

Структура языка программирования. Программные конструкции.

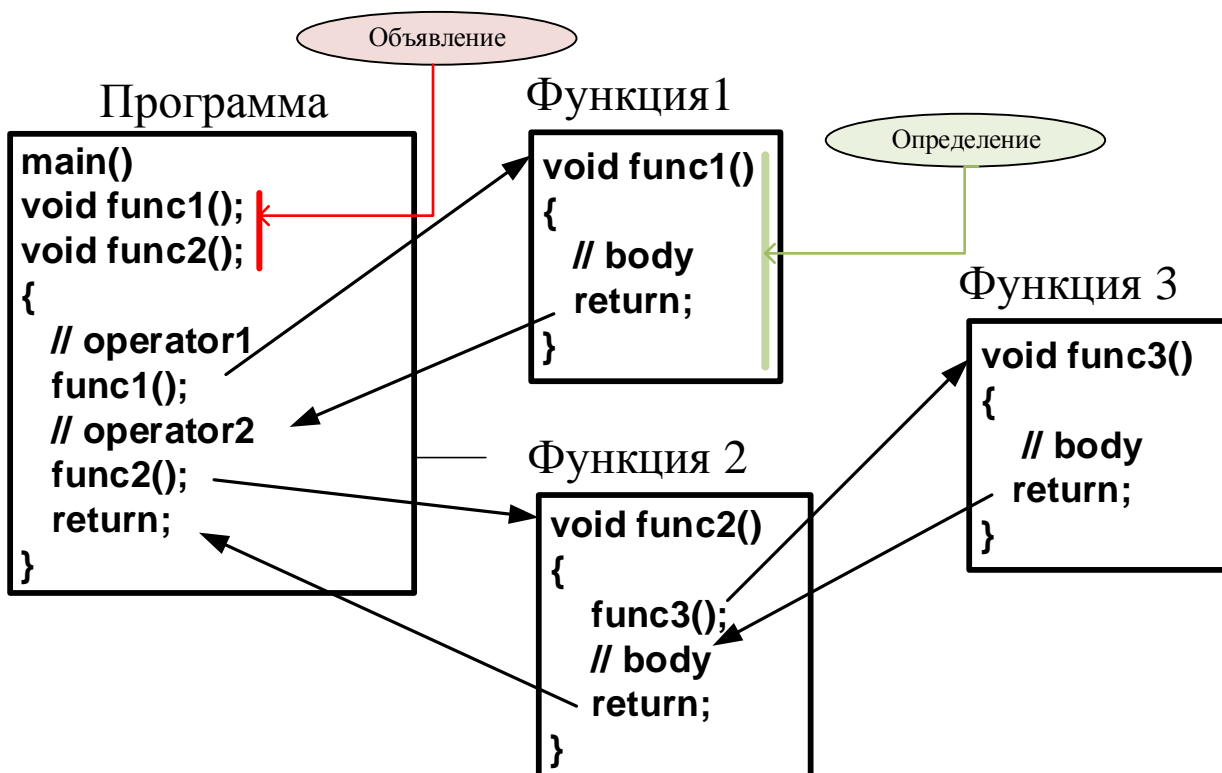
План лекции:

- программные конструкции;
- соглашения о вызовах;
- программные блоки;
- передача параметров в функции;
- функции с переменным числом параметров;
- перегрузка функций;
- шаблоны функций.

1. Программные конструкции:

- функции;
- объявление функции;
- определение;
- параметры.

Функция – фрагмент программного кода, к которому можно обратиться из другого места программы.



С функцией связывается **идентификатор** (имя функции). Некоторые языки программирования допускают безымянные функции.

С **именем функции** неразрывно связан **адрес ее первой инструкции**, которой передаётся управление при обращении к функции.

После выполнения функции управление возвращается в точку программы, где данная функция была вызвана (**адрес возврата**).

Функция может принимать параметры и может возвращать некоторое значение.

Функции, которые возвращают **пустое** значение, называют **процедурами**.

Пример анонимной (безымянной) функции C#:

$(x, y) \Rightarrow x + y$

Функция должна быть **объявлена** и **определена**.

Объявление функции содержит:

- имя функции;
- список имён и типов передаваемых параметров (или аргументов);
- тип возвращаемого функцией значения.

Определение функции содержит исполняемый код функции.

В одних языках программирования определение следует **непосредственно за объявлением** функции. В других языках необходимо сначала объявить функцию, а потом привести её определение.

Для **вызова** функции необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в функцию **параметры**.

Передача параметров в функцию:

- по значению;
- по ссылке.

Для переменной, переданной по **значению**, создаётся локальная копия и любые изменения, которые происходят в теле функции, происходят с локальной копией и **не изменяют** значения самой переменной.

Для переменной, переданной по **ссылке**, изменения происходят с самой переменной.

Функция определяет **локальную** (собственную) область видимости, куда входят входные параметры, а также те переменные, которые объявляются непосредственно в теле самой функции.

Функция – подпрограмма, выполняющая какие-либо операции и возвращающая значение.

Процедура – подпрограмма, которая выполняет операции, и не возвращает значения.

Метод – это функция или процедура, которая принадлежит классу или экземпляру класса.

В различных языках программирования объявления функций и процедур имеют различный синтаксис, могут использоваться различные ключевые слова:

| C# | Pascal | PHP |
|---|--|---|
| <pre>public void Name(string text) { System.Console.WriteLine(text); }</pre> | <pre>procedure name(var text: string) begin write(text); end;</pre> | <pre>function name(\$text) { echo \$text; }</pre> |
| Standard ML | Visual Basic | Python |
| <pre>fun name t = print t</pre> <p><i>или, что то же самое через лямбда-функцию:</i></p> <pre>val name = fn t => print t</pre> | <pre>Sub Name(text) Console.WriteLine(text) End Sub</pre> | <pre>def func(p): print(p)</pre> |
| JavaScript | PureBasic | C++ |
| <pre>function max (a, b) { return a >= b ? a : b; }</pre> | <pre>Procedure Name(text.s) PrintN(text) EndProcedure</pre> | <pre>void name(char* text) { std::cout<<text<< std::endl; }</pre> |
| GO | Java | Swift |
| <pre>func add (x, y int) (int){ var z = x + y fmt.Println("x + y = ", z) return z }</pre> | <pre>public void name(String text) { System.out.println(text); }</pre> | <pre>func printText(text: String) { print(text) }</pre> |

2. Программные конструкции C++:

- функции;
- вызов функции;
- стек и соглашения о вызовах.

Функция – подпрограмма, выполняющая какие-либо операции и возвращающая значение.

В C++11 параметр **auto** является допустимым типом возвращаемого значения, который позволяет компилятору *вывести* тип исходя из типа выражения оператора `return`.

Необязательные элементы объявления функции:

| | | |
|---|---|--|
| <code>constexpr</code> | — | возвращаемое значение функции является константой (определено во время компиляции) |
| <code>extern</code> или <code>static</code> | — | спецификация компоновки |
| <code>inline</code> | — | указывает компилятору на необходимость замены каждого вызова функции на саму функцию |

Примеры

1.

```
constexpr float exp(float x, int n)
{
    return n == 0 ? 1 : n * x;
};
```

2.

```
// декларация (объявление) func с C-подобным связыванием.
extern "C" int func( const char *fmt, ... );
```

3.

```
inline int sum(int a, int b)
{
    return a + b;
}
```

Соглашение о вызовах – это протокол взаимодействия вызывающей и вызываемой функций.

Механизм вызова функций на примере языка программирования C++ (компиляторы Microsoft, Visual Studio, x86).

Что такое функция в C++?

У функции есть **тело**, которое содержит код, исполняемый функцией.

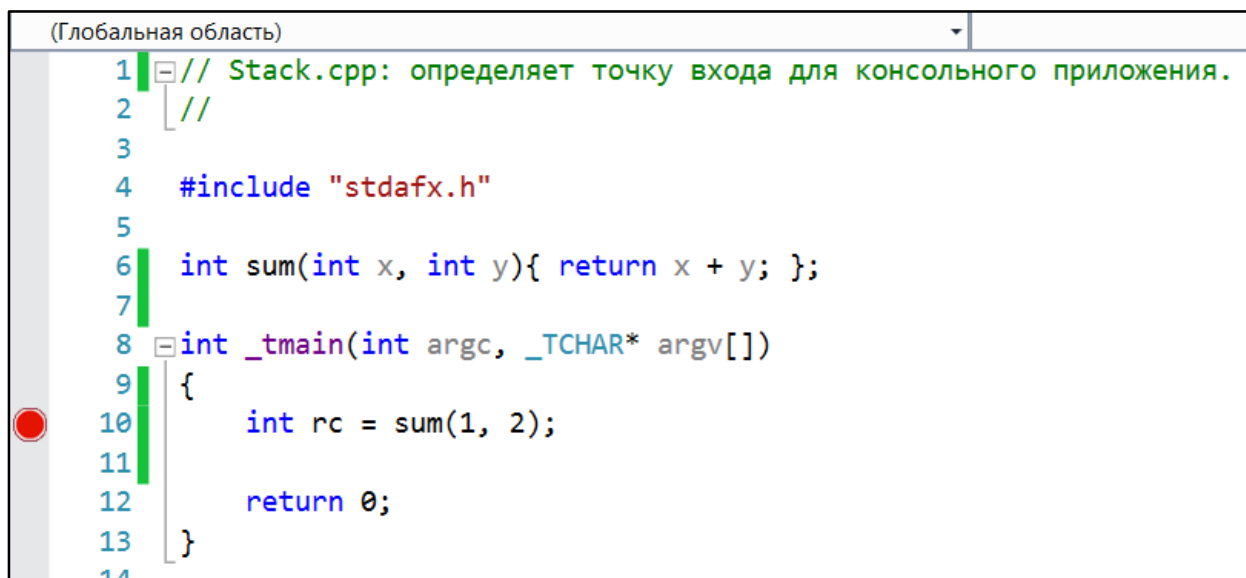
Функция может иметь **параметры** (аргументы), а может не иметь их (список параметров пуст).

Функция может **возвращать** некоторое значение или не возвращать ничего.

При **вызове** функции выполняется:

- определенный **способ передачи параметров** в функцию;
- порядок размещения параметров в стеке;
- очищение стека;
- порядок возврата вычисленного значения;
- конкретная инструкция для вызова.

Механизм вызова функций называется **"Соглашение о вызовах (Calling convention)"** – это протокол, в соответствии с которым вызывающий код и вызываемая функция согласны общаться друг с другом.



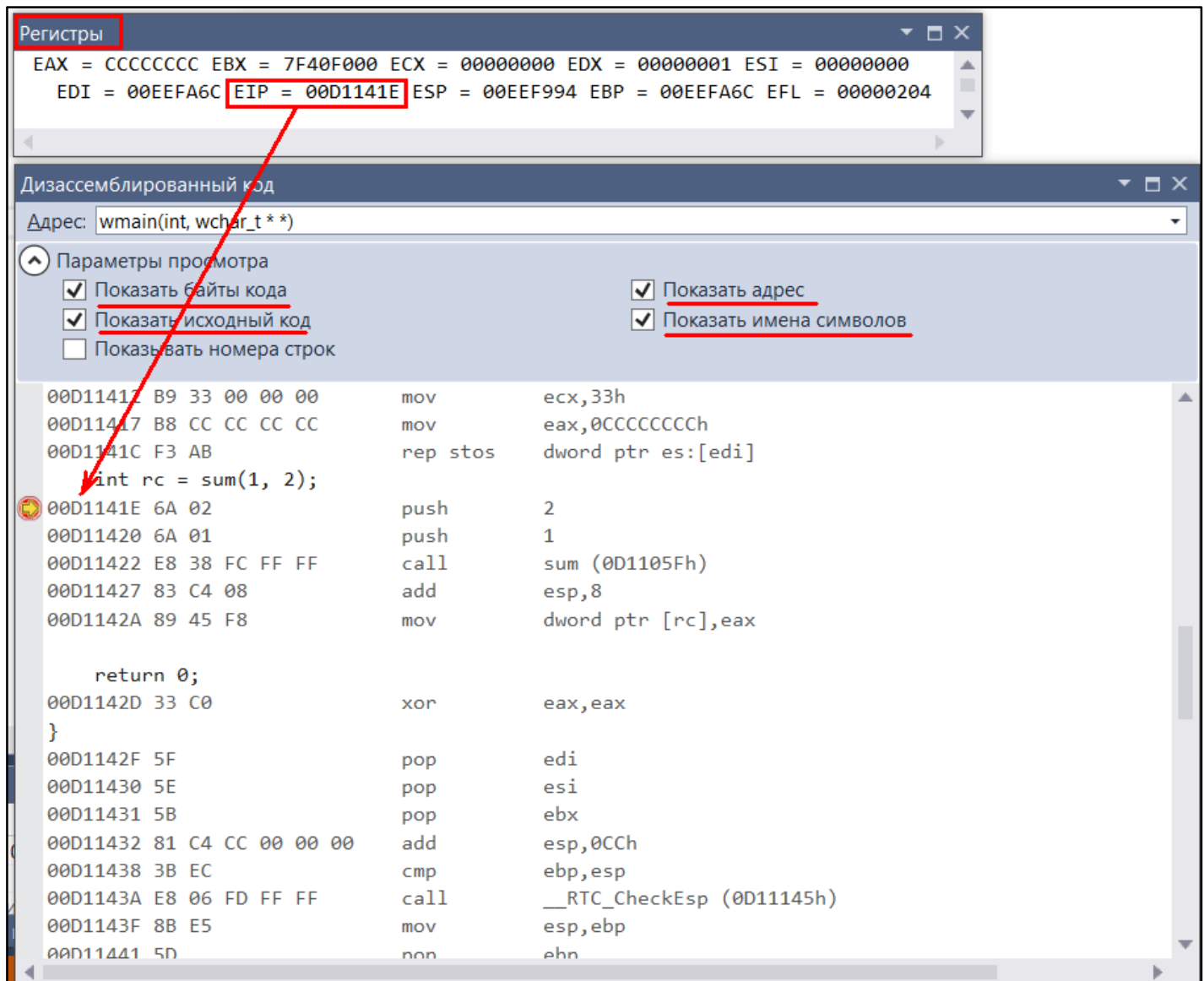
```
(Глобальная область)
1 // Stack.cpp: определяет точку входа для консольного приложения.
2 //
3
4 #include "stdafx.h"
5
6 int sum(int x, int y){ return x + y; };
7
8 int _tmain(int argc, _TCHAR* argv[])
9 {
10     int rc = sum(1, 2);
11
12     return 0;
13 }
14
```

Последовательность действий:

- 1). Запускаем программу в режиме отладки (F5). Выполнение программы остановится на 10-ой строке.
- 2). Открываем окно отладчика **Регистры**, отображающее содержимое регистров (в контекстном меню окна выбираем ЦП).
- 3). Открываем окно отладчика **Память**.
- 4). Устанавливаем курсор на строку 10 и вызываем в контекстном меню **Дизассемблированный код**.

