

Основы программной инженерии (ПОИТ)  
Технологии разработки программного обеспечения (ИСИТ)

Структура системы программирования. Программные конструкции

План лекции:

- программные конструкции (блоки, функции, процедуры);
- объявление функции, определение функции, передача параметров в функцию;
- способы передачи параметров;
- область видимости переменных;
- программные библиотеки;
- модель памяти C/C++ (классы памяти);
- среда разработки: понятие и назначение дизассемблера.

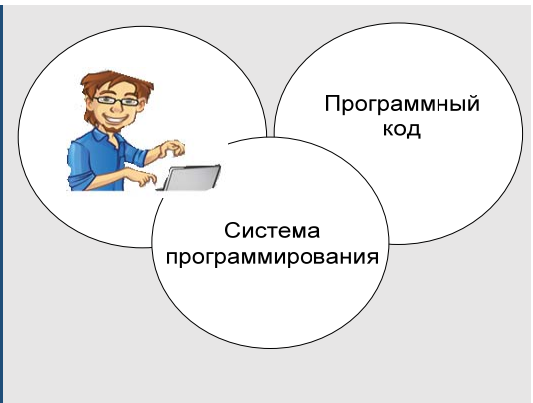
1. На прошлых лекциях:

Система программирования

**Система программирования:**

**комплекс программных средств**, предназначенных для автоматизации процесса разработки, отладки программного обеспечения и подготовки программного кода к выполнению

**Система**, образуемая языком программирования, компиляторами или интерпретаторами программ, представленных на этом языке, соответствующей документацией, а также вспомогательными средствами для подготовки программ к форме, пригодной для выполнения



**Язык программирования:**



**формальная знаковая система**, предназначенная для записи компьютерных программ.

Знаковая система определяет набор **лексических, синтаксических и семантических** правил написания программы (программного кода).

Язык программирования представляется в виде набора спецификаций, определяющих его синтаксис и семантику.

## Символы времени трансляции, символы времени выполнения:

**Алфавит языка программирования** – набор символов, разрешенных к использованию языком программирования. Основывается на одной из кодировок.

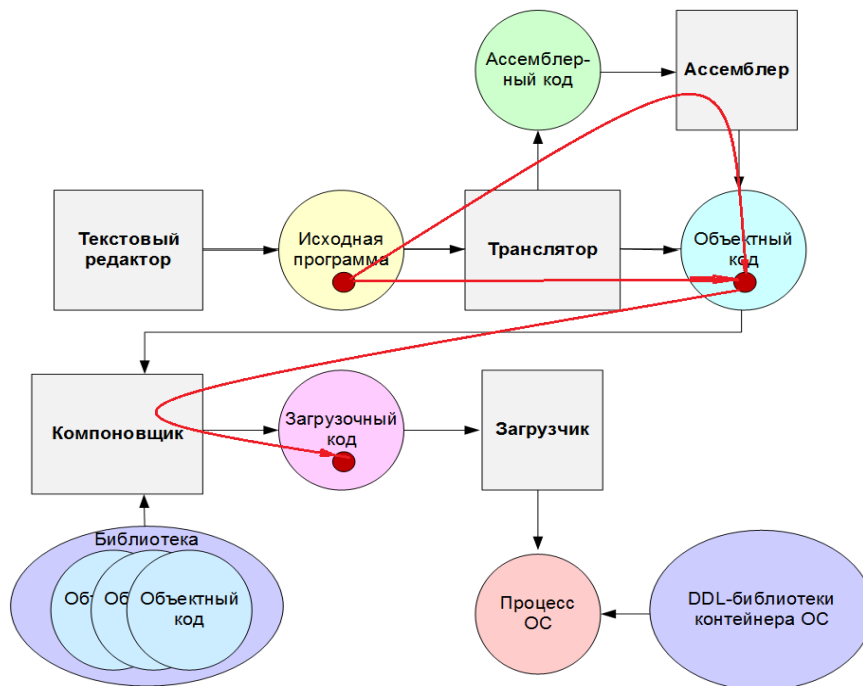
<b>Набор символов времени трансляции:</b>	текст программы на языке программирования хранится в исходных файлах и основан на определенной кодировке символов
<b>Набор символов времени выполнения:</b>	символы, отображаемыми в среде выполнения; дополнительные символы зависят от локализации

