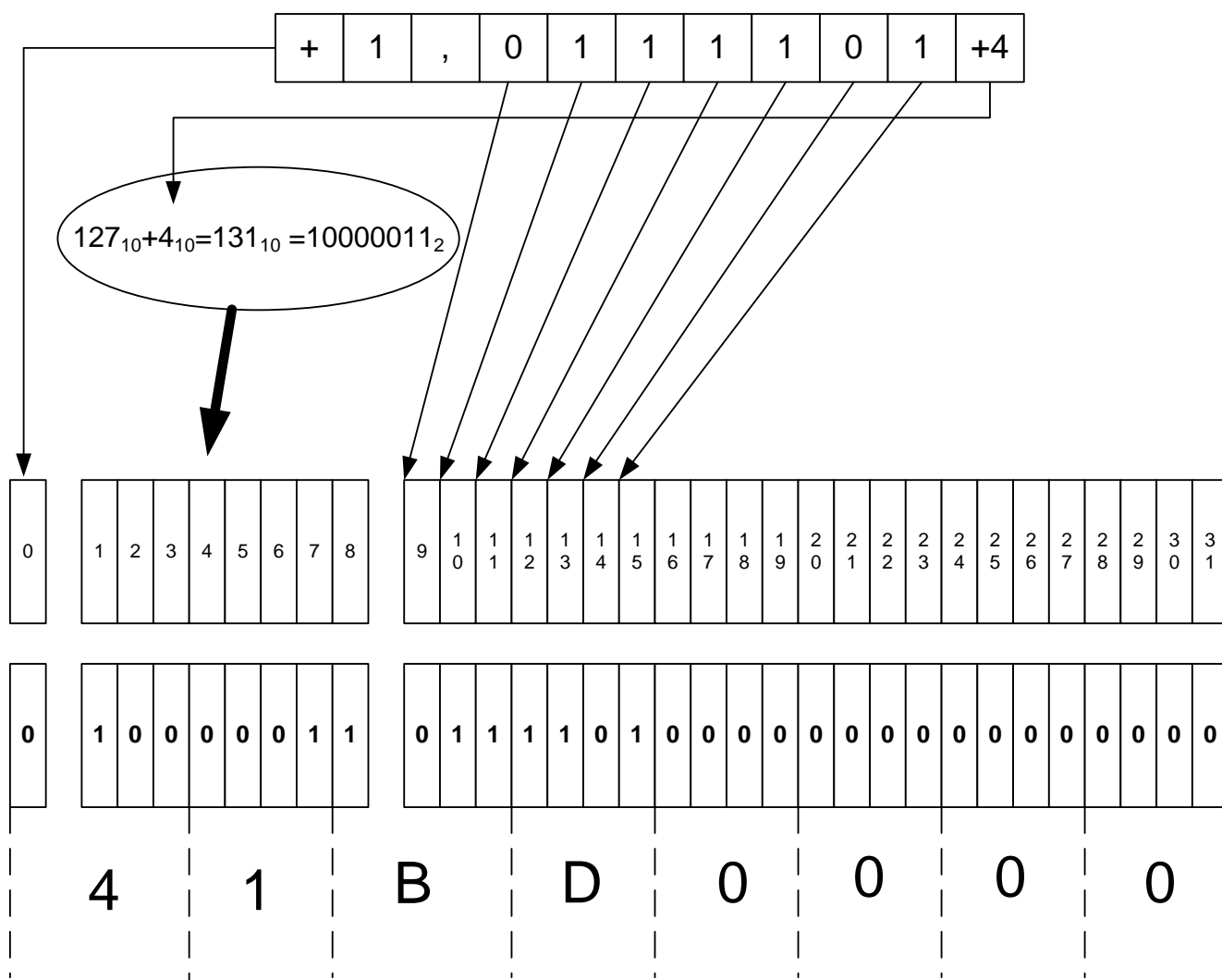
$$23,625_{10} = 10111,101_2 = 1,0111101_2(+4)$$

$$23_{10} = 10111_2$$

$$0,625_{10} = 0,101$$

$$+23,625 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

$$+23,625 = +10111,101 = +1,0111101 (+4)$$





```
// IEEE 754-1985 Standard for Binary Float-Point Arithmetic
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    int ld = sizeof(double);
    int lf = sizeof(float);

    float f1 = 23.625f;
    float f2 = - 23.625f;
    double d1 = 1.234;
    double d2 = -1.234;
    double d3 = 1.234E5;
    double d4 = 1.234E-5;

```

Память 2

Адрес: 0x0033F860

Имя	Значение
ld	0x00000008
lf	0x00000004
&d1	0x0033f890 {1.234}
&d2	0x0033f880 {-1.234}
&d3	0x0033f870 {12340}
&d4	0x0033f860 {1.234E-5}
&f1	0x0033f8ac {23.625}
&f2	0x0033f8a0 {-23.625}

Контрольные значения 1

Имя	Значение
&f1	0x0033f8ac {23.625}
&f2	0x0033f8a0 {-23.625}

0x0033F860 fc 80 93 af fc e0 e9 3e ЪЪ“Ібай>

0x0033F868 cc cc cc cc cc cc cc cc ММММММММ

0x0033F870 00 00 00 00 80 20 fe 40 ....Ъ ю@

0x0033F878 cc cc cc cc cc cc cc cc ММММММММ

0x0033F880 58 39 b4 c8 76 be f3 bf X9rIvsvi

0x0033F888 cc cc cc cc cc cc cc cc ММММММММ

0x0033F890 58 39 b4 c8 76 be f3 3f X9rIvsv?

0x0033F898 cc cc cc cc cc cc cc cc ММММММММ

0x0033F8A0 00 00 bd c1 cc cc cc cc ..SБММММ

0x0033F8A8 cc cc cc cc 00 00 bd 41 ММММ..SA

0x0033F8B0 cc cc cc cc cc cc cc cc ММММММММ

0x0033F8B8 04 00 00 00 cc cc cc cc ....ММММ

0x0033F8C0 cc cc cc cc 08 00 00 00 ММММ....

0x0033F8C8 cc cc cc cc 1c f9 33 00 ММММ.щЗ.

0x0033F8D0 09 4d c1 00 01 00 00 00 .МБ.....

0x0033F8D8 40 9f 71 00 00 cf 71 00 @uq..Пq.

0x0033F8E0 1a aa 06 41 00 00 00 00 .Е.А....

0x0033F8E8 00 00 00 00 00 b0 75 7f .....°u.

```
// IEEE 754-1985 Standard for Binary Float-Point Arithmetic
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    float f1 = 23.625f;
    float f2 = - 23.625f;
    float f3 = f1/0;
    float f4 = f2/0;
    float f5 = f3*5.0f;
    float f6 = f4*5.0f;
    float f7 = f5/5.0f;
    float f8 = sqrt(-2.0f);
    float f9 = f3*0.0f;
    float f10 = f3+f4;

```

Память 2

Имя	Значение
&f3	0x006cfcc4 {1.#INF0000}
&f4	0x006cfcb8 {-1.#INF0000}
&f5	0x006cfcac {1.#INF0000}
&f6	0x006cfca0 {-1.#INF0000}
&f7	0x006cfc94 {1.#INF0000}
&f8	0x006cfc88 {-1.#IND0000}
&f9	0x006cfc7c {-1.#IND0000}
&f10	0x006cfc70 {-1.#IND0000}

Контрольные значения 1

Имя	Значение
&f3	0x006cfcc4 {1.#INF0000}
&f4	0x006cfcb8 {-1.#INF0000}
&f5	0x006cfcac {1.#INF0000}
&f6	0x006cfca0 {-1.#INF0000}
&f7	0x006cfc94 {1.#INF0000}
&f8	0x006cfc88 {-1.#IND0000}
&f9	0x006cfc7c {-1.#IND0000}
&f10	0x006cfc70 {-1.#IND0000}

0x006CFC70 00 00 c0 ff ..Ая

0x006CFC74 cc cc cc cc ММММ

0x006CFC78 cc cc cc cc ММММ

0x006CFC7C 00 00 c0 ff ..Ая

0x006CFC80 cc cc cc cc ММММ

0x006CFC84 cc cc cc cc ММММ

0x006CFC88 00 00 c0 ff ..Ая

0x006CFC8C cc cc cc cc ММММ

0x006CFC90 cc cc cc cc ММММ

0x006CFC94 00 00 80 7f ..Ъ.

