

Тема 3-3.

Механизм вызова функций

для АСУБ и ЭВМб

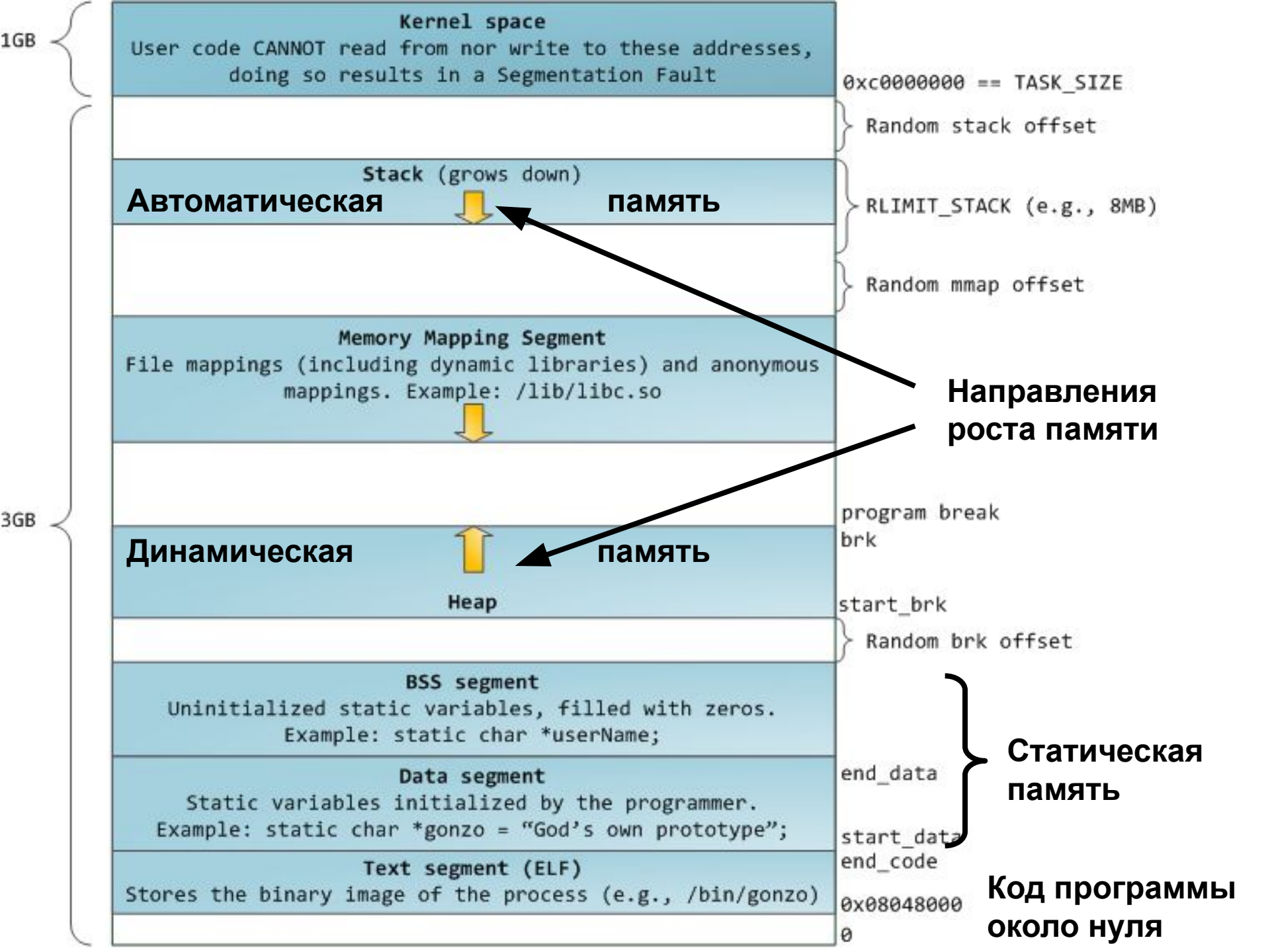
Темы лекции

- Модульное программирование и функциональная декомпозиция
- Использование функций в C++
- Особенности передачи параметров
- Сквозной пример II

Статическая память — это область памяти, выделяемая при запуске программы до вызова главной функции (*main*) из свободной оперативной памяти для размещения **глобальных и статических данных** программы.

