Лекция 14

Указатели в параметрах функций

Преподаватель Палехова Ольга Александровна,
кафедра И5 БГТУ «Военмех»

Передача параметров в функцию

В языке Си передача параметров в функцию осуществляется только ПО ЗНАЧЕНИЮ.

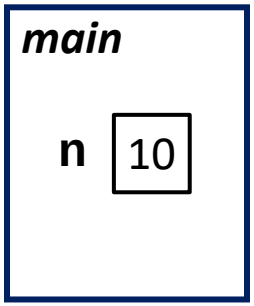
Функция работает с собственным экземпляром значений аргументов, любые изменения значений параметров внутри функции никак не отразятся на значениях передаваемых аргументов.

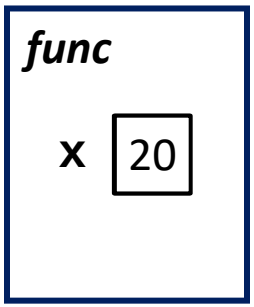
Передача параметров по значению

```c
#include <stdio.h>

void func (int x)
{
    x += 10;
    printf ("func function: %d \n", x);
}

int main ()
{
    int n = 10;
    func (n);
    printf("main function: %d \n", n);
    return 0;
}
```

определение функции

вызов функции





Результат работы программы:

```
func function: 20
main function: 10
```

Указатели как параметры функций

Задача 1. Написать функцию, которая поменяет местами значения двух переменных.

Задача 2. Написать функцию для решения квадратного уравнения.

Задача 3. Написать функцию ввода массива.

Проблема – каждая из этих функций должна реализовать вспомогательный алгоритм, возвращающий больше одного выходного значения.

Идея – дать вызывающей и вызываемой функции доступ к одному и тому же участку памяти

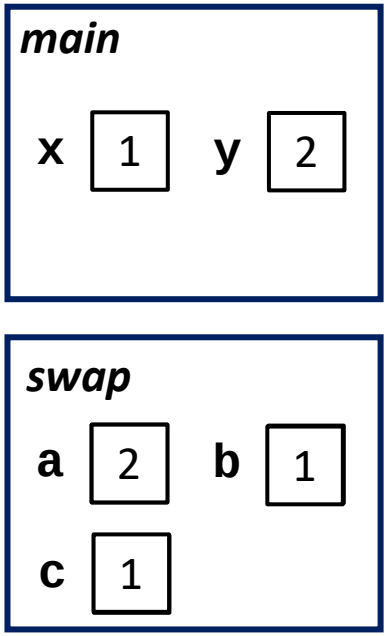
Решение – **передать** функции **адрес** блока памяти, в который нужно поместить выходное данное. Формальным параметром будет **указатель**.

Обмен значений переменных

Задача 1. Написать функцию, которая поменяет местами значения двух переменных.

```
void swap ( int a, int b )
{
    int c;
    c = a;
    a = b;
    b = c;
}
```

эта функция работает с копиями значений параметров




```
main()
{
  int x = 1, y = 2;
  swap ( x, y );
  printf ( "x = %d,  y = %d", x, y );
}
```

Результат работы программы:

```
x = 1, y = 2
```

Указатели в параметрах функции

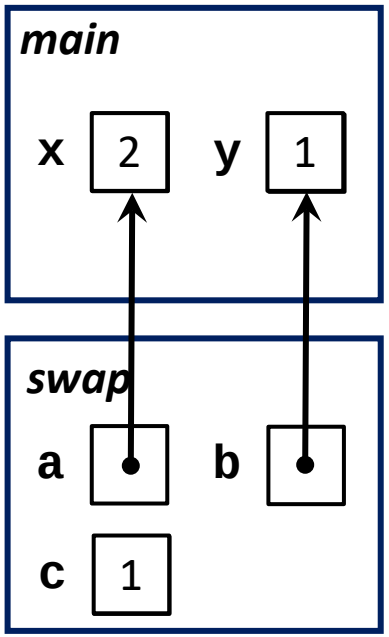
Проблема: надо, чтобы изменения, сделанные в функции обмена, стали известны вызывающей функции.

