Лекция 08
БГТУ, ФИТ, 1 семестр

Основы программной инженерии (ПИ)

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.

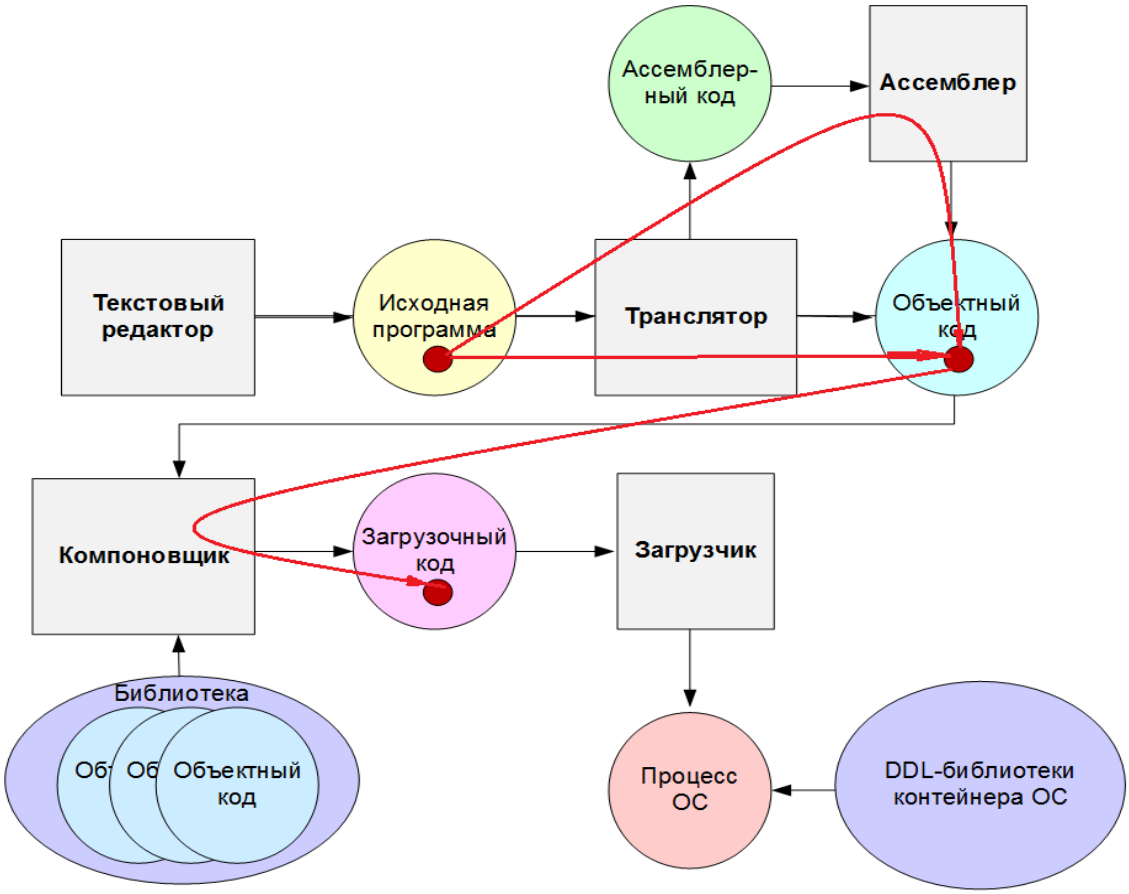
Набор символов времени трансляции:	текст программы на языке программирования хранится в исходных файлах и основан на определенной кодировке символов
Набор символов времени выполнения:	символы, отображаемыми в среде выполнения; дополнительные символы зависят от локализации

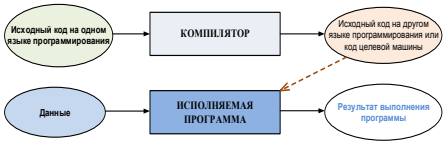
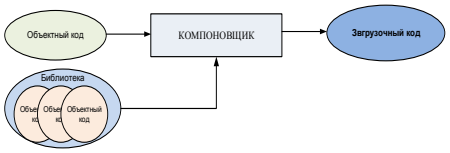
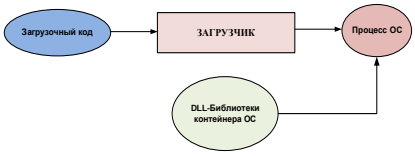
Лексемы	***идентификаторы;*** ***ключевые (зарезервированные) слова;*** ***знаки операций;*** ***константы;*** ***разделители*** (скобки, знаки операций, точка, запятая, пробельные символы и т.д.).

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:



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

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.