0x006CFC98 cc cc cc cc ММММ

0x006CFC9C cc cc cc cc ММММ

0x006CFCA0 00 00 80 ff ..Ъя

0x006CFCA4 cc cc cc cc ММММ

0x006CFCA8 cc cc cc cc ММММ

0x006CFCAC 00 00 80 7f ..Ъ.

0x006CFCB0 cc cc cc cc ММММ

0x006CFCB4 cc cc cc cc ММММ

0x006CFCB8 00 00 80 ff ..Ъя

0x006CFCBC cc cc cc cc ММММ

0x006CFCC0 cc cc cc cc ММММ

0x006CFCC4 00 00 80 7f ..Ъ.

0x006CFCC8 cc cc cc cc ММММ

0x006CFCCC cc cc cc cc ММММ

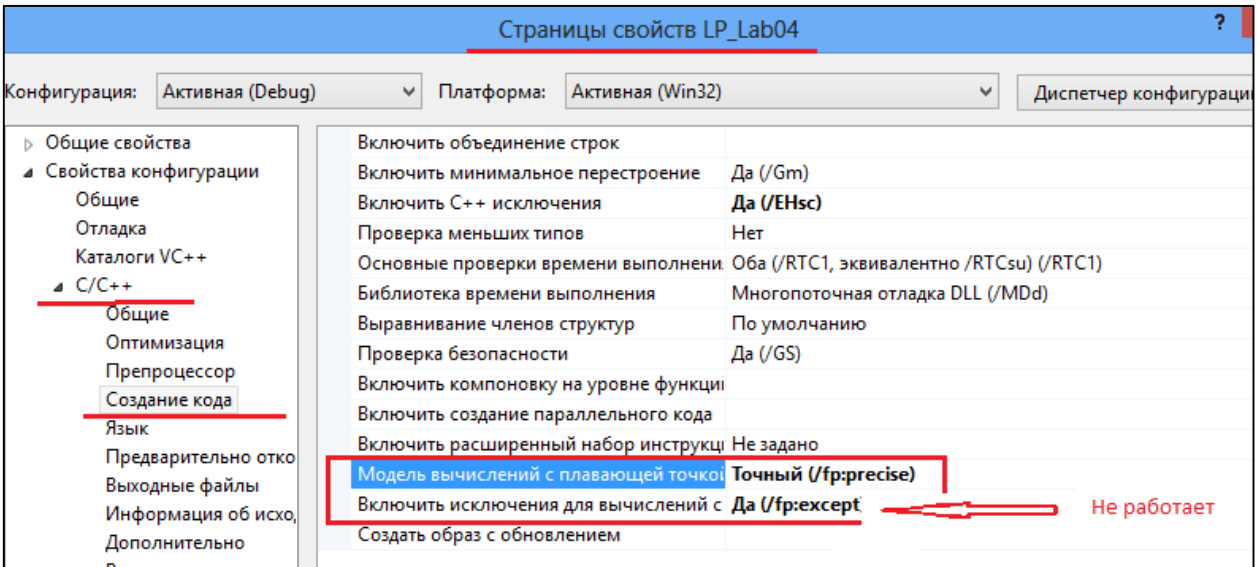
0x006CFCD0 00 00 bd c1 SБ



Стандарт представления значений с плавающей точкой (IEEE 754) оставляет несколько зарезервированных значений, соответствующих **не-числам** (NaN, not-a-number).

Стандартная библиотека Visual C++ выводит не-числа следующим образом:

Печатается	Означает
1.#INF	Положительная бесконечность
-1.#INF	Отрицательная бесконечность
-1.#SNAN	Отрицательное сигнальное не-число ( <i>signaling NaN</i> )
1.#QNAN	Положительное несигнальное не-число ( <i>quiet NaN</i> )
-1.#QNAN	Отрицательное несигнальное не-число ( <i>quiet NaN</i> )
1.#IND	Положительная неопределённость
-1.#IND	Отрицательная неопределённость



```
// IEEE 754-1985 Standard for Binary Float
int _tmain(int argc, _TCHAR* argv[])
{
    int lf = sizeof(float);
    int ld = sizeof(double);
    int lld = sizeof(long double);
}
```

Контрольные значения 1

Имя	Значение
lf	0x00000004
ld	0x00000008
lld	0x00000008

## 14. Фундаментальные типы C++:

*указатель;*

*\*void;*

*опасные и безопасные типы;*

*управляемый код (C#, Java).*

**Неуправляемый код** компилируется для конкретного типа процессора и при вызове просто исполняется. Вся **ответственность** за управлением памятью, безопасностью и т.д. лежит на разработчике.

При формировании **управляемого кода** компиляция состоит из двух шагов:

1. Компиляция исходного кода в **промежуточный байт-код**.
2. Компиляция байт-кода в специфичный для платформы код.

**Таким образом достигается независимость от платформы.**

**Указатель** (pointer) – это переменная, объявленная особым образом, в которой хранится адрес объекта определенного типа (как правило, другой переменной).

**Базовый тип указателя определяется** типом переменной, на которую он ссылается. Значением указателя является **адрес** ячейки памяти (4 байта).

- в языках программирования (ЯП) обычно существует нулевой указатель, который показывает, что эта **переменная-указатель** не ссылается (не указывает) ни на какой объект.
- в C++ – это 0 или макрос NULL, а в стандарте C++11 введено **ключевое слово** nullptr.
- В Pascal, Ruby – nil.
- В C#, Java все типы, кроме **базовых**, являются **ссылочными**: обращение к ним происходит по ссылке, однако явно передать параметр по ссылке нельзя.

В языке C/C++ **три вида указателей**:

- указатель на объект известного типа;
- указатель на объект неопределённого типа **void**;
- указатель на функцию.

Существует два оператора для работы с указателями:

оператор разыменования указателя \* и оператор получения адреса &.

- оператор & является унарным (имеет один операнд) и возвращает адрес своего операнда.
- унарный оператор разыменования указателя \* возвращает значение, хранящееся по указанному адресу.

## Синтаксис объявления указателя:

<тип\_указателя> \*<имя\_указателя>;

## Область применения:

- с помощью указателя легко индексировать элементы массивов;
- указатели позволяют функциям модифицировать свои параметры;
- на основе указателей можно создавать связанные списки и другие динамические структуры.

### а. Пример. Указатель на объект известного типа:

```
wchar_t wc = L'Ц'; // 0x0426
```

The screenshot displays a C++ development environment with three main components:

- Code Editor:** Contains variable declarations and pointer assignments. A red box highlights the declarations: `char c = 'A';`, `wchar_t wc = L'Ц';`, `int k = 5;`, `float f = 1.5f;`, and `long double ld = 2.3E-3;`. A blue box highlights the pointer declarations: `char *pc = &c;`, `wchar_t *pwc = &wc;`, `int *pk = &k;`, `float *pf = &f;`, and `long double *pld = &ld;`. To the right, corresponding dereference assignments are shown: `char cx = *pc;`, `wchar_t wx = *pwc;`, `int kx = *pk;`, `float fx = *pf;`, and `long double ldx = *pld;`.
- Контрольные значения 1 (Watch Window):** A table showing the memory addresses and values of the variables. A red box highlights the pointers `pc`, `pwc`, `pk`, `pf`, and `pld`. A blue box highlights the dereferenced pointers `&pc`, `&pwc`, `&pk`, `&pf`, and `&pld`.
- Память 1 (Memory Window):** A hex dump of memory starting at address 0x0110F99C. A red box highlights the memory address 0x0110F99C, and a blue box highlights the value 0x0426, which corresponds to the Unicode character 'Ц' stored in `wc`.

