

БГТУ, ФИТ, ПОИТ, 2 семестр,

Конструирование программного обеспечения

Структура языка программирования

План лекции:

Структура языка программирования

- ✓ **фундаментальные (встроенные) типы данных:**
 - указатели;
- ✓ **пользовательские типы данных**
 - типы, которые может создавать пользователь на основе фундаментальных типов (возможно описание их свойств и поведение);
 - массивы фундаментальных типов;
 - enum;
 - struct;
 - union;
 - typedef;
 - class (спецификатор типа).
- ✓ **инициализация памяти:** присвоение значения в момент объявления переменной;
- ✓ **область видимости переменных:** доступность переменных по их идентификатору в разных частях программы; пространства имен.

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 ЯП обычно существует нулевой указатель, который показывает, что эта **переменная-указатель** не ссылается (не указывает) ни на какой объект.
- в C++ – 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.

В C++ 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:

```
whar t      wc = L'U';    // 0x0426
```

[illegible]

! Пример. Логическая ошибка:

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

The screenshot shows a C++ program with several variables declared at the top:

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

Below these are pointers initialized to point to themselves:

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

A separate box contains more pointer assignments:

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

At the bottom, there's a section titled "Контрольные значения 1" (Control values 1) which lists various memory addresses and their corresponding values, likely representing the state of memory after some operations.

[illegible]

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

```
char      c = 'A';
wchar_t   wc = L'Q';
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 + 2;  
wchar_t *wcx = pwc + 2;  
int *kx = pk + 3;  
float *fx = pf + 3;  
long double *ldx = pld + 3;
```

```
int l1 = sizeof(char*);  
int l2 = sizeof(wchar_t*);
```

```
return 0;
```

Контрольные значения 1

Поиск (Ctrl+E) 🔍 ← → Глубина поиска: 3 ▾

Имя	Значение
c	65 'A'
wc	1062 'Ц'
pc	0x0033fb6b "AMMMMt;\af"ы3"
cx	0x0033fb6d "MMMt;\af"ы3"
pwc	0x0033fb5c L"L책책책책책책책책책책艾昇3ꠖ\x1
wcx	0x0033fb60 L"L책책책책책책책책책책艾昇3ꠖ\x1"
pk	0x0033fb50 {5}
kx	0x0033fb5c {-859044826}
pf	0x0033fb44 {1.50000000}
fx	0x0033fb50 {7.006e-45#DEN}
pld	0x0033fb34 {0.0023000000000000000}
ldx	0x0033fb4c {1.230757558720e-313#DEN}
I1	4
I2	4

Память 2

Адрес: 0x0033FAA4

0x0033FAA4	04 00 00 00	cc cc cc cc cc cc cc cc
0x0033FAB0	04 00 00 00	cc cc cc cc cc cc cc cc
0x0033FABC	4c fb 33 00	cc cc cc cc cc cc cc cc
0x0033FAC8	34 fb 33 00	cc cc cc cc cc cc cc cc
0x0033FAD4	50 fb 33 00	cc cc cc cc cc cc cc cc
0x0033FAE0	44 fb 33 00	cc cc cc cc cc cc cc cc
0x0033FAEC	5c fb 33 00	cc cc cc cc cc cc cc cc
0x0033FAF8	50 fb 33 00	cc cc cc cc cc cc cc cc
0x0033FB04	60 fb 33 00	cc cc cc cc cc cc cc cc
0x0033FB10	5c fb 33 00	cc cc cc cc cc cc cc cc
0x0033FB1C	6d fb 33 00	cc cc cc cc cc cc cc cc
0x0033FB28	6b fb 33 00	cc cc cc cc cc cc cc cc
0x0033FB34	48 50 fc 18 73 d7 62 3f	cc cc cc cc cc cc cc cc
0x0033FB40	00 00 c0 3f	cc cc cc cc cc cc cc cc
0x0033FB4C	05 00 00 00	cc cc cc cc cc cc cc cc
0x0033FB58	26 04 cc cc	cc cc cc cc cc cc cc cc
0x0033FB64	41	cc cc cc cc cc cc cc cc

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.