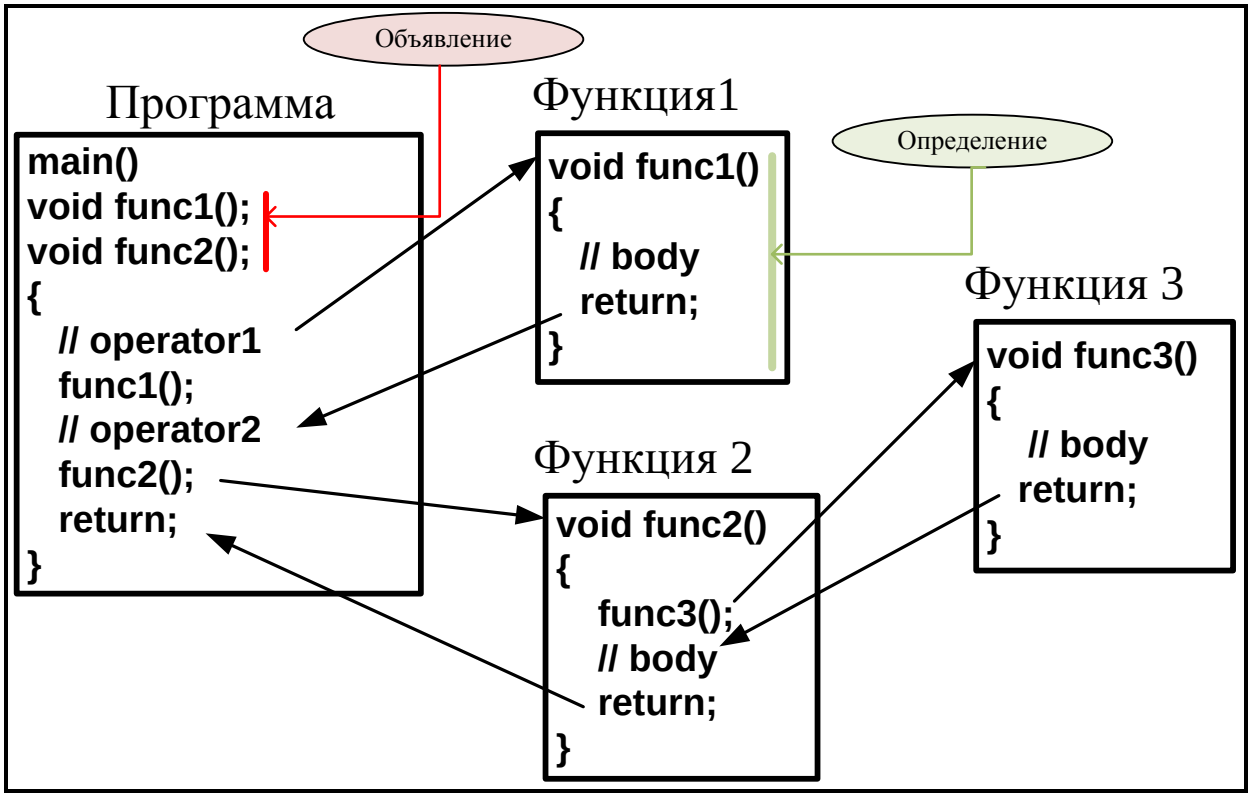
В языках программирования инструкции могут быть сгруппированы в специальные логически связанные вычислительные блоки следующего вида:

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.



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

- ✓ функция может ***принимать параметры***;
- ✓ функция 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

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

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:

C/C++	Visual Basic	Python

`int getMul(int x, int y) {`
`  return x * y;`
`};`
	`Sub Name(text)`
	
	`Console.WriteLine(text)`
	`End Sub`
		`def func(text):`
		`  print(text)`

4. Область видимости переменных:

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

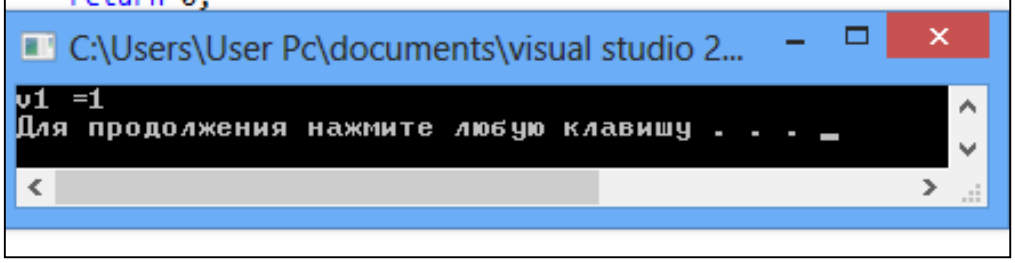
Область видимости переменных в С++:
– переменная должна быть объявлена до ее использования;
– переменная объявленная во внутреннем блоке (локальная переменная) не доступна во внешнем;
– переменная объявленная во внешнем блоке доступна во внутреннем;
– во внутреннем блоке переменная может быть переобъявлена.

