

Массивы

Лекция 4

Структуры данных

- ⊙ Элементарными единицами данных являются значения того или иного стандартного типа, связанные с литералами, поименованными константами или переменными
- ⊙ Эти значения можно группировать и создавать более или менее сложные *структуры данных*
- ⊙ Каждая такая структура может получить свое имя и рассматриваться как переменная *составного* или *агрегатного типа*

Доступ к элементам

- ⊙ Таким образом, с переменной составного типа (структурой данных) в каждый момент времени связано некоторое множество значений
- ⊙ Отдельные значения – *элементы структуры данных* – выделяются путем специальных *операций извлечения*

Определение массива

- ⊙ Наиболее простой и часто используемой структурой данных является *массив*
- ⊙ Массив – это набор некоторого числа однотипных данных, расположенных в последовательных ячейках памяти
- ⊙ Количество элементов массива называется его *размером*, а тип элементов – *типом массива*

Объявление массивов

- Синтаксис объявления массива:

<тип массива> <имя массива> [<размер массива>]

<размер массива> – это литерал или константное выражение

- В соответствии с объявлением массива для его размещения будет выделена область памяти длиной

*<размер массива> * **sizeof** <тип массива>*

байт

- Например: **int** a[5]; **float** x[n+m]; **double** q[4];

Инициализация массива

- ⊙ Объявление массива может сопровождаться его *инициализацией*
- ⊙ Синтаксис объявления массива с инициализацией:
<тип массива> <имя массива> [<размер массива>] = {<список значений>}
- ⊙ В этом случае элементы массива получают значения из списка инициализации
- ⊙ В список инициализации могут входить любые вычисляемые выражения

Примеры объявлений

- Одномерный массив:

```
int a[5] = { 3, 45, 11, -8, 74};
```

```
double q[4] = {1.7, 4.53};
```

- Во втором случае инициализируются только два первых элемента массива **x**, а оставшиеся два элемента получают нулевые значения

- При наличии списка инициализации размер массива можно не указывать, он определяется по числу инициализирующих значений:

```
int a[ ] = { 3, 45, 11, -8, 74};
```

Массивы в C++

В неинициализированном массиве (по аналогии с неинициализированной переменной) будут храниться заранее неизвестные значения (какой-то «мусор», ранее записанный другими программами или даже вашей программой в выделяемую для объявленного массива память).

Обращение к элементам массива

- ⊙ Производится с помощью числовых индексов, причем индексация начинается с нуля
- ⊙ В случае массива операция извлечения – это бинарная операция «квадратные скобки»
- ⊙ Первым операндом является имя массива, вторым – целочисленное выражение, заключенное в квадратные скобки

$a[0] = a[i] + a[2 * i + 1];$

Индексация элементов массива

- ⦿ Индексация элементов массива начинается с нуля
- ⦿ Таким образом, первому элементу массива соответствует значение индекса 0, второму – значение индекса 1, элементу с порядковым номером k – значение индекса $k-1$

Чтобы обратиться к какому-то из элементов массива для того, чтобы прочитать или изменить его значение, нужно указать имя массива и за ним индекс элемента в квадратных скобках.

Элемент массива с конкретным индексом ведёт себя также, как переменная.

Например, чтобы вывести значения первого и последнего элементов массива **mas1** надо написать в программе:

```
cout << mas1[0];  
cout << mas1[3];
```

А чтобы присвоить новые значения (10, 20, 30, 40) всем элементам массива, потребуется написать в программе:

- ⦿ `mas1[0] = 10;`
- ⦿ `mas1[1] = 20;`
- ⦿ `mas1[2] = 30;`
- ⦿ `mas1[3] = 40;`

Уже из последнего примера видно, что для того, чтоб обратиться ко всем элементам массива, придется повторять однотипные действия.

Для многократного повторения подобных операций используются циклы. Соответственно, мы могли бы заполнить массив нужными элементами с помощью цикла:

```
for(int i=0; i<4; i++)  
{  
    mas1[i] = (i+1) * 10;  
}
```



А после этого несложно вывести все элементы массива на экран:

```
for(int i=0; i<4; i++)  
{  
    cout << mas1[i] << ' ';  
}
```



Пример программы

```
#include <iostream>
using namespace std;
int main() {
    cout << "Укажите размер массива: ";
    int n;
    cin >> n;
    const int dim = n;
    int arr[dim];
    for(int i=0; i<dim; i++)
    {
        arr[i] = i+1;
    }
    for(int i=dim-1; i>=0; i--)
    {
        cout << arr[i] << ' ';
    }
}
```