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

- Code Editor:** Contains the following code:

```
#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;
}
```

Annotations include a red box around the variable declarations and a blue box around the void pointer declarations and the casted assignment.
- Memory Dump (Память 2):** Shows a hex dump starting at address 0x003AFD6C. A blue arrow points from the memory address 0x003AFD9C (containing '9c fd 3a 00') to the variable `&k` in the code editor. Another blue arrow points from the memory address 0x003AFDA4 (containing 'cc cc cc 41') to the variable `(int*)z` in the code editor.
- Control Values (Контрольные значения 1):** A table showing the memory addresses and values of the variables:

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

с. Указатели на функции

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

The screenshot shows a C++ IDE with the following components:

- Code Editor:** Contains a C++ program. The function pointer declaration `int(*f) (int, int);` is highlighted with a red box. The assignment `f = sum;` is highlighted with a blue box. The function calls `f(2, 3);` are also highlighted with blue boxes.
- Память 1 (Memory 1):** Shows the address `0x00BB8134` pointing to `&f`. The memory dump shows the address stored at `0x00BB8134` is `5f 10 bb 00`, which corresponds to the address `0x00BB810F` (the address of `f`).
- Память 2 (Memory 2):** Shows the address `0x00BB105F` pointing to `f`. The memory dump shows the address stored at `0x00BB105F` is `e9 9c 03 00`, which corresponds to the address `0x00BB810F`.
- Контрольные значения 1 (Control Values 1):** A table showing the values of `f` and `&f`.

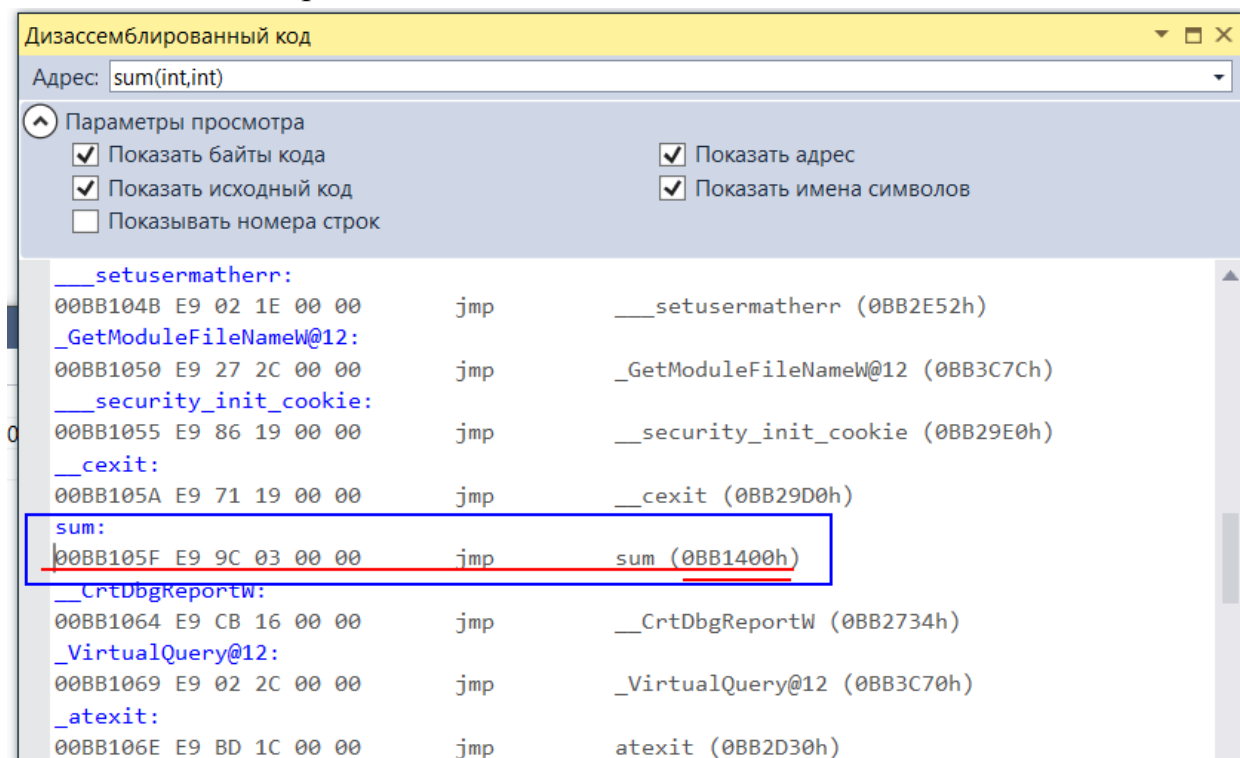
Имя	Значение	Тип
f	0x00bb105f {указатель на функцию.exe!sum(int,int)}	int (int, int) *
&f	0x00bb8134 {указатель на функцию.exe!int(*f)(int, int)} {0x00bb105f {указатель на функцию.exe!sum(int,int)}}	int (int, int) **

