

# Переменные-указатели и операции над указателями

## Лекция 6

1) Что будет выведено на экран?

2) Что надо исправить для получения правильного результата для всех целых чисел?

```
#include <iostream>
using namespace std;

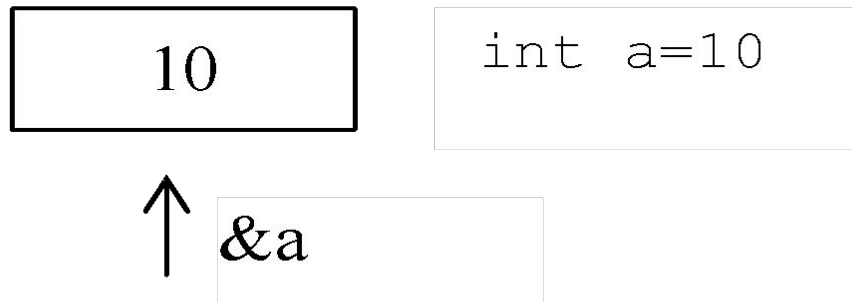
//Нахождение суммы цифр
целого числа
int sum_c(int m)
{int s=0;
  while (m!=0)
  {
    s+=m%10;
    m/=10;
  }
  return s;
}
```

```
int main ()
{
  int m,k;
  m=123;
  k=sum_c(m);
  cout<<"k="<<k<<"\n";
  cout<<"m="<<m<<"\n";
  m=-568;
  k=sum_c(m);
  cout<<"k="<<k<<"\n";
  cout<<"m="<<m<<"\n";
  system ("pause");
  return 0;
}
```

2

# Указатели

- Указатели – это переменные, предназначены для хранения адресов памяти.



Объявление указателя:           тип\* имя;  
Память под переменные-указатели  
выделяется только тогда, когда им  
присваивается какое-либо значение.

## Примеры

```
int* i;
```

```
double *f, *ff;
```

```
char* c;
```

```
int** a; // указатель на указатель
```

```
const int* pci;
```

- Указатель можно сразу проинициализировать:

Примеры:

```
int i;//целая переменная;
```

```
const int ci;//целая константа
```

```
//указатель на целую переменную
```

```
int* pi=&i;
```

```
//указатель на целую константу
```

```
const int* pci=&ci;
```

```
//указатель-константа на переменную целого типа
```

```
int* const cpi=&i;
```

```
//указатель-константа на целую константу
```

```
const int* const cpc=&ci;
```

# 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;`

Пример:

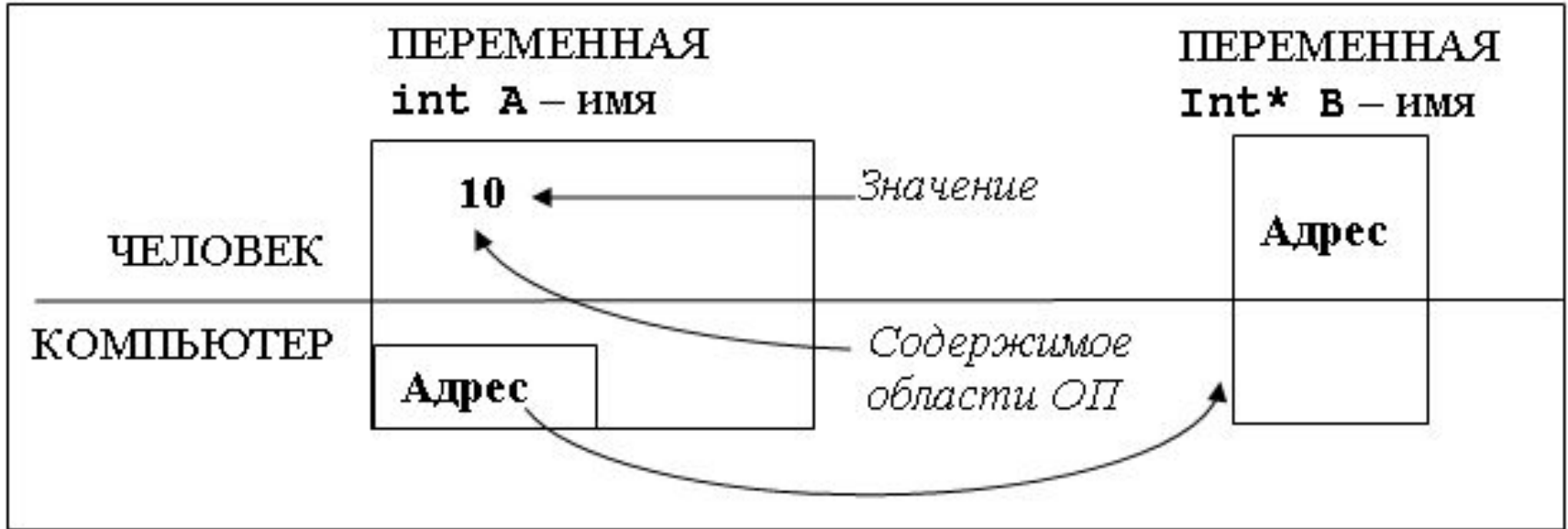
**int A;** // выделяется память

**int\* B;** // память не выделяется

...

