Алгоритмизация и программирование I

Лекция 6

Выпишите рекурентные соотношения для вычисления слагаемого

$$A) \quad 1 - \frac{z^2}{(2!)^2} + \frac{z^4}{(4!)^2} - \dots$$

$$B) \quad z + \frac{1}{2!} \cdot \frac{z^2}{2} + \frac{1}{3!} \cdot \frac{z^3}{3} + \dots$$

Лекция 8

- Указатели
- Функции

Повторение: объявление переменной

- **Что означает запись: int A=10;**
- **Доступ к объявленной переменной осуществляется *по ее имени*.**
- **При этом все обращения к переменной заменяются на адрес ячейки памяти, в которой хранится ее значение.**
- **При завершении программы или функции, в которой была описана переменная, память автоматически освобождается.**

Указатели

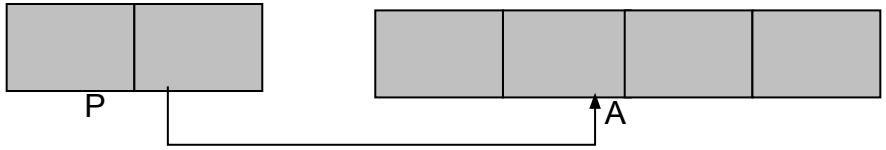
- Указатель – это переменная, в которой хранится адрес другой переменной или участка памяти.
- Объявление указателей:
  - Как и любая переменная, *указатель должен быть объявлен*.
  - При описании переменных-указателей перед именем переменной ставится «*».
  - При объявлении указателей всегда указывается тип объекта, который будет храниться по данному адресу:
  - **тип *имя_переменной;**
  - **Пример: int *a;**
- Звездочка в описании указателя относится непосредственно к имени, поэтому, чтобы объявить несколько указателей, ее ставят перед именем каждого из них:
  - **float *x, y, *z;**

Способы инициализации указателя

- с помощью операции получения адреса

  int a=5;

  int* p=&a; **// или int p(&a);**



- с помощью проинициализированного указателя

  int* r=p;

- адрес присваивается в явном виде

  char* cp=(char*)0x B800 0000;

  где 0x B800 0000 – шестнадцатеричная константа,

  (char*) – операция приведения типа.

- присваивание пустого значения:

  int* N=NULL;

  int* R=0;

Операция получения адреса &

- Операция получения адреса обозначается знаком &.
- Возвращает адрес своего операнда.
  - **float a; //объявлена вещественная переменная a**
  - **float *adr_a; //объявлен указатель на тип float**
  - **adr_a = &a; //оператор записывает в переменную *adr_a* адрес переменной *a***