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

## Структура языка программирования

- ✓ *алфавит языка;*
- ✓ *идентификаторы;*
- ✓ *фундаментальные (встроенные) и пользовательские типы данных;*
- ✓ *преобразование типов:* явное и неявное (автоматическое);
- ✓ *инициализация памяти:* присвоение значения в момент объявления переменной;
- ✓ *константное выражение:* выражение, вычисляемое компилятором;
- ✓ *область видимости переменных:* доступность переменных по их идентификатору в разных частях программы;
- ✓ *пространства имен;*
- ✓ *выражения;*
- ✓ *инструкции языка:*
  - присваивания;
  - инструкции объявления;
  - блок вычислений;
  - ветвление;
  - циклы;
  - инструкции перехода;
  - обработка исключений;
- ✓ *программные конструкции* (декомпозиция программного кода):  
процедуры, функции, методы, ...

## От исходного кода к исполняемому модулю:



<p><b>Компилятор (транслятор)</b> – программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке; результат – объектный модуль</p>	<p><b>Компоновщик</b> (linker, редактор связей) – программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль</p>	<p><b>Загрузчик</b> (loader) – программа, предназначенная для запуска процесса операционной системы на основе загрузочного модуля</p>

**Программа = алгоритм + данные**

## 2. Программные конструкции (блоки, функции, процедуры, модули)

**Инструкция** (или оператор (англ. statement) – это одна команда языка программирования; наименьшая законченная часть.

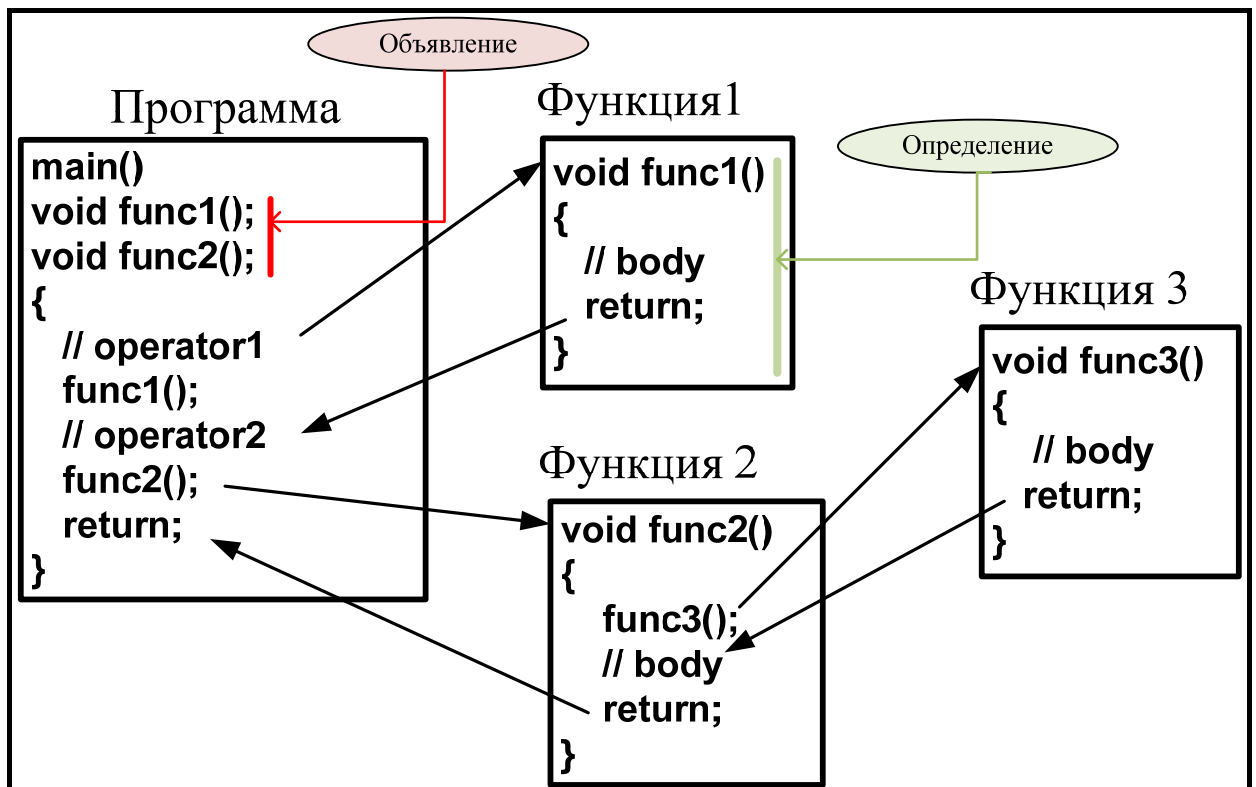
**Блок** (или составной оператор) – это группа логически связанных между собой инструкций языка программирования, рассматриваемых как единое целое.