В окне дизассемблированного кода должны быть отмечены следующие чекбоксы:

- ✓ Показать байты кода
- ✓ Показать исходный код
- ✓ Показать адрес
- ✓ Показать имена символов



Регистр EIP – указатель на **инструкцию**, которая должна быть выполнена процессором. Содержимое регистра **EIP** нельзя изменять явно. Он **обновляется** сам в следующих ситуациях:

1. Процессор закончил выполнение инструкции. Длина инструкции известна – это **определенное** количество байт выполняемого кода. Процессор знает, сколько байт занимает инструкция и, соответственно, сдвигает указатель на нужное количество байт после ее выполнения.

2. Выполнена инструкция **ret** – возврат.

3. Выполнена инструкция **call** – вызов.

5). Делаем шаг отладки (F10), чтобы выполнить данную инструкцию и перейти к следующей. Значение регистра EIP автоматически увеличилось на 2, так как инструкция использовала ровно 2 байта машинного кода (байты **6A 02** по адресу **0x00D1141E**). Значение регистра EIP изменяется автоматически:

The screenshot shows a debugger window with two main panes. The top pane, titled 'Регистры' (Registers), displays the current state of the CPU registers. The bottom pane, titled 'Дизассемблированный код' (Disassembled code), shows the instructions being executed.

Регистры (Registers):

- EAX = CCCCCCCC
- EBX = 7F40B000
- ECX = 00000000
- EDX = 00000001
- ESI = 00000000
- EDI = 00F8FB6C
- EIP = 00D11420**
- ESP = 00F8FA90**
- EBP = 00F8FB6C
- EFL = 00000206

Дизассемблированный код (Disassembled code):

Адрес: wmain(int, wchar_t **)

Параметры просмотра (View options):

- ☒ Показывать байты кода (Show code bytes)
- ☒ Показывать исходный код (Show source code)
- ☐ Показывать номера строк (Show line numbers)
- ☒ Показывать адрес (Show address)
- ☒ Показывать имена символов (Show symbol names)

Код (Code):

| Адрес (Address) | Байты кода (Code bytes) | Инструкция (Instruction) | Комментарий (Comment) |
|---------------------|-------------------------|--------------------------|-----------------------|
| 00D11401 | 8B EC | mov | ebp, esp |
| 00D11403 | 81 EC CC 00 00 00 | sub | esp, 0CCh |
| 00D11409 | 53 | push | ebx |
| 00D1140A | 56 | push | esi |
| 00D1140B | 57 | push | edi |
| 00D1140C | 8D BD 34 FF FF FF | lea | edi, [ebp-0CCh] |
| 00D11412 | B9 33 00 00 00 | mov | ecx, 33h |
| 00D11417 | B8 CC CC CC CC | mov | eax, 0CCCCCCCCh |
| 00D1141C | F3 AB | rep stos | dword ptr es:[edi] |
| int rc = sum(1, 2); | | | |
| 00D1141F | 6A 02 | push | 2 |
| 00D11420 | 6A 01 | push | 1 |
| 00D11422 | E8 38 FC FF FF | call | sum (0D1105Fh) |
| 00D11427 | 83 C4 08 | add | esp, 8 |

6). Нажимаем F10 еще один раз, и переходим к строке с адресом **0x00D11422**. Проверяем значение регистра **EIP**, оно опять увеличилось на 2.

Следующая строка кода (инструкция **call**) – это вызов функции **sum**. Эта инструкция переносит поток выполнения по указанному адресу. В коде, приведенном на рисунке выше, это адрес **0x0D1105Fh**. Обращаем внимание на адрес инструкции, следующей за **call**. В нашем примере это адрес **0x00D11427**. Сюда поток должен вернуться сразу после выполнения кода вызываемой функции, на который указывала инструкция **call**. Это адрес точки возврата.

7). Выполнение инструкции **call** (F11 - шаг с заходом) передаст управление в функцию **sum**. При этом значение EIP изменится на **0x0D113C0**.

Теперь поток выполнения находится внутри функции **sum**.

8). В окне памяти отладчика в поле для ввода «Адрес» вводим имя регистра: **ESP**. Содержимое памяти по адресу, хранящемуся в регистре ESP (в вершине стека) равно **0x00D11427** – это адрес точки возврата, т.е. адрес инструкции, следующей за командой **call**.

Регистр ESP.

Любые локальные переменные создаются в стеке. **Стек** – это область памяти, зарезервированная операционной системой. Стек растягивается или уменьшается по мере того, как функции вызываются или завершают свое выполнение. Помимо локальных переменных в стек также помещаются аргументы, передаваемые функции.

Указатель на стек хранится в регистре процессора, который называется **ESP**. В регистр ESP помещается адрес вершины стека.

Когда вызывается функция `sum` и начинается ее выполнение, указатели **EIP** и **ESP** обновляются, как показано на рисунке выше. Обращаем внимание на изменение адреса в регистре EIP (вызов из `main` функции `sum`) и на смещение вершины стека (ESP) с учетом аргументов, переданных в вызываемую функцию `sum`.

Рисунок. Передача параметров через стек:

The screenshot displays a debugger window with three main panes. The left pane shows the source code of `Stack.cpp` with the `sum` function and its call in `_tmain`. The middle pane, titled 'Память 3' (Memory 3), shows a memory dump starting at address `0x00ECFC84`, with the first four bytes `27 14 d1 00` highlighted in red. The right pane, titled 'Регистры' (Registers), shows the current state of the CPU registers, with `EIP = 00D113C0` and `ESP = 00ECFC84` highlighted in red. Below the registers, the 'Дизассемблированный код' (Disassembled code) pane shows the assembly instructions for the `sum` function, including `push ebp`, `mov ebp, esp`, `sub esp, 0C0h`, and `push ebx`.

Когда выполнение передается на другой участок памяти, процессор автоматически помещает в вершину стека адрес, к которому следует вернуться после завершения выполнения этого кода.

9). Продолжаем пошаговое выполнение (обратите внимание, что в памяти по адресу регистра **ESP** в вершине стека лежит значение адреса возврата **0x00D11427**).

10). Переходим к выполнению последней в функции инструкции `ret` (ее адрес **0x00D113EA**). Нажимаем F10, и выполнение передается по адресу **0x00D11427**. Это адрес точки возврата.

Выводы:

- каждый поток имеет свой собственный указатель на текущую инструкцию, и его значение всегда актуально. Этот указатель хранится в регистре **EIP**.
- каждый поток имеет свой собственный стек (для хранения аргументов функции, локальных переменных), адрес инструкции, которой будет передано управление после выхода из функции (точка возврата). Адрес вершины стека хранится в регистре **ESP**.
- вызов функций осуществляется с помощью инструкций **call**.
- возврат из функции происходит с помощью инструкции **ret**.

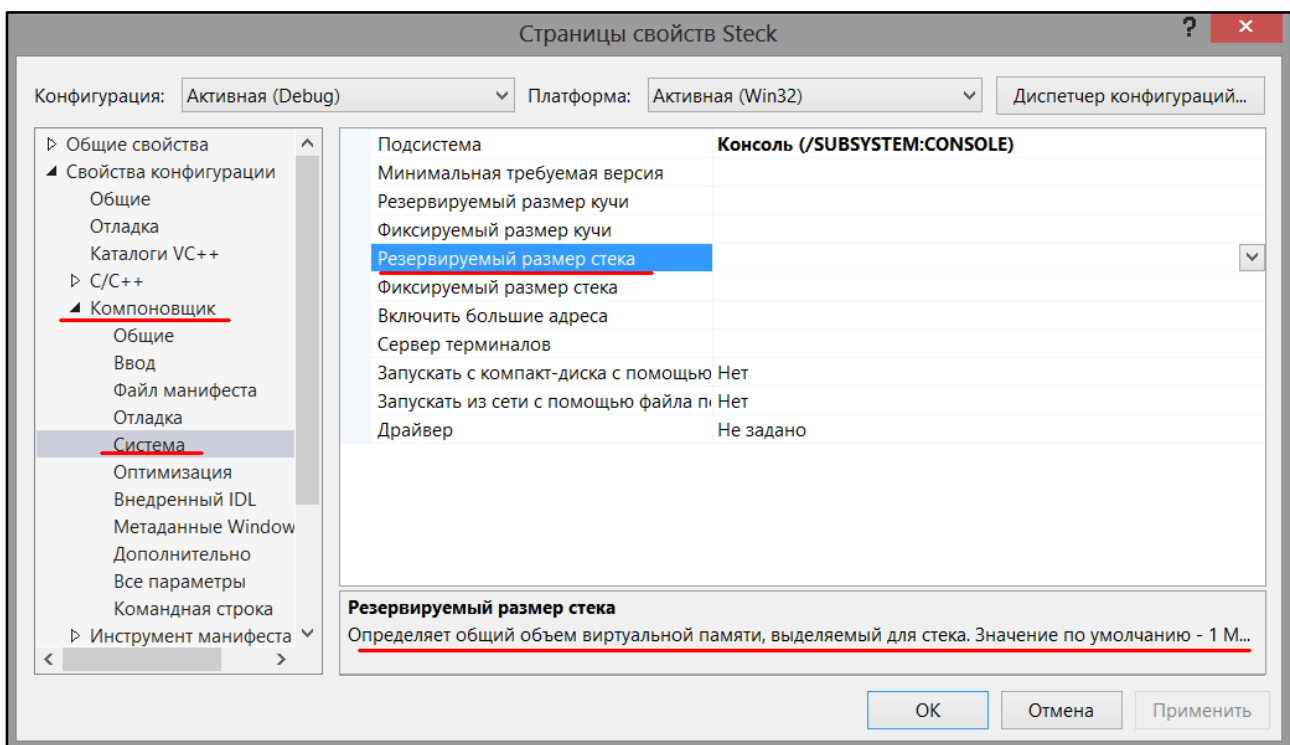
Инструкция **call** помещает в вершину стека (по указателю ESP) адрес точки возврата (адрес инструкции, следующей за **call**). Затем она обновляет регистр **EIP**, помещая в него адрес вызванного в данный момент кода, и выполнение потока продолжается с этого нового адреса, сохраненного в **EIP**.

Инструкция **ret** снимает с вершины стека, на которую указывает **ESP**, двойное слово (DWORD – 4 байта – в ассемблере соответствует типу **int** языка C/C++) и помещает его в регистр **EIP**. Затем выполнение потока продолжается с адреса, который теперь находится в **EIP**.

Подробнее о стеке.

Стек - это область памяти (в пределах, отведенных процессу четырех гигабайт), в которой поток может хранить **данные**, необходимые ему для выполнения. В стеке могут храниться локальные переменные, используемые кодом, временные переменные, используемые компилятором, аргументы функций и т.д. Доступ к данным в стеке организован по принципу: последним пришел, первым вышел (**LIFO** - Last In First Out).

По умолчанию размер стека равен 1 Мб:



Передача параметров в функцию

Передача параметров в функцию происходит через *стек*.

Вызывающая функция знает, сколько параметров надо ей передать и каковы значения этих параметров. В нашем примере, если код вызывает функцию `sum`, которая принимает два параметра типа `int`, то вызывающий код:

- помещает два параметра в стек с помощью двух инструкций `push`. В результате этого указатель стека (ESP) **уменьшается на 2×4 байта** (вершина стека сдвигается на **8 байт**);
- выполняет инструкцию `call`, которая передает управление функции `sum`. При этом значение ESP **уменьшается еще на 4 байта**, потому что в стек помещается *адрес точки возврата*.

Прием параметров

Вызываемая функция извлекает параметры из стека. При входе в функцию `sum`, в вершине стека (это первые 4-е байта в архитектуре X86) находится адрес точки возврата.

Значение, хранящееся в **ESP + 4** – это первый параметр, переданный в функцию `sum` (подчеркнут синим на рисунке «Передача параметров через стек»).

По адресу **ESP + 8** хранится второй параметр функции `sum`.

3. Соглашения о вызовах (*Calling conventions*). Определение.

Соглашение о вызовах – это протокол для передачи аргументов функциям.

Соглашение о вызовах – это договоренность между вызывающим и вызываемым кодом:

- о *способе* передачи параметров;
- о *порядке* их размещения в стеке;
- об *очистке* стека;
- о порядке *возврата* значения;
- о конкретной *инструкции* для вызова функции.