**! Пример. Логическая ошибка:**

```
whar_t      wc = 'Ц';    // код ASCII 0xd6 расширяется до 2-х байтов
                и значение 0xffd6 присваивается переменной wc
```

[illegible]

```

char          c = 'A';
wchar_t       wc = L'Ц';
int           k = 5;
float         f = 1.5f;
long double   ld = 2.3E-3;

char* pc = &c;
wchar_t* pwc = &wc;
int* pk = &k;
float* pf = &f;
long double* pld = &ld;

char cx = *pc;
wchar_t wcx = *pwc;
int kx = *pk;
float fx = *pf;
long double ldx = *pld;

```

Память 2

Адрес: 0x0091F77C

0x0091F77C	48 50 fc 18 73 d7 62 3f	cc cc cc cc	MF
0x0091F788	cc cc cc cc f8 f7 91 00	cc cc cc cc	MM
0x0091F794	cc cc cc cc 00 00 c0 3f	cc cc cc cc	MM
0x0091F7A0	cc cc cc cc 08 f8 91 00	cc cc cc cc	MM
0x0091F7AC	cc cc cc cc 05 00 00 00	cc cc cc cc	MM
0x0091F7B8	cc cc cc cc 14 f8 91 00	cc cc cc cc	MM
0x0091F7C4	cc cc cc cc 26 04	cc cc cc cc	MM
0x0091F7D0	cc cc cc cc 20 f8 91 00	cc cc cc cc	MM
0x0091F7DC	cc cc cc cc cc cc cc 41	cc cc cc cc	MM
0x0091F7E8	cc cc cc cc 2f f8 91 00	cc cc cc cc	MM
0x0091F7F4	cc cc cc cc 48 50 fc 18 73 d7 62 3f	cc cc cc cc	MM
0x0091F800	cc cc cc cc cc cc cc 00 00 c0 3f	cc cc cc cc	MM
0x0091F80C	cc cc cc cc cc cc cc 05 00 00 00	cc cc cc cc	MM
0x0091F818	cc cc cc cc cc cc cc 26 04	cc cc	MM
0x0091F824	cc cc cc cc cc cc cc cc 41	cc	MM
0x0091F830	cc cc cc cc e4 e8 05 93 58 f8 91 00	cc	MM

120 % Проблемы не найдены.

Контрольные значения 1

Поиск (Ctrl+E) Глубина поиска: 3

Имя	Значение	Тип
c	65 'A'	char
wc	1062 'Ц'	wchar_t
pc	0x0091f82f "AMMMMDm\х5"Хш"	char *
cx	65 'A'	char
pwc	0x0091f820 L"Ц책책책책책책책책책책.췔.췔\х1"	wchar_t *
wcx	1062 'Ц'	wchar_t
pk	0x0091f814 {5}	int *
kx	5	int
pf	0x0091f808 {1.50000000}	float *
fx	1.50000000	float
pld	0x0091f7f8 {0.00230000000000000000}	double *
ldx	0.00230000000000000000	double
&ldx	0x0091f77c {0.00230000000000000000}	double *

**Адресная арифметика** (значение адреса увеличивается/уменьшается на величину, *кратную размеру* того типа данного, для которого был объявлен указатель):

[illegible]

### ***б. Указатель на объект неопределённого типа void***

**\*void:**

В C++ существует специальный тип указателя, который называется *указателем на неопределённый тип*.

**Синтаксис:**

```
void *<имя указателя>;
```

Указателю на неопределённый тип в качестве значений разрешается присваивать значения лишь в сочетании с *операцией явного преобразования* типа. В этом случае указатель на объект неопределённого типа становится обычным указателем на объект некоторого конкретного типа.



```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    char    c = 'A';
    int      k = 5;

    void* x = &c;
    void* y = &k;
    void* z = (int*)y+1;

    return 0;
}
```

Память 2

Адрес: 0x003AFD6C

0x003AFD6C	cc cc cc cc cc cc cc cc	MMMMMMMM
0x003AFD74	9c fd 3a 00 cc cc cc cc	ъэ:..MMM
0x003AFD7C	cc cc cc cc 98 fd 3a 00	MMMM.э:.
0x003AFD84	cc cc cc cc cc cc cc cc	MMMMMMMM
0x003AFD8C	a7 fd 3a 00 cc cc cc cc	§э:..MMM
0x003AFD94	cc cc cc cc 05 00 00 00	MMMM....
0x003AFD9C	cc cc cc cc cc cc cc cc	MMMMMMMM
0x003AFDA4	cc cc cc 41 cc cc cc cc	MMMMMMMM
0x003AFDAC	5a 62 d0 61 00 fe 3a 00	ZbPa.ю:.
0x003AFDB4	a9 19 0b 01 01 00 00 00	θ.....
0x003AFDBC	98 86 7c 00 d0 be 7c 00	.. .Ps .
0x003AFDC4	ea 61 d0 61 00 00 00 00	каPa....
0x003AFDCC	00 00 00 00 00 60 a0 7e	.....`~
0x003AFDD4	80 f8 ff ff c9 58 f2 1f	ЪшыяЙХт.
0x003AFDDC	00 00 00 00 00 00 3b 00	.....;
0x003AFDE4	00 00 00 00 c4 fd 3a 00	....Дэ:.
0x003AFDEC	00 00 00 00 48 fe 3a 00	....Ню:.
0x003AFDF4	7d 10 0b 01 6a f1 e1 60	}....ic6`

Контрольные значения 1

Имя	Значение
&c	0x003afda7 "AMMMMZbPa"
&k	0x003afd98 {0x00000005}
(char*)x	0x003afda7 "AMMMMZbPa"
(int*)y	0x003afd98 {0x00000005}
(int*)z	0x003afd9c {0xcccccccc}

### с. Указатели на функции

Адрес функции задается ее **именем**, указанным без скобок и аргументов.

```
// указатель на функцию.cpp: определяет т
//

#include "stdafx.h"

int sum(int x, int y){ return x + y; };
int sub(int x, int y){ return x - y; };

int(*f) (int, int);

int _tmain(int argc, _TCHAR* argv[])
{
    f = sum;
    int x = f(2, 3);
    f = sub;
    int y = f(2, 3);

    return 0;
}
```

Память 1

Адрес: 0x00BB8134 ← &f

0x00BB8134	5f 10 bb 00 01 00 00 00 00 00 00 00 00	_.».....
0x00BB8140	00 00 00 00 00 00 00 00 01 00 00 00	.....
0x00BB814C	18 52 8d 00 d0 52 8d 00 00 00 00 00	.RĲ.PRĲ....
0x00BB8158	00 00 00 00 00 00 00 00 00 00 00 00	.....