В языках программирования инструкции могут быть сгруппированы в специальные логически связанные вычислительные блоки следующего вида:

Псевдокод	C/C++, C#, Java	Pascal/Delphi	Python
Начало <оператор> <оператор> ... Конец	{ <оператор> <оператор> ... }	begin <оператор> <оператор> ... end	<оператор> <оператор> ...

Вычислительный блок называют составным оператором.

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



<b>Имя функции</b>	С функцией связывается <i><b>идентификатор</b></i> (имя функции). Некоторые языки допускают безымянные функции
<b>Адрес функции</b>	С <i><b>именем функции</b></i> неразрывно связан <i><b>адрес ее первой инструкции</b></i> , которой передаётся управление при обращении к функции
<b>Адрес возврата</b>	После выполнения функции управление возвращается в точку программы, где данная функция была вызвана ( <i><b>адрес возврата</b></i> )

- ✓ функция может ***принимать параметры*** и может ***возвращать некоторое значение***;
- ✓ функции, которые возвращают ***пустое*** значение, называют ***процедурами***;
- ✓ функция должна быть объявлена и определена.

#### **Объявление функции содержит:**

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

**Определение функции** содержит исполняемый код функции.

#### **Вызов функции:**

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

#### **Способы передачи параметров в функцию:**

по значению;  
 по ссылке.

### 3. Способы передачи параметров в функцию

<b>по значению</b>	создается локальная копия переменной; любые изменения переменной в теле функции происходят с локальной копией; значение самой переменной не изменяется.
<b>по ссылке</b>	изменения происходят с самой переменной по адресу ее размещения в памяти.

Для переменной, переданной по ссылке функция определяет собственную (локальную) область видимости. В нее входят входные параметры и 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.

Примеры функций на различных языках программирования:

<b>C/C++</b>	<b>Visual Basic</b>	<b>Python</b>
<pre>int getMul(int x, int y) {     return x * y; };</pre>	<pre>Sub Name(text)     Console.WriteLine(text) End Sub</pre>	<pre>def func(text):     print(text)</pre>

#### 4. Область видимости переменных:

**Область видимости переменных** – доступность переменных по их идентификатору в разных частях (блоках программы).

##### **Область видимости переменных в C++:**

- переменная должна быть объявлена до ее использования;
- переменная объявленная во внутреннем блоке (локальная переменная) не доступна во внешнем;
- переменная объявленная во внешнем блоке доступна во внутреннем;
- во внутреннем блоке переменная может быть переобъявлена.

**Область видимости переменной** (идентификатора) зависит от места ее объявления в тексте программы.

**Область действия идентификатора** – это часть программы, в которой его можно использовать для доступа к связанной с ним области памяти.

