

# Основы программирования

## Функции C/C++

# Описание и вызов функции

**Формат объявления функции:**

**тип\_возвр** **имя** (**тип1** , **тип2** , ... ) ;

**тип\_возвр** – тип возвращаемого значения,

**имя** – имя функции,

**тип1** , **тип2** , ... – типы **формальных** параметров функции

**Формат определения функции:**

**тип\_возвр** **имя**(**тип1** **форм\_п1**, **тип2** **форм\_п2**,...)

{ **тело** функции; }

**форм\_п1** , **форм\_п2** , ... – **формальные** параметры

**Вызов функции:**

**имя** (**факт\_п1** , **факт\_п2** , ...)

**факт\_п1** , **факт\_п2** , ... – фактические параметры, значения которых передаются в функцию

# Формальные и фактические параметры

**Формальные параметры** – это внутренние входные переменные функции. Они создаются при вызове функции перед началом ее работы, получают **значения** от **фактических параметров**, видны только в теле функции и удаляются при выходе из функции.

**Число параметров** при определении и вызове функции должно быть **одинаковым**.

**Фактическим параметром** может быть константа, переменная или выражение такого же типа, как у соответствующего формального, или приводимого к нему (например, константа `int` для параметра типа `double`).

Передача параметров в С (не С++) всегда производится **по значению**, т.е. формальные параметры получают **копии значений** фактических параметров.

# Возвращаемое значение функции

**Тип возвращаемого значения** – это любой тип C/C++ или специальный пустой тип **void**. В последнем случае считается, что функция не возвращает значение (аналог процедуры в Паскале).

Выход из функции осуществляет оператор:

**return значение; (return; для возвращаемого типа void)**

он должен стоять везде, где возможен этот выход, чтобы функция обязательно возвращала значение.

**Значение** – это константа, переменная или выражение возвращаемого типа или приводимого к нему по умолчанию. **Результатом вызова функции** является именно это **значение**, поэтому вызов может стоять в выражениях в правой части присваиваний, в операторах вывода и т.д.