**A=10;**

**B=&A;** // &A – взятие адреса. Выделяется  
память под B и в нее записывается  
адрес области памяти, выделенной под  
A



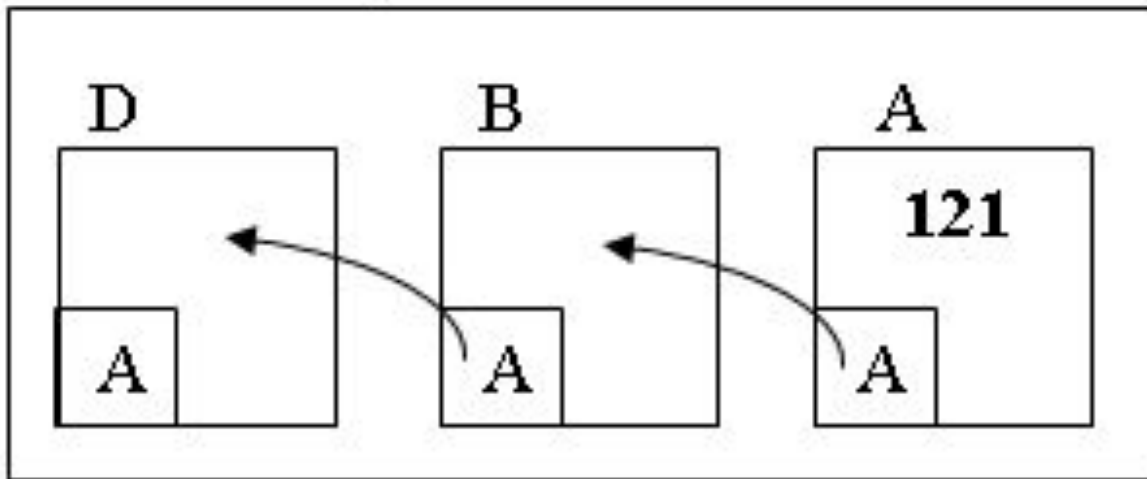


- Обращаться к содержимому области памяти можно через переменные-указатели, для этого используется операция разыменования. Для этого ставится «\*» перед именем переменной-указателем:



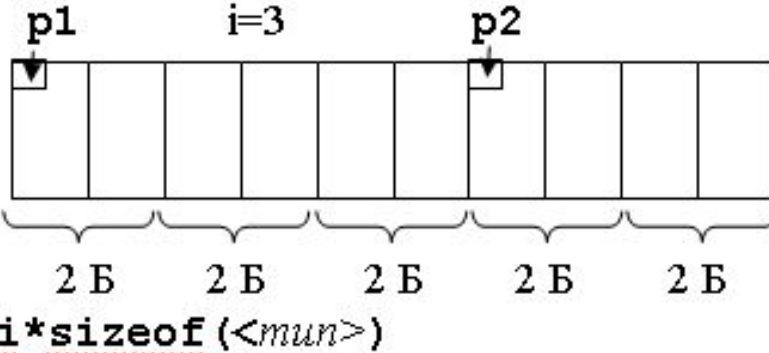
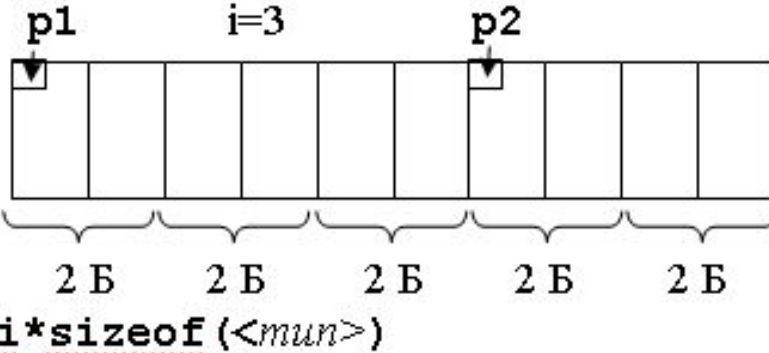
**int \*\*D;** // значением этой переменной  
является значение переменной типа  
указатель