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;
}
```

```
C:\Windows\system32\cmd.exe
local:  v1= 2

global: v1= 1

Для продолжения нажмите любую клавишу . . .
```

```
#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;
}
```

```
D:\Adel\Кафедра\ОПИ+ТРПО\Пример
var= 1
var= 2
var= -1
var= -1
var= -1
var= -1
var= -1
```

Бесконечный цикл!

Проблемы не найдены.

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»):

файл, содержащий машинный код (в Microsoft с расширением ***dll***, в Linux – ***so***, в Mac OS – ***dylib***);
Загружается в память процесса загрузчиком программ операционной системы либо при создании процесса, либо по запросу уже работающего процесса, то есть динамически.

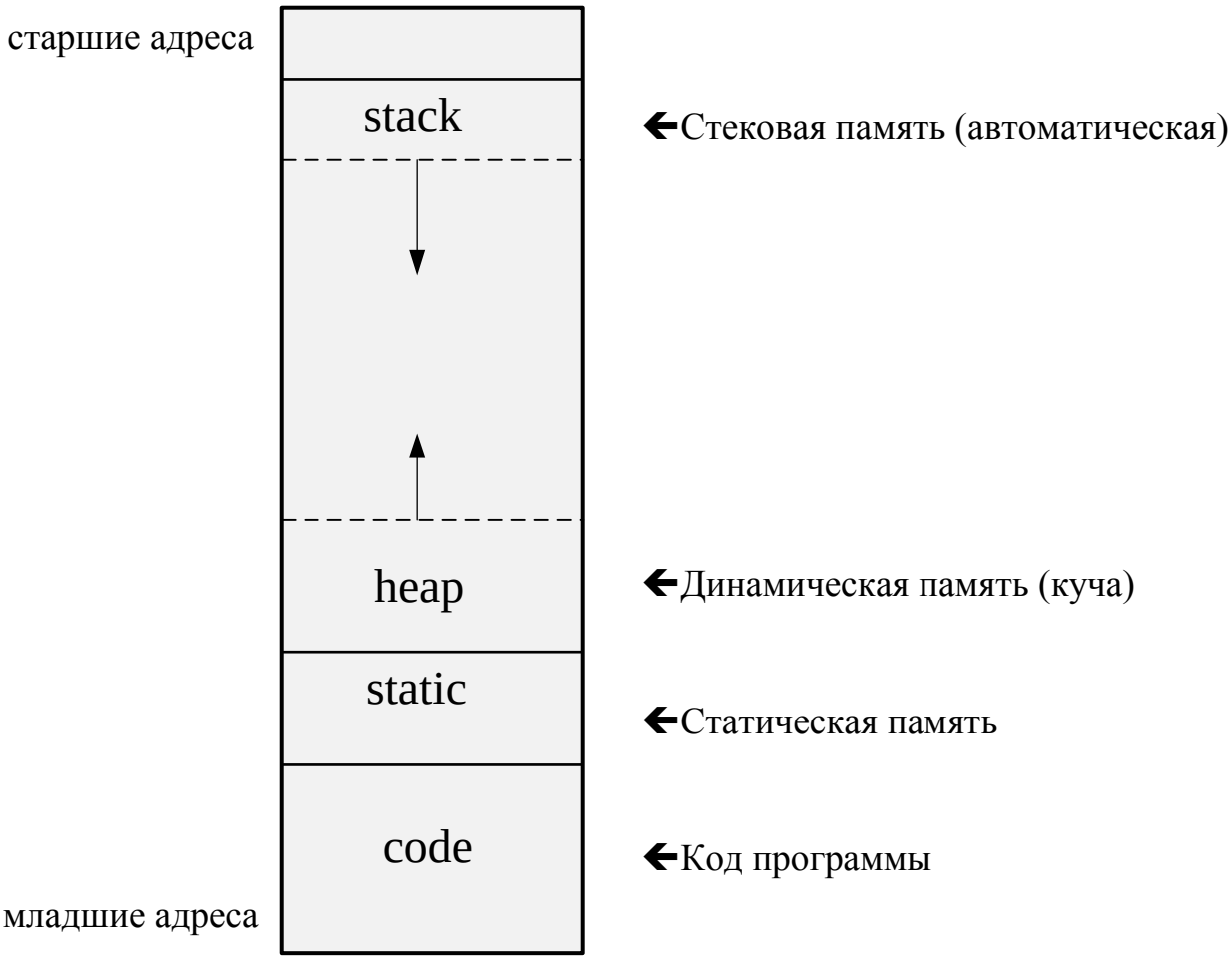
Преимущества:

- совместное использование одной копии динамической библиотеки несколькими программами (экономия 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++ представляет 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

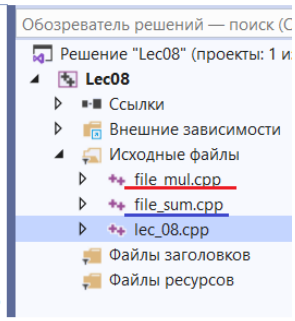
Heap или динамическая память, или куча – это область памяти, выделение которой в языке программирования C++ производится с помощью оператора `new`, освобождение – оператором `delete`.

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