Открываем окно дизассемблированного кода нажатием правой кнопки на указателе f:

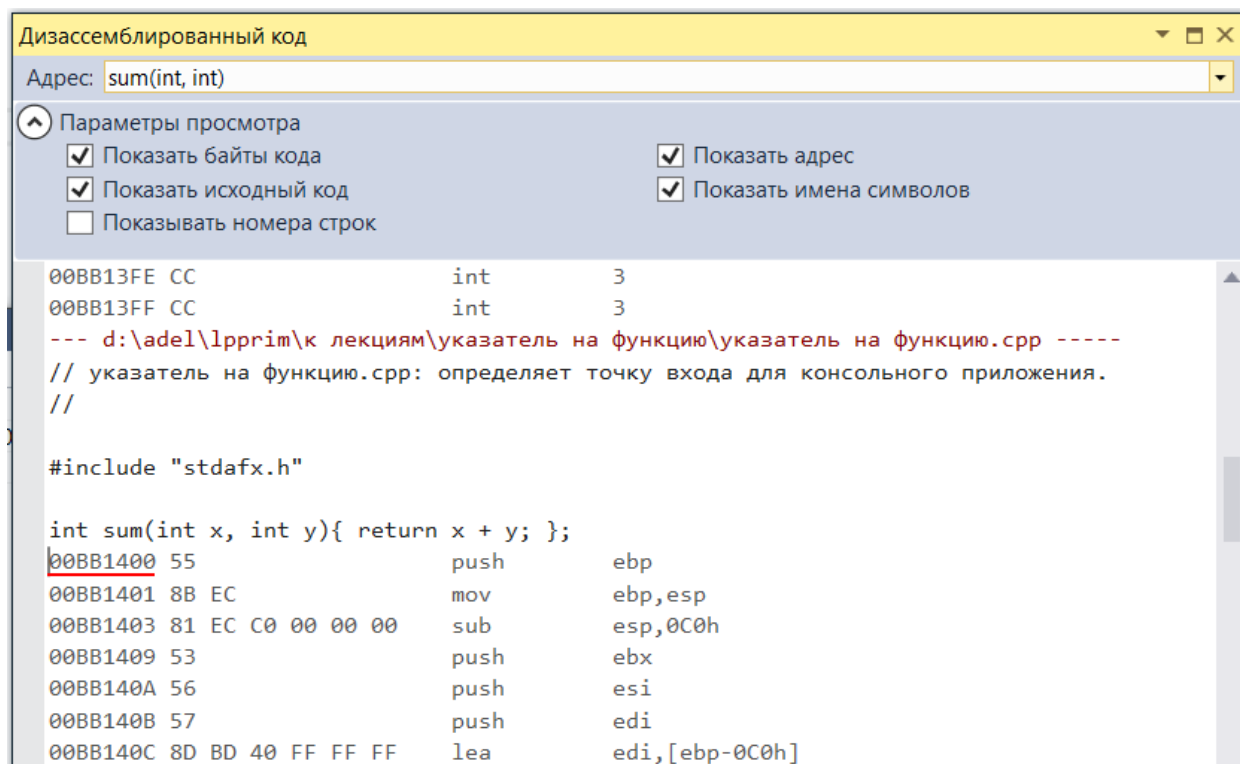
The screenshot shows the 'Контрольные значения 1' (Control Values 1) window. A right-click context menu is open over the variable `f`. The menu options are:

- Копировать (Ctrl+C)
- Вставить (Ctrl+V)
- Изменить значение
- Добавить контрольное значение
- Добавить параллельное контрольное значение
- Удалить контрольное значение
- Выделить все (Ctrl+A)
- Очистить все
- Шестнадцатеричный вывод
- Свернуть родительский элемент
- К исходному коду
- К дизассемблированному коду

Окно дизассемблированного кода:



Выполняем шаг с заходом в функцию:



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]

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

Размер одномерного массива в байтах вычисляется по формуле:

$$\text{количество_байтов} = \text{sizeof(базовый_тип)} * \text{количество_элементов}$$

```
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[] = "1234abcdцчщюя";
```

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

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

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

Контрольные значения 1

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

Память 2

Адрес: 0x00E1FBEC

0x00E1FBEC	0e 00 00 00	cc cc cc ccMMMM
0x00E1FBF4	cc cc cc cc	0d 00 00 00	MMMM....
0x00E1FBFC	cc cc cc cc	cc cc cc cc	MMMMMMMM
0x00E1FC04	05 00 00 00	cc cc cc ccMMMM
0x00E1FC0C	cc cc cc cc	05 00 00 00	MMMM....
0x00E1FC14	cc cc cc cc	cc cc cc cc	MMMMMMMM
0x00E1FC1C	31 32 33 34	61 62 63 64	1234abcd
0x00E1FC24	f6 f7 f9 fe ff	00 cc cc cc	цчщюя.MM
0x00E1FC2C	cc cc cc cc	cc cc cc cc	MMMMMMMM
0x00E1FC34	31 32 33 34	61 62 63 64	1234abcd
0x00E1FC3C	f6 f7 f9 fe ff	cc cc cc cc	цчщюяMMMM
0x00E1FC44	cc cc cc cc	cc cc cc cc	MMMMMMMM
0x00E1FC4C	01 00 01 00 01	cc cc cc ccMMMM
0x00E1FC54	cc cc cc cc	cc cc cc cc	MMMMMMMM
0x00E1FC5C	01 00 01 00 01	cc cc cc ccMMMM
0x00E1FC64	cc cc cc cc	cc cc cc cc	MMMMMMMM

```

1 // LP_Lab06.cpp: определяет точку входа для консольного приложения.
2 //
3 #include "stdafx.h"
4 int _tmain(int argc, _TCHAR* argv[])
5 {
6     wchar_t Lc1[] = {L'1', L'2', L'3'};
7     wchar_t Lc2[] = L"1234abcdцщюя";
8
9     int lLc1 = sizeof(Lc1);
10    int lLc2 = sizeof(Lc2);
11
12
13    return 0;
14 }
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	...pЕ~Ъ
0x00C9FAC1	f8 ff ff c9 e7 7c 1a	шяяЙз .
0x00C9FAC8	00 00 00 00 00 00 caK
0x00C9FACF	00 00 00 00 00 b0 fa°ь
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...

больные значения 1

я	Значение
lLc1	26
lLc2	28
Lc1	0x00c9fa7c L"1234abcdцщюя..."
Lc2	0x00c9fa58 L"1234abcdцщюя"

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    short s[] = { 1, 2, 3, 4, 5, 6, 7 };
    int i[] = { 1, 2, 3, 4, 5, 6, 7 };

    float f[] = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0 };
    double d[] = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0 };

    int ls = sizeof(s);
    int li = sizeof(i);
    int lf = sizeof(f);
    int ld = sizeof(d);

    return 0;
}
```

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]

Память 2

Адрес: 0x008EF6A4

Адрес	Данные	Символы
0x008EF6A4	00 00 00 00 00 00 f0 3fp?
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 ccMM
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.АСП.
0x008EF754	88 42 8c b4 00 00 00 00	€ВЪг....
0x008EF75C	00 00 00 00 00 e0 31 7ea1~
0x008EF764	80 f8 ff ff c9 18 c6 1c	ЪшяЯЙ.Ж.