Операция разадресации (разыменования) *

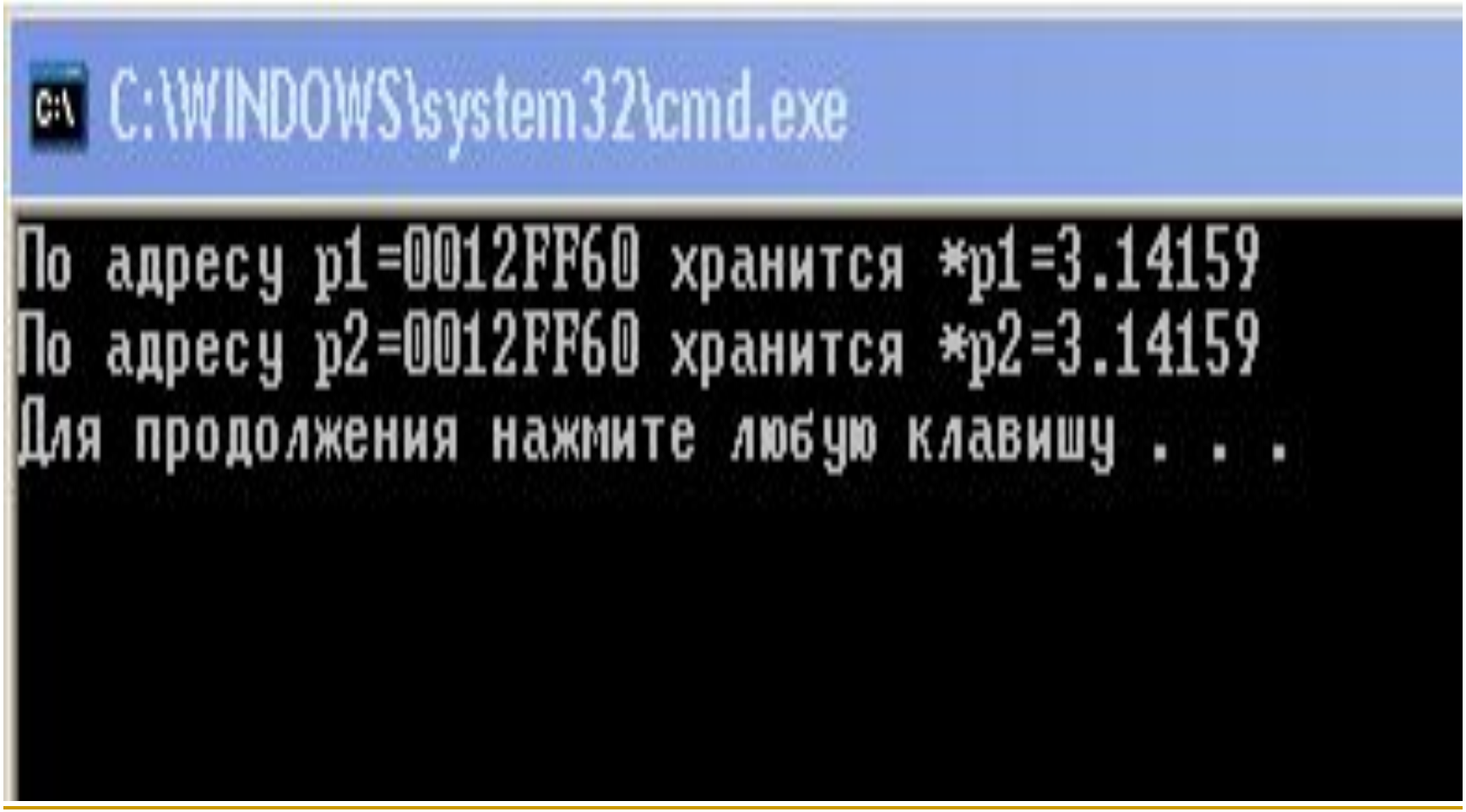
- *Операция разадресации* * возвращает значение переменной, хранящееся по заданному адресу, т.е. выполняет действие, обратное операции &:
  - float a; //Объявлена вещественная переменная *a*
  - float *adr_a; //Объявлен указатель на тип float
  - a=*adr_a; //Оператор записывает в переменную *a* вещественное значение, хранящееся по адресу *adr_a*.

Операции * и & при работе с указателями

Описание	Адрес	Значение, хранящееся по адресу
тип *p;	p	*p
тип p;	&p	p

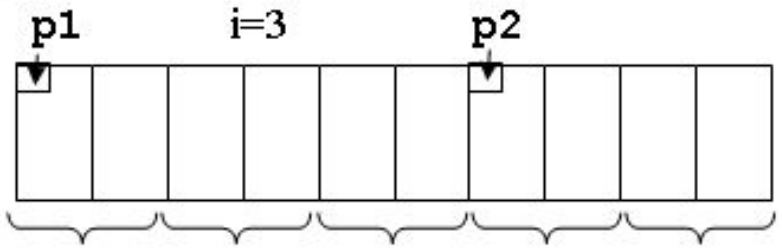
Пример

```
#include <iostream>
#include <locale>
using namespace std;
void main()
{
   setlocale(LC_ALL,"rus");
   float PI=3.14159, *p1, *p2;
   p1=p2=&PI;
  cout<<"По адресу p1="<<p1<<" хранится *p1="
<<*p1<<"\n";
  cout<<"По адресу p2="<<p2<<" хранится *p2="
<<*p2<<"\n";
}
```


C:\WINDOWS\system32\cmd.exe
По адресу p1=0012FF60 хранится *p1=3.14159
По адресу p2=0012FF60 хранится *p2=3.14159
Для продолжения нажмите любую клавишу . . .

Арифметические операции над указателями:

- *сложение* и *вычитание указателей с константой*;
- *вычитание одного указателя из другого*;
- *инкремент*; *декремент*.