Заполнение массивов

- ⊙ Для массивов больших размеров инициализация, как правило, не производится и их заполнение выполняется в процессе работы программы
- ⊙ Одним из способов решения проблемы заполнения массивов является использование псевдослучайных чисел
- ⊙ Генерация таких чисел осуществляется функцией `rand()` из библиотеки `stdlib` (заголовочный файл `<stdlib.h>`)

Функция rand()

- ◎ Целочисленная функция **rand()** возвращает псевдослучайное число из диапазона
0 .. **RAND_MAX**,
где константа **RAND_MAX = 0x7fff (32535)**
- ◎ Для задания другого диапазона следует использовать формулу:
rand() % (max-min+1)+min,
где **min** и **max** – нижняя и верхняя границы требуемого диапазона

Функция rand()

- ⊙ Для получения псевдослучайных вещественных значений в заданном диапазоне удобно использовать следующую формулу:
$$(\text{float}) \text{rand}() / \text{RAND_MAX} * (\text{max} - \text{min}) + \text{min}$$
- ⊙ В этом выражении целое значение, возвращаемое функцией **rand()** явным образом преобразуется в вещественное, т.к. в противном случае всегда будет получаться нулевое значение

```
#include <iostream>
#include <cstdlib>
#include <time.h>
using namespace std;
int main()
{
// определяем переменные
int a[10];
int i;
srand(time(0)); // инициализация генерации случайных чисел
//генерируем целый случайный массив из 10 элементов от 1
до 5
for (i = 0; i < 10; i++)
    a[i] = 1 + rand() % 5;
for (i = 0; i < 10; i++)
    cout<<a[i]<<" "; // вывод результата на экране
}
```

main.cpp

```
1 #include <iostream>
2 #include <cstdlib>
3 #include <time.h>
4 using namespace std;
5
6 int main()
7 {
8     // определяем переменные
9     int a[10];
10    int i;
11
12    srand(time(0)); // инициализация генерации случайных чисел
13
14    //генерируем целый случайный массив из 10 элементов от 1 до 5
15    for (i = 0; i < 10; i++)
16        a[i] = 1 + rand() % 5;
17    for (i = 0; i < 10; i++)
18        cout<<a[i]<<" "; // вывод результата на экране
19 }
20
```

input

4 2 3 4 5 1 5 5 4 2

...Program finished with exit code 0
Press ENTER to exit console.

Заполнение массива числами, введёнными пользователем

```
#include <iostream>
#include <time.h>
using namespace std;
int main()
{
// определяем переменные
int a[10];
int i;
for (i = 0; i < 10; i++)
{
cout << "Введите значение для эл-та массива " << i << ": ";
cin >> a[i];
}
for (i = 0; i < 10; i++)
cout<<a[i]<<" "; // вывод результата на экране
}
```

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     // определяем переменные
7     int a[10];
8     int i;
9
10    for (i = 0; i < 10; i++)
11    {
12        cout << "Введите значение для элемента массива " << i << ": ";
13        cin >> a[i];
14    }
15    for (i = 0; i < 10; i++)
16        cout << a[i] << " "; // вывод результата на экране
17 }
18
```

input

```
Введите значение для элемента массива 0: 1
Введите значение для элемента массива 1: 2
Введите значение для элемента массива 2: 3
Введите значение для элемента массива 3: 4
Введите значение для элемента массива 4: 5
Введите значение для элемента массива 5: 6
Введите значение для элемента массива 6: 7
Введите значение для элемента массива 7: 8
Введите значение для элемента массива 8: 9
Введите значение для элемента массива 9: 10
1 2 3 4 5 6 7 8 9 10
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

Двумерные массивы

- ⊙ В языке C++ такие массивы рассматриваются как одномерные массивы одномерных массивов
- ⊙ Поэтому такой массив может быть определен следующим образом:

```
int a[10] [5];
```

Инициализация массива

- ⊙ Двумерный массив может инициализироваться как одномерный массив:

```
int a[2][3] = { 3, 45, 11, -8, 74, -10};
```

или как массив массивов:

```
int a[2][3] = { {3, 45, 11}, {-8, 74, -10}};
```

- ⊙ При наличии инициализатора в определении двумерного массива можно не указывать размер по первому измерению, например:

```
int a[ ][3] = { {3, 45, 11}, {-8, 74, -10}};
```