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

Адрес	Данные	Символы
0x0035FC5C	80 fc 35 00 84 fc 35 00	ЪЪ5..Ъ5.
0x0035FC64	88 fc 35 00 8c fc 35 00	€Ъ5.ЪЪ5.
0x0035FC6C	90 fc 35 00 94 fc 35 00	ђЪ5."Ъ5.
0x0035FC74	98 fc 35 00 cc cc cc cc	..Ъ5.ММММ
0x0035FC7C	cc cc cc cc 01 00 00 00	ММММ....
0x0035FC84	02 00 00 00 03 00 00 00
0x0035FC8C	04 00 00 00 05 00 00 00
0x0035FC94	06 00 00 00 07 00 00 00
0x0035FC9C	cc cc cc cc 18 49 99 45	ММММ.И™Е

Массив указателей на функции:

The screenshot displays a C program and its memory dump. The program defines four functions: `f1`, `f2`, `f3`, and `f4`, each returning a `char`. In the `_tmain` function, an array `f` of type `char(*)` is initialized with the addresses of these functions. The memory dump window, titled "Память 2", shows the memory layout. The array `f` is located at address `0x00A2F730` and contains four pointers: `64 10 23 01` (pointing to `f1` at `0x00A2F724`), `a5 10 23 01` (pointing to `f2` at `0x00A2F728`), `a0 10 23 01` (pointing to `f3` at `0x00A2F72C`), and `14 10 23 01` (pointing to `f4` at `0x00A2F730`). The "Контрольные значения 1" window shows the variable `f` with its value `0x00a2f730 {0x01231064 {LP_Lab06.exelf1(char)}, 0x012310a5 {LP_Lab06.exelf2(char)}, C char (cha`.

```
#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;
}
```

Память 2

Адрес	Данные	Комментарий
0x00A2F724	10 00 00 00
0x00A2F728	cc cc cc cc	MMMM
0x00A2F72C	cc cc cc cc	MMMM
0x00A2F730	64 10 23 01	d.#.
0x00A2F734	a5 10 23 01	Г.#.
0x00A2F738	a0 10 23 01	.#.
0x00A2F73C	14 10 23 01	..#.
0x00A2F740	cc cc cc cc	MMMM
0x00A2F744	94 f7 a2 00	"чў.
0x00A2F748	79 1a 23 01	y.#.
0x00A2F74C	01 00 00 00	

Контрольные значения 1

Имя	Значение	Тип
lf	16	int
f	0x00a2f730 {0x01231064 {LP_Lab06.exelf1(char)}, 0x012310a5 {LP_Lab06.exelf2(char)}, C char (cha	

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

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 (не выделяется память).

19. Пользовательские типы C/C++:

enum;
struct;
union;
typedef;
class (спецификатор типа).

20. 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;
}
```

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 *

Память 2

Адрес	Содержимое	Символ
0x00F9FDE4	fe ff ff ff	юяяя
0x00F9FDE8	cc cc cc cc	ММММ
0x00F9FDEC	cc cc cc cc	ММММ
0x00F9FDF0	02 00 00 00
0x00F9FDF4	cc cc cc cc	ММММ
0x00F9FDF8	cc cc cc cc	ММММ
0x00F9FDFC	1c 00 00 00
0x00F9FE00	cc cc cc cc	ММММ
0x00F9FE04	cc cc cc cc	ММММ
0x00F9FE08	01 00 00 00
0x00F9FE0C	cc cc cc cc	ММММ
0x00F9FE10	cc cc cc cc	ММММ
0x00F9FE14	fe ff ff ff	юяяя
0x00F9FE18	cc cc cc cc	ММММ
0x00F9FE1C	cc cc cc cc	ММММ
0x00F9FE20	00 00 00 00
0x00F9FE24	cc cc cc cc	ММММ

Оператор sizeof - это оператор языка, применяемый для определения размера типа данных в байтах.