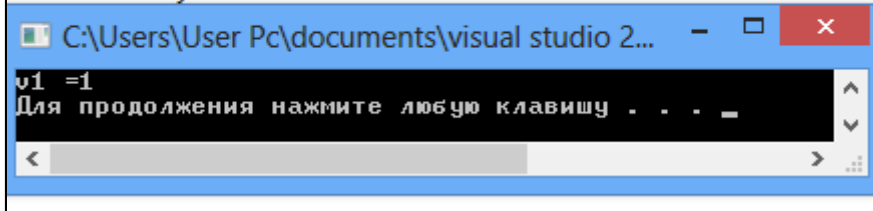
В зависимости от области действия переменная может быть *локальной* или *глобальной*.

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

int v1 = 1;

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout<<"v1 ="<< v1 <<std::endl;

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



## Оператор разрешения области видимости «::» (два двоеточия)

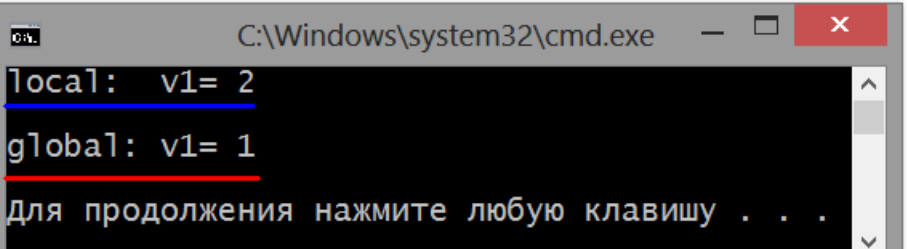
```
#include "stdafx.h"
#include <iostream>

int v1 = 1;

int _tmain(int argc, _TCHAR* argv[])
{
    int v1 = 2;
    std::cout << "local: v1= " << v1 << std::endl << std::endl;

    std::cout << "global: v1= " << ::v1 << std::endl << std::endl;

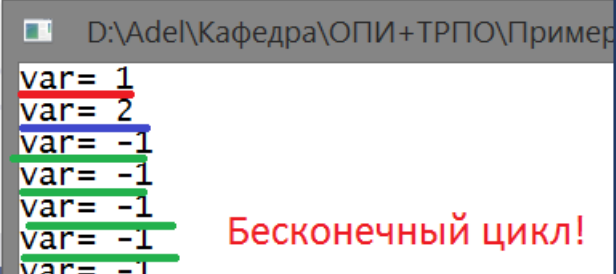
    return 0;
}
```



```
#include <iostream>

int var = 1;

int main()
{
    std::cout << "var= " << var << std::endl;
    int var = 2;
    std::cout << "var= " << var << std::endl;
    while (var > 0) {
        int var = 0;
        --var;
        std::cout << "var= " << var << std::endl;
    }
    system("pause");
    return 0;
}
```



Бесконечный цикл!



## 5. Программные библиотеки

**Назначение библиотек** – предоставить программисту стандартный простой и надёжный механизм повторного использования кода.

При выполнении определенных *соглашений* библиотеки можно использовать в программных проектах, реализуемых на нескольких языках программирования.

### *Классификация библиотек:*

- библиотеки на языках программирования (библиотеки классов, шаблонов, функций и т. п.). Компилируются *вместе с исходными файлами проекта*;
- библиотеки объектных модулей (статические библиотеки). Компилируются *вместе с объектными файлами проекта*;
- библиотеки исполняемых модулей (динамические библиотеки). Загружаются в память в момент запуска программы или во время ее исполнения, по мере надобности.

## Типы библиотек:

статические;  
динамические.

**Статическая библиотека** (англ. library) — набор подпрограмм или объектов, которые подключаются к исходной программе в виде объектных файлов во время *компоновки*.

### Использование библиотек:

- отлаженные функции, разработанные в больших проектах, помещают в библиотеку (время трансляции уменьшается);
- функции из библиотеки можно вызывать в разных программах.

### Статическая библиотека:

- это файл (в Microsoft с расширением *lib*, в Linux – *a*), содержащий объектные модули;
- является *входным файлом* для компоновщика (*Linker*).

### Преимущества:

- просто использовать;
- не требуется наличие самой библиотеки;
- исполняемый файл один (расширение .exe).