Память 2

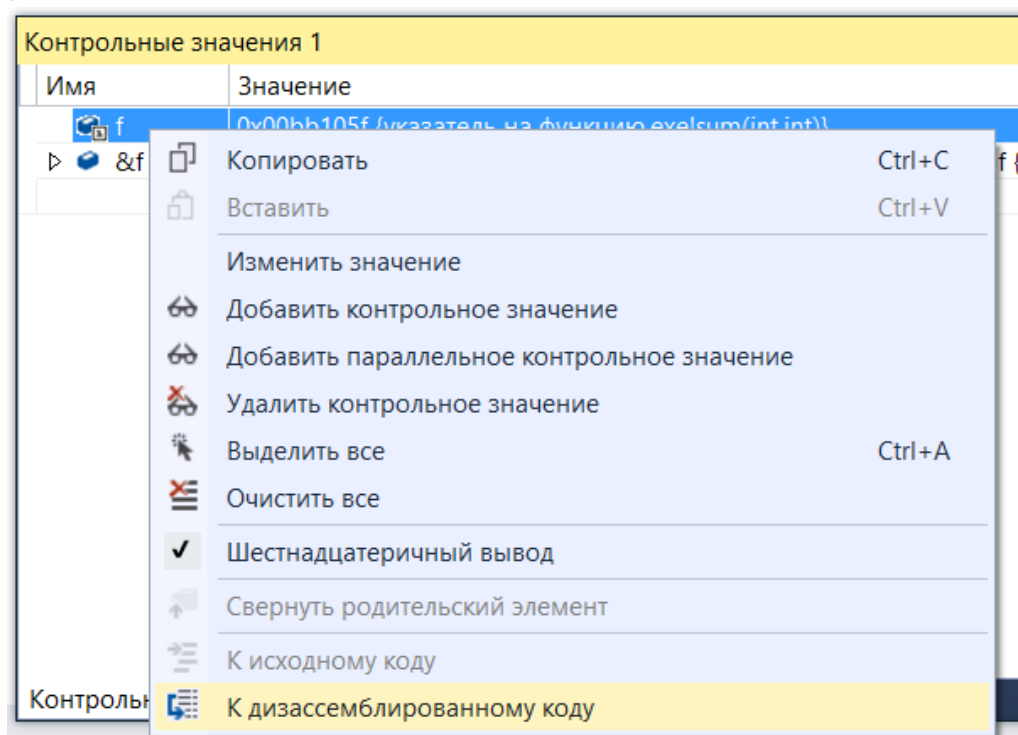
Адрес: 0x00BB105F ← f

0x00BB105F	e9 9c 03 00 00 e9 cb 16 00 00 e9 02	йнь...йл...й.
0x00BB106B	2c 00 00 e9 bd 1c 00 00 e9 bc 2b 00	,...йS...йж+.
0x00BB1077	00 e9 b3 15 00 00 e9 5a 0b 00 00 e9	й: йа й

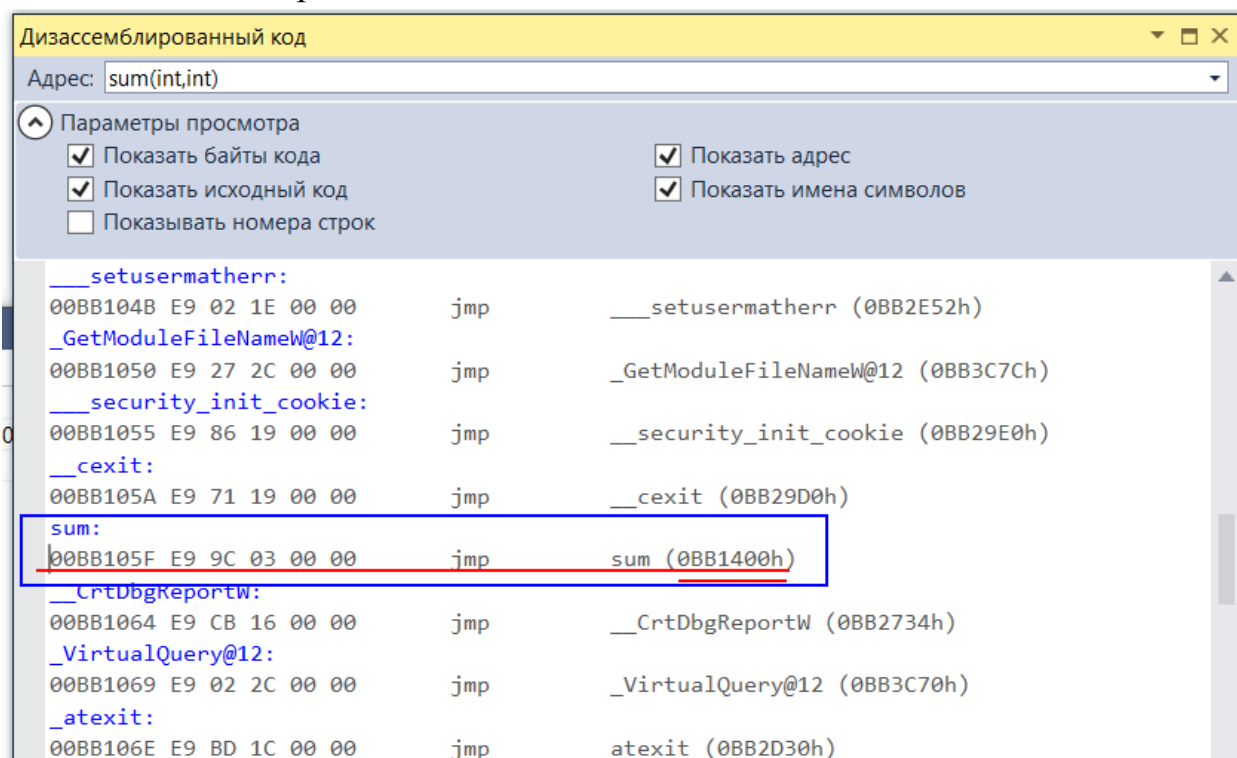
Контрольные значения 1

Имя	Значение	Тип
f	0x00bb105f (указатель на функцию.exe!sum(int,int))	int (int, int) *
&f	0x00bb8134 (указатель на функцию.exe!int(*f)(int, int)) {0x00bb105f (указатель на функцию.exe!sum(int,int))}	int (int, int) **

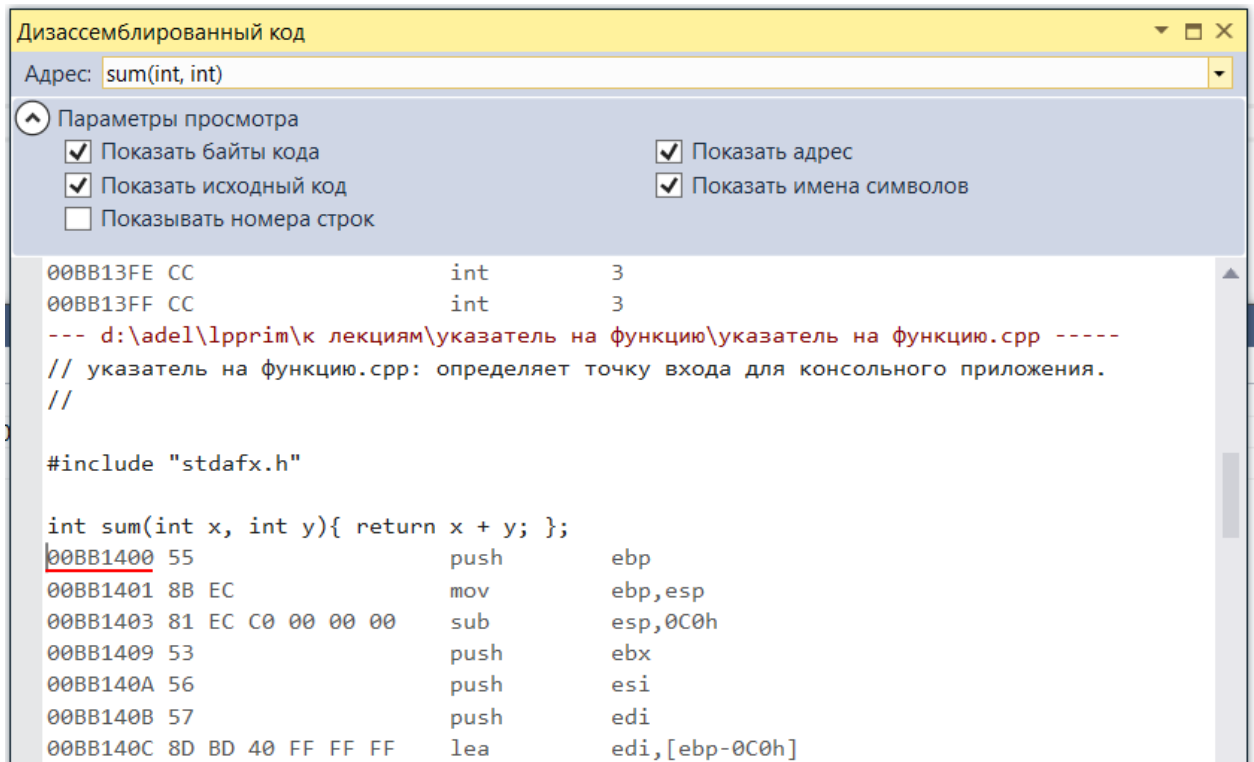
Открываем окно дизассемблированного кода нажатием правой кнопки на указателе f:



Окно дизассемблированного кода:



Выполняем шаг с заходом в функцию:



Дизассемблированный код

Адрес: `sum(int, int)`

Параметры просмотра

- ☒ Показать байты кода
- ☒ Показать исходный код
- ☐ Показывать номера строк
- ☒ Показать адрес
- ☒ Показать имена символов

```
00BB13FE CC          int      3
00BB13FF CC          int      3
--- d:\adel\lpprim\к лекциям\указатель на функцию\указатель на функцию.cpp -----
// указатель на функцию.cpp: определяет точку входа для консольного приложения.
//

#include "stdafx.h"

int sum(int x, int y){ return x + y; };
00BB1400 55          push     ebp
00BB1401 8B EC      mov      ebp,esp
00BB1403 81 EC C0 00 00 00  sub     esp,0C0h
00BB1409 53          push     ebx
00BB140A 56          push     esi
00BB140B 57          push     edi
00BB140C 8D BD 40 FF FF FF  lea     edi,[ebp-0C0h]
```



## 15. Фундаментальные типы C++: ссылки

**Ссылка** (reference) в языке C++ – это *псевдоним* другого объекта.

Независимые переменные ссылочного типа должны быть инициализированы при своем объявлении, т.к. они должны указывать на какой-либо объект. Таким образом мы присваиваем переменной ссылочного типа адрес уже **объявленного** объекта.