ОПЕРАЦИЯ	ПРИМЕР	ОПИСАНИЕ
<указатель>+<целое>	`p2=p1+i;`	В результате этой операции значением переменной **p2** будет являться адрес области памяти, отстоящей от области памяти, адрес которой хранится в переменной **p1**, на количество единиц типа, на данные которого указывают эти указатели:  **i*sizeof**(*<тип>*)

ОПЕРАЦИЯ	ПРИМЕР	ОПИСАНИЕ
<указатель>–<целое>	`p2=p1-i;`	Аналогично сложению
<указатель>++; *<указатель>*– –;	`p1++;` `p2--`	Изменения происходят в единицах типа: `1·sizeof(`*<тип>*`);`
<указатель>– *<указатель>*	`i=p2-p1;`	При выполнении этой операции указатели должны быть одного и того же типа; результат – целое число, абсолютная величина которого указывает, сколько единиц данного типа помещается между адресом памяти первого и второго указателей. Знак этого числа показывает, какой из указателей больше.

Задание

- Чему равно значение переменной A?

```
int A;      // выделяется память
int *P;     // память не выделяется
...
A=10;
P=&A;   // &A – взятие адреса. Выделяется память под P  и в
        нее записывается адрес области памяти, выделенной
        под A.
A++;
*P*=*P;
```

```
A++; // A=11
```

```
*P*=*P; // A=A*A
```

Функции

- Деление программы на функции является базовым принципом структурного программирования.
- Основные свойства и достоинства структурного программирования
  - Преодоление барьера сложности программ.
  - Возможность демонстрации правильности программ на различных этапах решения.
  - Наглядность.
  - Простота модификации

Функции

- Любая последовательность операторов, встречающаяся в программе более одного раза, будучи вынесенной в отдельную функцию, сокращает размер программы.

Объявление и определение функций в языке С

- Функция является
  - во-первых, одним из производных типов С++;
  - во-вторых, минимальным исполняемым модулем программы.

Функция

- Функция – это именованная последовательность описаний и операторов, выполняющая законченное действие.

Объявление и определение функции

- **Объявление функции (прототип, заголовок)** задает имя функции, тип возвращаемого значения и список передаваемых параметров.

//объявление

<тип> <имя_функции> ([<список_формальных_параметров>]);

- **Определение функции** содержит, кроме объявления, тело функции, которое представляет собой последовательность описаний и операторов.

//определение

<тип> <имя_функции> ([<список_формальных_параметров>])

{

<тело_функции>

}

<Тело_функции>::=<блок>| <составной оператор>**.**

- Внутри функции нельзя определить другую функцию

Формальные и фактические параметры

- Список формальных параметров – это те величины, которые требуется передать в функцию.
- Элементы списка разделяются запятыми. Для каждого параметра указывается тип и имя. В объявлении имена можно не указывать.
- При вызове указываются: имя функции и фактические параметры.
- Фактические параметры заменяют формальные параметры при выполнении операторов тела функции.

Пример

```
void func1(int, float);    // объявление функции (;)

void main()
{
    int a; float z;
    func1(a,z);         // вызов функции
}
void func1(int c, float x) // определение функции
{
     // тело функции
}
```

- **В определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать.**

Использование функций

- Объявление функции должно находиться в тексте раньше вызова функции, чтобы компилятор мог осуществить проверку правильности вызова.
- Если функция имеет тип не void, то ее вызов может быть операндом выражения.
- Если функция func () объявлена без параметров, то ее вызовом является имя с пустыми скобками: func().

Возвращаемое значение

- В теле функции может быть оператор, который возвращает полученное значение функции в точку вызова:
  - return <выражение>;
- Используется для возврата результата, поэтому выражение должно иметь тот же **тип**, что и **тип функции** в определении.
- Тип возвращаемого значения может быть любым, кроме массива и функции, но может быть указателем на массив или функцию.

Пример

```
int func1(int, float);// объявление функции

void main()
{
    int a; float z;
    a=func1(a+2,z);      // вызов функции
}
int  func1(int c, float x)     // определение функции
{
                  // тело функции
   return c;
}
```

Пример

- Вычислить значение *y*:

$$y = \frac{\max(a, \max(b, c)) + 4 * \max(a * b, c - a)}{\max(a, b) + \max(c * a - b, c * b - a)}$$

- Удобнее ввести функцию, которая вычисляет максимум из двух чисел: **max (x,z)**.