Соглашение о вызовах:

| Ключевое слово | Очистка стека | Передача параметров |
|-----------------------------------|--------------------|--|
| <u>cdecl</u> | Вызывающая функция | Параметры помещаются в стек в обратном порядке (справа налево) |
| <u>clrcall</u> | Н/Д | Параметры загружаются в стек выражений CLR по-порядку (слева направо). |
| <u>stdcall</u> | Вызываемая функция | Параметры помещаются в стек в обратном порядке (справа налево) |
| <u>fastcall</u> | Вызываемая функция | Хранятся в регистрах, затем помещаются в стек |
| <u>thiscall</u> | Вызываемая функция | Помещаются в стек; указатель this хранится в регистре ECX |
| <u>vectorcall</u> | Вызываемая функция | Хранятся в регистрах, затем помещаются в стек в обратном порядке (справа налево) |

3.1. Программные конструкции C++:

Соглашение вызовах __cdecl (используется по умолчанию). Параметры передаются через стек, порядок следования параметров справа налево, стек освобождает вызывающий код, возврат значения через регистр EAX.

Определение. Соглашение вызовах __cdecl:

1. Параметры функций помещаются в стек, порядок передачи параметров «справа налево». Параметры, размер которых меньше 4-х байт, *расширяются* до 4-х байт. Адрес возврата кладется в стек поверх параметров.
2. Стек освобождается **вызывающим** кодом: после инструкции CALL следует инструкция ADD, которая прибавляет к значению регистра ESP суммарный размер в байтах всех аргументов, т. о. целостность стека восстанавливается **вызывающим** кодом.
3. Возвращаемый параметр (результат) передается через регистр **EAX**.

Выполняем программу в режиме отладки:

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

int __cdecl funcA(int x, int y)
{
    return x+y;
};

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    int k1, k2;
    k1 = funcA(1,2);
    system("pause");
    return 0;
};
```

Дизассемблированный код (главная функция):

| | |
|-------------------------|------------------------|
| 18: k1 = funcA(1,2); | |
| 0013118E 6A 02 | push 2 |
| 00131190 6A 01 | push 1 |
| 00131192 E8 13 FF FF FF | call funcA (01310AAh) |
| 00131197 83 C4 08 | add esp,8 |
| 0013119A 89 45 FC | mov dword ptr [k1],eax |

Дизассемблированный код (вызываемая функция):

```
9: int _cdecl funcA(int x, int y)
10: {
00131150 55          push     ebp
00131151 8B EC      mov      ebp,esp
11:     return x+y;
00131153 8B 45 08    mov      eax,dword ptr [ebp+8]
00131156 03 45 0C    add      eax,dword ptr [ebp+0Ch]
12: };
00131159 5D          pop      ebp
0013115A C3          ret
```

(Глобальная область)

```
1 // Stack.cpp: определяет точку входа для консольного приложения.
2 //
3
4 #include "stdafx.h"
5
6 int sum(int x, int y){ return x + y; };
7
8 int _tmain(int argc, _TCHAR* argv[])
9 {
10     int rc = sum(1, 2);
11
12     return 0;
13 }
```

Память 3

Адрес: 0x00FCF6B4

| | | |
|------------|---|------------------|
| 0x00FCF6B4 | 00 00 00 00 00 00 00 00 00 00 e0 a6 7e cc |a ~M |
| 0x00FCF6C1 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMMMMMM |
| 0x00FCF6CE | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMMMMMM |
| 0x00FCF6DB | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMMMMMM |
| 0x00FCF6E8 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMMMMMM |
| 0x00FCF6F5 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMMMMMM |

Регистры

EAX = CCCCCCCC EBX = 7EA6E000 ECX = 00000000 EDX = 00000001 ESI = 00000000
EDI = 00FCF78C EIP = 00D2141E ESP = 00FCF6B4 EBP = 00FCF78C EFL = 00000204

Дизассемблированный код

Адрес: `wmain(int, wchar_t *)`

Параметры просмотра

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

```
9: {
00D21400 55          push     ebp
00D21401 8B EC       mov      ebp, esp
00D21403 81 EC CC 00 00 00 sub     esp, 0CCh
00D21409 53          push     ebx
00D2140A 56          push     esi
00D2140B 57          push     edi
00D2140C 8D BD 34 FF FF FF lea      edi, [ebp-0CCh]
00D21412 B9 33 00 00 00 mov      ecx, 33h
00D21417 B8 CC CC CC CC mov      eax, 0CCCCCCCCh
00D2141C F3 AB       rep stos dword ptr es:[edi]
10: int rc = sum(1, 2);
00D2141E 6A 02       push     2
00D21420 6A 01       push     1
```

Передача аргументов функции по протоколу `__cdecl`: параметры кладутся в стек «справа налево». Поверх них в стек кладется адрес точки возврата.

! Запомним значение регистра ESP.

Выполним первую инструкцию в строке по адресу `0x00D2141E`:
`push 2`

Обратите внимание, что значение регистра ESP уменьшилось на 4 и стало равно `0x00FCF6B4`.

Регистры

EAX = CCCCCCCC EBX = 7EA6E000 ECX = 00000000 EDX = 00000001 ESI = 00000000
EDI = 00FCF78C EIP = 00D21420 ESP = 00FCF6B0 EBP = 00FCF78C EFL = 00000206

Выполняем вторую инструкцию `push (push 2)` и значение ESP уменьшилось еще на 4 байта и равно `0x00FCF6B0`.

Теперь входим (F11) в вызываемую функцию `sum`.

В окне *дизассемблера* значение, хранимое в ESP, равно `0x00FCF6A8` и в стековой памяти по этому адресу:

Память 3

Адрес: `0x00FCF6A8`

| | | | |
|------------|---|----|-----------------|
| 0x00FCF6A8 | 27 14 d2 00 01 00 00 00 02 00 00 00 00 | 00 | ' . T |
| 0x00FCF6B5 | 00 00 00 00 00 00 00 00 e0 a6 7e cc cc | |a ~MM |
| 0x00FCF6C2 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | | MMMMMMMMMMMMMM |
| 0x00FCF6CF | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | | MMMMMMMMMMMMMM |
| 0x00FCF6DC | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | | MMMMMMMMMMMMMM |
| 0x00FCF6E9 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | | MMMMMMMMMMMMMM |

Регистры

EAX = CCCCCCCC EBX = 7EA6E000 ECX = 00000000 EDX = 00000001 ESI = 00000000
EDI = 00FCF78C EIP = 00D213C0 ESP = 00FCF6A8 EBP = 00FCF78C EFL = 00000206

Дизассемблированный код

Адрес: `sum(int, int)`

Параметры просмотра

- ☒ Показывать байты кода
- ☒ Показывать исходный код
- ☒ Показывать номера строк
- ☒ Показывать адреса
- ☒ Показывать имена символов

```
00D213BB CC          int      3
00D213BC CC          int      3
00D213BD CC          int      3
00D213BE CC          int      3
00D213BF CC          int      3
--- d:\adel\lpprim\к лекциям\steck\steck.cpp ---
1: // Stack.cpp: определяет точку входа для консольного приложения.
2: //
3:
4: #include "stdafx.h"
5:
6: int sum(int x, int y){ return x + y; };
00D213C0 55          push     ebp
00D213C1 8B EC        mov      ebp, esp
```

Начальное значение регистра ESP до размещения аргументов в стеке было равно **0x00FCF6B4**. После выполнения последней инструкции вызываемой функции `ret` значение ESP равно **0x00FCF6AC** (в этот момент адрес точки возврата уже удален из стека):

Память 3

Адрес: `0x00FCF6AC`

| | | |
|------------|---|----------------|
| 0x00FCF6AC | 01 00 00 00 02 00 00 00 00 00 00 00 00 00 | |
| 0x00FCF6B9 | 00 00 00 00 e0 a6 7e cc cc cc cc cc cc | ...a!~MMMMMM |
| 0x00FCF6C6 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMMMM |
| 0x00FCF6D3 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMMMM |
| 0x00FCF6E0 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMMMM |
| 0x00FCF6ED | cc cc cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMMMM |

Регистры

EAX = 00000003 EBX = 7EA6E000 ECX = 00000000 EDX = 00000001 ESI = 00000000
EDI = 00FCF78C EIP = 00D21427 **ESP = 00FCF6AC** EBP = 00FCF78C EFL = 00000204

В точке возврата в вызывающей функции:

```
Дизассемблированный код
Адрес: sum(int, int)
Параметры просмотра
[x] Показать байты кода
[x] Показать исходный код
[x] Показывать номера строк
[x] Показать адрес
[x] Показать имена символов
00D21401 8B EC      mov     ebp, esp
00D21403 81 EC CC 00 00 00  sub     esp, 0CCh
00D21409 53          push    ebx
00D2140A 56          push    esi
00D2140B 57          push    edi
00D2140C 8D BD 34 FF FF FF  lea     edi, [ebp-0CCh]
00D21412 B9 33 00 00 00     mov     ecx, 33h
00D21417 B8 CC CC CC CC     mov     eax, 0CCCCCCCCh
00D2141C F3 AB      rep stos dword ptr es:[edi]
10:      int rc = sum(1, 2);
00D2141E 6A 02      push    2
00D21420 6A 01      push    1
00D21422 E8 38 FC FF FF     call    sum (0D2105Fh)
00D21427 83 C4 08      add     esp, 8
```

Значение регистра **ESP** все еще не равно изначальное значению.

После выполнения инструкции

`add esp, 8`

с адресом **0x00D21427** (точка возврата, подчеркнута красным), получим: **0x00FCF6AC + 0x8 = 0x00FCF6B4** – это значение регистра **ESP** до размещения аргументов в стеке. Инструкция, восстанавливающая **целостность стека**, принадлежит **вызывающему** коду.

```
Память 3
Адрес: 0x00FCF6B4
0x00FCF6B4 00 00 00 00 00 00 00 00 00 e0 a6 7e cc .....a|~M
0x00FCF6C1 cc cc cc cc cc cc cc cc cc cc cc cc cc cccccccccccccccccc
0x00FCF6CE cc cc cc cc cc cc cc cc cc cc cc cc cc cccccccccccccccccc
0x00FCF6DB cc cc cc cc cc cc cc cc cc cc cc cc cc cccccccccccccccccc
0x00FCF6E8 cc cc cc cc cc cc cc cc cc cc cc cc cc cccccccccccccccccc
0x00FCF6F5 cc cc cc cc cc cc cc cc cc cc cc cc cc cccccccccccccccccc

Регистры
EAX = 00000003 EBX = 7EA6E000 ECX = 00000000 EDX = 00000001 ESI = 00000000
EDI = 00FCF78C EIP = 00D2142A ESP = 00FCF6B4 EBP = 00FCF78C EFL = 00000216
```

Как только управление передается в вызываемую функцию, до выполнения каких-либо инструкций внутри нее, первые 4 байта по адресу, хранящиеся в ESP, будут содержать адрес возврата, 4 байта по адресу (ESP + 4) содержат первый параметр, следующие 4 байтах по адресу (ESP + 8) содержат второй параметр:

| | | |
|----------|----------|----------|
| 2714D200 | 01000000 | 02000000 |
|----------|----------|----------|

3.2. Программные конструкции C++:

Соглашение вызова `__stdcall` (Windows API): параметры передаются через стек, порядок параметров справа налево, стек освобождает вызываемый код, возврат через регистр EAX.

Определение. Соглашение вызова `__stdcall`:

Параметры помещаются в стек: порядок параметров «справа налево». Адрес возврата кладется в стек поверх параметров.

Стек освобождает **вызываемый** код. В последней инструкции вызываемого кода **RET** указывается *значение* равное суммарному размеру в байтах всех параметров функции. Команда **RET** после извлечения адреса возврата прибавляет к регистру ESP указанное значение. **Целостность стека** восстанавливается *вызываемым* кодом.

Значение возвращаемого параметра передается через регистр EAX.

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

int __stdcall funcA(int x, int y)
{
    return x+y;
};

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    int k1, k2;
    k1 = funcA(1,2);
    system("pause");
    return 0;
};
```

| | |
|-------------------------|------------------------|
| 18: k1 = funcA(1,2); | |
| 0012118E 6A 02 | push 2 |
| 00121190 6A 01 | push 1 |
| 00121192 E8 18 FF FF FF | call funcA (01210AFh) |
| 00121197 89 45 FC | mov dword ptr [k1],eax |

```

9: int stdcall funcA(int x, int y)
10: {
00121150 push     ebp
00121151 mov      ebp,esp
11:      return x+y;
00121153 mov      eax,dword ptr [x]
00121156 add      eax,dword ptr [y]
12: };
00121159 pop      ebp
0012115A ret      8

```

Как только управление передается в вызываемую функцию, до выполнения каких-либо инструкций внутри нее, первые 4 байта по адресу, хранящемуся в ESP, будут содержать *адрес возврата*, 4 байта по адресу (ESP + 4) будут содержать *первый параметр*, следующие 4 байтах по адресу (ESP + 8) будут содержать *второй параметр*.

Следовательно, для восстановления целостности стека значение, хранимое в регистре ESP, должно увеличиться на 12.

The screenshot displays a debugger interface with three main panels:

- Память 3 (Memory 3):** Shows memory addresses from 0x001EFB30 to 0x001EFB71. The first three 4-byte blocks are highlighted with red boxes: 0x001EFB30 (27 14 8e 00), 0x001EFB34 (01 00 00 00), and 0x001EFB38 (02 00 00 00). These correspond to the return address and the two parameters passed to the function.
- Регистры (Registers):** Shows the current state of registers. The ESP register is highlighted with a red box and contains the value 001EFB30.
- Дизассемблированный код (Disassembled code):** Shows the assembly code for the function `sum(int, int)`. The instruction `ret 8` at address 008E13EA is highlighted with a red box, indicating the return instruction that adjusts the stack pointer.