[illegible]



**16. Массивы данных фундаментальных типов:** коллекция однородных данных, размещенных последовательно в памяти и допускающие доступ по индексу, который определяется:

$$\text{смещение} = \text{индекс\_элемента} * \text{sizeof(базовый\_тип)}.$$

**Массив** (array) – это совокупность переменных, имеющих одинаковый тип и объединенных под одним именем. Доступ к отдельному элементу массива осуществляется с помощью индекса.

Объем памяти, необходимый для хранения массива, зависит от его типа и количества элементов в нем.

Размер одномерного массива в байтах вычисляется по формуле:

$$\text{количество\_байтов} = \text{sizeof(базовый\_тип)} * \text{количество\_элементов}$$

The screenshot displays a C++ program and its execution state. The code defines two boolean arrays, two character arrays, and calculates their sizes. It also declares pointers to the first array. The 'Контрольные значения 1' (Check values 1) window shows the memory addresses and values of these variables. The 'Память 2' (Memory 2) window shows a hex dump of memory, with a pink arrow pointing from the 'pc1' variable to its corresponding memory location.

```
int _tmain(int argc, _TCHAR* argv[])
{
    bool b1[5] = { true, false, true, false, true };
    bool b2[] = { true, false, true, false, true };

    char c1[] = { '1', '2', '3', '4', 'a', 'b', 'c', 'd', 'ц', 'ч', 'щ', 'ю', 'я' };
    char c2[] = "1234abcdцщюя";

    int lb1 = sizeof(b1);
    int lb2 = sizeof(b2);

    int lc1 = sizeof(c1);
    int lc2 = sizeof(c2);

    bool *pb1 = b1;
    char *pc1 = c1;
}
```

Имя	Значение	Тип
pb1	0x00e1fc5c {true}	bool *
pc1	0x00e1fc34 "1234abcdцщюя"	char *
&lc1	0x00e1fbf8 {13}	int *
&lc2	0x00e1fbec {14}	int *
c2	0x00e1fc1c "1234abcdцщюя"	char[14]
c1	0x00e1fc34 "1234abcdцщюя"	char[13]
b2	0x00e1fc4c {true, false, true, false, true}	bool[5]
b1	0x00e1fc5c {true, false, true, false, true}	bool[5]

Memory dump (Память 2) showing hex values and ASCII characters. A pink arrow points from the 'pc1' variable to the memory address 0x00e1fc34, which contains the string "1234abcdцщюя".

```

1 //
2 #include "stdafx.h"
3
4 int _tmain(int argc, _TCHAR* argv[])
5 {
6
7     wchar_t Lc1[] = {L'1', L'2', L'3'};
8     wchar_t Lc2[] = L"1234abcdцщющя";
9
10    int llc1 = sizeof(Lc1);
11    int llc2 = sizeof(Lc2);
12
13
14    return 0;
15 }
16
17

```

Память 2

Адрес: 0x00C9FA58

0x00C9FA58	31 00 32 00 33 00 34	1.2.3.4
0x00C9FA5F	00 61 00 62 00 63 00	.a.b.c.
0x00C9FA66	64 00 46 04 47 04 49	d.F.G.I
0x00C9FA6D	04 4e 04 4f 04 00 00	.N.O...
0x00C9FA74	cc cc cc cc cc cc cc	MMMMMM
0x00C9FA7B	cc 31 00 32 00 33 00	M1.2.3.
0x00C9FA82	34 00 61 00 62 00 63	4.a.b.c
0x00C9FA89	00 64 00 46 04 47 04	.d.F.G.
0x00C9FA90	49 04 4e 04 4f 04 cc	I.N.O.M
0x00C9FA97	cc cc cc cc cc ec fa	MMMMMMь
0x00C9FA9E	c9 00 19 1a 40 00 01	Й...@..
0x00C9FAA5	00 00 00 40 7f e1 00	...@.6.
0x00C9FAAC	30 80 e1 00 55 a8 c0	0Ъ6.UËA
0x00C9FAB3	bc 00 00 00 00 00 00	j.....
0x00C9FABA	00 00 00 f0 45 7e 80	...pE~Ъ
0x00C9FAC1	f8 ff ff c9 e7 7c 1a	шяяЙЪ .
0x00C9FAC8	00 00 00 00 00 00 00	ca .....K
0x00C9FACF	00 00 00 00 00 b0 fa	.....°ъ
0x00C9FAD6	c9 00 00 00 00 00 34	Й.....4
0x00C9FADD	fb c9 00 7d 10 40 00	ый.}.@.
0x00C9FAE4	b1 3d 49 bc 00 00 00	±=Ij...

Контрольные значения 1

Имя	Значение
llc1	26
llc2	28
Lc1	0x00c9fa7c L"1234abcdцщющя..."
Lc2	0x00c9fa58 L"1234abcdцщющя"

```

1 #include "stdafx.h"
2
3 int _tmain(int argc, _TCHAR* argv[])
4 {
5
6     short s[] = { 1, 2, 3, 4, 5, 6, 7 };
7     int i[] = { 1, 2, 3, 4, 5, 6, 7 };
8
9     float f[] = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0 };
10    double d[] = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0 };
11
12    int ls = sizeof(s);
13    int li = sizeof(i);
14    int lf = sizeof(f);
15    int ld = sizeof(d);
16
17    return 0;
18 }

```

Память 2

Адрес: 0x008EF6A4

0x008EF6A4	00 00 00 00 00 00 f0 3f	.....p?
0x008EF6AC	00 00 00 00 00 00 00 40	.....@
0x008EF6B4	00 00 00 00 00 00 08 40	.....@
0x008EF6BC	00 00 00 00 00 00 10 40	.....@
0x008EF6C4	00 00 00 00 00 00 14 40	.....@
0x008EF6CC	00 00 00 00 00 00 18 40	.....@
0x008EF6D4	00 00 00 00 00 00 1c 40	.....@
0x008EF6DC	cc cc cc cc cc cc cc cc	MMMMMMMM
0x008EF6E4	00 00 80 3f 00 00 00 40	..Ъ?...@
0x008EF6EC	00 00 40 40 00 00 80 40	..@@...Ъ@
0x008EF6F4	00 00 a0 40 00 00 c0 40	.. @...A@
0x008EF6FC	00 00 e0 40 cc cc cc cc	..a@MMMM
0x008EF704	cc cc cc cc 01 00 00 00	MMMM....
0x008EF70C	02 00 00 00 03 00 00 00	.....
0x008EF714	04 00 00 00 05 00 00 00	.....
0x008EF71C	06 00 00 00 07 00 00 00	.....
0x008EF724	cc cc cc cc cc cc cc cc	MMMMMMMM
0x008EF72C	01 00 02 00 03 00 04 00	.....
0x008EF734	05 00 06 00 07 00 cc cc	.....MM
0x008EF73C	cc cc cc cc 90 f7 8e 00	MMMMђчЪ.
0x008EF744	19 1b 05 00 01 00 00 00	.....
0x008EF74C	20 53 cf 00 c0 53 cf 00	СП.ASP.
0x008EF754	88 42 8c b4 00 00 00 00	€BЪг....
0x008EF75C	00 00 00 00 00 e0 31 7e	.....a1~
0x008EF764	80 f8 ff ff c9 18 c6 1c	ЪшяяЙ.Ж.