Автоматическая память — это область памяти, *резервируемая при запуске программы* до вызова главной функции (*main*) из свободной оперативной памяти и *используемый в дальнейшем* для размещения **локальных данных**.

Динамическая память — это совокупность блоков памяти, выделяемых из доступной свободной оперативной памяти непосредственно **во время выполнения программы** под размещение конкретных данных



Стек

- Автоматическая область памяти организована в форме стек
- Стек поддерживается аппаратно центральным процессором
- Слово “стек” (stack) можно перевести как “стопка”. Объекты добавляются на стек сверху и снимаются потом в обратном порядке.



Стек можно представить в виде трубки с подпружиненным дном, расположенной вертикально.

Верхний конец трубки открыт, в него можно добавлять, или, как говорят, заталкивать элементы

Стек и функции

- Одно из главных назначений стека — поддержка вызовов функций (подпрограмм).
- При вызове функций надо сохранить адрес возврата, чтобы подпрограмма могла по окончании своей работы вернуть управление вызвавшей ее программе.

Кадр стека

- **Кадр стека** (*stack frame*) — часть стека, сформированный одним вызовом функции.
- **Кадр стека** – механизм передачи аргументов и выделения временной памяти с использованием *системного* стека
- Во время отладки отладчик позволяет “проходить” по кадрам стека вызовов, сформированных на данный момент.
- Кадр стека позволяет реализовать **рекурсию**



Стек используется для:

- Выделения и освобождения памяти под локальные переменные
- Вызова функции call **name**
 - поместить в стек адрес команды, следующей за командой call
 - передать управление по адресу метки name
- Возврата из функции
 - извлечь из стека адрес возврата address
 - передать управление на адрес address

Стек используется для:

передачи параметров в функцию

соглашение о вызове:

- расположение входных данных;
- порядок передачи параметров;
- какая из сторон очищает стек;
- etc

cdecl

- аргументы передаются через стек, справа налево;
- очистку стека производит вызывающая сторона;
- результат функции возвращается через регистр EAX

Организация автоматической ПАМЯТИ

```
void f_1(int a)
{
    char b;
    // ...
}
void f_2(double c)
{
    int d = 1;
    f_1(d);
    // ...
}
int main(void)
{
    double e = 1.0;
    f_2(e);
    // ...
}
```

1. **Вызов main**
2. Создание e
3. **Вызов f_2**
4. Создание c
5. Создание d
6. **Вызов f_1**
7. Создание a
8. Создание b
9. **Завершение f_1**
10. Разрушение b
11. Разрушение a
12. **Завершение f_2**
13. Разрушение d
14. Разрушение c
15. **Завершение main**
16. Разрушение e