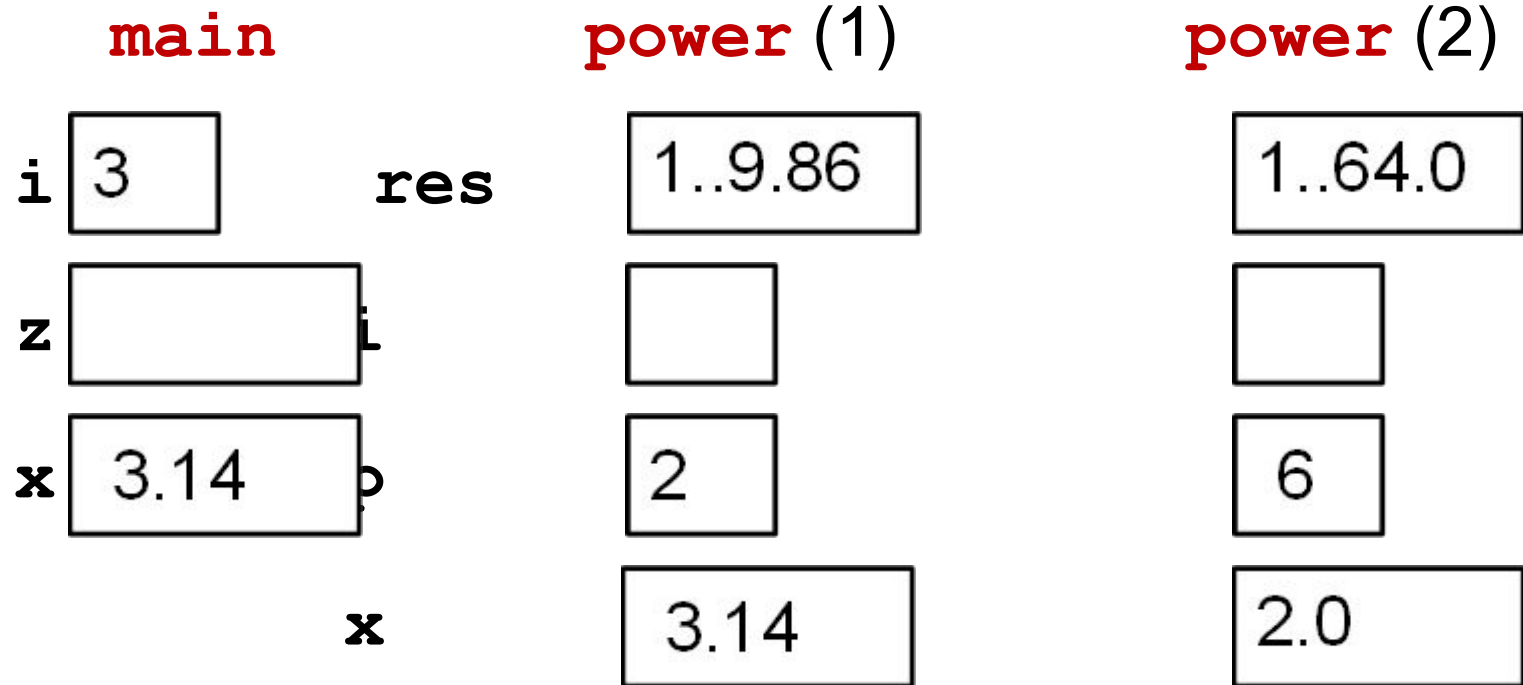
# I Пример: возведение числа в целую

натуральную степень  $x^p$

```
double power(double x, int p)
{
    double res; int i;
    if (p < 0) return 0;
    for (res = 1, i = 1; i <= p; i++)
        res *= x;
    return res;
}
void main(int argc, char *argv[])
{
    double x = 3.14, z; int i = 3;
    z = power(x, i-1); cout << z;
    cout << power(1+1, 6);
    return; // МОЖНО ОПУСТИТЬ
}
```

# Что происходит в памяти

Переменные и параметры обеих функций – **локальные**, они создаются в стеке



# Обмен значениями - ошибка

```
void swap(int a, int b)
{
    int c = a; a = b; b = c;
}
void main(...)
{
    int x = 3, y = 18; swap(x, y);
}
```

**main**

x 

3
---

y 

18
----

**swap** (ВХОД)

a 

3
---

b 

18
----

**swap** (ВЫХОД)

18
----

3
---

# Обмен значениями через указатели

```
void swap(int *a, int *b)
{
    int c = *a; *a = *b; *b = c;
}
void main(...)
{
    int x = 3, y = 18; swap(&x, &y);
}
```

**main**

**x** 3

**y** 18

**swap**

**a** addr x

**b** addr y

**main** после **swap**

18

3



# Передача параметров по ссылке

В C++ возможен еще один способ передачи параметров в функцию – **по ссылке**. Ссылка – это **скрытый указатель**, который используется, как **другое имя** передаваемой переменной. Поэтому фактическими параметрами, которые передаются в функцию по ссылке, могут быть только **переменные**.

```
void swap(int &a, int &b)
{
    int c = a; a = b; b = c;
}
void main(...)
{
    int x = 3, y = 18; swap(x, y);
}
```

# Передача одномерного массива

Если в функцию передается одномерный массив, то он никогда не дублируется: в качестве параметра передается **указатель** с адресом начала массива (статического или динамического). Кроме того, как правило, в функцию необходимо передать и текущую длину массива.

**Эквивалентные способы** описания заголовка функции, которая получает вещественный массив длины  $n$ :

```
int func(double *arr, int n);
```

```
int func(double arr[], int n);
```

```
int func(double arr[1000], int n);
```

## Функция вычисления минимума в массиве

```
double getmin(double *x, int n)
{
    int i; double minval;
    minval = x[0];
    for (i = 1; i < n; i++)
        if (minval > x[i]) minval = x[i];
    return minval;
}

void main(...)
{
    double z, a[100], *b; int k;
    cin >> k; b = new double[k];
    ... // заполнение массивов a и b
    z = getmin(a, 100);
    cout << getmin(b, k);
}
```

# Передача двумерного массива

При передаче в функцию **двумерного статического массива** необходимо явно указать в соответствующем параметре число элементов в строке. Это связано с тем, что такой массив является, фактически «массивом массивов фиксированной длины», поэтому и функция может обрабатывать только матрицы с одинаковым числом столбцов. Кроме того, в функцию нужно непосредственно передавать **количество строк и столбцов**.