**D=&B;**



# Действия над указателями

Описание: **int \*p1, \*p2, i;**

ОПЕРАЦИЯ	ПРИМЕР	ОПИСАНИЕ
<code>&lt;указатель&gt;+&lt;целое&gt;</code>	<code>p2=p1+i ;</code>	<p>В результате этой операции значением переменной <code>p2</code> будет являться адрес области памяти, отстоящей от области памяти, адрес которой хранится в переменной <code>p1</code>, на количество единиц типа, на данные которого указывают эти указатели:</p>  <p style="text-align: center;"> <code>p1</code>                      <code>i=3</code>                      <code>p2</code>    2 Б    2 Б    2 Б    2 Б    2 Б  <code>i*sizeof(&lt;тип&gt;)</code> </p>

ОПЕРАЦИЯ	ПРИМЕР	ОПИСАНИЕ
<code>&lt;указатель&gt;-&lt;целое&gt;</code>	<code>p2=p1-i ;</code>	Аналогично сложению
<code>&lt;указатель&gt;++; &lt;указатель&gt;--;</code>	<code>p1++; p2--</code>	Изменения происходят в единицах типа: <code>1*sizeof(&lt;тип&gt;) ;</code>
<code>&lt;указатель&gt;- &lt;указатель&gt;</code>	<code>i=p2-p1 ;</code>	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.

# 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.

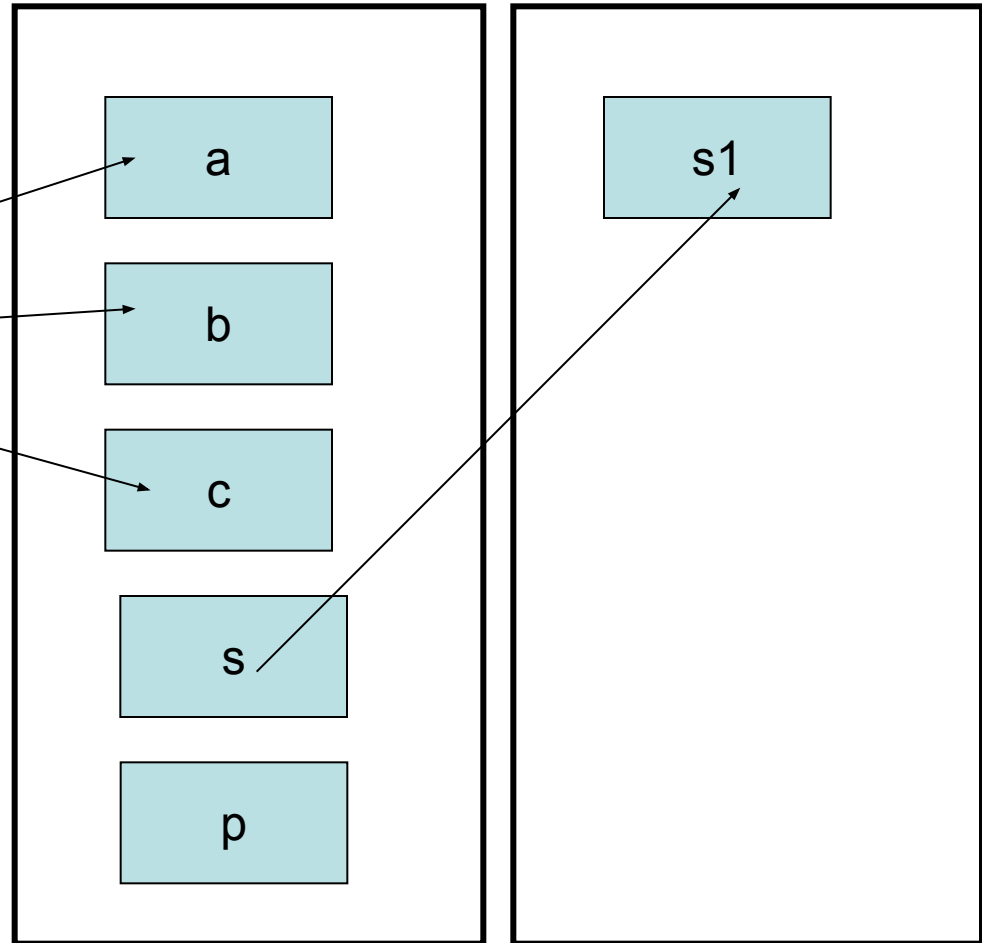
//функция возвращает площадь треугольника, заданного длинами сторон a,b,c

```
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

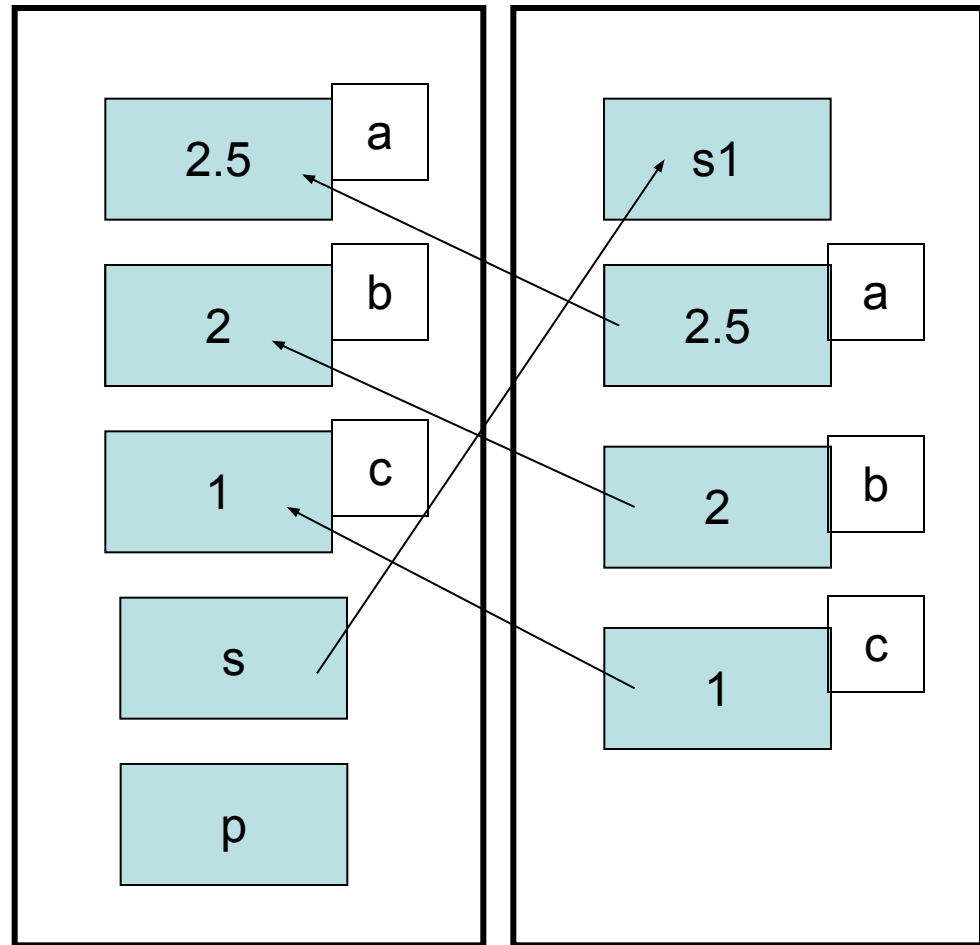
```
double s1=square(2.5,2,1);
```

Стек функции square    Стек функции main