```
#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;
}

```

21. Пользовательские типы C++: struct.

Структура обеспечивает удобный способ организации взаимосвязанных данных.

```

3  #include "stdafx.h"
4
5
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

```

3  #include "stdafx.h"
4
5
6  struct MyStruct
7  {
8      char c;
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;

```

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

77 1

&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
17 int _tmain(int argc, _TCHAR* argv[])
18 {
19     MyStruct mystruct = {'a', 'b', 'c', 77777, 77, 2.8E-5};
20
21     int size = sizeof(MyStruct);
22
23     return 0;
24 }

```

ольные значения 1

	Значение
size	0x00000010
&mystruct	0x0094fb10 {c=0x61 'a' c1=0x62 'b' wc=0xffff6 '□' ...}

```

15
16 int _tmain(int argc, _TCHAR* argv[])
17 {
18     MyStruct mystruct = {'a', 'b', 'c', 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

```

ольные значения 1

	Значение
&mystruct	0x00eaf80c {c=0x61 'a' c1=0x62 'b' wc=0xffff6 '□' ...}
&ms2	0x00eaf7f4 {c=0x61 'a' c1=0x62 'b' wc=0xffff6 '□' ...}

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	
pms1	0x0035fbd8	c=0xcc 'M' c1=0xcc 'M' wc=0xcccc 'M' .

```
#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

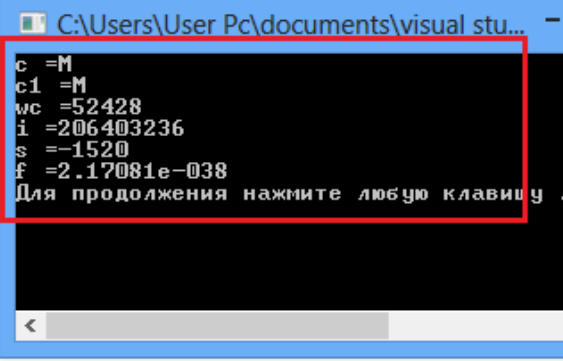
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus"); // настройка кодировки

    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 ;