Обращение к элементу массива

- ⊙ Для двумерных массивов каждый из индексов записывается в отдельных квадратных скобках:
 $a[0][2] = a[1][2] + 4;$
- ⊙ Поскольку элементы двумерного массива располагаются в оперативной памяти в виде непрерывной последовательности, то возможно обращение к элементу массива с использованием одного индексного выражения

Пример обращения

- ⊙ Пусть определение массива имеет вид:

int a [m] [n],

где **m**, **n** – константы

- ⊙ Тогда эквивалентными являются два обращения: **a**
[i] [j] и **a[i*m+j]**

Примеры программ

- Программа «Заполнение целыми числами»
- Листинг программы

- Программа «Заполнение вещественными числами»
- Листинг программы

Поиск в массиве

- ⊙ Существует две основных формулировки задачи поиска:
 - найти элемент массива (первый или последний), удовлетворяющий заданному условию;
 - найти все элементы массива, удовлетворяющие некоторому условию;
- ⊙ Любой поиск связан с последовательным просмотром элементов массива и проверкой их соответствия условию поиска

```
#include <iostream>
using namespace std;
int main()
{
    int a[5];
    int i,x;
    x=3;
    for (i = 0; i < 5; i++)
    {
        a[i]=i+1;
        cout<<a[i]<<"    "<<endl;
    }
    for (i = 0; i < 5; i++)
        if(a[i] = x)
            cout<<a[i]<<" "; // вывод результата на экране
}
```

Поиск единственного элемента

- ⊙ В этом случае основу алгоритма решения задачи составляет цикл, содержащий в качестве условия продолжения отрицание условия поиска
- ⊙ Например, требуется проверить, есть ли среди элементов массива A длиной n элемент со значением, равным заданному значению x

Результаты поиска

- ⊙ Возможны две ситуации:
 - такой элемент существует, тогда при некотором значении индекса i выполняется условие $A[i]=x$;
 - такого элемента в массиве нет
- ⊙ В первом случае поиск нужно завершать при обнаружении искомого элемента, во втором – при достижении конца массива

Условие завершения

- Формально такое условие завершения поиска записывается в виде:

$$A[i] = x \text{ ИЛИ } i=n$$

- Отрицание этого условия, в соответствии с правилом де Моргана, имеет вид:

$$A[i] \neq x \text{ И } i < n$$

- Поскольку основная задача поиска решается при проверке условия, то тело цикла должно содержать только инкремент индексной переменной

Цикл поиска

- ◎ Цикл поиска в нотации C++ принимает вид:
 i=0;
 while (*A*[*i*] != *x* && *i*<*n*) *i*++;
- ◎ Условие *i*≤*n* заменено на *i*<*n*, чтобы не допустить выхода за границу массива

Результат поиска

- ⦿ Поскольку условие цикла является конъюнкцией двух простых условий, то после завершения цикла необходимо проверить основное из них:

```
if (i < n) printf(“Элемент найден”);
```

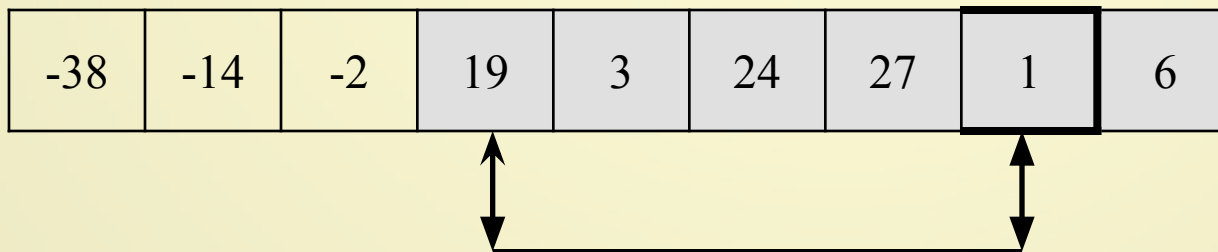
```
else printf(“Элемент не найден”);
```

Сортировка массива

- ◎ Сортировкой массива называется упорядочение значений его элементов по возрастанию или убыванию
- ◎ Рассмотрим три простых алгоритма сортировки:
 - сортировка методом выбора,
 - сортировка методом включения,
 - сортировка методом обмена

Сортировка методом выбора

- Основная идея этого метода заключается в последовательном формировании отсортированной части массива путем добавления в ее конец очередного элемента, выбранного в его неотсортированной части