```
//вызов функции  
double a=2.5,b=2,c=1;  
double s1=square (a, b, c);
```

Стек функции square      Стек функции main



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.



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

```
#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);
}
int main ()
{
int x,y,nod;
cin>>x>>y;
nod=evklid(evklid(x,y),x+y);
cout<<"NOD="<<nod<<"\n";
system ("pause");
return 0;
}
```

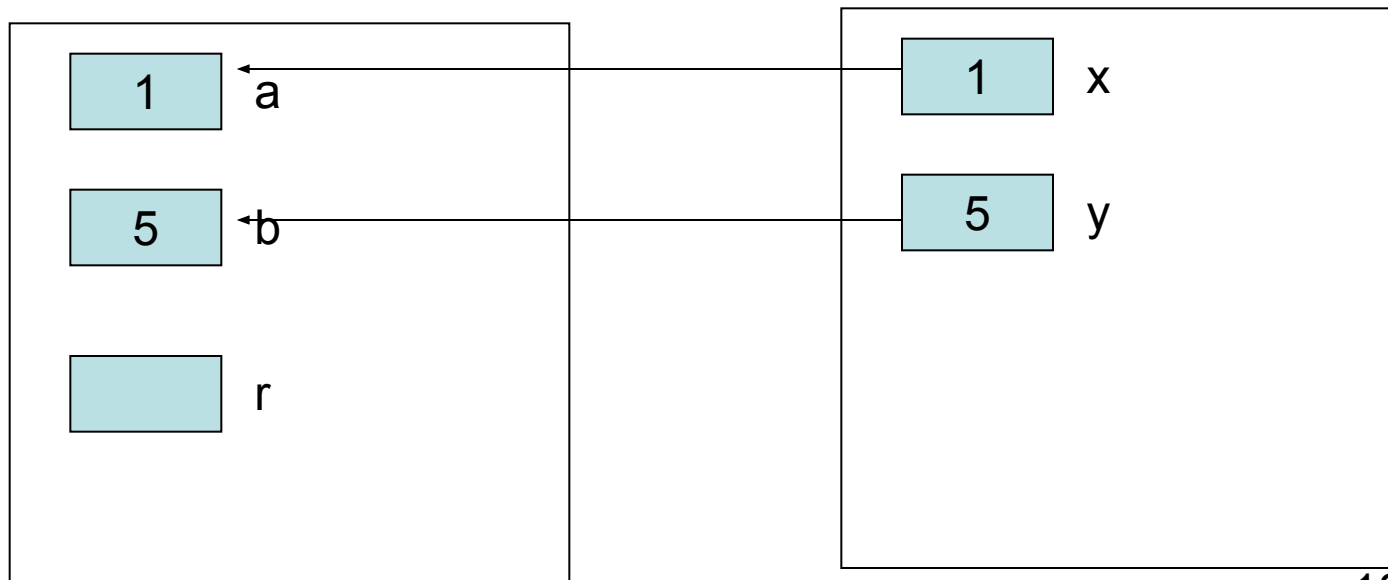
```
void Change (int a,int b) //передача по значению
{
    int r=a;
    a=b;
    b=r;
}
```

//ВЫЗОВ функции

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