Решение: передать функции адреса переменных. Формальным параметром будет **указатель**.

```
void swap ( int *a, int *b )
{
   int c;
   c = *a;
   *a = *b;
   *b = c;
}

main()
{
  int x = 1, y = 2;
  swap ( &x, &y );
  printf ( "x = %d,  y = %d", x, y );
}
```

Эта функция тоже работает с копиями значений параметров

обращение по адресу, записанному в указатель



Результат работы программы:

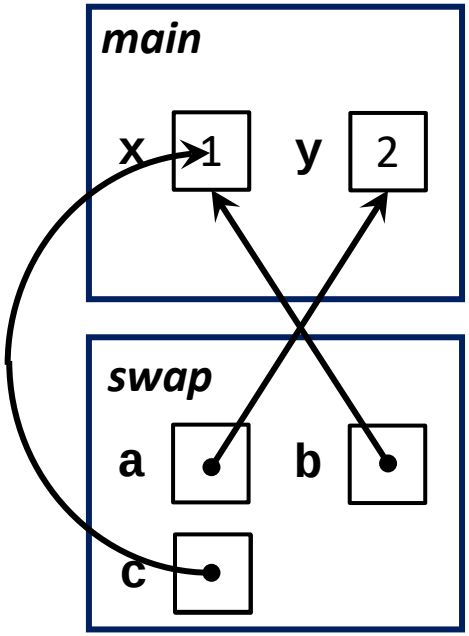
```
x = 2, y = 1
```

Указатели в параметрах функции

? Что будет, если изменить в первом примере только тип переменных, а тело функции не менять?

```
void swap ( int *a, int *b )
{
    int *c;
    c = a;
    a = b;
    b = c;
}
```

Меняем местами значения *a* и *b*



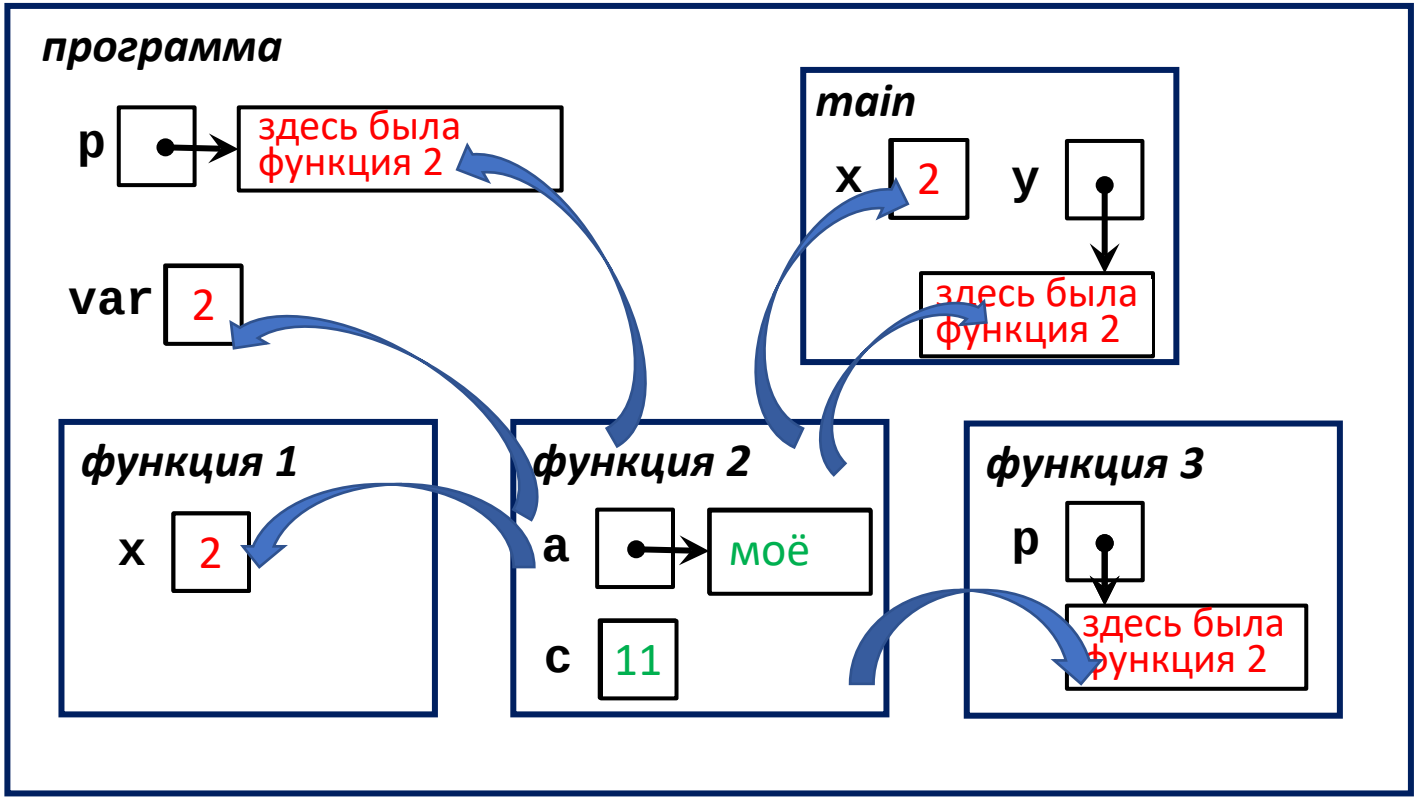

```
main()
{
  int x = 1, y = 2;
  swap ( &x, &y );
  printf ( "x = %d,  y = %d", x, y );
}
```