**Примеры эквивалентных заголовков:**

```
int func(int arr[][8], int nrow, int ncol);
```

```
int func(int (*arr)[8], int nrow, int ncol);
```

# Функция для вывода матрицы

```
void prinstat(int x[][4], int nrow, int ncol)
{
    int i, j;
    for (i = 0; i < nrow; i++)
    {
        for (j = 0; j < ncol; j++)
            cout << x[i][j] << " ";
        cout << endl;
    }
}

void main(...)
{
    int a[5][4];
    ...
    prinstat(a, 5, 4);
}
```

# Передача матрицы как одномерного массива

В статическом двумерном массиве строки располагаются в памяти последовательно. Массив с  $n$  строками и  $m$  столбцами – это  $n*m$  последовательных элементов, которые можно рассматривать как одномерный массив длины  $n*m$ . Поэтому функцию для работы с матрицей можно построить по-другому:

- в качестве формальных параметров задать **одномерный массив**, а также число строк и столбцов матрицы
- в теле функции **изменить индексацию** на одномерную
- при вызове функции передавать ей не матрицу, а **адрес ее начального элемента**

# Вывод матрицы как одномерного массива

```
void prin1dim(int *x, int nrow, int ncol)
{ int i, j;
  for (i = 0; i < nrow; i++)
  {
    for (j = 0; j < ncol; j++)
      cout << x[i*ncol+j] << " ";
    cout << endl;
  }
}

void main(...)
{
  int a[5][4];
  ...
  prin1dim(&a[0][0], 5, 4);
}
```

# Передача динамического массива

**Двумерный динамический массив** фактически является самоопределенным:

- двойной указатель (имя массива) хранит адрес массива указателей на строки
- каждый элемент последнего массива является указателем на одномерный массив (строку).

Поэтому в функцию надо передать просто **ИМЯ массива, число строк и столбцов**. В теле функции нужно использовать обычную двойную индексацию.



# Вывод двумерного динамического массива

```
void prindyn(int **x, int nrow, int ncol)
{ int i, j;
  for (i = 0; i < nrow; i++)
  { for (j = 0; j < ncol; j++)
    cout << x[i][j] << " ";
    cout << endl;
  }
}

void main(...)
{ int n, m, **a, i, j; cin >> n >> m;
  a = new int* [n];
  for (i = 0; i < n; i++)
    a[i] = new int [m];
  ...
  prindyn(a, n, m);
}
```

# Координаты минимума в матрице

```
void mincoor(double **x, int nrow, int ncol,
             int *rmin, int *cmin)
{
    int i, j, rm = 0, cm = 0;
    for (i = 0; i < nrow; i++)
        for (j = 0; j < ncol; j++)
            if (x[rm][cm] > x[i][j])
                { rm = i; cm = j; }
    *rmin = rm; *cmin = cm;
}

void main(...)
{
    double **a; int rmin, cmin, nrow, ncol;
    ...
    mincoor(a, nrow, ncol, &rmin, &cmin);
    cout << rmin << " " << cmin;
}
```

# Координаты минимума в матрице

```
void mincoor(double **x, int nrow, int ncol,
             int &rmin, int &cmin)
{
    int i, j; rmin = cmin = 0;
    for (i = 0; i < nrow; i++)
        for (j = 0; j < ncol; j++)
            if (x[rmin][cmin] > x[i][j])
                { rmin = i; cmin = j; }
}

void main(...)
{
    double **a; int rmin, cmin, nrow, ncol;
    ...
    mincoor(a, nrow, ncol, rmin, cmin);
    cout << rmin << " " << cmin;
}
```