Контрольные значения 1

Имя	Значение	Тип
ld	56	int
lf	28	int
li	28	int
ls	14	int
d	0x008ef6a4 {1.0000000000000000, 2.0000000000000000, 3.0000000000000000, 4.0000000000000000, 5.0000000000000000, 6.0000000000000000, 7.0000000000000000}	double[7]
f	0x008ef6e4 {1.00000000, 2.00000000, 3.00000000, 4.00000000, 5.00000000, 6.00000000, 7.00000000}	float[7]
i	0x008ef708 {1, 2, 3, 4, 5, 6, 7}	int[7]
s	0x008ef72c {1, 2, 3, 4, 5, 6, 7}	short[7]

28

Массив указателей, элементами которого являются ссылки на элементы массива `i`:

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int i[] = { 1, 2, 3, 4, 5, 6, 7 };
    int *pi[] = { &i[0], &i[1], &i[2], &i[3], &i[4], &i[5], &i[6] };
    int lpi = sizeof(pi);

    return 0;
}
```

Контрольные значения 1

Имя	Значение
pi	0x0035fc5c {0x0035fc80 {1}, 0x0035fc84 {2}, 0x0035fc88 {3}, 0x0035fc8c {4}, 0x0035fc90 {5}, 0x0035fc94 {6}, 0x0035fc98 {7}}
i	0x0035fc80 {1, 2, 3, 4, 5, 6, 7}
lpi	28

Память 2

Адрес	0x0035FC5C	0x0035FC64	0x0035FC6C	0x0035FC74	0x0035FC7C	0x0035FC84	0x0035FC8C	0x0035FC94	0x0035FC9C
80 fc 35 00	84 fc 35 00	90 fc 35 00	98 fc 35 00	cc cc cc cc	02 00 00 00	04 00 00 00	06 00 00 00	07 00 00 00	cc cc cc cc
88 fc 35 00	8c fc 35 00	94 fc 35 00	cc cc cc cc	01 00 00 00	03 00 00 00	05 00 00 00	07 00 00 00	07 00 00 00	18 49 99 45

Массив указателей на функции:

```
#include "stdafx.h"

char f1(char c) { return c; }
char f2(char c) { return c; }
char f3(char c) { return c; }
char f4(char c) { return c; }

int _tmain(int argc, _TCHAR* argv[])
{
    char(*f[])(char) = { f1, f2, f3, f4 };
    int lf = sizeof(f);

    return 0;
}
```

Контрольные значения 1

Имя	Значение	Тип
lf	16	int
f	0x00a2f730 {0x01231064 {LP_Lab06.exelf1(char)}, 0x012310a5 {LP_Lab06.exelf2(char)}, 0x012310e6 {LP_Lab06.exelf3(char)}, 0x01231127 {LP_Lab06.exelf4(char)}}	

Память 2

Адрес	0x00A2F724	0x00A2F728	0x00A2F72C	0x00A2F730	0x00A2F734	0x00A2F738	0x00A2F73C	0x00A2F740	0x00A2F744	0x00A2F748	0x00A2F74C
10 00 00 00	cc cc cc cc	cc cc cc cc	64 10 23 01	a5 10 23 01	a0 10 23 01	14 10 23 01	cc cc cc cc	94 f7 a2 00	79 1a 23 01	01 00 00 00	

## 17. Многомерные массивы C/C++: многомерность моделируется в виде «массива массивов»

Доступ к элементу, стоящему на пересечении первой строки и третьего столбца, можно получить двумя способами:

- либо индексируя массив – `mm[0][2]`;
- либо используя указатель – `*(( <базовый_тип> *)mm+2)`.

Правила адресной арифметики требуют приведения типа указателя на массив к его базовому типу.

Двухмерный массив можно представить с помощью указателей на одномерные массивы.

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int mm[][3] = { { 1, 2, 3 }, { 4, 5, 6 } };

    int *m1 = mm[0];
    int *m2 = mm[1];

    int m11 = m1[0];
    int m12 = m1[1];
    int m13 = m1[2];

    int m21 = m2[0];
    int m22 = m2[1];
    int m23 = m2[2];
}
```

**Память 2**

Адрес	Значение (hex)	Значение (dec)
0x0034F920	44 f9 34 00	Дц4.
0x0034F924	cc cc cc cc	MMMM
0x0034F928	cc cc cc cc	MMMM
0x0034F92C	38 f9 34 00	8ц4.
0x0034F930	cc cc cc cc	MMMM
0x0034F934	cc cc cc cc	MMMM
0x0034F938	01 00 00 00	....
0x0034F93C	02 00 00 00	....
0x0034F940	03 00 00 00	....
0x0034F944	04 00 00 00	....
0x0034F948	05 00 00 00	....
0x0034F94C	06 00 00 00	....
0x0034F950	cc cc cc cc	MMMM

**Контрольные значения 1**

Имя	Значение	Тип
m23	6	int
m22	5	int
m21	4	int
m13	3	int
m12	2	int
m11	1	int
m2	0x0034f944 {4}	int *
m1	0x0034f938 {1}	int *
mm	0x0034f938 {0x0034f938 {1, 2, 3}, 0x0034f944 {4, 5, 6}}	int[2][3]



## 18. Пользовательские типы:

типы, создаваемые пользователем, всегда есть объявление типа.

объявление типа – существует на уровне транслятора (память не выделяется).

## 19. Пользовательские типы C/C++:

enum;

struct;

union;

typedef;

class (спецификатор типа).

### Пользовательские типы C++: enum.

```
#include "stdafx.h"

enum EEE {A,B,C,D = -2};
int _tmain(int argc, _TCHAR* argv[])
{
    int D = 28;
    EEE k = EEE::A;
    EEE m = EEE::D;
    int j = EEE::B;
    int l = D;
    int v = C;
    int g = ::D;

    return 0;
}
```

**Память 2**

Адрес	Содержимое (hex)	Содержимое (ASCII)
0x00F9FDE4	fe ff ff ff	юяяя
0x00F9FDE8	cc cc cc cc	MMMM
0x00F9FDEC	cc cc cc cc	MMMM
0x00F9FDF0	02 00 00 00	....
0x00F9FDF4	cc cc cc cc	MMMM
0x00F9FDF8	cc cc cc cc	MMMM
0x00F9FDfc	1c 00 00 00	....
0x00F9FE00	cc cc cc cc	MMMM
0x00F9FE04	cc cc cc cc	MMMM
0x00F9FE08	01 00 00 00	....
0x00F9FE0C	cc cc cc cc	MMMM
0x00F9FE10	cc cc cc cc	MMMM
0x00F9FE14	fe ff ff ff	юяяя
0x00F9FE18	cc cc cc cc	MMMM
0x00F9FE1C	cc cc cc cc	MMMM
0x00F9FE20	00 00 00 00	....
0x00F9FE24	cc cc cc cc	MMMM

**Контрольные значения 1**

Имя	Значение	Тип
&D	0x00f9fe2c {0x0000001c}	int *
&k	0x00f9fe20 {A (0x00000000)}	EEE *
&m	0x00f9fe14 {D (0xffffffff)}	EEE *
&j	0x00f9fe08 {0x00000001}	int *
&l	0x00f9fdfc {0x0000001c}	int *
&v	0x00f9fdf0 {0x00000002}	int *
&g	0x00f9fde4 {0xffffffff}	int *

```
#include "stdafx.h"

enum EEE { A, B, C, D = -2 };
int _tmain(int argc, _TCHAR* argv[])
{
    int s = sizeof(EEE);
    return 0;
}
```

Контрольные значения 1

Имя	Значение	Тип
s	0x00000004	int

```
#include "stdafx.h"

enum {ZERO, ONE, TWO}; // определение целочисленных констант
int _tmain(int argc, _TCHAR* argv[])
{
    int k0 = ZERO;
    int k1 = ONE;
    int k2 = TWO;
    return 0;
}
```