Программа

```
float max (float x, float y)   // Заголовок
{  float r;          // Локальная переменная
    if (x>y) r=x; else r=y;
    return r;                    //тело функции
}
void main ()
{  int a,b;
   float y,c,d;
   scanf(“%d%d”,&a,&b);
   scanf(“%f%f”,&c,&d);
   y=(max(a,max(b,c))–4*max(d*c–b,a*b+c))/
       (max(a*b–c,c*b)+max(d*a,b-c));
  printf(“\ny=%5.2f”,y);
}
```

Задание

- Заданы координаты сторон треугольника, если такой треугольник существует, то найти его площадь.

Программа

```
#include <iostream.h>
#include <math.h>
/*функция возвращает длину отрезка, заданного координатами x1,y1 и x2,y2*/
double line(double x1,double y1,double x2,double y2)
{
return sqrt(pow(x1-x2,2)+pow(y1-y2,2));
}
```

```
/*функция возвращает площадь треугольника, заданного длинами сторон a,b,c*/
double square(double a, double b, double c)
{
double s, p=(a+b+c) *0.5;
return s=sqrt(p*(p-a)*(p-b)*(p-c));
                    // формула Герона
}
//возвращает true, если треугольник существует
bool triangle(double a, double b, double c)
{
      return (a+b>c&&a+c>b&&c+b>a);
}
```

Программа (продолжение)

```
void main()
{
double x1=1,y1,x2,y2,x3,y3;
double point1_2,point1_3,point2_3;

cout<<"\nEnter koordinats of triangle:";
cin>>x1>>y1>>x2>>y2>>x3>>y3;

point1_2=line(x1,y1,x2,y2);
point1_3=line(x1,y1,x3,y3);
point2_3=line(x2,y2,x3,y3);

If   (triangle(point1_2,point1_3,point2_3))
cout<<"S="<<square(point1_2,point2_3,point1_3)<<"\n";
    else cout<<"\nTriagle doesnt exist";
}
```