В дизассемблированном коде последняя инструкция вызываемой функции:

ret 8

Команда **ret** сначала неявно увеличивает значение **ESP** на 4 байта (извлекает из стека адрес точки возврата). И поскольку после **ret** указано число 8, то **ESP** увеличивается еще на 8 байт.

Целостность стека восстановлена!

The screenshot displays a debugger interface with three main panels:

- Память 3 (Memory 3):** Shows a memory dump starting at address 0x001EFB3C. The first row (0x001EFB3C) contains eight zero bytes followed by the hex sequence c0 e2 7f cc, which corresponds to the ASCII string "Ав.М".
- Регистры (Registers):** Lists the current state of CPU registers. The ESP register is highlighted with a red box and shows the value 001EFB3C.
- Дизассемблированный код (Disassembled code):** Shows the assembly code for the function `sum(int, int)`. The instruction at address 008E1427, `mov dword ptr [rc], eax`, is highlighted with a red line and a yellow arrow, indicating it is the current instruction being executed.

On the left side of the debugger, a snippet of C code is visible:

```
int y){
    CHAR* argv
```

3.3. Программные конструкции C++:

Соглашение вызова **_fastcall** (не стандартизированный, для внутренних вызовов), параметры передаются через регистры (первые 2 – через регистры, остальные справа налево в стек), стек освобождает вызываемый код, возврат через регистр EAX. В Borland (Delphi) (параметры слева на право)

Определение. Соглашение вызова **_fastcall**:

Первые два параметра передаются через 2 регистра. Для передачи остальных параметров, используется стек; порядок параметров "справа налево".

Целостность стека восстанавливается вызываемым кодом.

Возвращаемый параметр передается через регистр EAX.

! В компиляторе фирмы Borland параметры передаются через регистры слева направо, если параметров больше двух, остальные помещаются в стек.

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

int _fastcall funcA(int x, int y, int p1, int p2, int p3, int p4)
{
    return x+y + p1+p2+p3+p4;
};

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    int k1, k2;
    k1 = funcA(1,2,3,4,5,6);
    system("pause");
    return 0;
};
```

```
18:      k1 = funcA(1,2,3,4,5,6);
00121B1E  push     6
00121B20  push     5
00121B22  push     4
00121B24  push     3
00121B26  mov      edx,2
00121B28  mov      ecx,1
00121B30  call     funcA (01210B9h)
00121B35  mov      dword ptr [k1],eax
19:      system("pause");
```

```

9: int _fastcall funcA(int x, int y, int p1, int p2, int p3, int p4)
10: {
00121AB0  push      ebp
00121AB1  mov       ebp,esp
00121AB3  sub       esp,8
00121AB6  mov       dword ptr [y],0CCCCCCCCh
00121ABD  mov       dword ptr [x],0CCCCCCCCh
00121AC4  mov       dword ptr [y],edx
00121AC7  mov       dword ptr [x],ecx
11:      return x+y + p1+p2+p3+p4;
00121ACA  mov       eax,dword ptr [x]
00121ACD  add       eax,dword ptr [y]
00121AD0  add       eax,dword ptr [p1]
00121AD3  add       eax,dword ptr [p2]
00121AD6  add       eax,dword ptr [p3]
00121AD9  add       eax,dword ptr [p4]
12: };
00121ADC  mov       esp,ebp
00121ADE  pop       ebp
00121ADF  ret       10h

```

3.4. Другие соглашения:

Соглашения вызова: _

- **pascal** (применялось в Windows 3.x);
- **_safecall** (COM),
- **_thiscall** (для объектно-ориентированных языков).

4. Программные конструкции:

- программные блоки;
- функции, не возвращающие значения (в других языках, часто называются процедурами); return не обязателен, но желателен для явного указания завершения функции;
- функции возвращающие значения; return обязателен.

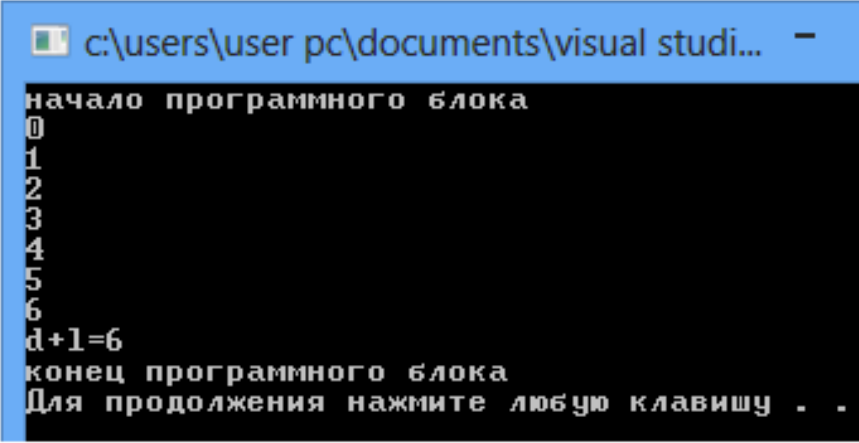
4.1. Программные конструкции C++: программные блоки

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

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

    {    // программный блок
        std::cout<<"начало программного блока"<<std::endl;
        int d = 2, l = 4;
        for (int k = 0; k < 7; k++) std::cout<<k<<std::endl;
        std::cout<<"d+l="<<d+l<<std::endl;
        std::cout<<"конец программного блока"<<std::endl;
    };

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

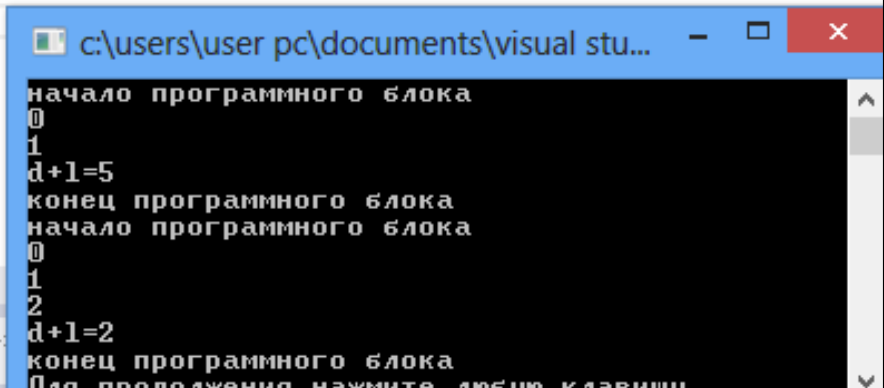


```
начало программного блока
0
1
2
3
4
5
6
d+l=6
конец программного блока
Для продолжения нажмите любую клавишу . .
```

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

#define BLOCK(x,y,z) {
    std::cout<<"начало программного блока"<<std::endl; \
    int d = x, l = y; \
    for(int k = 0; k < z; k++)std::cout<<k<<std::endl; \
    std::cout<<"d+l="<<d+l<<std::endl; \
    std::cout<<"конец программного блока"<<std::endl; \
}

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    BLOCK(2,3,2);
    BLOCK(1,1,3);
    system("pause");
    return 0;
};
```



Внимание: после символа продолжения (\) должен быть конец строки (Enter).

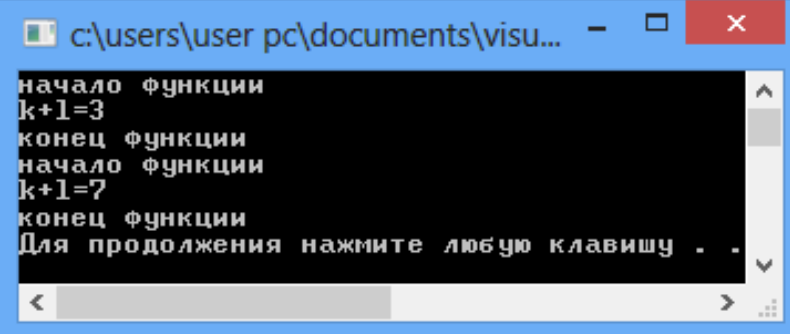
4.2. Программные конструкции C++:

функции *не возвращающие* значения (в других языках, часто называются процедурами), `return` не обязателен, но желателен для явного указания завершения функции.

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

void func(int k, int l) // функция не возвращающая значение
{
    std::cout<<"начало функции"<<std::endl;
    std::cout<<"k+l="<<k+l<<std::endl;
    std::cout<<"конец функции"<<std::endl;
    return; // не обязателен, но желателен
};

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    func (1,2);
    func (3,4);
    system( "pause" );
    return 0;
};
```



| | | | |
|----------|-------------------|------|-------------------------|
| 00AF5075 | 6A 00 | push | 0 |
| 00AF5077 | FF 15 60 04 B0 00 | call | dword ptr ds:[0B00460h] |
| 00AF507D | 83 C4 08 | add | esp,8 |
| 00AF5080 | 3B F4 | cmp | esi,esp |
| 00AF5082 | E8 C1 C2 FF FF | call | RTC CheckEsp (0AF1348h) |
| 19: | func (1,2); | | |
| 00AF5087 | 6A 02 | push | 2 |
| 00AF5089 | 6A 01 | push | 1 |
| 00AF508B | E8 BE C1 FF FF | call | func (0AF124Eh) |
| 00AF5090 | 83 C4 08 | add | esp,8 |
| 20: | func (3,4); | | |
| 00AF5093 | 6A 04 | push | 4 |
| 00AF5095 | 6A 03 | push | 3 |
| 00AF5097 | E8 B2 C1 FF FF | call | func (0AF124Eh) |
| 00AF509C | 83 C4 08 | add | esp,8 |
| 21: | system("pause"); | | |

4.3. Программные конструкции C++: функции, *возвращающие* значения:

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

int func(int k, int l) // функция возвращающая значение
{
    std::cout<<"начало функции"<<std::endl;
    std::cout<<"конец функции"<<std::endl;
    return k+l; // обязателен
};

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    int x = func (1,2);
    func (3,4);
    system("pause");
    return 0;
};
```

Показывать номера строк

```
18:      int x = func (1,2);
01265087 6A 02          push     2
01265089 6A 01          push     1
0126508B E8 1B C4 FF FF  call     func (012614ABh)
01265090 83 C4 08        add      esp,8
01265093 89 45 F8        mov     dword ptr [x],eax
19:      func (3,4);
```

4.3. Программные конструкции C++: параметры, передаваемые по ссылке.

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

int func(int& k, int l)
{
    k += 100;
    return k + l;
}

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

    int y = 3;
    int x = func(y, 2);

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

Память 3

Адрес: 0x00D6FA80

| Адрес | Содержимое | Столбцы |
|------------|---|-------------------|
| 0x00D6FA80 | 03 00 00 00 cc cc cc cc b5 5f 69 c8 dc fa d6 00 |ММММм_ийьЦ. |
| 0x00D6FA90 | a9 5d 92 00 01 00 00 00 60 9b f4 00 f0 a3 f4 00 | @]'.....`.ф.р]ф. |
| 0x00D6FAA0 | e5 5f 69 c8 00 00 00 00 00 00 00 00 80 27 7e | e_ий.....Ъ~ |
| 0x00D6FAB0 | 80 f8 ff ff c9 58 6b 24 00 00 00 00 00 d7 00 | ЪшяяЙХk\$......Ч. |
| 0x00D6FAC0 | 00 00 00 00 a0 fa d6 00 00 00 00 00 24 fb d6 00 |ьЦ.....\$ыЦ. |
| 0x00D6FAD0 | 09 11 92 00 81 4d 2d c8 00 00 00 00 e4 fa d6 00 | ..'гМ-И.....дьЦ. |

Регистры

EAX = 00F48A8C EBX = 7E278000 ECX = C7B13892 EDX = 00F4DFD0 ESI = 00D6F9A4
EDI = 00D6FA8C EIP = 00925428 ESP = 00D6F9A4 EBP = 00D6FA8C EFL = 00000246

Дизассемблированный код

Адрес: wmain(int, wchar_t**)

Параметры просмотра

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

```
0092541C E8 CD BE FF FF  call     __RTC_CheckEsp (09212EEh)

      int y = 3;
00925421 C7 45 F4 03 00 00 00  mov     dword ptr [y],3
      int x = func(y, 2);
00925428 6A 02          push     2
0092542A 8D 45 F4        lea      eax,[y]
0092542D 50          push     eax
0092542E E8 97 BF FF FF  call     func (09213CAh)
00925433 83 C4 08        add      esp,8
00925436 89 45 E8        mov     dword ptr [x],eax
```

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

int func(int& k, int l)
{
    k += 100;
    return k + 1;
}

int _tmain(int argc, _TCHAR* argv)
{
    setlocale(LC_ALL, "rus");

    int y = 3;
    int x = func(y, 2);

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

Память 3

Адрес: 0x00D6FA80

Столбцы: Авто

| | | | | | |
|------------|-------------------|--------------|----------------------|-----------------|-------------------|
| 0x00D6FA80 | 67 00 00 00 cc cc | Y=0x00000067 | 8 dc fa d6 00 | g...ММММм_ийьц. | |
| 0x00D6FA90 | a9 5d 92 00 01 00 | | 00 00 00 00 9b 14 00 | f0 a3 f4 00 | 0]'.....`ф.р]ф. |
| 0x00D6FAA0 | e5 5f 69 c8 00 00 | | 00 00 00 00 00 00 80 | 27 7e | e_иИ.....Б'~ |
| 0x00D6FAB0 | 80 f8 ff ff c9 58 | | 6b 24 00 00 00 00 00 | d7 00 | ЪшяяЙХк\$......Ч. |
| 0x00D6FAC0 | 00 00 00 00 a0 fa | | d6 00 00 00 00 00 24 | fb d6 00 |ьЦ.....\$ыЦ. |
| 0x00D6FAD0 | 09 11 92 00 81 4d | | 2d c8 00 00 00 00 e4 | fa d6 00 | ..'.ғМ-И....дьЦ. |

Регистры

EAX = 00000069 EBX = 7E278000 ECX = 00000067 EDX = 00D6FA80 ESI = 00D6F9A4
EDI = 00D6FA8C EIP = 00925439 ESP = 00D6F9A4 EBP = 00D6FA8C EFL = 00000212

Дизассемблированный код

Адрес: wmain(int, wchar_t **)

Параметры просмотра

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

| | | | |
|---------------------|----------------------|------|---------------------------|
| 0092541C | E8 CD BE FF FF | call | __RTC_CheckEsp (09212EEh) |
| int y = 3; | | | |
| 00925421 | C7 45 F4 03 00 00 00 | mov | dword ptr [y],3 |
| int x = func(y, 2); | | | |
| 00925428 | 6A 02 | push | 2 |
| 0092542A | 8D 45 F4 | lea | eax,[y] |
| 0092542D | 50 | push | eax |
| 0092542E | E8 97 BF FF FF | call | func (09213CAh) |
| 00925433 | 83 C4 08 | add | esp,8 |
| 00925436 | 89 45 E8 | mov | dword ptr [x],eax |

4.4. Программные конструкции C++:

передача адреса в качестве параметра.