Результат работы программы:

```
x = 1, y = 2
```

Побочный эффект функции

Изменение функцией содержимого любых блоков памяти, кроме блоков, занимаемых объектами, принадлежащими этой функции, называется **побочным эффектом функции**



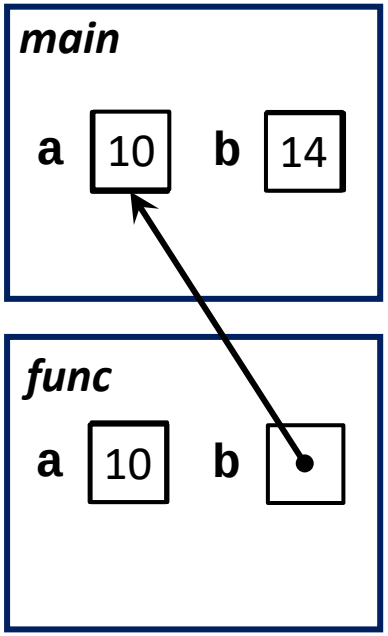
Побочный эффект функции

```
int func (int a, int *b)
{
    a += 2;
    *b += 2;
    return a + *b;
}

int main()
{
    int a = 1, b = 0;
    a += func (a, &b);
    b += func (b, &a);
    a += func (a, &a);
```

a = 6, b = 2

a = 8, b = 14

a = 30, b = 14



! Имена переменных **в функциях** могут совпадать, но это совершенно **разные переменные**

Функция решения уравнения

Задача 2. Написать функцию для решения уравнения вида $ax^2 + bx + c = 0$. Коэффициенты уравнения могут быть только целыми.

Сколько корней может иметь это уравнение?

Анализ задачи:

- если $a \neq 0$, то уравнение квадратное, корней может быть 2, 1 или 0 действительных корней;
- если $a = 0$, то уравнение линейное, 1 корень;
- если $a = 0$ и $b = 0$, то уравнение вырождается в равенство $c = 0$:
  - если $c = 0$, то равенство будет верным при любых значениях x, следовательно, количество корней бесконечно;
  - если $c \neq 0$, то равенство не будет верным ни при каких значениях x, следовательно, корней нет.

Функция решения уравнения

Задача 2. Написать функцию для решения уравнения вида $ax^2 + bx + c = 0$. Коэффициенты уравнения могут быть только целыми.

Проблема: корней может быть разное количество.

Идея: возвращать не только значения корней, но и их количество.

Проблемы:

- как обозначить бесконечность?
- как различать отсутствие корней в принципе и отсутствие действительных корней?

Идея: ввести систему обозначений:

Проблема: как вернуть бесконечное число корней?