### Недостатки:

- платформенно зависима;
- загружается в память с каждым экземпляром запущенного приложения;
- при изменении кода библиотеки необходима компоновка всех приложений, которые используют библиотеку.

### Динамическая библиотека (или «общая библиотека»):

- набор подпрограмм, которые подгружаются в программу во время её выполнения (в Microsoft с расширением *dll*, в Linux – *so*), содержащий объектные модули;
- является *входным файлом* для компоновщика (*Linker*).

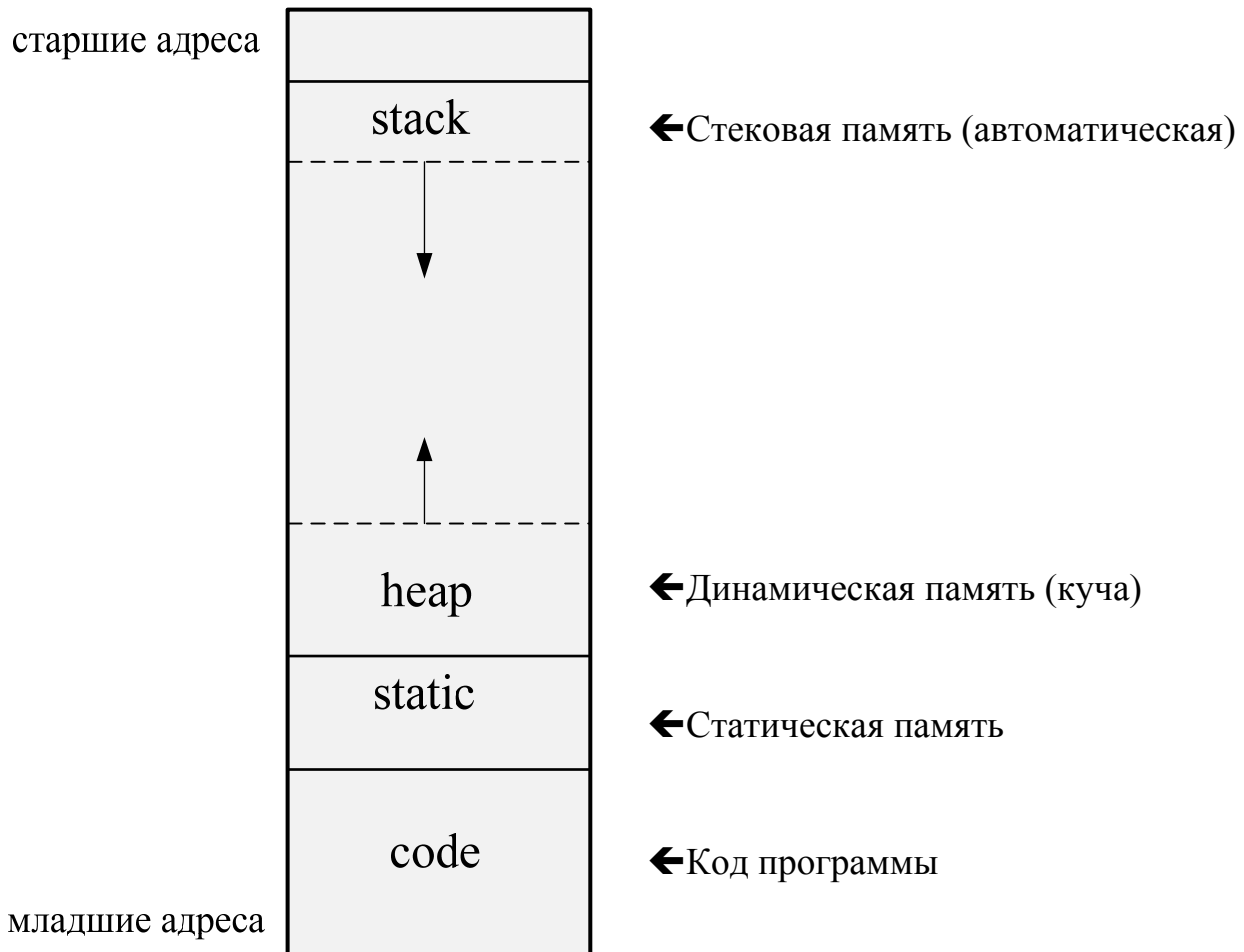
### Преимущества:

- совместное использование одной копии динамической библиотеки несколькими программами (экономит адресного пространства);
- возможность обновления динамической библиотеки до более новой версии без необходимости перекомпиляции всех исполняемых файлов, которые ее используют.