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

float func(float m[], int k)
{
    float rc = m[k];
    m[k] *= 2;
    return rc;
}

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

    float mf[] = { 1.0f, 2.0f, 3.0f, 4.0f, 5.0f };
    float f = func(mf, 2);

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

Локальные

| Имя | Значение |
|--------------|---|
| argc | 0x00000001 |
| argv | 0x012c9b60 {0x012c9b68 L"D:\\Adel\\Кафедра\\ЯП_Лаб\\Ла69\\LPLab09\\D |
| mf | 0x011ffd60 {1.00000000, 2.00000000, 6.00000000, 4.00000000, 5.00000000} |
| [0x00000000] | 1.00000000 |
| [0x00000001] | 2.00000000 |
| [0x00000002] | 6.00000000 |
| [0x00000003] | 4.00000000 |
| [0x00000004] | 5.00000000 |
| f | 3.00000000 |

Локальные

Контрольные значения 1

Стек вызовов

Точки останова

Окно команд

Окно интерпр

4.5. Программные конструкции C++: передача структуры в качестве параметра.

Передача структуры по значению:

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

struct WWW{ int k; char cc[50]; };

WWW func(WWW w)
{
    w.k = 1;
    strcpy_s(w.cc, 50, "Минск");

    return w;
}

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

    WWW parm, rparm;
    parm.k = 0;
    strcpy_s(parm.cc, 50, "");

    rparm = func(parm);

    return 0;
}
```

| Локальные | | |
|-----------|--|------------|
| Имя | Значение | Тип |
| argc | 0x00000001 | int |
| argv | 0x00ff9b60 {0x00ff9b68 L"D:\Adel\Кафедра\ЯП_Ле | wchar_t ** |
| parm | {k=0x00000000 cc=0x00c9fb98 "" } | WWW |
| rparm | {k=0x00000001 cc=0x00c9fb58 "Минск" } | WWW |

Локальн... Контрол... Стек выз... Точки ос... Окно ко... Окно ин... Вывод Сг

[illegible]

```

01347FFE 3B F4          cmp     esi,esp
01348000 E8 E9 92 FF FF  call    __RTC_CheckEsp (013412EEh)

    rparm = func(parm);
01348005 83 EC 38       sub     esp,38h

    rparm = func(parm);
01348008 B9 0E 00 00 00  mov     ecx,0Eh
0134800D 8D 75 C0       lea     esi,[parm]
01348010 8B FC         mov     edi,esp
01348012 F3 A5         rep movs dword ptr es:[edi],dword ptr [esi]
01348014 8D 85 80 FE FF FF  lea     eax,[ebp-180h]
0134801A 50           push    eax
0134801B E8 18 94 FF FF  call    func (01341438h)
01348020 83 C4 3C       add     esp,3Ch
01348023 B9 0E 00 00 00  mov     ecx,0Eh
01348028 8B F0         mov     esi,eax
0134802A 8D BD 40 FE FF FF  lea     edi,[ebp-1C0h]
01348030 F3 A5         rep movs dword ptr es:[edi],dword ptr [esi]
01348032 B9 0E 00 00 00  mov     ecx,0Eh
01348037 8D B5 40 FE FF FF  lea     esi,[ebp-1C0h]
0134803D 8D 7D 80       lea     edi,[rparm]
01348040 F3 A5         rep movs dword ptr es:[edi],dword ptr [esi]

    return 0;
01348042 33 C0         xor     eax,eax

```

В регистр **ECX** поместили значение $0Eh_{16} = 14_{10}$.

Это повторитель: структура занимает 56 байт, следовательно, в стек надо поместить 14 раз по 4 байта целочисленных значений, составляющих структуру.

Команда **LEA** (Load Effective Address – загрузить текущий адрес) загружает адрес **rparm** в регистр **EDI**.

Команда **MOVS** с префиксом повторения **REP** (копирование двойного слова – 4 байта) – позволяет повторить команду столько раз, сколько указано в регистре **ECX** (14_{10} , $14 \cdot 4 = 56_{10}$). Команда по адресу $0x01348012$ копирует данные из участка памяти, адрес которого указан в регистре **ESI**, в другой участок памяти, адрес которого указан в регистре **EDI**.

PUSH – кладет результат в регистр **EAX**.

Команда LEA (Load Effective Address – загрузить текущий адрес) загружает адрес **rparm** и **REP MOVS** копирует данные из участка памяти, адрес которого указан в регистре **ESI**, в участок памяти, адрес которого указан в регистре **EDI**, повторяя команду столько раз, сколько указано в регистре **ECX**.

Режим отладки (дизассемблированный код).

1). Непосредственно перед вызовом функции **func** в стеке: в вершине стека – адрес возвращаемого значения типа **WWW** (типа структура); на дне стека параметр типа **WWW** (структура), переданный по значению, размером 56 байтов:

The screenshot displays a debugger interface with three main panels. The left panel shows the source code of a C program with a function `func` and a `main` function. The middle panel, titled 'Память 1', shows the stack memory starting at address `0x008DFBA4`, which is labeled as `<- ESP`. A red box highlights the return address `0x008DFBA4`. The right panel, titled 'Дизассемблированный код', shows the assembly code for the function `wmain`. A red box highlights the `call func (07142Eh)` instruction at address `000753CB`. The bottom panel, titled 'Память 4', shows the stack memory starting at address `ebp-180h`. A red box highlights the return address `0x008DFC30`.

2). Вход в функцию. В вершину стека помещен адрес точки возврата в главную функцию (не показано).

3). Возврат в главную функцию (адрес **0x000753D0**). С вершины стека снят адрес точки возврата. По адресу **0x008DFC30** видим результат выполнения функции – структуру с полями 1, “Минск”.

Память 1

Адрес: 0x008DFBA4 <- ESP

Дизассемблированный код

Адрес: wmain(int, wchar_t**)

Параметры просмотра

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

000753AE 3B F4 cmp esi,esp

000753B0 E8 2F BF FF FF call __RTC_CheckEsp (0712E4h)

rparm = func(parm);

000753B5 83 EC 38 sub esp,38h

000753B8 B9 0E 00 00 00 mov ecx,0Eh

000753BD 8D 75 C0 lea esi,[parm]

000753C0 8B FC mov edi,esp

000753C2 F3 A5 rep movs dword ptr es:[edi],dword ptr [esi]

000753C4 8D 85 80 FE FF FF lea eax,[ebp-180h]

000753CA 50 push eax

000753CB E8 5E C0 FF FF call func (07142Eh)

000753D0 83 C4 3C add esp,3Ch

000753D3 B9 0E 00 00 00 mov ecx,0Eh

000753D8 B8 F0 mov esi,eax

000753DA 8D BD 40 FE FF FF lea edi,[ebp-1C0h]

4). Очистку стека выполняет вызывающий код. Инструкция **add esp, 3Ch** очищает стек – удаляет 60 байтов: адрес возвращаемого значения (4 байта) и структуру (56 байтов).

Передача адреса структуры:

Дизассемблированный код

Адрес: wmain(int, wchar_t**)

Параметры просмотра

- ☒ Показывать байты кода
- ☒ Показывать исходный код
- ☒ Показывать номера строк
- ☒ Показывать адреса
- ☒ Показывать имена символов

000B7FF4 50 push eax

000B7FF5 FF 15 A0 01 0C 00 call dword ptr ds:[0C01A0h]

000B7FFB 83 C4 0C add esp,0Ch

000B7FFE 3B F4 cmp esi,esp

000B8000 E8 E9 92 FF FF call __RTC_CheckEsp (0B12EEh)

25: rparm = func(parm);

000B8005 8D 45 C0 lea eax,[parm]

000B8008 50 push eax

000B8009 8D 8D 80 FE FF FF lea ecx,[ebp-180h]

000B800F 51 push ecx

000B8010 E8 2D 94 FF FF call func (0B1442h)

000B8015 83 C4 08 add esp,8

000B8018 B9 0E 00 00 00 mov ecx,0Eh

000B801D B8 F0 mov esi,eax

000B801F 8D BD 40 FE FF FF lea edi,[ebp-1C0h]

000B8025 F3 A5 rep movs dword ptr es:[edi],dword ptr [esi]

000B8027 B9 0E 00 00 00 mov ecx,0Eh

000B802C 8D B5 40 FE FF FF lea esi,[ebp-1C0h]

000B8032 8D 7D 80 lea edi,[rparm]

000B8035 F3 A5 rep movs dword ptr es:[edi],dword ptr [esi]

27: return 0;

28: return 0;

| Контрольные значения 1 | | |
|------------------------|-------------------------|----------|
| Имя | Значение | Тип |
| sizeof(WWW) | 56 | unsigned |
| parm | {k=0 cc=0x0107f940 "" } | WWW |

Передача адреса структуры:

параметру присваивается адрес структуры;

в функции по этому адресу открываем доступ к самой структуре.

Вызов функции: это инструкция **call func (0B1442h)**

0x000B8015 – точка возврата (адрес инструкции, следующей за **call func (0B1442h)**)

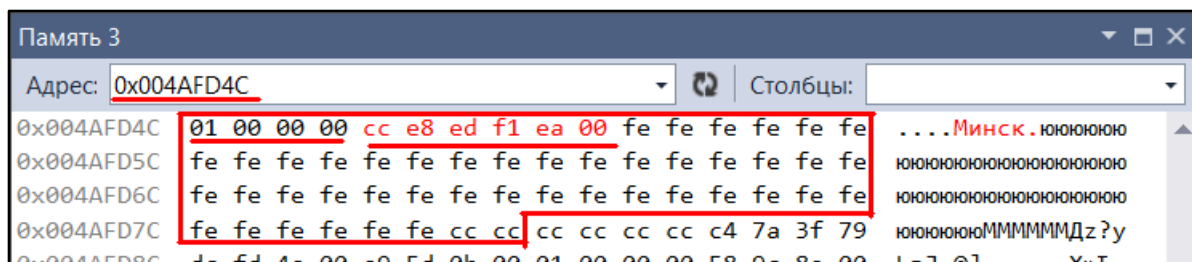
0x004AFD4C – адрес объекта **parm** типа **WWW** – это структура, передаваемая в функцию по ссылке.

Команда **LEA** (Load Effective Address – загрузить текущий адрес) загружает адрес **parm** в регистр **EAX**. Следующая инструкция **push EAX** помещает этот адрес в стек.

Функция func(WWW &):