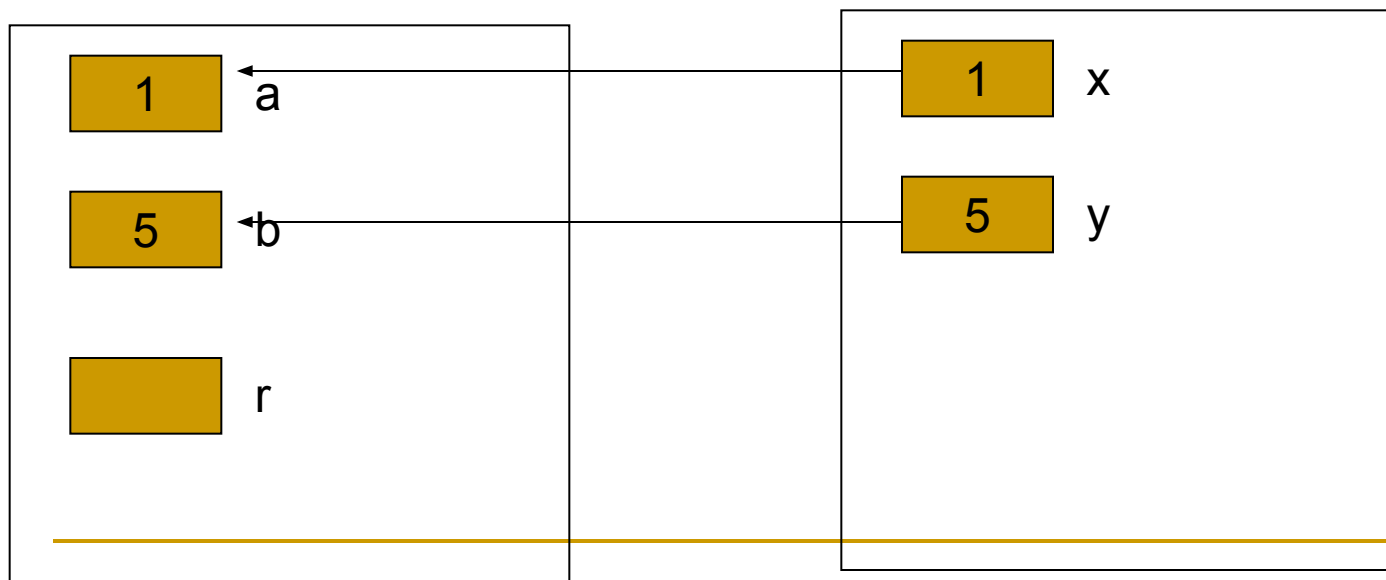
Передача параметров по значению

1. Вычисляются значения выражений, стоящие на месте фактических параметров
2. В стеке выделяется память под формальные параметры функции;
3. Каждому формальному параметру присваивается значение фактического параметра, при этом проверяются соответствия типов и при необходимости выполняются их преобразования.
4. В стек заносятся копии значений аргументов, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, а, следовательно, **нет и возможности их изменить.**

```
void swap (int a, int b) //передача параметра по значению
{
    int r=a;
    a=b;
    b=r;
}
```

//ВЫЗОВ ФУНКЦИИ

```
int x=1,y=5;
swap(x,y);
cout << "x = " << x << " y = " << y;
```