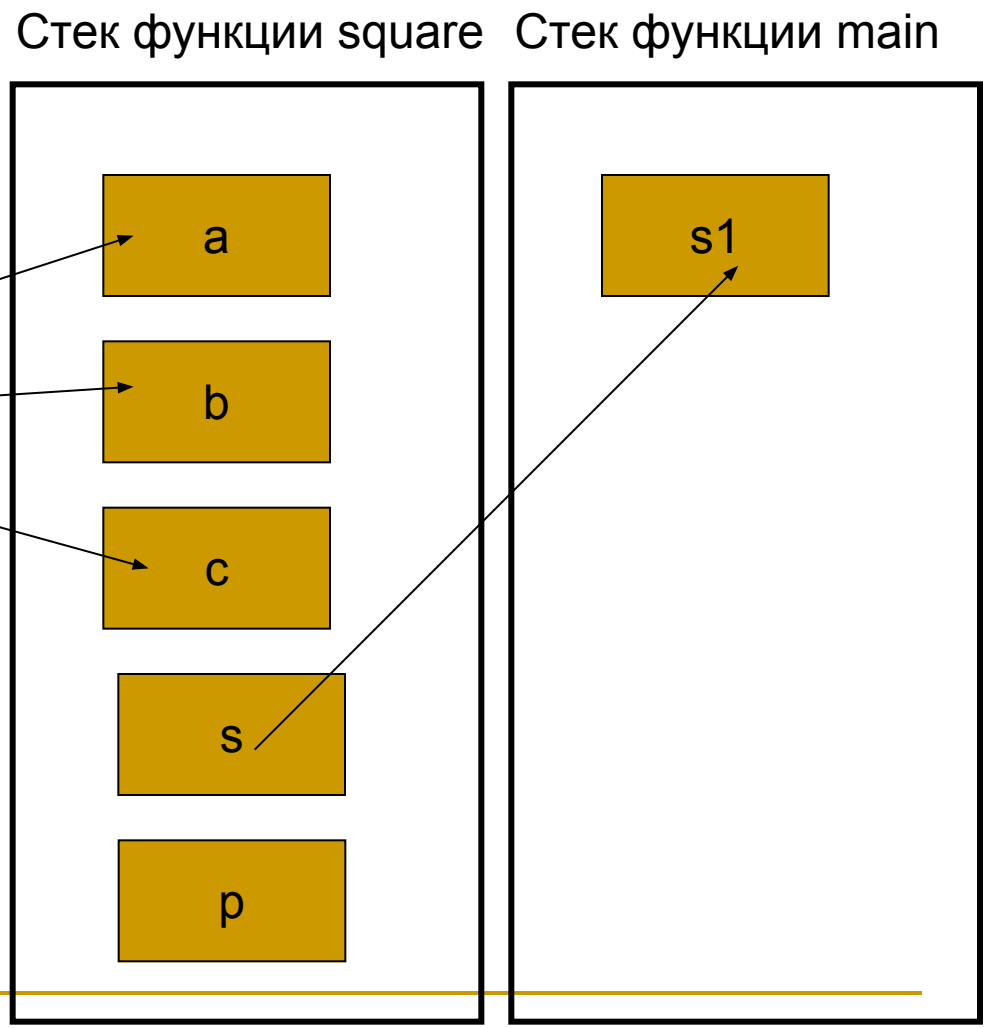
Способы передачи параметров

- Существует два способа передачи параметров в функцию:
  - по адресу
  - по значению

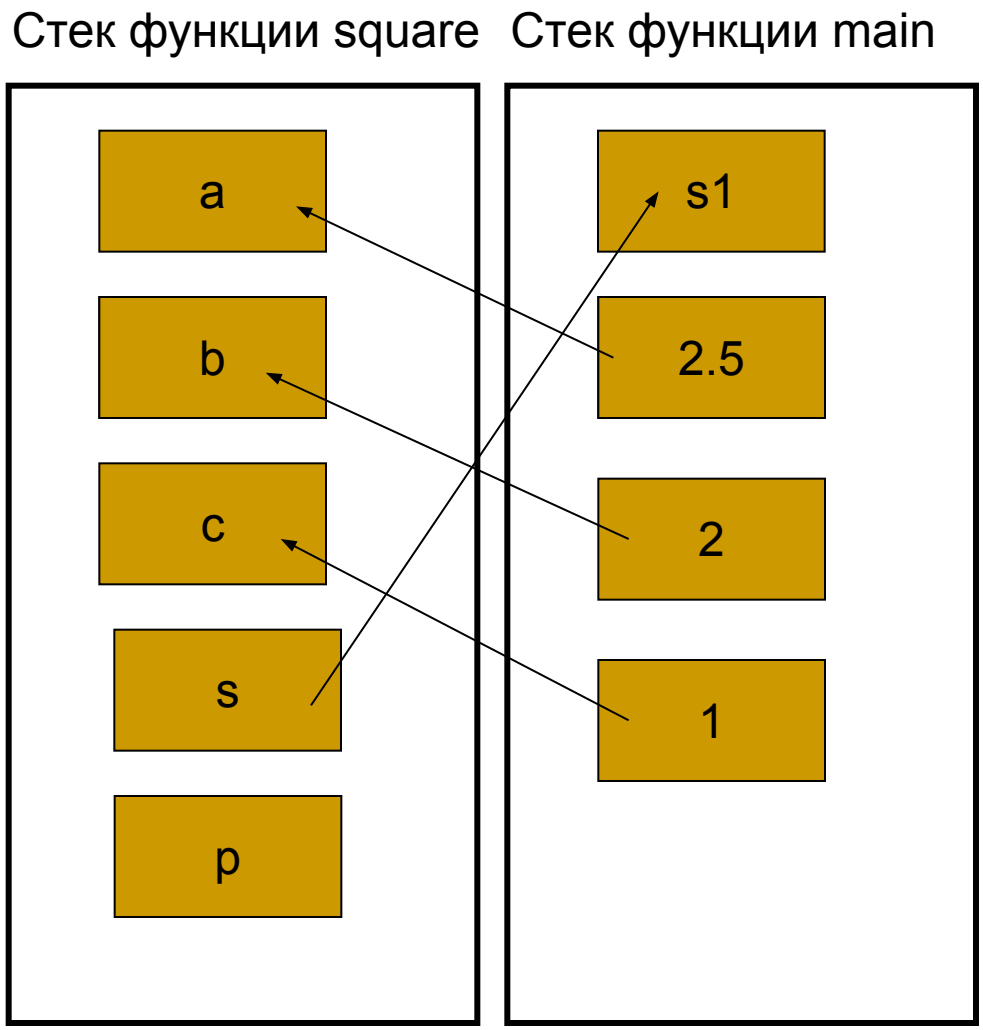
Передача параметров по значению

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.

```
//функция возвращает площадь треугольника, заданного длинами сторон a,b,c

double square (double a, double b, double c)
{
double s, p=(a+b+c)/2;
return s=sqrt(p*(p-a)*(p-b)*(p-c));
}

//вызов функции

double s1=square(2.5,2,1);
```



```
//вызов функции
double a=2.5,b=2,c=1;
double s2=square (a, b, c);
```