3 – бесконечность
2 – 2 корня
1 – 1 корень
0 – нет действительных корней
-1 – нет никаких корней

Идея: корнем может быть любое число, возвращать конкретные значения не нужно.

Функция решения уравнения

Задача 2. Написать функцию для решения уравнения вида $ax^2 + bx + c = 0$. Коэффициенты уравнения могут быть только целыми.

Входные и выходные данные для алгоритма:

Вход – коэффициенты уравнения

Выход – количество корней,
значения корней (максимум два)

Проблема: как вернуть три значения из функции?

Идея: возвращать через *return* количество корней, значения корней размещать в переменных, которые будут использоваться в вызывающей функции.

Входные и выходные данные для функции:

Вход – коэффициенты уравнения,
адреса переменных для записи значений корней.

Выход – количество корней.

Функция решения уравнения

```
int equation (int a, int b, int c, double *x1, double *x2)

    int d;
    if (a) /* квадратное уравнение */
    {
        d = b * b - 4 * a * c;
        if ( d > 0 )
        {
            *x1 = ( -b + sqrt (d) ) / ( 2 * a );
            *x2 = ( -b - sqrt (d) ) / ( 2 * a );
            return 2;
        }
        if ( d < 0 )
            return 0;
        *x1 = *x2 = -b / ( 2. * a );
        return 1;
    }
    if (b) /* линейное уравнение */
    {
        *x1 = *x2 = -c / (double)b;
        return 1;
    }
    if (c) /* корней нет */
        return -1;
    return 3;
}
```

количество корней

адреса для записи корней

не забываем разыменовывать

? Если забудем про разыменование?

Передача массива в функцию и из функции

Особенности:

- массивы никогда **не копируются**, все функции работают с оригиналом;
- в функцию передается **адрес** первого элемента массива и его **размер**
- объявить формальный параметр-указатель можно двумя способами:

```
тип_элемента_массива * имя_параметра
тип_элемента_массива имя_параметра []
```

- если по одному алгоритму обрабатываются массивы, элементы которых имеют разные типы, нужно написать несколько подобных функций
- если функция не должна изменять значения элементов массива, формальный параметр-указатель лучше объявить с квалификатором ***const***

Функция ввода массива

Задача 3. Написать функцию ввода целочисленного массива.

Вариант 1 Без контроля ввода

```
void input_array ( int array[], int n )
{
   int i;
   for ( i = 0 ; i < n ; i++ )
      scanf ("%d", &array[i]);
}
```

ничего не возвращает

адрес первого элемента массива

размер массива

значения *n* и *array* не изменяются

Тот же алгоритм иначе:

```
void input_array ( int array[], int n )
{
   while ( n-- )
      scanf ("%d", array++);
}
```

значения формальных параметров можно менять

Функция ввода массива

Задача 3. Написать функцию ввода целочисленного массива.

Вариант 2 С контролем ввода

```
int in_array ( int array[], int n )
{
   int i;
   for ( i = 0 ; i < n ; i++ )
      if ( ! scanf ("%d", array++) )
         return i;
   return n;
}
```

возвращает целое число

возвращаем количество считанных элементов

! Все **изменения**, сделанные в вызываемой функции **влияют** на массив в вызывающей функции

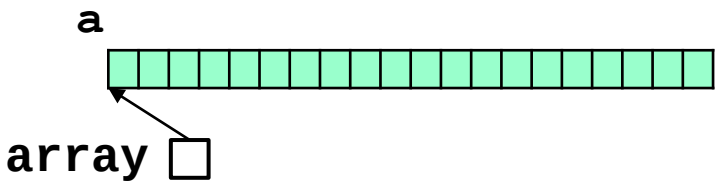
Вызов функции ввода массива

Применить функцию ввода массива можно

- для заполнения статического одномерного массива

```
int a[20];
input_array (a, 20);
```