```
Change(x,y);
```

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



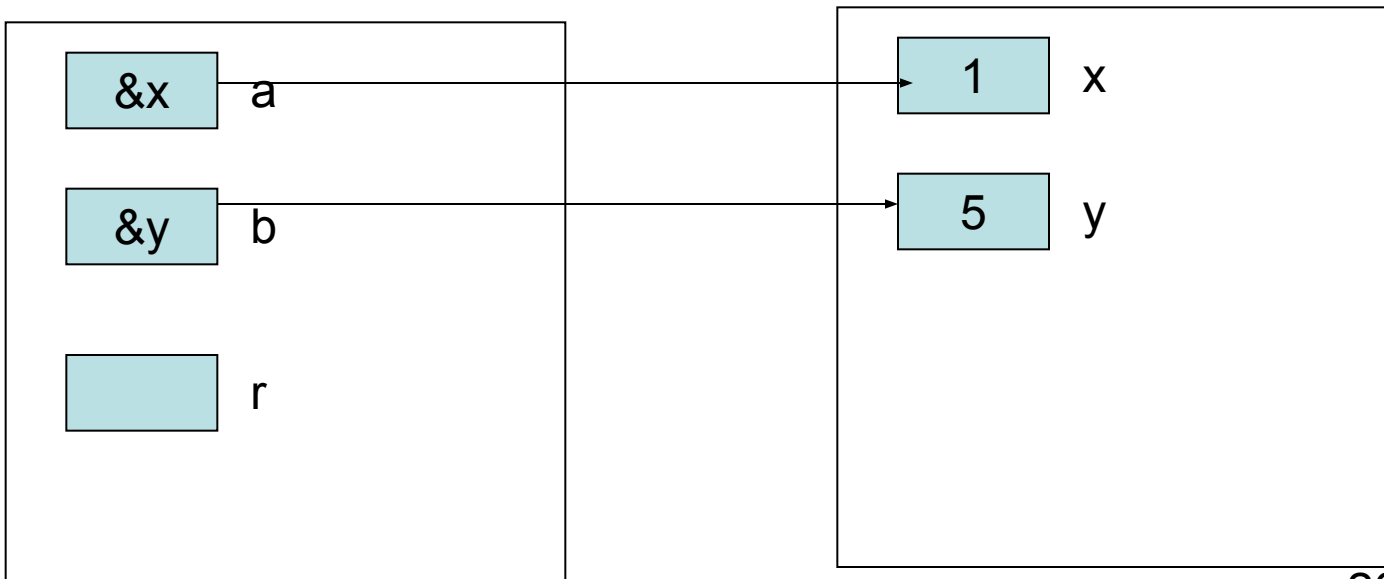
# Передача параметров по адресу

- 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 Change (int* a, int* b) //передача по адресу
{
int r=*a;
*a=*b;
*b=r;
}
```