## 20. Пользовательские типы C++: struct.

Структура обеспечивает удобный способ организации логически связанных данных.

```
6 struct MyStruct
7 {
8     char c;           // 1
9     wchar_t wc;       // 2
10    int i;             // 4
11    short s;           // 2
12    float f;           // 4
13 };                  // --> 13 байт
14
15 int _tmain(int argc, _TCHAR* argv[])
16 {
17     MyStruct mystruct = {'a', 'ц', 77777, 77, 2.8E-5};
18
19     int size = sizeof(MyStruct);
20
21     return 0;
22 }
```

Контрольные значения 1	
Имя	Значение
size	0x00000010

Адрес: 0x00A5FA0C Столбцы:

0x00A5FA0C	61	cc	f6	ff	d1	2f	01	00	4d	00	cc	cc	8b	e1	ea	37
0x00A5FA1C	cc	cc	cc	cc	70	fa	a5	00	89	19	11	00	01	00	00	00

&mystruct

0x00a5fa0c {c=0x61 'a' wc=0xffff6 'ц' i=0x00

c	0x61 'a'
wc	0xffff6 'ц'
i	0x00012fd1
s	0x004d
f	2.80000004e-005

```

3  #include "stdafx.h"
4
5
6  struct MyStruct
7  {
8      char c;           // 1
9      char c1;          // 1
10     wchar_t wc;        // 2
11     int i;             // 4
12     short s;           // 2
13     float f;           // 4
14     };                // --> 14 байт
15
16  int _tmain(int argc, _TCHAR* argv[])
17  {
18      MyStruct mystruct = {'a', 'b', 'ц', 77777, 77, 2.8E-5};
19
20      int size = sizeof(MyStruct);
21
22      return 0;
23  }

```

Имя переменной	Значение
size	0x00000010
&mystruct	0x0094fb10 {c=0x61 'a' c1=0x62 'b' wc=0xffff 'ц' ...}

```

15
16  int _tmain(int argc, _TCHAR* argv[])
17  {
18      MyStruct mystruct = {'a', 'b', 'ц', 77777, 77, 2.8E-5};
19      MyStruct ms2;
20      MyStruct* pms;
21      int size = sizeof(MyStruct);
22
23      char c = mystruct.c;
24      int i = mystruct.i;
25
26      ms2 = mystruct;
27
28      pms = &mystruct;
29
30      short cs = pms->s;
31      float cf = pms->f;
32      return 0;
33  }
34

```

Имя переменной	Значение
&mystruct	0x00eaf80c {c=0x61 'a' c1=0x62 'b' wc=0xffff 'ц' ...}
&ms2	0x00eaf7f4 {c=0x61 'a' c1=0x62 'b' wc=0xffff 'ц' ...}

Адресная арифметика:

**<адрес в указателе> + <значение\_int\_выражения>\*sizeof(<тип>),**

где

**значение\_int\_выражения** – это количество объектов;

**тип** – это тип данных, на которые ссылается указатель.

```
6 struct MyStruct
7 {
8     char c;           // 1
9     char c1;          // 1
10    wchar_t wc;        // 2
11    int i;             // 4
12    short s;          // 2
13    float f;           // 4
14 };                   // --> 14 байт
15
16 int _tmain(int argc, _TCHAR* argv[])
17 {
18     MyStruct mystruct = {'a', 'b', 'ц', 77777, 77, 2.8E-5};
19     MyStruct *pms, *pms1;
20
21     pms = &mystruct;
22     pms1 = pms+1;
23
24     return 0;
25 }
```

Адрес:	0x0035FBC8	Столбцы:	A
0x0035FBC8	61 62 f6 ff d1 2f 01 00 4d 00 cc cc 8b e1 ea 37		
0x0035FBD8	cc cc cc cc c5 2a e7 cd 30 fc 35 00 89 19 25 00		
0x0035FBE8	01 00 00 00 f8 81 42 00 e8 82 42 00 15 2d e7 cd		

...	0x0035fbc8	c=0x61 'a' c1=0x62 'b' wc=0xffff6 'ц'.
pms	0x0035fbc8	c=0x61 'a' c1=0x62 'b' wc=0xffff6 'ц'.
pms1	0x0035fbd8	c=0xcc 'M' c1=0xcc 'M' wc=0xcccc ' '.

```

#include "stdafx.h"
#include <locale>
#include <iostream>
struct MyStruct
{
    char c;           // 1
    char c1;          // 1
    wchar_t wc;       // 2
    int i;             // 4
    short s;          // 2
    float f;          // 4
};                    // --> 14 байт

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus"); // настройка кодировки

    MyStruct mystruct = {'a', 'b', 'ц', 77777, 77, 2.8E-5};
    MyStruct *pms, *pms1;

    pms = &mystruct;
    pms1 = pms+1;

    std::cout << "c =" << pms1->c <<std::endl
               << "c1 =" << pms1->c1 <<std::endl
               << "wc =" << pms1->wc <<std::endl
               << "i =" << pms1->i <<std::endl
               << "s =" << pms1->s <<std::endl
               << "f =" << pms1->f <<std::endl ;

    system("pause");
    return 0;
}

```

```

C:\Users\User Pc\documents\visual stu...
c =M
c1 =M
wc =52428
i =206403236
s =-1520
f =2.17081e-038
Для продолжения нажмите любую клавишу

```

namespace std

## 21. Пользовательские типы C++:

struct, поля битов.

В языке C/C++ реализована встроенная поддержка **битовых полей** (bit-fields), предоставляющих доступ к отдельным битам.

```

4 #include <locale>
5 #include <iostream>
6
7 struct bit32 { char a0:8; char a1:8; char a2:8; char a3:8; };
8 struct bit8 { short b0:1, b1:1, b2:1, b3:1, b4:1, b5:1, b6:1, b7:1; };
9 int _tmain(int argc, _TCHAR* argv[])
10 {
11     int x = 0x01020304;
12
13     bit32 *px = (bit32*)&x;
14     bit8 *px1 = (bit8*)((char*)&x+1);
15     bit8 *px2 = (bit8*)((char*)&x+2);
16
17     std::cout<<(short)px->a0<<(short)px->a1<<(short)px->a2<<(short)px->a3<<std::endl;
18     std::cout<<-px1->b0<<-px1->b1<<-px1->b2<<-px1->b3
19         <<-px1->b4<<-px1->b5<<-px1->b6<<-px1->b7 << std::endl;
20     std::cout<<-px2->b0<<-px2->b1<<-px2->b2<<-px2->b3
21         <<-px2->b4<<-px2->b5<<-px2->b6<<-px2->b7 << std::endl;
22
23     system("pause");
24     return 0;
25 }

```

Имя	Значение
px->a0	0x04 '\x4'
px->a1	0x03 '\x3'
px->a2	0x02 '\x2'
px->a3	0x01 '\x1'
px1	0x00f2f975 {b0=0xffff b1=0xffff b2=0x0000 ...}

$$0x03_{16} = 000000011_2$$

Имя	Значение
&x	0x0103fb78 {0x01020304}
px	0x0103fb78 {a0=0x04 '\x4' a1=0x03 '\x3' a2=0x02 '\x2' ...}
px1	0x0103fb79 {b0=0xffff b1=0xffff b2=0x0000 ...}
b0	0xffff
b1	0xffff
b2	0x0000
b3	0x0000
b4	0x0000
b5	0x0000
b6	0x0000
b7	0x0000
px2	0xcccccc {b0=??? b1=??? b2=??? ...}
px1->b0	0xffff
-px1->b0	0x00000001

```
#include "stdafx.h"
#include <iostream>

struct bit32{ char a0 : 8; char a1 : 8; char a2 : 8; char a3 : 8; };
struct bit8{ short b0 : 1, b1 : 1, b2 : 1, b3 : 1, b4 : 1, b5 : 1, b6 : 1, b7 : 1; };

int _tmain(int argc, _TCHAR* argv[])
{
    int x = 0x01020304;

    bit32 *px = (bit32*)&x;
    bit8 *px1 = (bit8*)((char*)&x + 1);
    bit8 *px2 = (bit8*)((char*)&x + 2);
    px2->b0 = px2->b1 = px2->b2 = px2->b3 = 1;

    system("pause");
    return 0;
}

/*
struct bbit8{ bool b0 : 1,
bbit8 *px3 = (bbit8*)&x ;

```

Память 2

Адрес: &x

0x00E9FC14 04 03 02 01 cc cc cc cc 7c 81 b8 99 70 fc e9 ....MMMM|гё"рьй

Память 4

Адрес: 0x00E9FC15 px1

0x00E9FC15 03 02 01 cc cc cc cc 7c 81 b8 99 70 fc e9 00 ...MMMM|гё"рьй.