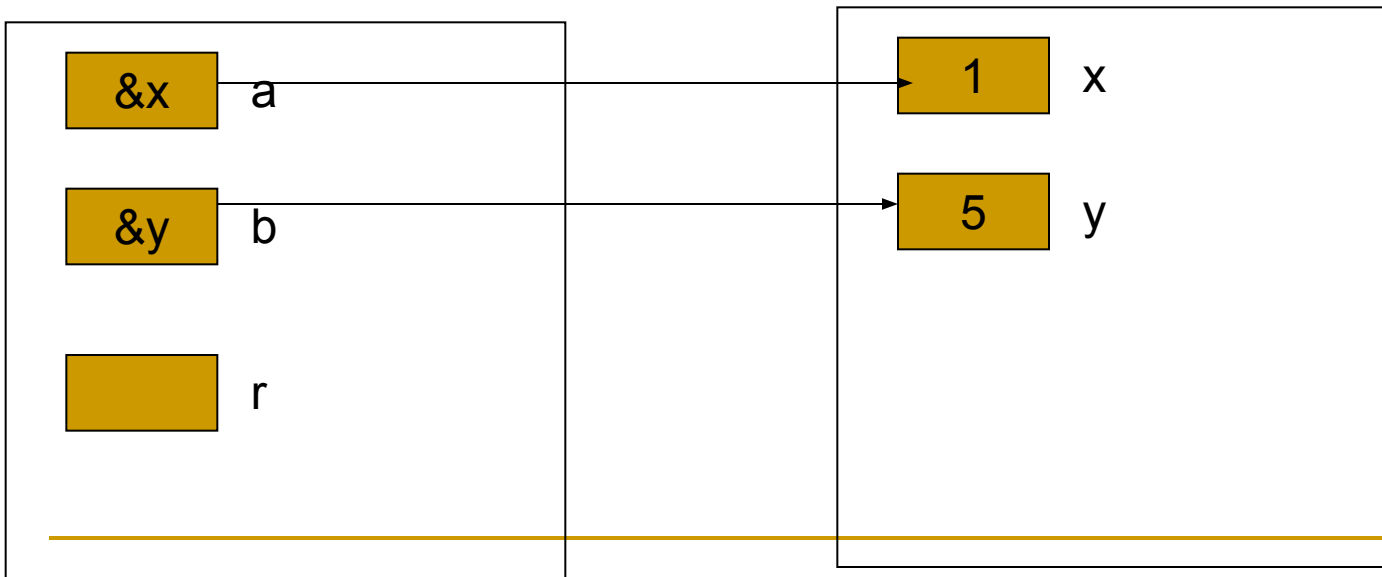
Передача параметров по адресу

- В стек заносятся копии адресов параметров, следовательно, у функции появляется доступ к ячейке памяти, в которой находится фактический параметр и она может его изменить.

```
void swap (int* a, int* b) //передача по адресу (с помощью указателей)
{
int r=*a;
*a=*b;
*b=r;
}
```

//ВЫЗОВ функции

```
int x=1,y=5;
swap(&x,&y);
cout << "x = " << x << " y = " << y;;
```



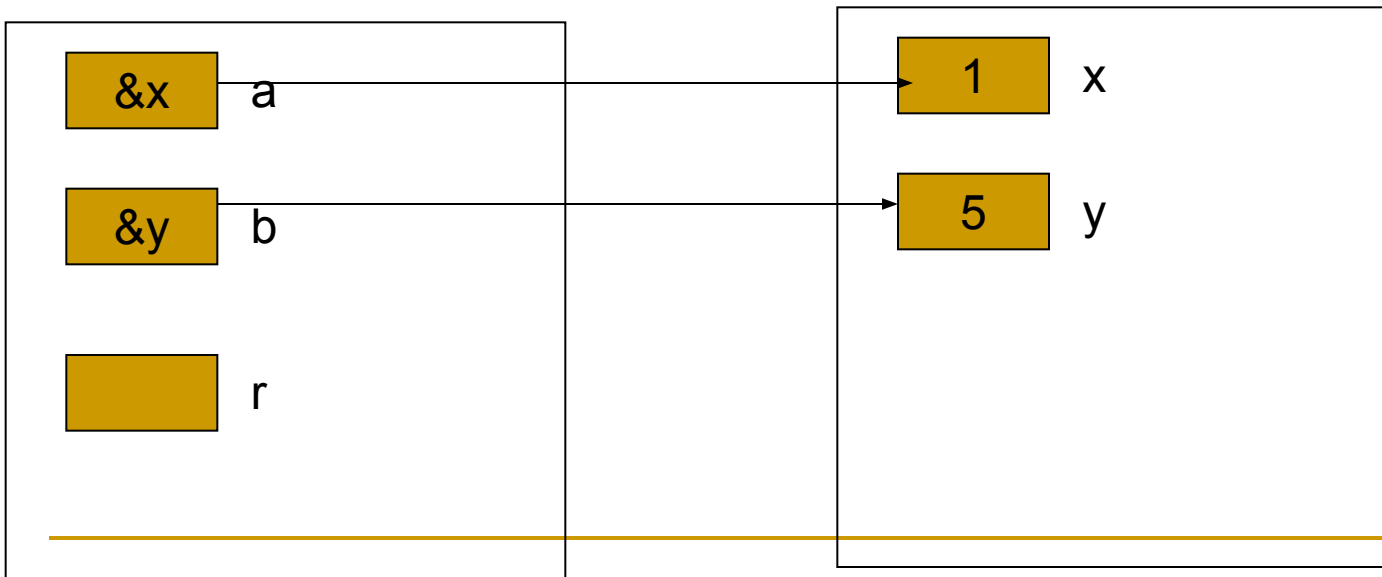
```
void swap (int& a, int& b) //передача по адресу (с помощью ссылки)
{
int r=a;
a=b;
b=r;
}
```