//ВЫЗОВ функции

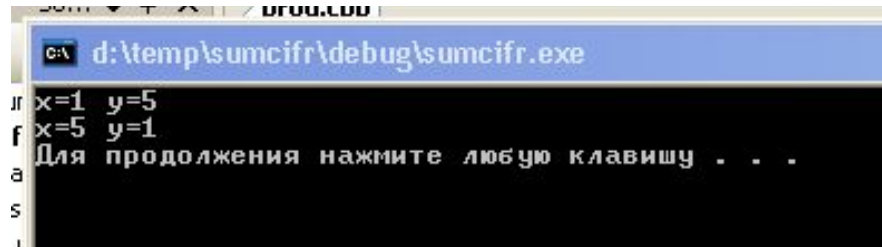
```
int x=1,y=5;
Change(&x,&y);
cout<<"x="<<x<<" y="<<y;
```



```

void Change (int& a, int& b)    //передача по адресу (ссылке)
{
int r=a;
a=b;
b=r;
}

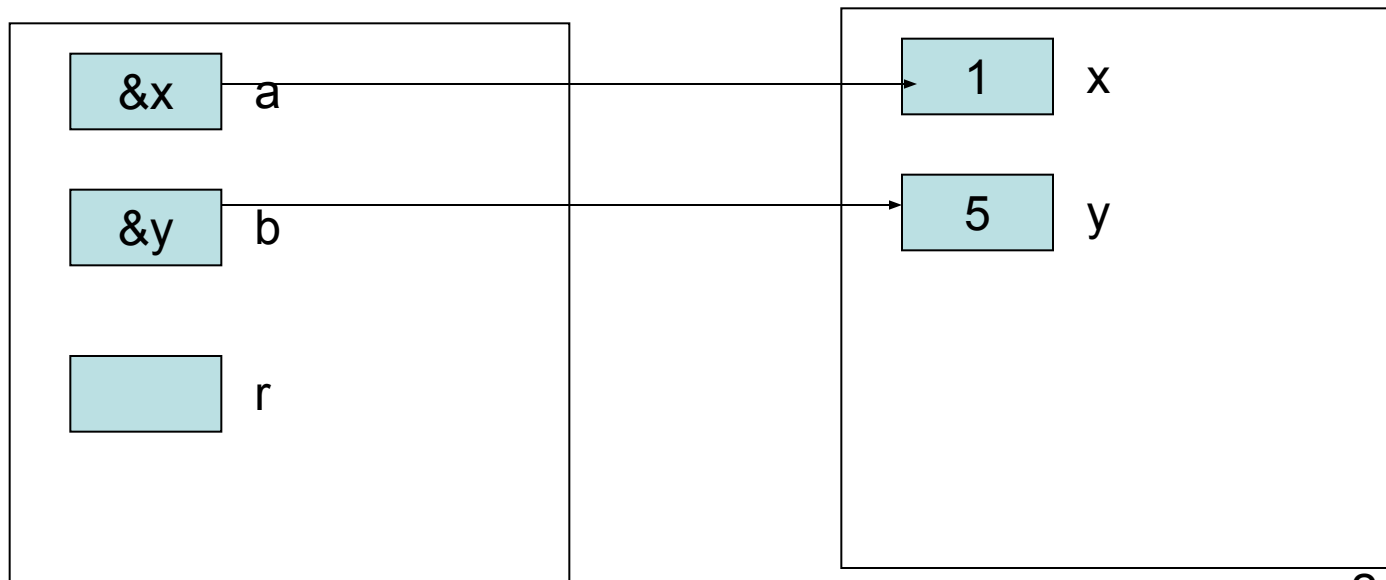
```



```

//ВЫЗОВ функции
int x=1,y=5;
cout<<"x="<<x<<" y="<<y<<"\n";
Change(x,y);
cout<<"x="<<x<<" y="<<y<<"\n";

```



# Локальные переменные

- Переменные, которые используются внутри данной функции, называются локальными. Память для них выделяется в стеке, поэтому после окончания работы функции они удаляются из памяти.
- **Нельзя возвращать указатель на локальную переменную**, т. к. память, выделенная такой переменной, будет освобождаться.

```
int* f()
{
    int a;
    ...
    return &a; // ОШИБКА!
}
```

# Глобальные переменные

- Глобальные переменные – это переменные, описанные вне функций. Они видны во всех функциях, где нет локальных переменных с такими именами.

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

```
#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;
}
```

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



# Подставляемые (inline) функции

- Спецификатор inline определяет для функции так называемое внутреннее связывание, которое заключается в том, что компилятор вместо вызова функции подставляет команды ее кода. При этом может увеличиваться размер программы, но исключаются затраты на передачу управления к вызываемой функции и возврата из нее.
- Подставляемыми **не могут быть**:
  - рекурсивные функции;
  - функции, у которых вызов размещается до ее определения;
  - функции, которые вызываются более одного раза в выражении;
  - функции, содержащие циклы, переключатели и операторы переходов;
  - функции, которые имеют слишком большой размер, чтобы сделать подстановку.

```
/* функция возвращает расстояние от  
точки с координатами (x1,y1) (по  
умолчанию центр координат) до точки с  
координатами (x2,y2)*/
```

```
inline float Line(float x1,float y1,  
                  float x2=0,float y2=0)  
{  
return sqrt(pow(x1-x2)+pow(y1-y2,2));  
}
```