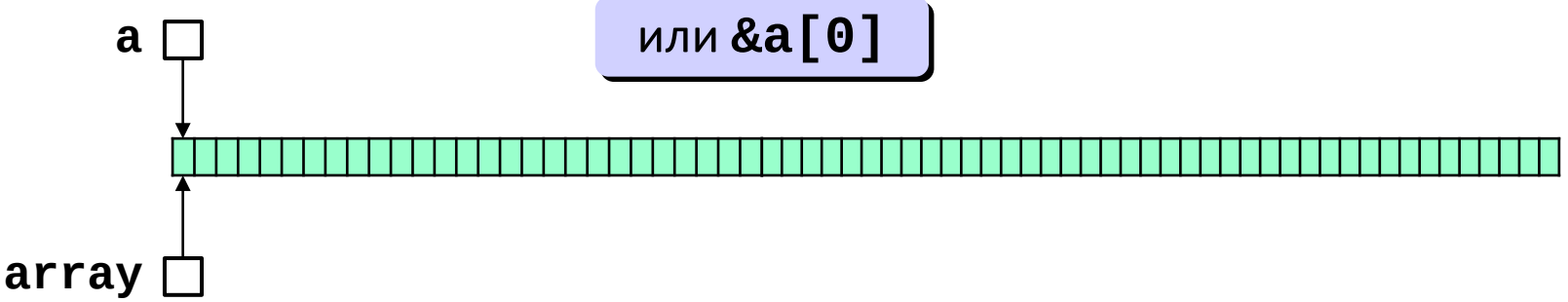
или `&a[0]`



- для заполнения динамического одномерного массива

```
int *a;
a = malloc (100*sizeof(int));
input_array (a, 100);
```

или `&a[0]`

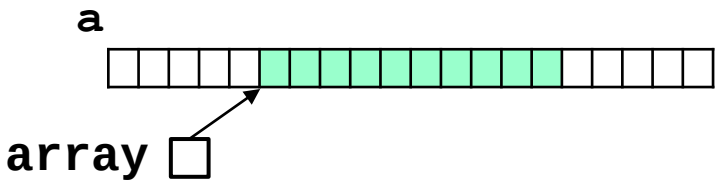


Вызов функции ввода массива

- для заполнения части статического одномерного массива

```
int a[20];
input_array (a+5, 10);
```

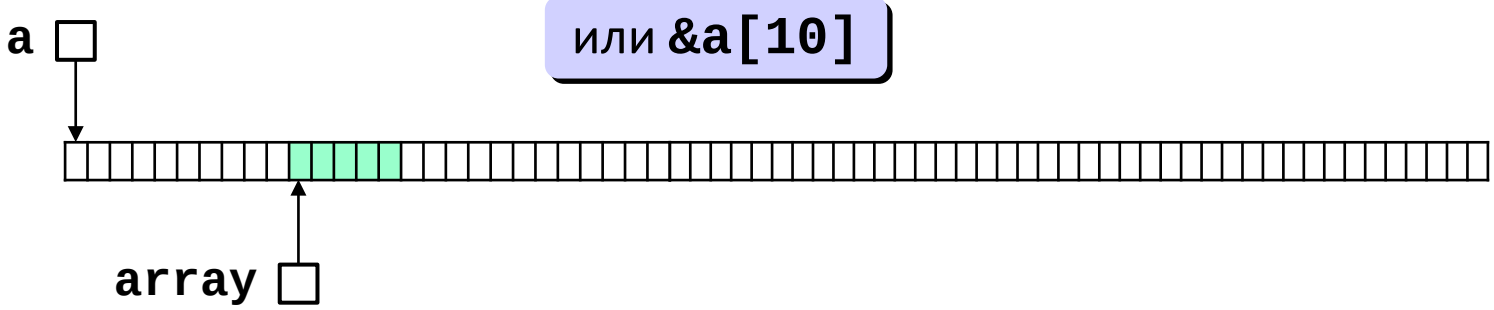
или &a[5]



- для заполнения части динамического одномерного массива

```
int *a;
a = calloc (100, sizeof(int));
input_array (a+10, 5);
```

или &a[10]

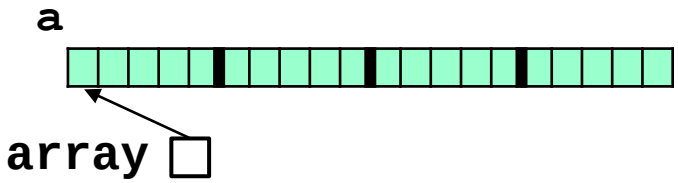


Вызов функции ввода массива

- для заполнения статического двумерного массива

```
int a[4][5];
input_array (a[0], 4*5);
```