```
6  #include <locale>
7
8  struct WWW{ int k; char cc[50]; };
9
10 WWW func(WWW& w)
11 {
12     w.k = 1;
13     strcpy_s(w.cc, 50, "Минск");
14
15     return w;
16 }
17
18 int _tmain(int argc, _TCHAR* argv[])
19 {
20     setlocale(LC_ALL, "rus");
21
22     WWW parm, rparm;
23     parm.k = 0;
24     strcpy_s(parm.cc, 50, "");
25
26     rparm = func(parm);
27 }
```

| Контрольные значения 1 | | |
|------------------------|---|----------|
| Имя | Значение | Тип |
| sizeof(WWW) | 56 | unsigned |
| parm | {k=0 cc=0x004afd50 "" } | WWW |
| &parm | 0x004afd4c {k=0 cc=0x004afd50 "" } | WWW * |
| w | {k=1 cc=0x004afd50 "Минск" } | WWW & |
| &w | 0x004afd4c {k=1 cc=0x004afd50 "Минск" } | WWW * |

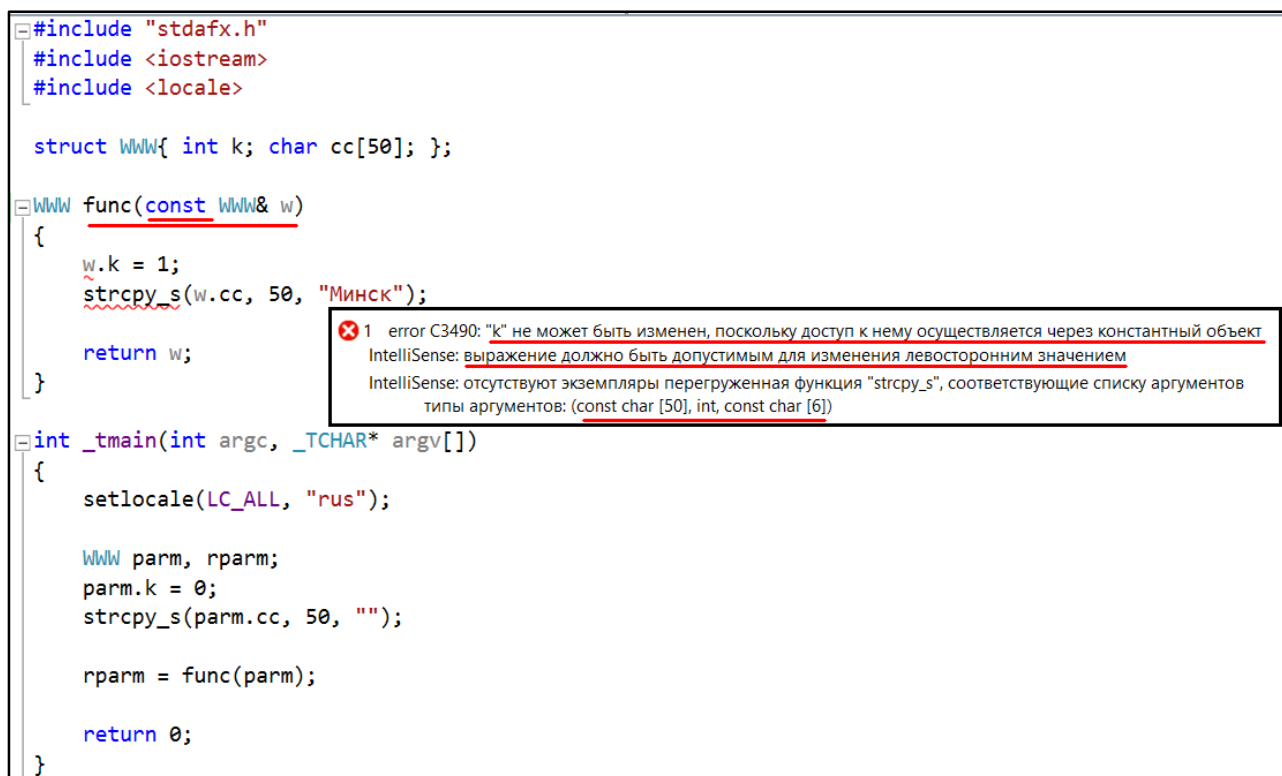


Возврат:

| Контрольные значения 1 | | |
|------------------------|---|--------------|
| Имя | Значение | Тип |
| sizeof(WWW) | 56 | unsigned int |
| parm | {k=1 cc=0x004afd50 "Минск" } | WWW |
| &parm | 0x004afd4c {k=1 cc=0x004afd50 "Минск" } | WWW * |
| rparm | {k=1 cc=0x004afd10 "Минск" } | WWW |

Программные конструкции: функции, параметры const.

Квалификатор типов const – зарезервированное ключевое слово, которое не позволяет модифицировать значения переменных.




```

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

struct WWW{ int k; char cc[50]; };

WWW func(const WWW& w)
{
    WWW ww = w;
    ww.k = 1;
    strcpy_s(ww.cc, 50, "Минск");

    return w;
}

```

5. Программные конструкции C++:

Функции, принимающие значения, заданные по умолчанию.

При обращении к функции, можно опускать некоторые из ее аргументов. Для этого необходимо при объявлении прототипа функции проинициализировать эти параметры значениями, которые будут использоваться в функции, как значения по умолчанию. Аргументы, заданные по умолчанию, должны быть последними аргументами.

```

#include <iostream>
#include <locale>

int func(int k, int l = 2, int m = 3)
{
    k += 100;
    return k + l + m;
}

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

    int x1 = func(1, 5, 6);
    int x2 = func(1, 5);
    int x3 = func(1);

    system("pause");
    return 0;
}

```

```

18:      int x1 = func(1, 5, 6);
00D57FD7  push      6
00D57FD9  push      5
00D57FDB  push      1
00D57FDD  call      func (0D5144Ch)
00D57FE2  add       esp,0Ch
00D57FE5  mov       dword ptr [x1],eax

19:      int x2 = func(1, 5);
00D57FE8  push      3
00D57FEA  push      5
00D57FEC  push      1
00D57FEE  call      func (0D5144Ch)
00D57FF3  add       esp,0Ch
00D57FF6  mov       dword ptr [x2],eax

20:      int x3 = func(1);
00D57FF9  push      3
00D57FFB  push      2
00D57FFD  push      1
00D57FFF  call      func (0D5144Ch)
00D58004  add       esp,0Ch
00D58007  mov       dword ptr [x3],eax

21:

```

Если при вызове функции не передавать ей значения формальных параметров *l* и *m*, то по умолчанию будут использоваться значения 2 и 3. В примере показаны различные способы использования функции `func` с аргументами по умолчанию.

5. Программные конструкции C++:

Переменное число параметров, указатель.

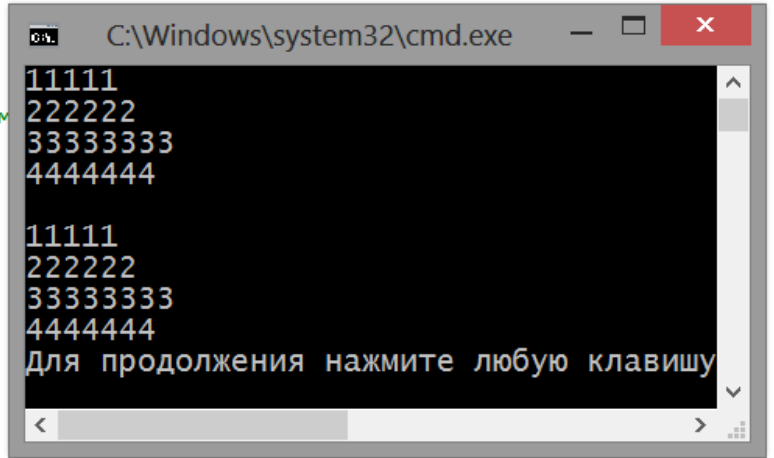
```
#include "stdafx.h"
#include <iostream>
#include <locale>

int func1(char* c, ...) // функция с переменным числом параметров
{
    char **p = &c;
    int k = 0;
    while (*(p + k)[0] != 0x00)
    {
        std::cout << *(p + k) << std::endl;
        k++;
    }
    std::cout << std::endl;
    return k;
}

int func2(int k, char* c, ...) // функция с переменным числом параметров
{
    char **p = &c;
    for (int i = 0; i < k; i++) std::cout << *(p + i) << std::endl;
    return k;
}

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

    int x1 = func1("11111", "222222", "33333333", "4444444", "");
    int x2 = func2(4, "11111", "222222", "33333333", "4444444");
    system("pause");
    return 0;
}
```



Косвенная адресация: указатель на указатель, который в свою очередь содержит адрес памяти, где хранятся данные.

6. Программные конструкции C++:

Функции с переменным числом параметров.

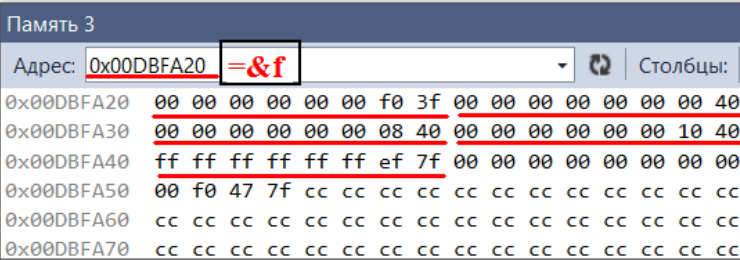
Пример: **int**, **double**.

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

int func1(int f, ...) // функция с переменным числом параметров int
{
    int *p = &f;
    int k = 0;
    while (p[k] != INT_MAX)
    {
        std::cout << p[k] << std::endl;
        k++;
    }
    std::cout << std::endl;
    return k;
}

int func2(double f, ...) // функция с переменным числом параметров double
{
    double *p = &f;
    int k = 0;
    while (p[k] != DBL_MAX)
    {
        std::cout << p[k] << std::endl;
        k++;
    }
    std::cout << std::endl;
    return k;
}

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    int x1 = func1(1, 2, 3, 4, INT_MAX);
    int x2 = func2(1.0, 2.0, 3.0, 4.0, DBL_MAX);
    system("pause");
    return 0;
}
```



По умолчанию параметры передаются в функцию через стек. Поэтому, технически, нет ограничения на количество передаваемых параметров. Функции с переменным числом параметров объявляются как обычные функции, но вместо недостающих аргументов ставится многоточие. Количество параметров и их типы известны только при вызове функции.

Передать функции параметры можно двумя способами:

- 1) явно передать обязательный аргумент, задающий число параметров;
- 2) добавить в конец списка параметр с уникальным значением, по которому будет определяться конец списка параметров.

Общий принцип работы следующий: в функции для доступа к списку параметров устанавливается указатель, значением которого будет адрес явного параметра в списке, далее изменяется значение этого указателя, чтобы переместиться на следующий параметр. В примере использован 2-ой способ передачи параметров.

7. Программные конструкции C++:

Функции с переменным числом параметров.

Пример: float, short, char

Пример для типа short: используется 1-ый способ передачи параметров (первый параметр – число передаваемых аргументов):

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

int func3(float f, ...) // функция с переменным числом параметров float
{
    double *p = (double*)&f+1;
    std::cout << f << std::endl;
    int k = 0;
    while(p[k] != (double)FLT_MAX) std::cout << p[k++] << std::endl;
    return k+1;
}

int func4(char c, ...) // функция с переменным числом параметров char
{
    int *p = (int*)&c;
    int k = 0;
    while (p[k] != 0) std::cout << p[k++] << std::endl;
    return k+1;
}

int func5(short s, ...) // функция с переменным числом параметров short
{
    int *p = (int*)&s;
    int k = 0;
    while (p[k] != (int)SHRT_MAX) std::cout << p[k++] << std::endl;
    return k+1;
}

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    int x5 = func5(1, 2, 3, 4, SHRT_MAX);
    int x4 = func4('a', 'b', 'c', 'd', (char)0x00);
    int x3 = func3(1.0, 2.0, 3.0, 4.0, FLT_MAX);
    system("pause");
    return 0;
}
```

Память 3

| Адрес: | 0x0078FD3C | ← ESP |
|------------|----------------------------------|-------|
| 0x0078FD3C | 09 4d 1e 01 01 00 00 00 02 00 00 | .M |
| 0x0078FD47 | 00 03 00 00 00 04 00 00 00 ff 7f | .. |
| 0x0078FD52 | 00 00 00 00 00 00 00 00 00 00 00 | .. |
| 0x0078FD5D | 40 97 7f cc cc cc cc cc cc cc cc | @- |
| 0x0078FD68 | cc cc cc cc cc cc cc cc cc cc cc | MM |

Память 4

| Адрес: | 0x0078fd40 | |
|------------|-------------------------------------|-----|
| 0x0078FD40 | 01 00 00 00 02 00 00 00 03 00 00 | ... |
| 0x0078FD4C | 04 00 00 00 ff 7f 00 00 00 00 00 | ... |
| 0x0078FD58 | 00 00 00 00 00 40 97 7f cc cc cc cc | ... |

Память 1 Память 2 Память 4

Передача параметров в функцию происходит через стек.

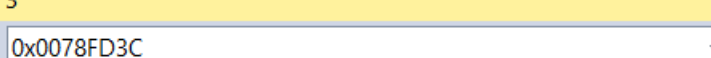
Указатель на адрес вершины стека хранится в регистре ESP.

push уменьшает значение ESP на 4 и помещает 32-битное значение по указанному адресу (стек увеличивается на 4 байта).

pop извлекает 32 бита по адресу, на который указывает ESP, и затем увеличивает значение ESP на 4 (стек уменьшается на 4 байта).

Указатель должен быть типизирован.

В случае передачи параметров типа char применен 2-ой способ передачи параметров. При вызове функции параметры помещаются в стек справа налево, при этом происходит выравнивание на границу 4 байта.



Память 3

Адрес: 0x0078FD3C

| Адрес | Данные (HEX) | Данные (ASCII) |
|------------|--|----------------|
| 0x0078FD3C | 1e 4d 1e 01 <u>61</u> 00 00 00 <u>62</u> 00 00 00 | .M..a...b... |
| 0x0078FD48 | <u>63</u> 00 00 00 <u>64</u> 00 00 00 <u>00</u> 00 00 00 | c...d..... |
| 0x0078FD54 | 00 00 00 00 00 00 00 00 00 40 97 7f |@-. |
| 0x0078FD60 | cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMM |
| 0x0078FD6C | cc cc cc cc cc cc cc cc cc cc cc cc | MMMMMMMMMMMM |

Память 1

Адрес: 0x0078FD40

| | | | | |
|------------|-------------|-------------|-------------|---------------|
| 0x0078FD40 | 61 00 00 00 | 62 00 00 00 | 63 00 00 00 | a...b...c... |
| 0x0078FD4C | 64 00 00 00 | 00 00 00 00 | 00 00 00 00 | d..... |
| 0x0078FD58 | 00 00 00 00 | 00 40 97 7f | cc cc cc cc |@-..MMMM |
| 0x0078FD64 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | |

Память 1 Память 2 Память 4