//ВЫЗОВ функции

```
int x=1,y=5;
```

```
swap(x,y);
```

```
cout << "x = " << x << " y = " << y;;
```



Передача массивов в функции

- Когда массив используется в качестве аргумента функции, передается только адрес массива, а не копия всего массива.
- При вызове функции с именем массива в функцию передается указатель на первый элемент массива. Надо помнить, что в C++ имена массивов без индекса - это указатели на первый элемент массива.
- Параметр должен иметь тип, совместимый с указателем. Имеется три способа объявления параметра, предназначенного для получения указателя на массив. Но все три метода объявления параметра приводят к одинаковому результату - указателю.

Передача массивов в функции. Первый способ

```
#include <stdio.h>
void display(int num[10]);
int main (void) /* вывод чисел */
{
    int t [10], i;
    for (i=0; i<10; ++i) t[i]=i;
    display(t);
    return 0;
}
void display(int num[10])
{
    int i;
    for (i=0; i<10; i++) printf ("%d", num[i]);
}
```

0123456789

Process returned 0 (0x0) execution time : 0.859 s

Press any key to continue.

Передача массивов в функции. Первый способ

- Формальный параметр может быть объявлен как массив.
- Хотя параметр `num` объявляется как целочисленный массив из десяти элементов, C++ автоматически преобразует его к целочисленному указателю, поскольку не существует параметра, который мог бы на самом деле принять весь массив.
- Передается только указатель на массив, поэтому должен быть параметр, способный принять его.

Передача массивов в функции. Второй способ

Следующий способ состоит в объявлении параметра для указания на **безразмерный массив**, как показано ниже:

```
void display(int num[])  
{  
int i;  
for (i=0; i<10; i++) printf("%d ", num[i]);  
}
```

где `num` объявлен как целочисленный массив неизвестного размера. Поскольку C++ не предоставляет проверку границ массива, настоящий размер массива не имеет никакого отношения к параметру (но, естественно, не к программе). Данный метод объявления также определяет `num` как целочисленный указатель.

Передача многомерного массива в функцию

При передаче массивов более высоких измерений только первое измерение может быть открыто, в то время как другие должны быть известны во время компиляции.

- `const unsigned int DIM1 = 3;`
- `const unsigned int DIM2 = 5;`
- `void func(int ary[DIM1][DIM2]) { ... } // (1)`
- `void func(int ary[][DIM2]) { ... } // (2)`

Передача массивов в функции. Третий способ

Через указатель

```
void display(int *num)  
{  
int i;  
for (i=0; i<10; i++) printf ("%d ", num[i]);  
}
```

Он допустим, поскольку любой указатель может быть индексирован с использованием [], если он является массивом.

Передача многомерного массива в функцию

```
#include <iostream.h>
using namespace std;
long InputMatrix(int *matrix ,int Height ,int Weight);
void main(){
    const int sHeight = 6;
    const int sWeight = 6;
    int mat[sHeight][sWeight] = {};
    //Вызов функции
    InputMatrix(&mat[0][0],sHeight,sWeight);}
```

Передача многомерного массива в функцию

```
long InputMatrix(int *matrix ,int Height ,int Weight){  
    for(int i=0; i<Height;i++){  
        for(int j=0;j<Weight;j++){  
            cout<<"INPUT"<<' '<<i<<' '<<j<<'\t';  
  
            cin>>matrix[i*Weight+j]; //ВВОД  
        }  
    }  
    return 0;  
}
```