

М А С С И В Ы

Массив – сложный (составной) тип данных, представляющий собой последовательность (конечную) элементов одного типа. Число элементов массива называют его **размером**.

Каждый элемент массива определяется именем массива и порядковым номером – *индексом*. **Индекс** – целое число, по которому производится доступ к элементу массива.

Индексов может быть несколько. В этом случае массив называют многомерным, а количество индексов одного элемента массива является его **размерностью**.

Одномерные массивы

Объявление одномерного **статического** массива:

Тип **Имя_Массива** [**Размер**] = { **Список значений** };

Тип – базовый тип, или известный тип Пользователя;

Размер – максимальное количество элементов (меньше можно использовать, больше – НЕТ).

Список значений может использоваться для инициализации, а может и отсутствовать.

При объявлении массива можно использовать атрибуты «класс памяти» и *const*.

Размер массива задается **только константой или константным выражением**, т.к. размер массива вместе с его типом определяет объем выделяемой **на этапе компиляции** памяти (статический массив).

Для работы с массивами переменного размера – создание динамических массивов.

Примеры объявлений:

1) `double a[20];`

2) `const int N = 20;`
`double a[N];`

3) `#define N 20 // Макроподстановка определить`

`...`

`double a[N];`

4) объявления массива с инициализацией:

`int a[5] = { 1, 2, 3 };`

Если в группе `{...}` список значений короче, то оставшимся элементам присваивается 0.

Индексы массивов в языке Си начинаются с 0, т.е. в массиве

```
int a[5];
```

первый элемент: $a[0]$, второй – $a[1]$, ..., пятый (последний) – $a[4]$.

Обращение к элементу массива осуществляется с помощью **операции индексации []** (квадратные скобки) :

Имя_Массива [**Индекс**]

Индекс – любое выражение целого типа, значение которого не выходит за указанный в объявлении *Размер*, например

```
a[3] = a[1] + 2;      a[i + 1]++;
```

Внимание. В языке Си *нет* контроля выхода индексов за границы размера массивов. При необходимости такой механизм должен быть запрограммирован явно.

Массив – последовательность данных и поэтому большинство операций выполняются в цикле. Рассмотрим некоторые из них для массива ***a***.

Имеем следующее объявление

```
int a[50], i, n;
```

50 – размер (максимальное количество, можно использовать меньше, но не больше), ***i*** – текущий индекс, ***n*** – размер.

Рассмотрим наиболее часто используемые участки работы наших программ.

1) Организация ввода исходных данных с клавиатуры с проверкой ошибочного ввода размера n :

```
cout << " Input n (<=50) ";
```

```
cin >> n;
```

```
if ( n < 0 || n > 50 ) {
```

```
cout << " Error ! " << endl;
```

```
getch();
```

```
return;      Если void main()
```

Или

```
return 0;  если int main(...)
```

```
}
```

```
for (i = 0; i < n; i++) {
```

```
cout << " a[ " << i+1 << " ] = " ;
```

```
cin >> a[i];
```

```
}
```

2) Заполнение массива **a** случайными числами в диапазоне $[-10, 10]$ и вывод их на экран

а) в столбик:

```
for (i = 0; i < n; i++) {  
    a[i] = random(21) - 10;  
    cout << a[ i ] << endl ;  
}
```

б) в строчку:

```
for (i = 0; i < n; i++) {  
    a[i] = random(21) - 10;  
    cout << setw(5) << a[ i ] ;  
}
```

Функция ***random(m)*** генерирует случайные числа в диапазоне $[0, m - 1]$;

функция ***rand()*** генерирует случайные числа в диапазоне $[0, Max_Int - 1]$ (описаны в файле ***stdlib.h***).

3) Поиск максимального элемента массива **a** :

а) по номеру (индекс максимального `i_max`):

```
int i_max = 0;
for (i = 1; i < n; i++)
    if ( a[i] > a[i_max] ) i_max = i;
cout << " Max = " << a[i_max]
     << " Index = " << i_max << endl ;
```

б) по значению:

```
int max;
max = a[0];
for (i = 1; i < n; i++)
    if ( a[i] > max ) max = a[i];
cout << " Max = " << max << endl ;
```

4) Сортировка ??? элементов массива a :

а) пузырьек с перестановками (r – дополнительная переменная для перестановки элементов):

```
for (i = 0; i < n-1; i++)  
    for (j = i+1; j < n; j++)  
        if ( a[i] > a[j] ) {  
            r = a[i];           - Переставляем  
            a[i] = a[j];       элементы a[i]  
            a[j] = r;          и a[j]  
        }
```

??? – Какая здесь сортировка: по возрастанию, или по убыванию?

б) пузырек с выбором и перестановкой (*int i_v* – дополнительная переменная для выбора индекса нужного элемента):

```
for (i = 0; i < n-1; i++) {  
    i_v = i;  
    for (j = i+1; j < n; j++)  
        if ( a[i_v] > a[j] )  
            i_v = j;  
    r = a[i];          - Переставляем  
    a[i] = a[i_v];     элементы a[i]  
    a[j_v] = r;        и a[i_v]  
}
```

Перед перестановкой еще можно поставить проверку

```
if(i_v != i) { Выполняем перестановку }
```

Рассмотрим некоторые примеры в помощь к выполнению индивидуальных заданий лабораторной работы № 5 (в одномерном *int* массиве из не более **10** элементов). Для решения задачи **обязательно** должен быть ввод массива с клавиатуры.