```
#include <iostream>

int getSum(int x, int y);
int getMul(int x, int y);

int parm1 = 2;
int parm2 = 3;

int main(int argc, char* argv[])
{
    std::cout << "getSum(" << parm1 << "," << parm2 << ") = " << getSum(2, 3) << std::endl;

    parm1 = 5;
    parm2 = 5;
    std::cout << "getMul(" << parm1 << "," << parm2 << ") = " << getMul(2, 3) << std::endl;

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

```
int getSum(int x, int y) { return x + y; };
```

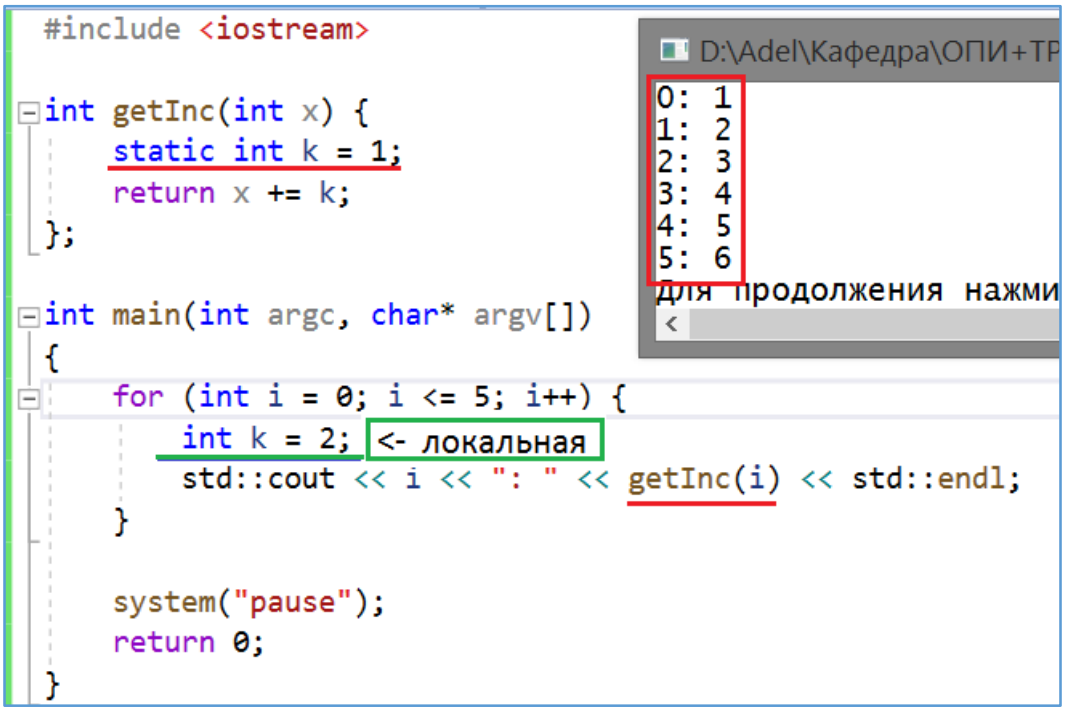
```
int getMul(int x, int y) { return x * y; };
```




```
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.
```

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

Локальная статическая память.



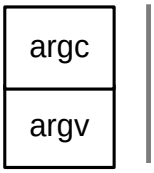
Переменная с ключевым словом `static` – это **статическая** переменная. Время ее жизни – постоянное.
Область видимости статической переменной ограничена одним файлом, внутри которого она определена, ее можно использовать только после ее объявления.