```

8 int func3(float f, ...) // функция с переменным числом параметров float
9 {
10     double *p = (double*)&f+1;
11     std::cout << f << std::endl;
12     int k = 0;
13     while(p[k] != (double)FLT_MAX) std::cout << p[k++] << std::endl;
14     return k+1;
15 }
16 int func4(char *p)
17 {
18     int *p = (int*)p;
19     int k = 0;
20     while (p[k] != 0)
21     {
22         return k+1;
23     }
24 }
25 int func5(short *p)
26 {
27     int *p = (int*)p;
28     int k = 0;
29     while (p[k] != 0)
30     {
31         return k;
32     }
33 }

```

Память 3

| Адрес: | 0x009AFD44 | 0x009AFD45 | 0x009AFD46 | 0x009AFD47 | 0x009AFD48 | 0x009AFD49 | 0x009AFD4A | 0x009AFD4B | 0x009AFD4C | 0x009AFD4D | 0x009AFD4E | 0x009AFD4F | 0x009AFD50 | 0x009AFD51 | 0x009AFD52 | 0x009AFD53 | 0x009AFD54 | 0x009AFD55 | 0x009AFD56 | 0x009AFD57 | 0x009AFD58 | 0x009AFD59 | 0x009AFD5A | 0x009AFD5B | 0x009AFD5C | 0x009AFD5D | 0x009AFD5E | 0x009AFD5F | 0x009AFD60 | 0x009AFD61 | 0x009AFD62 | 0x009AFD63 | 0x009AFD64 | 0x009AFD65 | 0x009AFD66 | 0x009AFD67 | 0x009AFD68 | 0x009AFD69 | 0x009AFD6A | 0x009AFD6B | 0x009AFD6C | 0x009AFD6D | 0x009AFD6E | 0x009AFD6F | 0x009AFD70 | 0x009AFD71 | 0x009AFD72 | 0x009AFD73 | 0x009AFD74 | 0x009AFD75 | 0x009AFD76 | 0x009AFD77 | 0x009AFD78 | 0x009AFD79 | 0x009AFD7A | 0x009AFD7B | 0x009AFD7C | 0x009AFD7D | 0x009AFD7E | 0x009AFD7F | 0x009AFD80 | 0x009AFD81 | 0x009AFD82 | 0x009AFD83 | 0x009AFD84 | 0x009AFD85 | 0x009AFD86 | 0x009AFD87 | 0x009AFD88 | 0x009AFD89 | 0x009AFD8A | 0x009AFD8B | 0x009AFD8C | 0x009AFD8D | 0x009AFD8E | 0x009AFD8F | 0x009AFD90 | 0x009AFD91 | 0x009AFD92 | 0x009AFD93 | 0x009AFD94 | 0x009AFD95 | 0x009AFD96 | 0x009AFD97 | 0x009AFD98 | 0x009AFD99 | 0x009AFD9A | 0x009AFD9B | 0x009AFD9C | 0x009AFD9D | 0x009AFD9E | 0x009AFD9F | 0x009AFDA0 | 0x009AFDA1 | 0x009AFDA2 | 0x009AFDA3 | 0x009AFDA4 | 0x009AFDA5 | 0x009AFDA6 | 0x009AFDA7 | 0x009AFDA8 | 0x009AFDA9 | 0x009AFDAA | 0x009AFDAB | 0x009AFDAC | 0x009AFDAD | 0x009AFDAE | 0x009AFDAF | 0x009AFDB0 | 0x009AFDB1 | 0x009AFDB2 | 0x009AFDB3 | 0x009AFDB4 | 0x009AFDB5 | 0x009AFDB6 | 0x009AFDB7 | 0x009AFDB8 | 0x009AFDB9 | 0x009AFDBA | 0x009AFDBB | 0x009AFDBC | 0x009AFDBD | 0x009AFDBE | 0x009AFDBF | 0x009AFDC0 | 0x009AFDC1 | 0x009AFDC2 | 0x009AFDC3 | 0x009AFDC4 | 0x009AFDC5 | 0x009AFDC6 | 0x009AFDC7 | 0x009AFDC8 | 0x009AFDC9 | 0x009AFDCA | 0x009AFDCB | 0x009AFDCC | 0x009AFDCD | 0x009AFDCE | 0x009AFDCF | 0x009AFDD0 | 0x009AFDD1 | 0x009AFDD2 | 0x009AFDD3 | 0x009AFDD4 | 0x009AFDD5 | 0x009AFDD6 | 0x009AFDD7 | 0x009AFDD8 | 0x009AFDD9 | 0x009AFDDA | 0x009AFDDB | 0x009AFDDC | 0x009AFDDD | 0x009AFDDE | 0x009AFDDF | 0x009AFDE0 | 0x009AFDE1 | 0x009AFDE2 | 0x009AFDE3 | 0x009AFDE4 | 0x009AFDE5 | 0x009AFDE6 | 0x009AFDE7 | 0x009AFDE8 | 0x009AFDE9 | 0x009AFDEA | 0x009AFDEB | 0x009AFDEC | 0x009AFDED | 0x009AFDEE | 0x009AFDEF | 0x009AFDF0 | 0x009AFDF1 | 0x009AFDF2 | 0x009AFDF3 | 0x009AFDF4 | 0x009AFDF5 | 0x009AFDF6 | 0x009AFDF7 | 0x009AFDF8 | 0x009AFDF9 | 0x009AFDFA | 0x009AFDFB | 0x009AFDFC | 0x009AFDFD | 0x009AFDFE | 0x009AFDFF | 0x009AFD80 | 0x009AFD81 | 0x009AFD82 | 0x009AFD83 | 0x009AFD84 | 0x009AFD85 | 0x009AFD86 | 0x009AFD87 | 0x009AFD88 | 0x009AFD89 | 0x009AFD8A | 0x009AFD8B | 0x009AFD8C | 0x009AFD8D | 0x009AFD8E | 0x009AFD8F | 0x009AFD90 | 0x009AFD91 | 0x009AFD92 | 0x009AFD93 | 0x009AFD94 | 0x009AFD95 | 0x009AFD96 | 0x009AFD97 | 0x009AFD98 | 0x009AFD99 | 0x009AFD9A | 0x009AFD9B | 0x009AFD9C | 0x009AFD9D | 0x009AFD9E | 0x009AFD9F | 0x009AFDA0 | 0x009AFDA1 | 0x009AFDA2 | 0x009AFDA3 | 0x009AFDA4 | 0x009AFDA5 | 0x009AFDA6 | 0x009AFDA7 | 0x009AFDA8 | 0x009AFDA9 | 0x009AFDAA | 0x009AFDAB | 0x009AFDAC | 0x009AFDAD | 0x009AFDAE | 0x009AFDAF | 0x009AFDB0 | 0x009AFDB1 | 0x009AFDB2 | 0x009AFDB3 | 0x009AFDB4 | 0x009AFDB5 | 0x009AFDB6 | 0x009AFDB7 | 0x009AFDB8 | 0x009AFDB9 | 0x009AFDBA | 0x009AFDBB | 0x009AFDBC | 0x009AFDBD | 0x009AFDBE | 0x009AFDBF | 0x009AFDC0 | 0x009AFDC1 | 0x009AFDC2 | 0x009AFDC3 | 0x009AFDC4 | 0x009AFDC5 | 0x009AFDC6 | 0x009AFDC7 | 0x009AFDC8 |
|--------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
|--------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|

double:

The screenshot shows a C++ program with a function `fvarparam` that takes a float `f` and a variable number of arguments. The function calculates the sum of the arguments and returns the count. A memory dump window shows the stack frame for `fvarparam`, with the return value `k` set to 3. The memory dump also shows the arguments `p` and `p[k]` pointing to memory locations containing the values 2, 3, and 4.

```
int fvarparam(float f, ...)
{
    double *p = (double*)&f + 1;
    std::cout << f << ' ';
    int k = 0;
    while (p[k] != (double)FLT_MAX) std::cout << "p[" << k << "] = " << p[k++] << ' ';
    std::cout << std::endl;
    return k;
};

double dvarparam(...)
{
    double *p = ...
}
```

| Имя | Значение | Тип |
|------|---------------------------------|----------|
| k | 3 | int |
| p | 0x0109f7cc {2.0000000000000000} | double * |
| p[k] | 3.4028234663852886e+038 | double |

Память 4
Адрес: 0x0109f7c4
1.0 (4 байта) 2.0 (8 байтов)
0x0109f7c4 db a6 39 00 00 00 80 3f 00 00 00 00 00 00 00 00 ы!9...Ъ?.....
0x0109f7d3 40 00 00 00 00 00 00 08 40 00 00 00 00 00 00 00 @.....@.....
0x0109f7e2 10 40 00 00 00 e0 ff ff ef 47 00 00 00 00 00 00 .@...аяяпG.....
0x0109f7f1 00 00 00 00 f0 96 70 66 66 66 66 66 66 66 66 66 pMMMMMMMM

8. Программные конструкции C++: переменное число параметров функции.

Пример: **STRUCT**

Первый параметр явно задает число передаваемых параметров в функцию:

The screenshot shows a C++ program with a function `func6` that takes a short `k` and a variable number of arguments of type `PARM`. The function prints the values of the arguments. The `main` function calls `func6` with 3 arguments. The console output shows the values 1 111, 2 222, and 3 333.

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

struct PARM {int id; char m[4];};

int func6 (short k, PARM p, ...) // с переменным числом параметров
{
    PARM *pp = &p;
    for (int i = 0; i < k; i++)
    {
        std::cout<<pp[i].id<< " "<<pp[i].m<<std::endl;
    }
    return 0;
};

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    PARM p1 = {1,"111"}, p2 = {2,"222"}, p3 = {3,"333"};
    func6(3, p1, p2, p3);
    system("pause");
    return 0;
};
```

C:\Users\Us...
1 111
2 222
3 333
Для продолжения нажмите

9. Программные конструкции C++: переменное число параметров функции.

Пример: макросы **va_list**, **va_start**, **va_arg**, **va_end**.

В стандарт языка входит набор макросов для работы со списками параметров переменной длины, определенный в **stdarg.h**. При использовании этих макросов точно так же требуется указывать в списке:

- явный параметр,
- объявить и установить на него указатель и
- перемещаться по списку, изменяя его.

Способы передачи параметров:

- 1) явно определяя обязательный аргумент, задающий число параметров,
- 2) добавляя в конец списка параметр с уникальным значением.

Назначение макросов:

- макрос **va_list** определяет тип указателя;
- макрос **va_start** устанавливает указатель типа **va_list** на явный параметр;
- макрос **va_arg** перемещает указатель на следующий параметр;
- макрос **va_end** обнуляет указатель.

Использование макросов:

- 1) объявляем указатель **va_list arg** в теле функции с переменным числом параметров до первого использования макросов;
- 2) указанный объект связывается с последним явно заданным формальным параметром (перед многоточием) переменного списка параметров с помощью макроса **va_start(arg, P)** (инициализация указателя);
- 3) перемещение по переменному списку параметров выполняется макросом **va_arg**. Для этого необходимо *явно* указывать тип очередного параметра, то есть разработчик должен его знать в момент написания программы. Например, если все параметры в списке **целого типа**, то вызов **va_arg** выглядит так:
va_arg(arg, int);
- 4) макрос **va_end(arg)** завершает обработку.

Пример использования для типа float и структуры типа PARM.

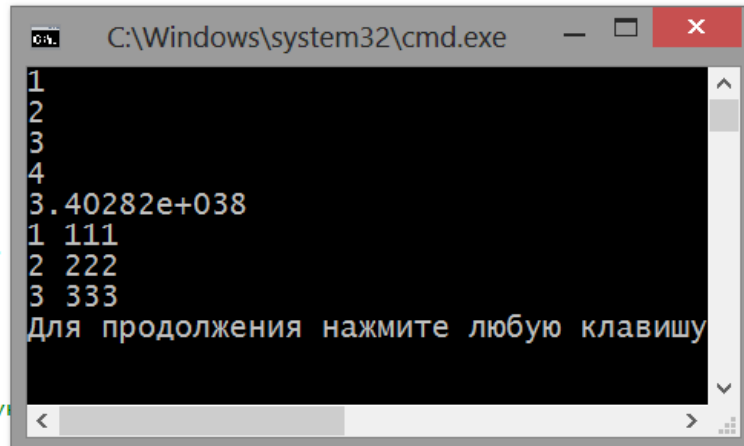
```
#include <iostream>
#include <locale>
#include <cstdlib>

struct PARM{ int id; char m[4]; };

int func3x(float f, ...) // функция с переменным числом параметров float
{
    va_list arg;
    va_start(arg, f);
    std::cout << f << std::endl;
    float f1;
    do{
        f1 = va_arg(arg, double);
        std::cout << f1 << std::endl;
    } while (f1 != FLT_MAX);
    va_end(arg);
    return 4;
}

int func6x(short k, PARM p...) // функция с переменным числом параметров PARM
{
    va_list arg;
    va_start(arg, p);
    PARM pp = p;
    std::cout << pp.id << " " << pp.m << std::endl;
    for (int i = 0; i < k - 1; i++)
    {
        pp = va_arg(arg, PARM);
        std::cout << pp.id << " " << pp.m << std::endl;
    }
    va_end(arg);
    return k-1;
}

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    int x3 = func3x(1.0, 2.0, 3.0, 4.0, FLT_MAX);
    PARM p1 = { 1, "111" }, p2 = { 2, "222" }, p3 = { 3, "333" };
    int x6 = func6x(3, p1, p2, p3);
    system("pause");
    return 0;
}
```



```
C:\Windows\system32\cmd.exe
1
2
3
4
3.40282e+038
1 111
2 222
3 333
Для продолжения нажмите любую клавишу
```

10. Программные конструкции: перегружаемые функции

Перегрузка процедур и функций обеспечивает возможность использования одноимённых подпрограмм: процедур или функций.