1. Найти сумму элементов, расположенных до первой **ПЯТЕРКИ**.

Рассмотрим возможные варианты значений массива:

а) 1 2 3 2 : Пятерки **НЕТ!!!**

б) 5 2 3 2 : Пятерка **ПЕРВАЯ**

в) 1 2 -3 5 : Сумма есть и = 0

В вариантах а) и б) **НЕТ** суммы.

Для проверки варианта **a)** можно использовать значение искомого индекса (обозначим его *int i5*), равного любому отрицательному значению, т.к. такого индекса в массиве не может быть!!!

Вариант решения:

```
int a[10], i, n, i5 = -2, и другие ...;
```

```
...
```

```
for (i = 0; i < n; i++)      - Поиск первой пятерки
```

```
    if ( a[i] == 5 ) {
```

```
        i5 = i;      - Индекс найденной «5»
```

```
        break;     - Заканчиваем поиск (цикл)
```

```
    }
```

```
if ( i_5 < 0 ) {                - Вариант а)  
    cout << " Not 5" << endl;    - НЕТ «5»  
    return;                      - Заканчиваем программу  
}
```

```
if ( i_5 == 0 ) {              - Вариант б)  
    cout << " First 5" << endl;  - Первая «5»  
    return;                      - Заканчиваем программу  
}
```

```
for(sum = i=0; i<i5;i++)      - Ищем сумму в)  
    sum += a[i];  
cout << "Sum = " << sum << endl;  
...
```

2. Найти сумму элементов, расположенных после последней **ПЯТЕРКИ**.

Рассмотрим возможные варианты значений массива:

а) 1 2 3 2 : Пятерки **НЕТ!!!**

б) 1 2 3 5 : «5» **ПОСЛЕДНЯЯ**

в) 5 1 2 -3 : Сумма есть и = 0

В вариантах а) и б) **НЕТ** суммы.

Для проверки варианта **а)** используем значение искомого индекса (***int i5***), равного любому отрицательному значению, как в предыдущем случае.

Вариант решения (объявления те же):

...

for (i = n-1; i >= 0; i--) - Поиск последней пятерки
выполняем в цикле начиная с конца массива

```
if ( a[i] == 5 ) {
```

```
    i5 = i;    - Индекс найденной «5»
```

```
    break;    - Заканчиваем поиск (цикл)
```

```
}
```



```
if ( i_5 < 0 ) {           - Вариант а)
    cout << " Not 5" << endl;    - НЕТ «5»
    return;                - Заканчиваем программу
}
```

```
if ( i_5 == n-1 ) {      - Вариант б)
    cout << " End 5" << endl;  - Последняя «5»
    return;              - Заканчиваем программу
}
```

```
for(sum = 0, i=i5+1; i<n;i++) - Ищем сумму в)
    sum += a[i];
cout << "Sum = " << sum << endl;
...

```

3. Найти сумму элементов, расположенных между первой и последней **ПЯТЕРКАМИ**.

Рассмотрим возможные варианты значений массива:

а) 1 2 3 2 : Пятерки **НЕТ!!!**

б) 1 2 3 5 : «5» **ОДНА**

в) 5 5 2 -3 : Пятерки **РЯДОМ**

г) 5 -7 7 5 : Сумма есть и = 0

В вариантах а), б) и в) **НЕТ** суммы.

Для проверки вариантов **а)** и **б)** используем значения искомых индексов (*int i51* – индекс первой пятерки, *i52* – индекс последней пятерки), равных любому отрицательному значению (как в предыдущих примерах).

Вариант решения:

```
int a[10], i, n, i51 = -2, i52 = -2, и другие ...;
```

...

Поиск первой пятерки выполняем как в примере 1:

```
for (i = 0; i < n; i++)
```

```
    if ( a[i] == 5 ) {
```

```
        i51 = i;    - Индекс первой «5»
```

```
        break;    - Заканчиваем поиск (цикл)
```

```
    }
```

```
if ( i51 < 0 ) {          - Вариант а)
```

```
    cout << " Not 5" << endl;    - НЕТ «5»
```

```
    return;              - Заканчиваем программу
```

```
}
```

Продолжаем, если *нашли первую ПЯТЕРКУ*, ищем последнюю пятерку, как в примере 2) *до найденной i51*:

```
for (i = n-1; i > i51; i--)  
    if ( a[i] == 5 ) {  
        i52 = i;    - Индекс последней «5»  
        break;    - Заканчиваем поиск (цикл)  
    }  
  
if ( i_52 < 0 ) {    - Вариант б)  
    cout << " One 5" << endl;    - ОДНА «5»  
    return;    - Заканчиваем программу  
}
```

```
if ( i52 - i51 == 1 ) {           - Вариант в)
    cout << "5 - 5" << endl; - 5-ки РЯДОМ
    return;           - Заканчиваем программу
}
for(sum = 0, i=i5+1; i<i52;i++) - Ищем сумму в)
    sum += a[i];
cout << "Sum = " << sum << endl;
. . .
```