## 6. Модель памяти C/C++:

Модель памяти языка C++ предоставляет области для хранения кода и данных.

Структура памяти C/C++-программ:



**Область кода** – память, в которой размещается код программы.

### **Статическая память**

**Static** или **статическая память** выделяется до начала работы программы, на стадии компиляции и служит для хранения статических переменных.

Типы статических переменных: **глобальные переменные** и **статические переменные**.

**Глобальные** переменные – это переменные, определенные вне функций. Память для глобальных переменных выделяется на этапе компиляции.

Глобальные переменные доступны в любой точке программы во всех ее файлах.

**Статические** переменные – это переменные, в описании которых присутствует ключевое слово `static`. Компилятор выделяет для таких переменных постоянное место хранения в статической области памяти.

При объявлении переменной в функции ключевое слово `static` указывает, что переменная *удерживает свое состояние между вызовами этой функции*.

### **Стековая память**

**Stack** или **стековая (автоматическая) память** предназначена для хранения локальных переменных.

**Локальные** переменные хранятся в стеке.

**Стек** – это непрерывная область оперативной памяти, организованная по принципу LIFO (последний вошел, первый вышел).

### **Динамическая память**

**Heap** или динамическая память, или куча – это область памяти, выделение которой в языке программирования C++ производится с помощью оператора `new`, освобождение — оператором `delete`.

**Пример.** Объявление глобальных статических переменных (компонуются редактором связей (linker)).

```
1  #include <iostream>
2
3  int getSum(int x, int y);
4  int getMul(int x, int y);
5
6  int parm1 = 2;
7  int parm2 = 3;
8
9  int main(int argc, char* argv[])
10 {
11     std::cout << "getSum(" << parm1 << "," << parm2 << ") = " << getSum(2, 3) << std::endl;
12
13     parm1 = 5;
14     parm2 = 5;
15     std::cout << "getMul(" << parm1 << "," << parm2 << ") = " << getMul(2, 3) << std::endl;
16
17     system("pause");
18     return 0;
19 }
```

```
1  int getSum(int x, int y) { return x + y; };
2
3  int getMul(int x, int y) { return x * y; };
4
```

Обозреватель решений — поиск (C...  
Решение "Lec08" (проекты: 1 и...  
Lec08  
Ссылки  
Внешние зависимости  
Исходные файлы  
file\_mul.cpp  
file\_sum.cpp  
lec\_08.cpp  
Файлы заголовков  
Файлы ресурсов