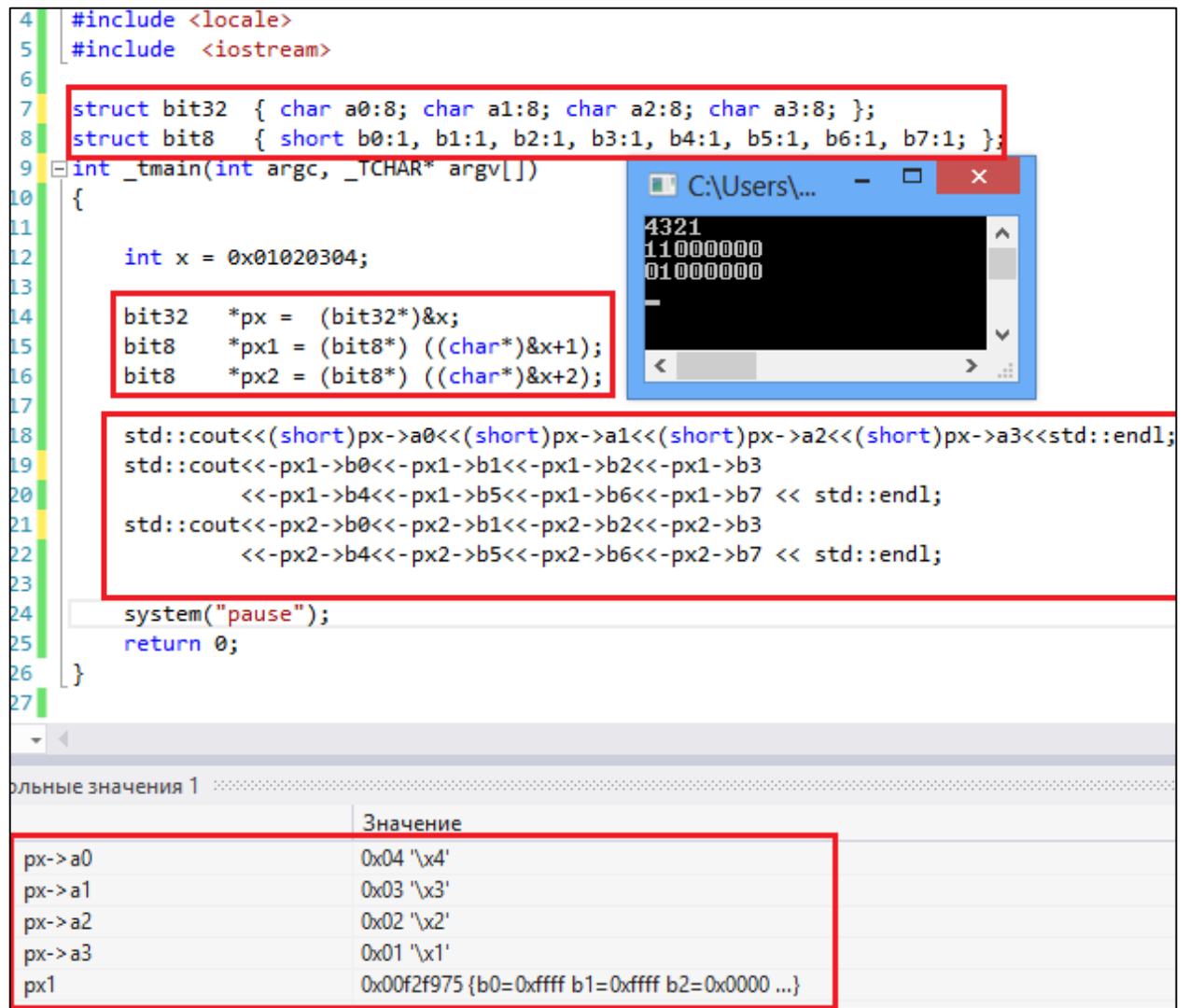
    system("pause");
    return 0;
}
```



```
c =M
c1 =M
wc =52428
i =206403236
s =-1520
f =2.17081e-038
Для продолжения нажмите любую клавишу .
```

22. 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
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
18     std::cout<<(short)px->a0<<(short)px->a1<<(short)px->a2<<(short)px->a3<<std::endl;
19     std::cout<<-px1->b0<<-px1->b1<<-px1->b2<<-px1->b3
20         <<-px1->b4<<-px1->b5<<-px1->b6<<-px1->b7 << std::endl;
21     std::cout<<-px2->b0<<-px2->b1<<-px2->b2<<-px2->b3
22         <<-px2->b4<<-px2->b5<<-px2->b6<<-px2->b7 << std::endl;
23
24     system("pause");
25     return 0;
26 }

```

Console output:

```

4321
11000000
01000000

```

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	0xffffffff {b0=??? b1=??? b2=??? ...}
px1->b0	0xffff
-px1->b0	0x00000001

Память 2

Адрес: 0x0103FB79

Адрес	Данные
0x0103FB79	03 02 01 cc cc cc cc 95 86
0x0103FB91	89 4f 01 10 0b 50 01 01 00
0x0103FBA9	24 36 00 11 81 c8 71 5e 11


```
#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 ...ММММ гё"рьй
Память 4
Адрес: 0x00E9FC15 px1
0x00E9FC15 03 02 01 cc cc cc cc 7c 81 b8 99 70 fc e9 00 ...ММММ гё"рьй.
Память 1
Адрес: 0x00E9FC16 px2
0x00E9FC16 02 01 cc cc cc cc 7c 81 b8 99 70 fc e9 00 09 ..ММММ 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 *