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

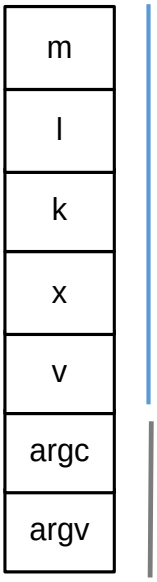
- как указание типа памяти: переменная располагается в статической памяти;
- как способ ограничить область видимости переменной рамками одного файла (в случае описания переменной вне функции).

Стек

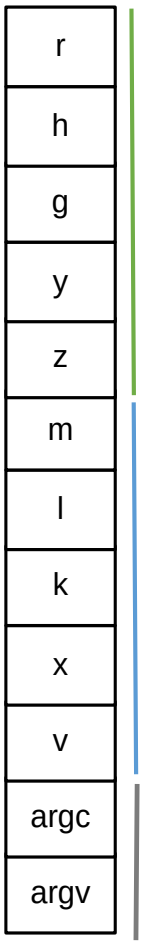
```
#include "stdafx.h"
#include <locale>
#include <iostream>

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;
}
```



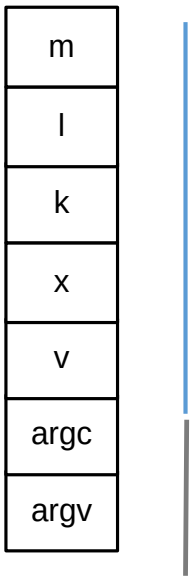

```
#include "stdafx.h"
#include <locale>
#include <iostream>

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;
}
```



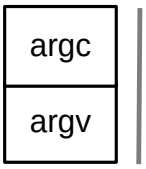
```
#include "stdafx.h"
#include <locale>
#include <iostream>

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;
}
```



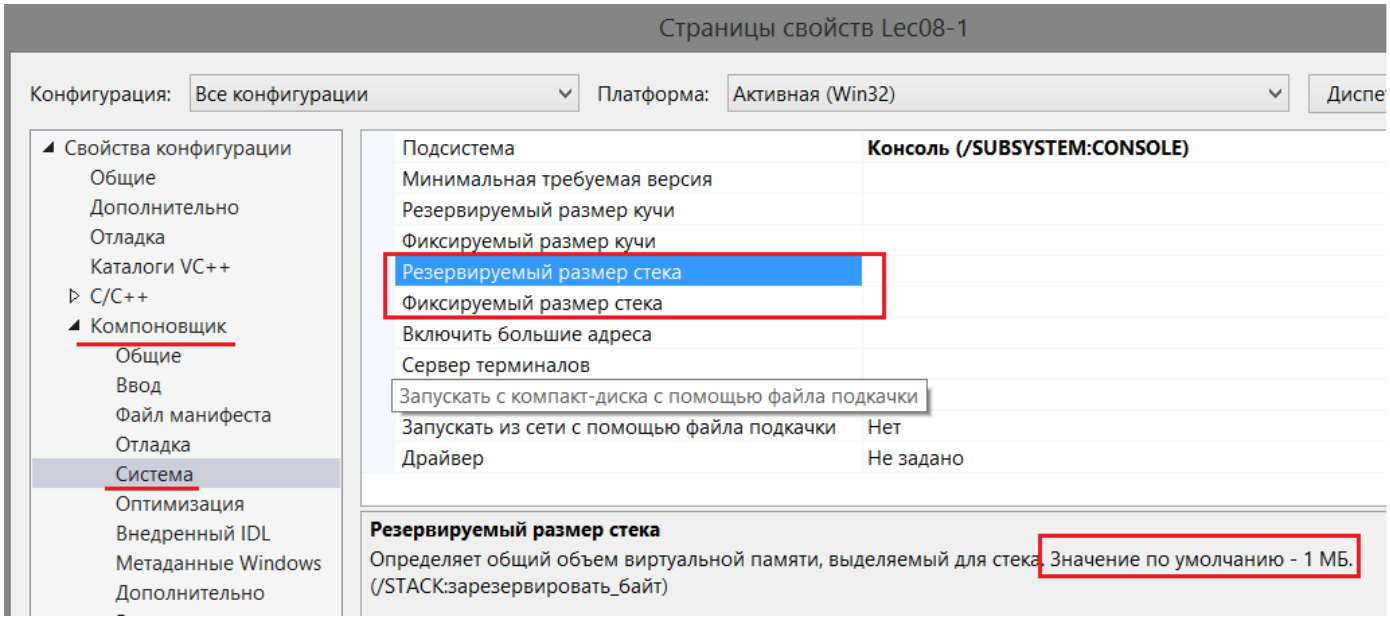

```
#include "stdafx.h"
#include <locale>
#include <iostream>

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;
}
```



```
#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;
}
```



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



Heap (куча).

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