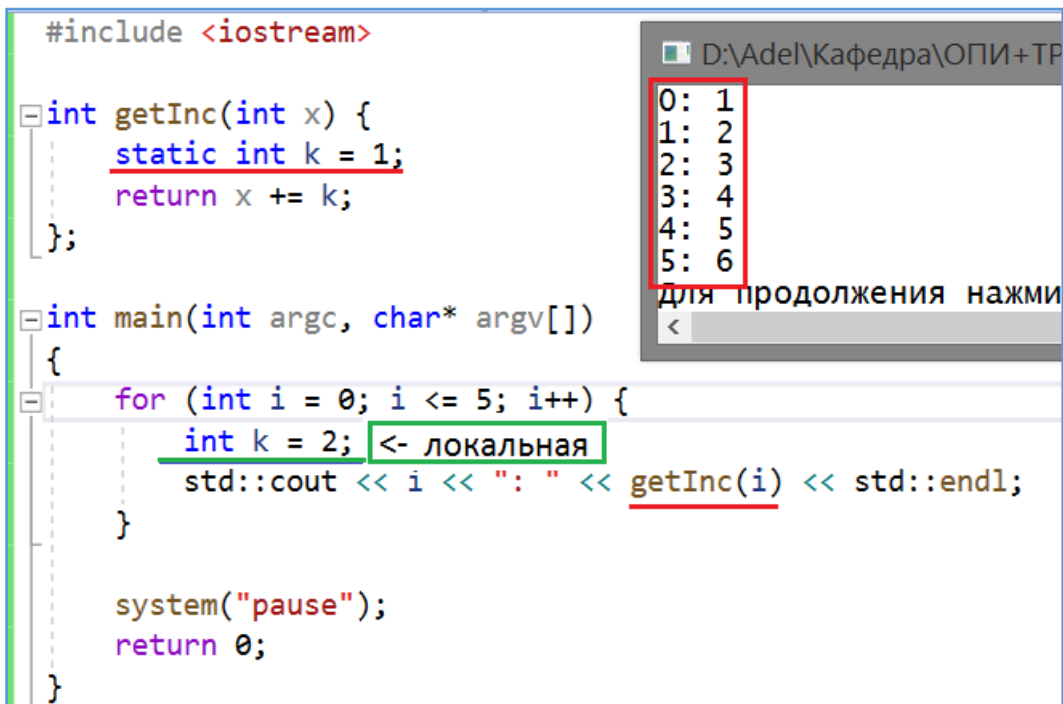
Вывод

Показать выходные данные из: Сборка

```
1>Целевой объект InitializeBuildStatus:
1> Создание "Debug\Lec08.tlog\unsuccessfulbuild", так как было задано "AlwaysCreate".
1>Целевой объект ClCompile:
1> file_mul.cpp
1> file_sum.cpp
1> lec_08.cpp
1> Создание кода...
1>Целевой объект Link:
1> Lec08.vcxproj -> D:\Adel\Кафедра\ОПИ+ТРПО\Примеры к лабораторным работам\Lec08\Debug\Lec08.exe
1>Целевой объект FinalizeBuildStatus:
1> Файл "Debug\Lec08.tlog\unsuccessfulbuild" удаляется.
1> Обработка "Debug\Lec08.tlog\Lec08.lastbuildstate"
```

```
D:\Adel\Кафедра\ОПИ+ТРПО\Примеры к лабораторн
getSum(2,3) = 5
getMul(5,5) = 6
Для продолжения нажмите любую клавишу . . .
```

## Локальная статическая память.



```
#include <iostream>

int getInc(int x) {
    static int k = 1;
    return x += k;
};

int main(int argc, char* argv[])
{
    for (int i = 0; i <= 5; i++) {
        int k = 2; <- локальная
        std::cout << i << ": " << getInc(i) << std::endl;
    }

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

The output window shows the following results:

0:	1
1:	2
2:	3
3:	4
4:	5
5:	6

Для продолжения нажми  
<

Переменная с ключевым словом `static` — это **статическая** переменная. Время ее жизни — постоянное.

Область видимости статической переменной ограничена одним файлом, внутри которого она определена, ее можно использовать только после ее объявления.

Ключевое слово `static` в языке C/C++ используется для двух различных целей:

- как указание типа памяти: переменная располагается в статической памяти;
- как способ ограничить область видимости переменной рамками одного файла (в случае описания переменной вне функции).

## Стек

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

int func2 (int y, int z)
{
    int g = 8;
    int h = 9;
    int r = 10;
    return y + z + g + h + r;
};

int func1 (int x, int v)
{
    int k = 5;
    int l = 6;
    int m = 7;
    return k+= func2(k, 7);
};

int main(int argc, _TCHAR* argv[])
{
    std::cout << func1(3, 4)<< std::endl;
    return 0;
}
```

argc

argv

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

int func2 (int y, int z)
{
    int g = 8;
    int h = 9;
    int r = 10;
    return y + z + g + h + r;
};

int func1 (int x, int v)
{
    int k = 5;
    int l = 6;
    int m = 7;
    return k+= func2(k, 7);
};

int main(int argc, _TCHAR* argv[])
{
    std::cout << func1(3, 4)<< std::endl;
    return 0;
}
```

m

l

k

x

v

argc

argv