Память 1

Адрес: 0x00E9FC16 px2

0x00E9FC16 02 01 cc cc cc cc 7c 81 b8 99 70 fc e9 00 09 ..MMMM|гё"рьй..

Контрольные значения 1

Имя	Значение	Тип
x	0x01020304	int
&x	0x00e9fc14 {0x01020304}	int *
px	0x00e9fc14 {a0=0x04 'x4' a1=0x03 'x3' a2=0x02 'x2' ...}	bit32 *
px1	0x00e9fc15 {b0=0xffff b1=0xffff b2=0x0000 ...}	bit8 *
px2	0x00e9fc16 {b0=0x0000 b1=0xffff b2=0x0000 ...}	bit8 *

(ГЛОБАЛЬНАЯ ОБЛАСТЬ)

```
4 #include <locale>
5 #include <iostream>
6
7 struct bit32 { char a0:8; char a1:8; char a2:8; char a3:8; };
8 struct bit8 { short b0:1, b1:1, b2:1, b3:1, b4:1, b5:1, b6:1, b7:1; };
9 int _tmain(int argc, _TCHAR* argv[])
10 {
11
12     int x = 0x01020304;
13
14     bit32 *px = (bit32*)&x;
15     bit8 *px1 = (bit8*)((char*)&x+1);
16     bit8 *px2 = (bit8*)((char*)&x+2);
17     px2->b0 = px2->b1 = px2->b2 = px2->b3 = 1;
18
19     system("pause");
20     return 0;
21 }
22
```

Память 2

0x0082FCB8 04 03 02 01 ....

0x0082FCBC cc cc cc cc MMMM

```

4  #include <locale>
5  #include <iostream>
6
7  struct bit32 { char a0:8; char a1:8; char a2:8; char a3:8; };
8  struct bit8  { short b0:1, b1:1, b2:1, b3:1, b4:1, b5:1, b6:1, b7:1; };
9  int _tmain(int argc, _TCHAR* argv[])
10 {
11
12     int x = 0x01020304;
13
14     bit32 *px = (bit32*)&x;
15     bit8 *px1 = (bit8*)((char*)&x+1);
16     bit8 *px2 = (bit8*)((char*)&x+2);
17     px2->b0 = px2->b1= px2->b2=px2->b3 = 1;
18
19     system("pause");
20     return 0;
21 }
22

```

Память 2

0x0082FCB8	04 03 0f 01
0x0082FCBC	cc cc cc cc

Контрольные значения 1

Имя	Значение
x	0x010f0304

```

4  #include <locale>
5  #include <iostream>
6
7  struct bit8b { bool b0:1, b1:1, b2:1, b3:1, b4:1, b5:1, b6:1, b7:1; };
8  int _tmain(int argc, _TCHAR* argv[])
9  {
10
11     int x = 0x01020304;
12     bit8b *px3 = (bit8b*)&x;
13     system("pause");
14     return 0;
15 }
16

```

Контрольные значения 1

Имя	Значение
x	0x01020304
*px3	{b0=false b1=false b2=true ...}
b0	false
b1	false
b2	true
b3	false
b4	false
b5	false
b6	false
b7	false

## 22. Пользовательские типы C++: union.

Объединение – это пользовательская переменная, которая может хранить объекты различного типа и размера. Для их размещения выделяется одна общая память, размерность которой определяется размерностью максимального элемента объединения.

The screenshot shows a C++ program defining a union named `bit32` and using it in `_tmain`. The union contains a `short`, a `char` array, and an `int`. The `_tmain` function initializes an `int` variable `x` and assigns it to the `bit32` union.

```
#include "stdafx.h"
#include <iostream>
union bit32
{
    short s : 8;
    char c[4];
    int i;
};

int _tmain(int argc, _TCHAR* argv[])
{
    int x = 0x01020304;
    bit32 *pbit32 = (bit32*)&x;
}
```

Memory dump (Память 2) showing the address `0x0066FA80` and its contents:

Адрес:	0x0066FA80	04 03 02 01	cc cc cc cc	2d bd ca f9	dc fa 66 00
0x0066FA90	09 5c 2a 01	01 00 00 00	60 98 7c 00	60 ce 7c 00	
0x0066FAA0	7d bd ca f9	00 00 00 00	00 00 00 00	b0 bd 7e	
0x0066FAB0	80 f8 ff ff	c9 78 cb 1f	00 00 00 00	00 00 67 00	
0x0066FAC0	00 00 00 00	a0 fa 66 00	00 00 00 00	24 fb 66 00	
0x0066FAD0	04 11 2a 01	39 af 86 f8	00 00 00 00	e4 fa 66 00	

Control values (Контрольные значения 1) table:

Имя	Значение	Тип
pbit32->i	0x01020304	int
pbit32->c[0]	0x04 '\x4'	char
pbit32->c[1]	0x03 '\x3'	char
pbit32->c[2]	0x02 '\x2'	char
pbit32->c[3]	0x01 '\x1'	char
pbit32->s	0x0004	short
&x	0x0066fa80 {0x01020304}	int *
pbit32	0x0066fa80 {s=0x0004 c=0x0066fa80 "\x4\x3\x2\x1... i=0x01020304 }	bit32 *

## 23. Пользовательские типы C++: typedef.

С помощью ключевого слова **typedef** можно определить новое имя типа данных. Новый тип при этом не *создается*, уже существующий тип получает новое имя.

The screenshot shows a C++ program using `typedef` to create new type names for `unsigned int`, `unsigned short`, and `wchar_t`. These types are then used to declare variables in the `_tmain` function.

```
#include "stdafx.h"
#include <locale>
#include <iostream>

typedef unsigned int    uint32;
typedef unsigned short uint16;
typedef wchar_t        unicode;

int _tmain(int argc, _TCHAR* argv[])
{
    uint32 k = 15;
    uint16 m = 12;
    unicode s[] = L"Привет мир";

    return 0;
}
```



## 24. Пользовательские типы C++: class

**Объектно-ориентированное программирование (ООП)** – *методология* программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса (типа особого вида), а классы образуют иерархию наследования. (Гради Буч)

В центре ООП находится понятие *объекта*.

**Объект** – это сущность, которой можно посылать сообщения и которая может на них реагировать, используя свои данные. Объект – это экземпляр класса. Данные объекта скрыты от остальной программы. Скрытие данных называется инкапсуляцией.

### **Абстракция данных**

Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой.

В ООП рассматривают абстракцию данных, подразумевая набор значимых характеристик объекта, доступный остальной программе.

### **Инкапсуляция**

Инкапсуляция – свойство системы, позволяющее объединить в классе данные и методы, работающие с ними.

Одни языки (например, C++, Java или Ruby) отождествляют инкапсуляцию с сокрытием, а другие (Smalltalk, Eiffel, OCaml) различают эти понятия.

### **Наследование**

Наследование – свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствуемой функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом.

Новый класс – потомком, наследником, дочерним или производным классом.

### **Полиморфизм подтипов**

Полиморфизм подтипов – свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Другой вид полиморфизма – параметрический – в ООП называют обобщенным программированием.

### **Класс**

Класс – в ООП, представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний (инициализация полей-данных и реализация поведения функций или методов).

Классы разрабатываются для обеспечения соответствия природе объекта и решаемой задаче, целостности данных объекта, удобного и простого интерфейса.