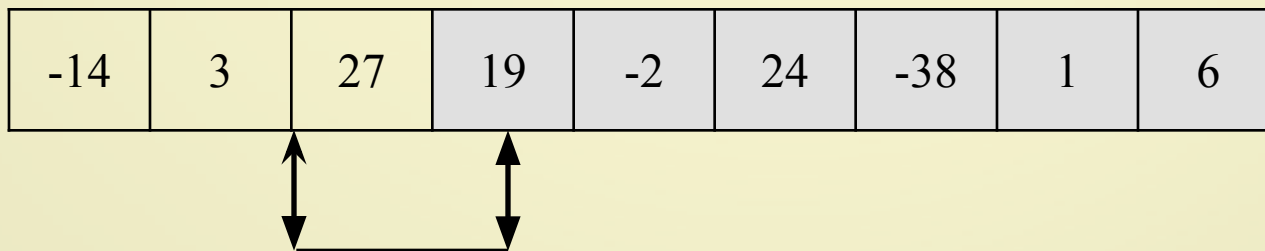


Текст программы

```
const int N = 10;
void main()
{ int i, j, nMin, A[N], c;
  // здесь нужно ввести массив A
  for ( i = 0; i < N-1; i ++ ) // i – индекс первого элемента в неотсорт. части
  { nMin = i; // ищем минимальный элемент в неотсортированной части
    for ( j = i+1; j < N; j ++ );
    if ( A[j] < A[nMin] ) nMin = j;
    if ( nMin != i ) // перемещаем минимальный элемент в начало
    { c = A[i]; A[i] = A[nMin]; A[nMin] = c; } // неотсортированной части
  }
  printf("\n Отсортированный массив:\n");
  for ( i = 0; i < N; i ++ )
    printf("%d ", A[i]);
}
```

Сортировка методом вставок

- Отсортированная часть массива также формируется путем последовательного добавления в нее элементов из его неотсортированной части
- Однако теперь в качестве очередного берется первый элемент неотсортированной части
- Место его размещения в отсортированной части выбирается так, чтобы сохранить уже имеющийся там порядок сортировки

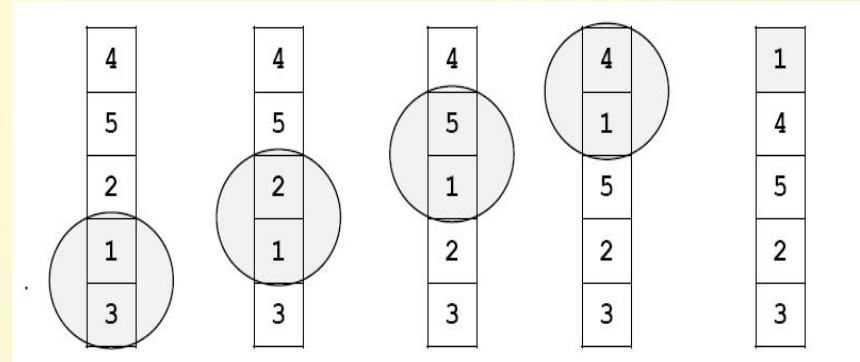


Текст программы

```
const int N = 10;
void main()
{ int i, j, nMin, A[N], c;
  // здесь нужно ввести массив A
  for ( i = 1; i < N; i ++ )
  { c = A[i];
    j = i - 1; // ищем в отсортированной части место для размещения
    while ( j >= 0 && A[j] > c) A[j+1] = A[j--]; // очередного элемента
    A[j+1] = c;
  }
  printf("\n Отсортированный массив:\n");
  for ( i = 0; i < N; i ++ )
    printf("%d ", A[i]);
}
```

Сортировка методом обмена

- Этот метод сортировки имеет жаргонное наименование «метод пузырька» и заключается в многократном упорядочении пар соседних элементов



Текст программы

```
const int N = 10;
void main()
{
    int i, j, A[N], c;
    // здесь надо ввести массив A
    for ( i = 0; i < N-1; i ++ ) // цикл повторных проходов по массиву
        for ( j = N-2; j >= i; j -- ) // идем с конца массива в начало
            if ( A[j] > A[j+1] ) // если они стоят неправильно, ...
                {
                    c = A[j]; A[j] = A[j+1]; A[j+1] = c; // переставить A[j] и A[j+1]
                }
    printf("\n Отсортированный массив:\n");
    for ( i = 0; i < N; i ++ ) printf("%d ", A[i]);
}
```

Сравнение методов

- ⦿ Все три алгоритма имеют, в среднем, одинаковую эффективность и выбор одного из них может определяться особенностями задачи, а также личными пристрастиями программиста

Конец лекции