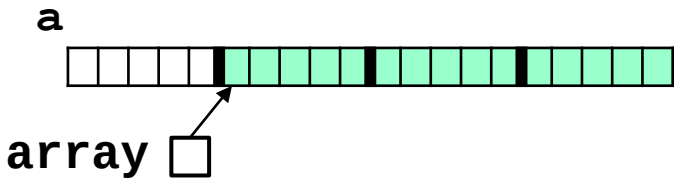
или `&a[0][0]`
или `*a`
или `(int*)a`



- для заполнения трех строк матрицы, запрограммированной статическим двумерным массивом

```
int a[4][5];
input_array (a[1], 3*5);
```

или `&a[1][0]`
или `*(a+1)`



Вызов функции ввода массива

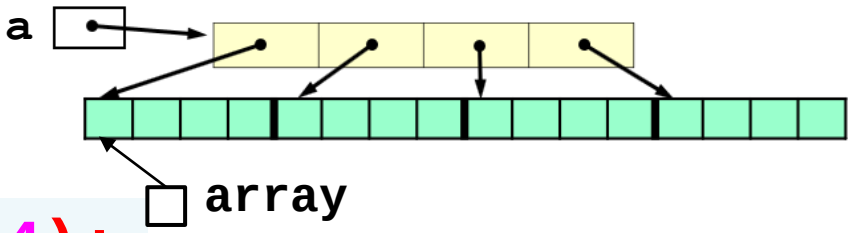
- для заполнения динамического двумерного массива или его части

```
int **a, i;
a = calloc (4, sizeof(int*));
a[0] = calloc (16, sizeof(int));
for ( i = 1; i < 4; i++ )
    a[i] = a[i-1] + 4;
```

```
input_array (a[0], 4*4);
```

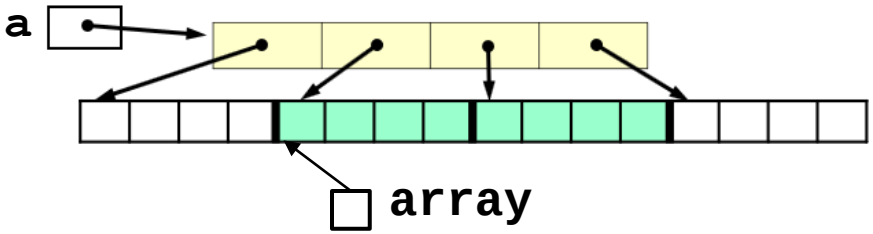
не a, не &a, не (int*)a

или &a[0][0] или *a




```
input_array (a[1], 2*4);
```

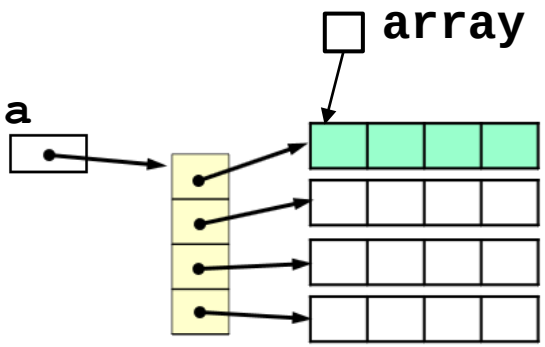
или &a[1][0] или *(a+1)



Вызов функции ввода массива

- для заполнения одной строки динамической матрицы

```
int **a, i;
a = calloc (4, sizeof(int*));
for ( i = 0; i < 4; i++ )
    a[i] = calloc (4, sizeof(int));
```




```
input_array (a[0], 4);
```

или **&a[0][0]**
или ***a**

не a, не &a,
не (int*)a

?

Можно заполнить динамическую матрицу целиком?

```
for ( i = 0; i < 4; i++ )
    input_array (a[i], 4);
```

или **&a[i][0]**
или ***(a+i)**

!

Такая функция применима только к непрерывному блоку памяти