# Рекурсия

- Рекурсией называется ситуация, когда какой-то алгоритм вызывает себя прямо (прямая рекурсия) или через другие алгоритмы (косвенная рекурсия) в качестве вспомогательного. Сам алгоритм называется рекурсивным.
- Рекурсивное решение задачи состоит из двух этапов:
  - исходная задача сводится к новой задаче, похожей на исходную, но несколько проще;
  - подобная замена продолжается до тех пор, пока задача не станет тривиальной, т. е. очень простой.

**Рекурсия** — это способ определения множества объектов через само это множество на основе заданных простых базовых случаев.

# Задачи

- Вычислить факториал ( $n!$ ), используя рекурсию.
- Вычислить степень, используя рекурсию.
- Вычислить  $n$ -ое число Фиббоначи

**Задача 1.** Вычислить факториал ( $n!$ ), используя рекурсию.

*Исходные данные:*  $n$

*Результат:*  $n!$

Рассмотрим эту задачу на примере вычисления факториала для  $n=5$ . Более простой задачей является вычисление факториала для  $n=4$ . Тогда вычисление факториала для  $n=5$  можно записать следующим образом:

$$5! = 4! * 5.$$

Аналогично:

$$4! = 3! * 4;$$

$$3! = 2! * 3;$$

$$2! = 1! * 2 ;$$

$$1! = 0! * 1$$

Тривиальная (простая) задача:

$$0! = 1.$$

Можно построить следующую *математическую модель*:

$$f(n) = \begin{cases} 1, & n = 0 \\ f(n-1) * n, & n \geq 1 \end{cases}$$

```
#include <iostream.h>
int fact(int n)
{
    if (n==0)return 1;    //тривиальная задача
    return (n*fact(n-1));
}

void main()
{
    cout<<"n?";
    int k;
    cin>>k; //вводим число для вычисления факториала

    //вычисление и вывод результата
    cout<<k<<"!="<<fact(k);
}
```

## Числа Фибоначчи:

- $F_1 = F_2 = 1$
- $F_n = F_{n-1} + F_{n-2}$  при  $n > 2$

**1, 1, 2, 3, 5, 8, 13, 21, 34, ...**



**Задача 2.** Вычислить степень, используя рекурсию.

*Исходные данные:  $x, n$*

*Результат:  $x^n$*

*Математическая модель:*

$$pow(x, y) = \begin{cases} 1, & n = 0 \\ pow(x, n - 1) * x, & n \geq 1 \end{cases}$$

```

#include <iostream.h>
int pow( int x,int n)
{
    if(n==0)return 1;//тривиальная задача
    return(x*pow(x,n-1));
}
void main()
{
    int x,k;
    cout<<"n?";
    cin>>x; //вводим число
    cin>>k; //вводим степень
    //вычисление и вывод результата
    cout<<x<<"^"<<k<<"="<<pow(x,k);
}

```

# Вычисление суммы цифр числа

```

int sumDig ( int n )
{
    int sum;
    sum = n % 10;
    if ( n >= 10 )
        sum += sumDig ( n / 10 );
    return sum;
}

```

последняя цифра

рекурсивный вызов

?

Где условие окончания рекурсии?

sumDig ( 1234 )

4 + sumDig ( 123 )

4 + 3 + sumDig ( 12 )

4 + 3 + 2 + sumDig ( 1 )

4 + 3 + 2 + 1

# Алгоритм Евклида

**Алгоритм Евклида.** Чтобы найти НОД двух натуральных чисел, нужно вычитать из большего числа меньшее до тех пор, пока меньшее не станет равно нулю. Тогда второе число и есть НОД исходных чисел.

```
int NOD ( int a, int b )
{
    if ( a == 0 || b == 0 )
        return a + b;
    if ( a > b )
        return NOD ( a - b, b );
    else return NOD ( a, b - a );
}
```

условие окончания  
рекурсии

рекурсивные вызовы

# Как работает рекурсия?

Факториал:

$$N! = \begin{cases} 1, & N = 1 \\ N \cdot (N-1)!, & N > 1 \end{cases}$$

```
int Fact ( int N )
{
    int F;
    cout << "-> N=" << N << endl;
    if ( N == 1 )
        F = 1;
    else F = N * Fact ( N - 1 );
    cout << "<- N=" << N << endl;
    return F;
}
```