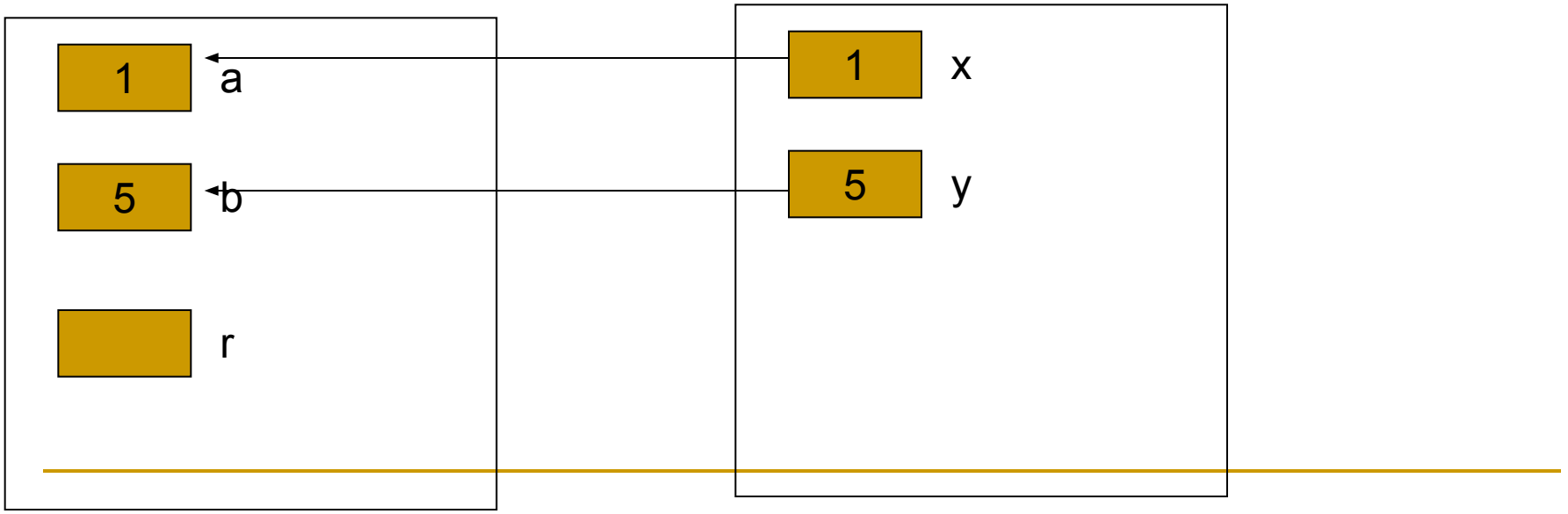


Таким образом, в стек заносятся копии фактических параметров, и операторы функции работают с этими копиями. Доступа к самим фактическим параметрам у функции нет, следовательно, нет возможности их изменить.