```
(любая 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 ММММ

```

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	MMMM

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

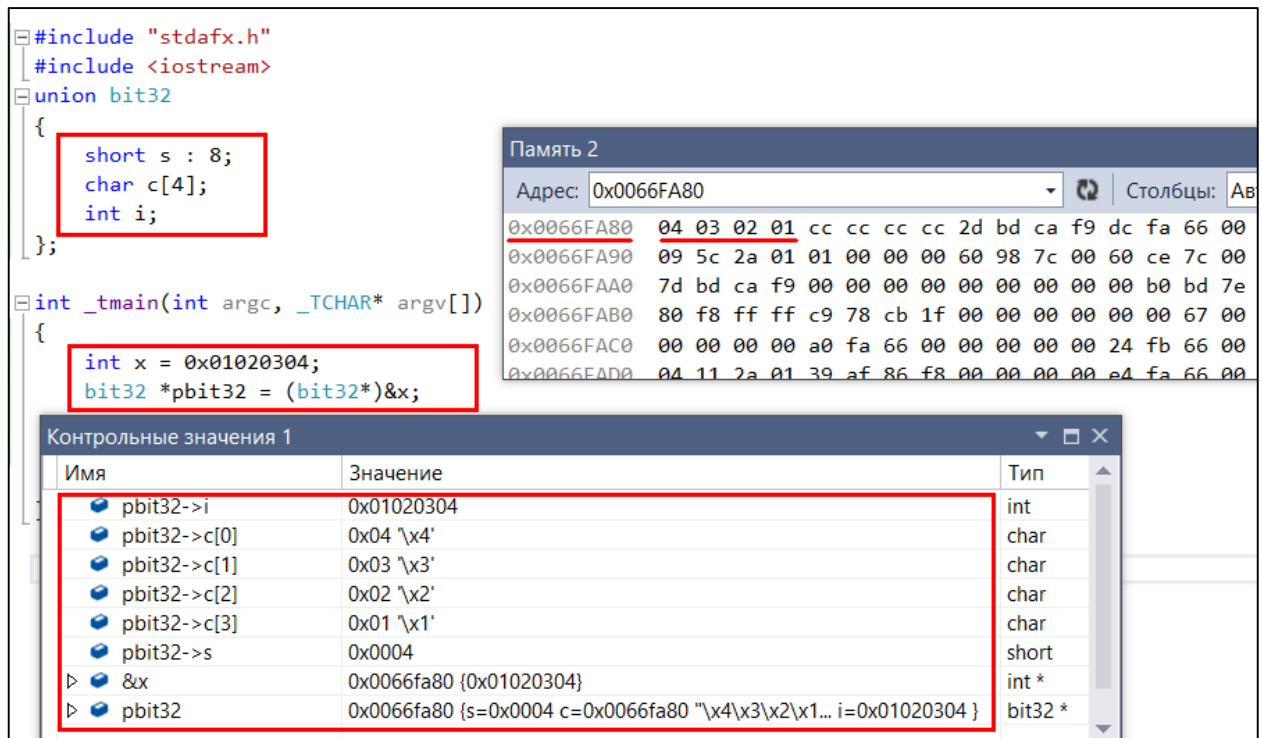
```

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

23. Пользовательские типы C++: union.

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



```
#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;
}
```

Память 2

Адрес:	0x0066FA80	Столбцы:	Ав
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 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		

Контрольные значения 1

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 *

24. 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.

```
#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;
}
```

25. Пользовательские типы C++: class (объектно-ориентированное программирование).

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. Соккрытие данных называется инкапсуляцией.

Абстракция данных

Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой.

В ООП рассматривают лишь 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 соккрытием, а другие (Smalltalk, Eiffel, OCaml) различают эти понятия.

Наследование

Наследование – свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся 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.