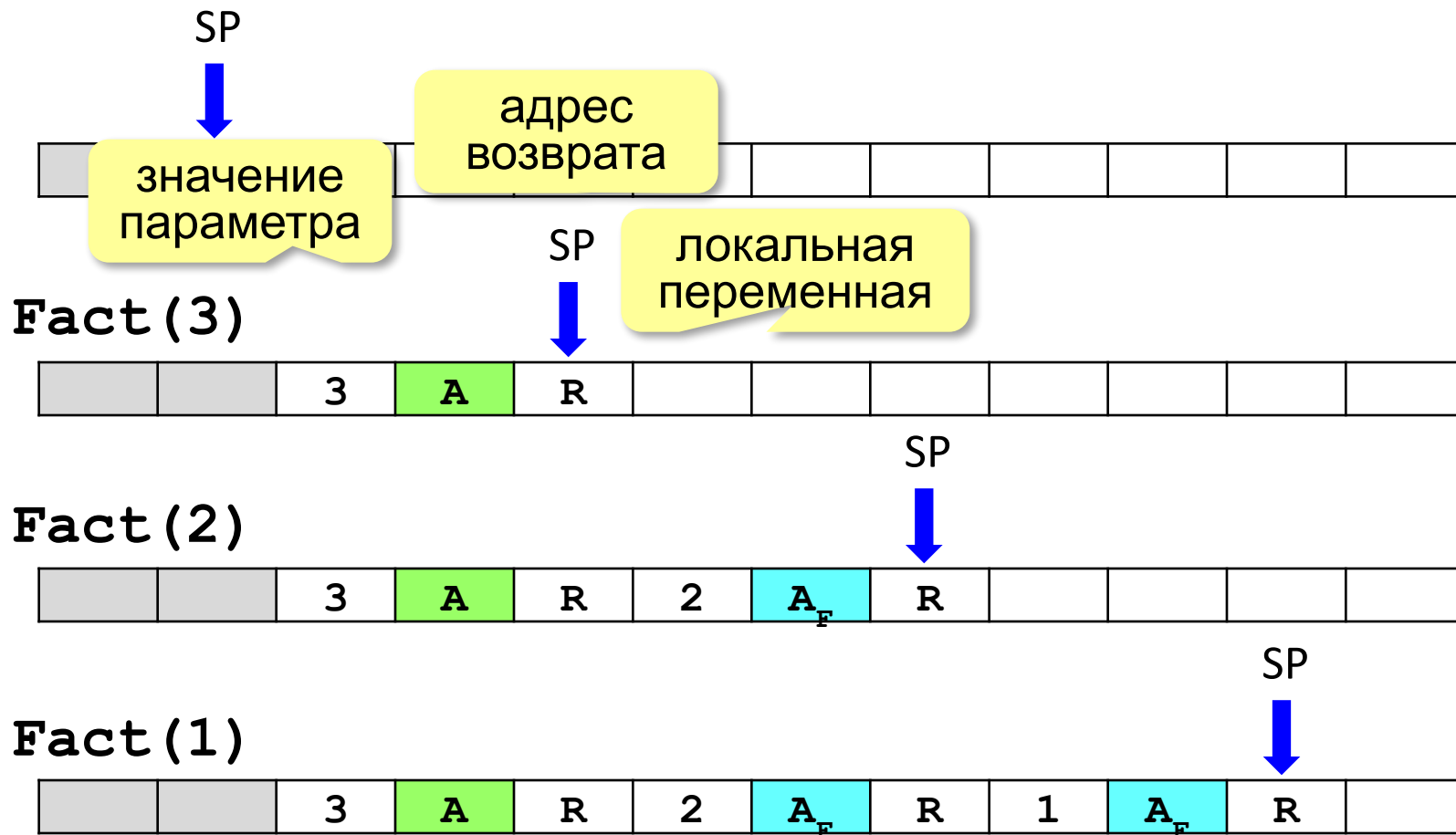
```
-> N = 3
    -> N = 2
        -> N = 1
            <- N = 1
        <- N = 2
    <- N = 3
```



Как сохранить состояние функции перед рекурсивным вызовом?

# Стек

**Стек** – область памяти, в которой хранятся локальные переменные и адреса возврата.



# Рекурсия – «за» и «против»

- с каждым новым вызовом расходуется память в стеке (возможно переполнение стека)
- затраты на выполнение служебных операций при рекурсивном вызове



▪ программа становится более короткой и понятной



▪ возможно переполнение стека

▪ замедление работы



Любой рекурсивный алгоритм можно заменить итерационным!

итерационный  
алгоритм

```
int Fact ( int N )
{
    int F;
    F = 1;
    for (i = 2; i <= N; i++)
        F = F * i;
    return F;
}
```





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

```
#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;
}
```

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