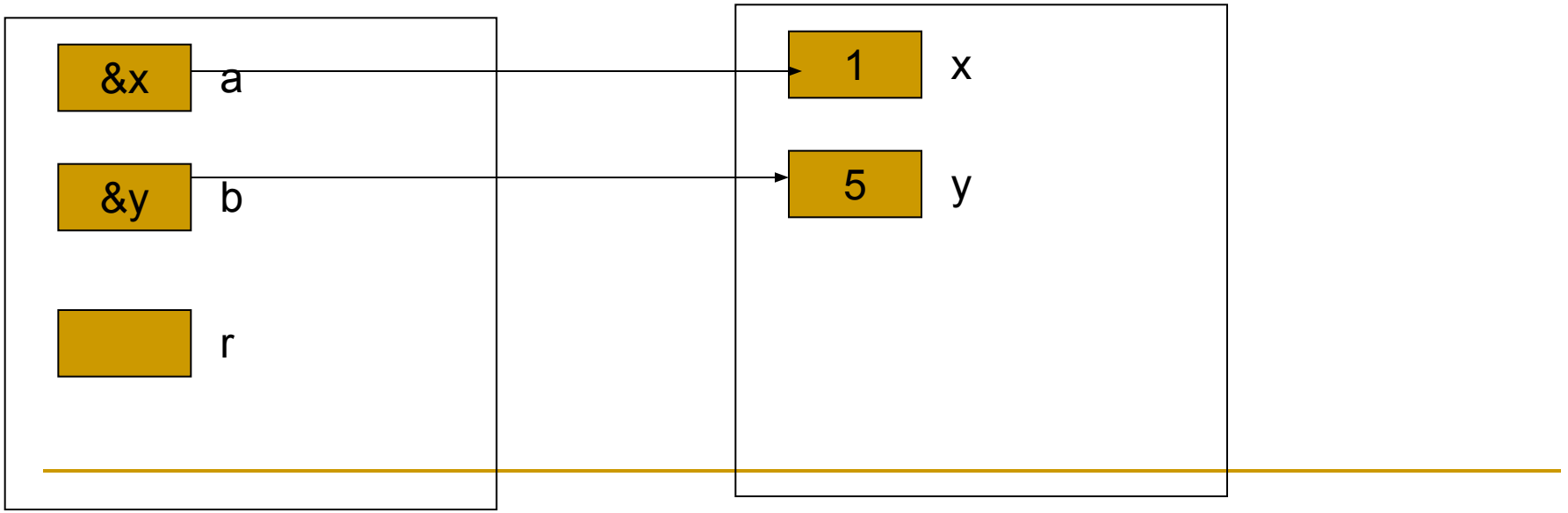
Передача параметров по адресу

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

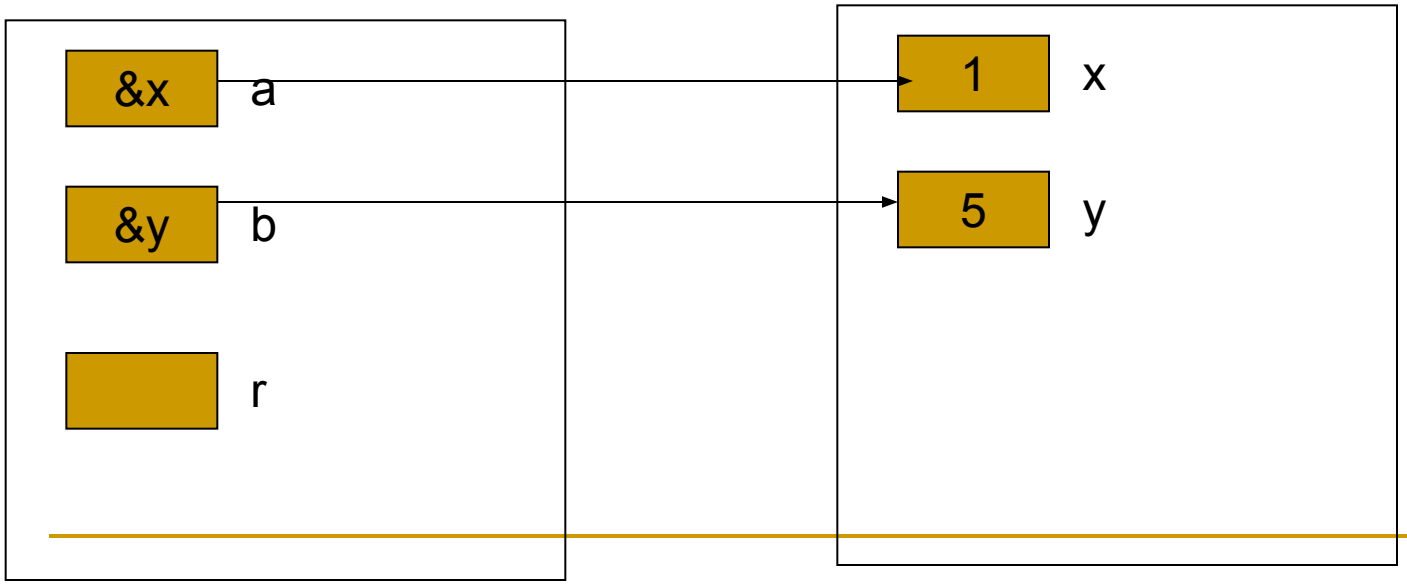
```
void swap (int a, int b) //передача по значению
{
        int r=a;
        a=b;
        b=r;
}

//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 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;
```



Задание

- Какой результат будет выведен на экран?

```
#include <iostream>
using namespace std;
void f(int i, int* j, int& k) {
    i++;
    (*j)++;
    k++;
}
void main()
{
    int i = 1, j = 2, k = 3;
    cout << "i j k\n";
    cout << i << ' ' << j << ' ' << k << '\n';
    f(i, &j, k);
    cout << i << ' ' << j << ' ' << k;
}
```

Задание

- **Найти наибольший общий делитель** (НОД) **для значений** x, y, x+y.