```

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

int func2 (int y, int z)
{
    int g = 8;
    int h = 9;
    int r = 10;
    return y + z + g + h + r;
};

int func1 (int x, int v)
{
    int k = 5;
    int l = 6;
    int m = 7;
    return k+= func2(k, 7);
};

int main(int argc, _TCHAR* argv[])
{
    std::cout << func1(3, 4)<< std::endl;
    return 0;
}

```

r
h
g
y
z
m
l
k
x
v
argc
argv

```

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

int func2 (int y, int z)
{
    int g = 8;
    int h = 9;
    int r = 10;
    return y + z + g + h + r;
};

int func1 (int x, int v)
{
    int k = 5;
    int l = 6;
    int m = 7;
    return k+= func2(k, 7);
};

int main(int argc, _TCHAR* argv[])
{
    std::cout << func1(3, 4)<< std::endl;
    return 0;
}

```

m
l
k
x
v
argc
argv



```

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

int func2 (int y, int z)
{
    int g = 8;
    int h = 9;
    int r = 10;
    return y + z + g + h + r;
};

int func1 (int x, int v)
{
    int k = 5;
    int l = 6;
    int m = 7;
    return k+= func2(k, 7);
};

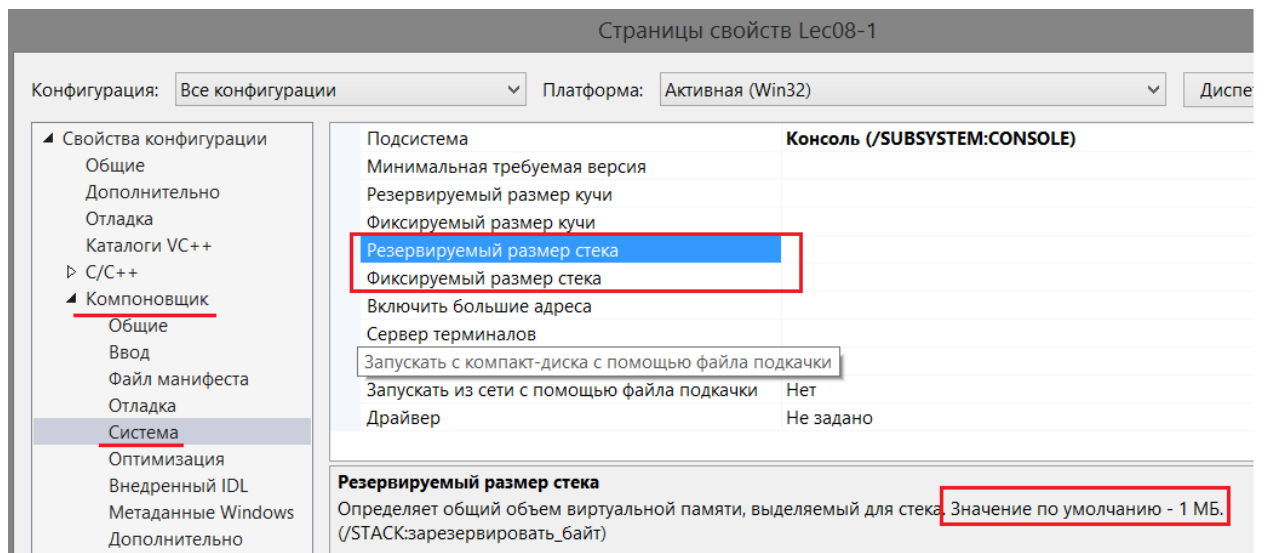
int main(int argc, _TCHAR* argv[])
{
    std::cout << func1(3, 4)<< std::endl;
    return 0;
}

```

argc

argv

Заданный по умолчанию резервируемый размер стека можно изменить в свойствах проекта раздела Компоновщик → система.



Неар (куча).

Выделение памяти в C++ производится с помощью оператора `new`, освобождение — оператором `delete`. Память, выделяемая функциями динамического распределения памяти, находится в куче (heap).