При трансляции происходит контроль одноимённых процедур и функций, чтобы они различались по **сигнатуре** (либо разным числом аргументов, либо другими типами аргументов). В этом случае транслятор может однозначно определить вызов нужной подпрограммы.

Чтобы исключить ошибку программиста, давшего случайно имя подпрограмме, которое уже используется, в некоторых языках вводится требование написания дополнительного ключевого слова. Так сделано, например, в языке Delphi (ключевое слово `overload`).

Под **перегрузкой** функции понимается, определение двух и более функций с одинаковым именем в одной области видимости, но с различными параметрами. Наборы параметров перегруженных функций могут отличаться порядком следования, количеством, типом.

Говорят, что две функции, с разной сигнатурой, но одинаковыми именами – перегруженные функции.

Сигнатура – это комбинация имени функции с её параметрами.

Компилятор самостоятельно выбирает нужную функцию, анализируя сигнатуры перегруженных функций.

```
#include "stdafx.h"
#include <cstring>
struct WWW {int k; char cc[50];};

WWW func(const WWW& w)
{
    WWW ww = w;
    ww.k=1;
    strcpy_s(ww.cc, 50, "Минск");
    return ww;
}

int _tmain(int argc, _TCHAR* argv[])
{
    WWW parm, rparm;

    parm.k = 0;
    strcpy_s(parm.cc, 2, "");

    rparm = func(parm);

    return 0;
}
```

```

#include <cstring>
struct WWW {int k; char cc[50];};

WWW func(const WWW& w)
{
    WWW ww = w;
    ww.k=1;
    strcpy_s(ww.cc, 50, "МИНСК");
    return ww;
}

int func(int x, int y)
{
    return x+y;
}

float func(float x, float y)
{
    return x+y;
}

int _tmain(int argc, _TCHAR* argv[])
{
    WWW parm, rparm;

    parm.k = 0;
    strcpy_s(parm.cc, 2, "");
    rparm = func(parm);

    int i = func(4, 5);

    float f = func(4.1f, 2.8f);

    return 0;
}

```

Пример неправильного использования перегрузки:

```

float func(float x, float y)
{
    return x+y;
}

int func(float x, float y)
{
    return(int) x+y;
}

```

IntelliSense: не удастся перегрузить функции, различаемые только по типу возвращаемого значения

IntelliSense: не удастся перегрузить функции, различаемые только по типу возвращаемого значения

IntelliSense: не существует подходящей функции преобразования из "WWW" в "float"

error C2556: int func(float,float): перегруженная функция отличается от "float func(float,float)" только возвращаемым типом

error C2371: func: перепреопределение; различные базовые типы

```

rparm = func(parm);

```

11. Программные конструкции C++: inline (подставляемые, встраиваемые) функции; **inline** – указание компилятору для выполнения подстановки, компилятор может *не выполнить* подстановку, если не установить параметры.

Встроенные функции или **inline**-функции определяются с помощью ключевого слова **inline**. Каждый раз, вместо вызова этой функции, компилятор будет заменять его фактическим кодом функции.

Вызовы функций занимают больше времени, чем выполнение последовательного кода. Но при такой замене увеличивается размер программы.

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

int funcA(int x, int y)
{
    return x+y;
};

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    int k1, k2;
    k1 = funcA(1,2);
    k2 = funcA(1,4);
    system("pause");
    return 0;
};
```

| | | | |
|----------|----------------|------|-------------------------|
| 008D1189 | E8 12 1A 00 00 | call | _RTC_CheckEsp (08D2BA0) |
| 17: | | | int k1, k2; |
| 18: | | | k1 = funcA(1,2); |
| 008D118E | 6A 02 | push | 2 |
| 008D1190 | 6A 01 | push | 1 |
| 008D1192 | E8 13 FF FF FF | call | funcA (08D10AAh) |
| 008D1197 | 83 C4 08 | add | esp,8 |
| 008D119A | 89 45 FC | mov | dword ptr [k1],eax |
| 19: | | | k2 = funcA(1,4); |
| 008D119D | 6A 04 | push | 4 |
| 008D119F | 6A 01 | push | 1 |
| 008D11A1 | E8 04 FF FF FF | call | funcA (08D10AAh) |
| 008D11A6 | 83 C4 08 | add | esp,8 |
| 008D11A9 | 89 45 F8 | mov | dword ptr [k2],eax |
| 20: | | | system("pause"); |

```

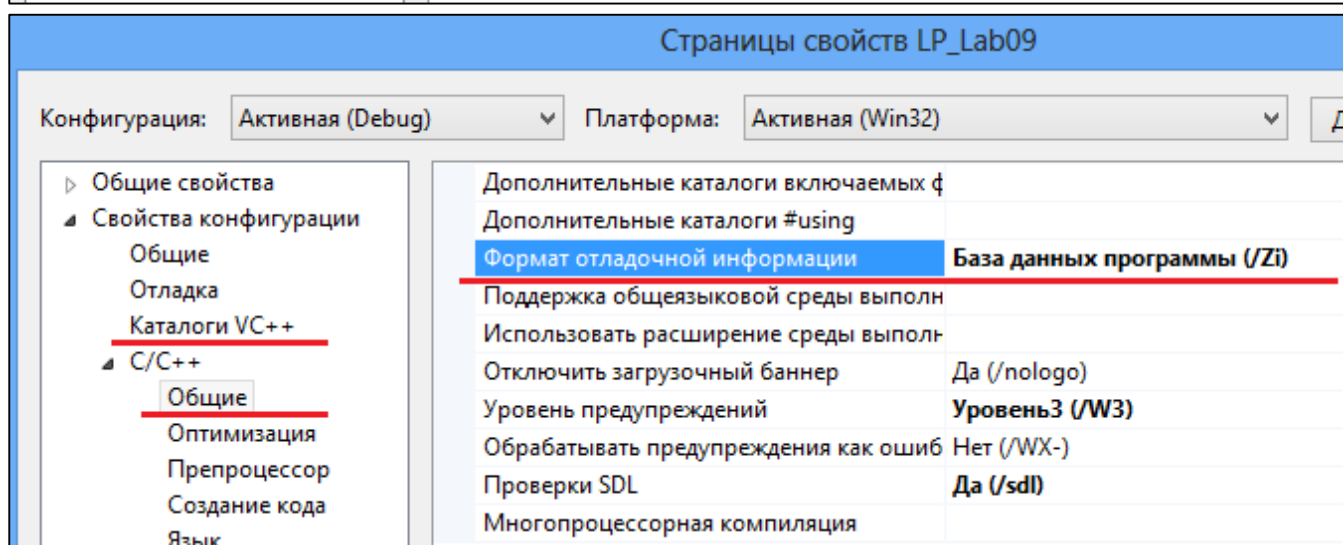
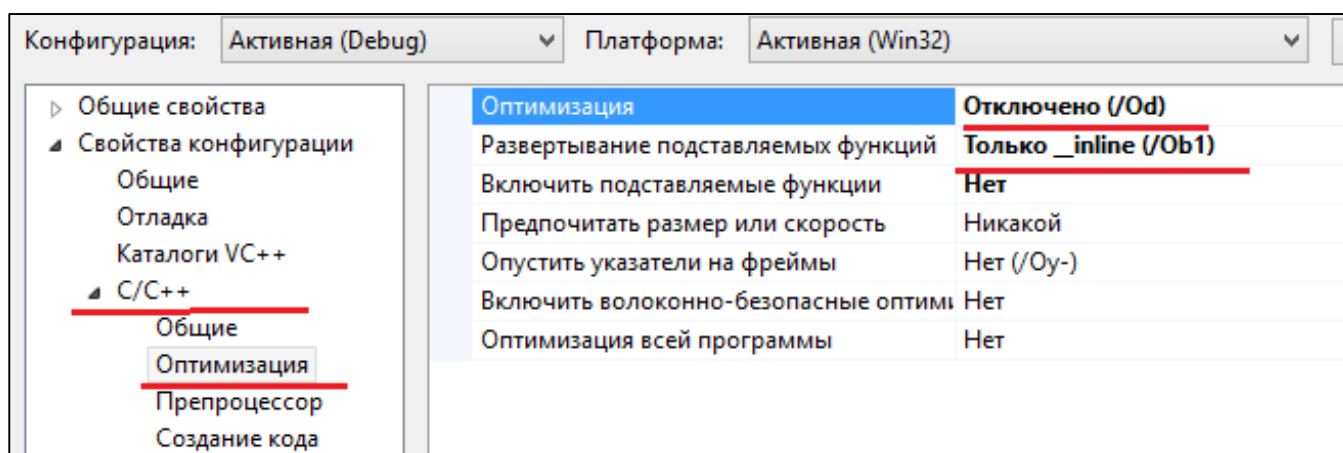
#include "stdafx.h"
#include <iostream>
#include <locale>
#include <cstdint>

inline int funcA(int x, int y)
{
    return x+y;
};

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    int k1, k2;
    k1 = funcA(1,2);
    k2 = funcA(1,4);
    system("pause");
    return 0;
};

```

Следует понимать, что компилятор может *игнорировать* попытки сделать функцию встроенной, если не установить параметры:



Так выглядит подстановка `inline`-функции в код:

| | | | |
|----------|-------------------|------|---------------------------|
| 0109116E | FF 15 C0 E3 09 01 | call | dword ptr ds:[109E3C0h] |
| 01091174 | 83 C4 08 | add | esp,8 |
| 01091177 | 3B F4 | cmp | esi,esp |
| 01091179 | E8 22 1A 00 00 | call | _RTC_CheckEsp (01092BA0h) |
| 17: | int k1, k2; | | |
| 18: | k1 = funcA(1,2); | | |
| 0109117E | B8 01 00 00 00 | mov | eax,1 |
| 01091183 | 83 C0 02 | add | eax,2 |
| 01091186 | 89 45 FC | mov | dword ptr [k1],eax |
| 19: | k2 = funcA(1,4); | | |
| 01091189 | B9 01 00 00 00 | mov | ecx,1 |
| 0109118E | 83 C1 04 | add | ecx,4 |
| 01091191 | 89 4D F8 | mov | dword ptr [k2],ecx |
| 20: | system("pause"); | | |
| 01091194 | 8B F4 | mov | esi,esp |
| 01091196 | 68 F4 AA 09 01 | push | 109AAF4h |
| 0109119B | FF 15 C8 E3 09 01 | call | dword ptr ds:[109E3C8h] |
| 010911A1 | 83 C4 04 | add | esp,4 |
| 010911A4 | 3B F4 | cmp | esi,esp |
| 010911A6 | E8 F5 19 00 00 | call | _RTC_CheckEsp (01092BA0h) |
| 21: | return 0; | | |

12. Программные конструкции: шаблоны функций: обобщенное описание семейства функций.

Шаблоны функций определяют семейство функций. С помощью шаблонов функций можно задавать наборы функций, основанных на одном коде, но действующих для разных типов. Шаблон функции начинается с ключевого слова **template**, за которым в *угловых скобках* следует ключевое слово **typename** и затем список параметров шаблона:

```
#include "stdafx.h"
template<typename T>
T max(T x, T y)
{
    return (x > y ? x : y);
}

int _tmain(int argc, _TCHAR* argv[])
{
    int r = max<int>(3, 5);
    float fr = max<float>(3.2f, 1.2f);
    double dr = max<double>(3.14, 1.25);
    char cr = max<char>('a', 'b');

    return 0;
}
```

| Имя | Значение | Тип |
|-----|-------------|-----------|
| arg | 0x00000001 | int |
| arg | 0x00f68588 | wchar_t** |
| dr | 3.140000000 | double |
| fr | 3.200000005 | float |
| cr | 0x62 'b' | char |
| ir | 0x00000005 | int |

С помощью утилиты **dumpbin** можно просмотреть *таблицы объектного кода*:

```
Командная строка разработчика для VS2013

D:\Adel\Кафедра\ЯП_Наб\Наб9\LP Lab09\Ftemplate\Debug>dumpbin /symbols Ftemplate.obj
Microsoft (R) COFF/PE Dumper Version 12.00.21005.1
Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file Ftemplate.obj

File Type: COFF OBJECT

COFF SYMBOL TABLE
000 00E1520D ABS      notype      Static      | @comp.id
001 80000391 ABS      notype      Static      | @feat.00
002 00000000 SECT1    notype      Static      | .drectve
      Section length  F4, #relocs    0, #linenums    0, checksum      0
```

```
      Section length 140, #relocs    5, #linenums    0, checksum      0, selection    5 (pick associative Section 0xC)
Relocation CRC 166946A1
029 00000000 SECTC    notype ()      External    | _wmain
02A 00000000 SECT6    notype ()      External    | ??$max@H@@YAHHH@Z (int __cdecl max<int>(int,int))
02B 00000000 SECT8    notype ()      External    | ??$max@M@@YAMMM@Z (float __cdecl max<float>(float,float))
02C 00000000 SECTA    notype ()      External    | ??$max@N@@YANNNG@Z (double __cdecl max<double>(double,double))
02D 00000000 SECT4    notype ()      External    | ??$max@D@@YADDD@Z (char __cdecl max<char>(char,char))
02E 00000000 UNDEF    notype ()      External    | __RTC_CheckEsp
02F 00000000 UNDEF    notype ()      External    | __RTC_InitBase
030 00000000 UNDEF    notype ()      External    | __RTC_Shutdown
031 00000000 SECTE    notype      Static      | .rtc$IMZ
```