Программа

```
#include <iostream>
using namespace std;
int evklid(int m,int n) //данные передаются по значению
{
     while (m!=n)
       if (m>n) m=m-n;
       else n=n-m;
return (m);
}
void main ()
{
  int x,y,nod;
  cin>>x>>y;
  nod=evklid(evklid(x,y),x+y);
  cout<<"NOD="<<nod<<"\n";
}
```

Задание

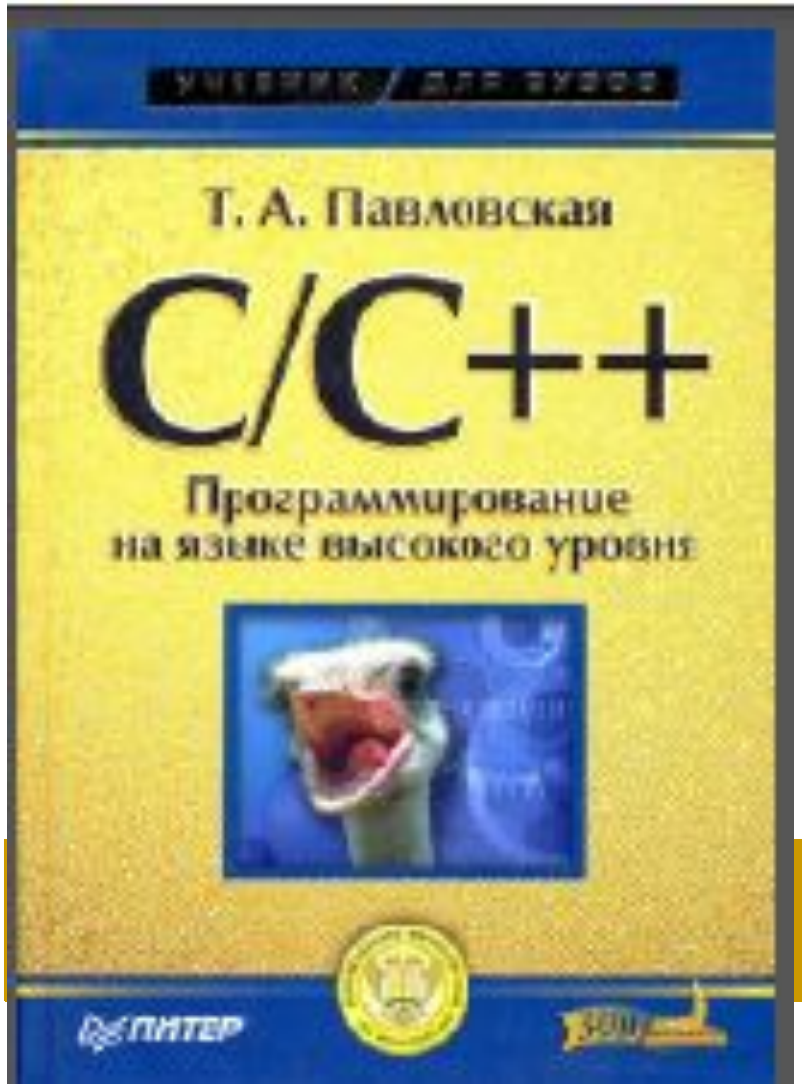
- **Написать программу, запрашивающую *N* целых чисел и выводящих в текстовый файл все цифры этих чисел через запятую в обратном порядке.**

Программа

```
#include <iostream>
#include <fstream>
using namespace std;
ofstream f;
void vyvod(int n) //данные передаются по значению
{     int k;
      while (n!=0)
      {
            k=n%10;
            f<<k;
            n=n/10;
            if (n!=0) f<<",";
      }
      f<<endl;
}
```

```
void main ()
{
int x,i,n;
f.open("a.txt",ios::out);
cin>>n;
for (i=1;i<=n;i++)
{
cin>>x;
vyvod(x);
}
f.close();
}
```

Домашнее задание



1) Стр. 52 - 54
2) Стр. 73 - 78