Массивы

Лекция 4

Массивы

- Последовательная группа ячеек памяти, имеющих одинаковое имя и одинаковый тип
- Массивы бывают динамические и статические
- Размер статического массива известен в момент компиляции программы (**`build-time`**) и не может меняться
- Размер динамического массива может изменяться во время работы программы (**`run-time`**)
- Работа со статическими массивами проще, обращение к элементам происходит быстрее
- Невнимательность в работе с динамическими массивами может приводить к ошибкам времени выполнения

Индекс массива

- На любой элемент массива можно сослаться, указав имя массива и номер позиции элемента, заключенный в квадратные скобки
- Номер позиции, указанный в квадратных скобках, называется **индекс**
- **Индекс** - это расстояние между первым элементом массива и тем, на который указывает индекс
- Индекс – целое число. Индексом может быть
  - константа
  - переменная
  - выражение
- Индексы элементов начинаются с нуля

Объявление и инициализация массивов

- Для объявления массива необходимо указать:
  - тип его элементов
  - имя массива
  - количество элементов массива в квадратных скобках (целочисленная константа)
- Для инициализации массива в момент объявления необходимо
  - После объявления поставить знак присваивания
  - В фигурных скобках через запятую перечислить все элементы массива
  - Если массив нужно инициализировать нулями, достаточно указать один ноль в фигурных скобках

Объявление и инициализация массивов

Объявление массива:

```
int arr[8] = {1,2,3,4,5,6,7,8};
```

Выделение памяти и ее заполнение:

1	2	3	4	5	6	7	8

Условные адреса элементов массива:

A0	A1	A2	A3	A4	A5	A6	A7
1	2	3	4	5	6	7	8

Объявление и инициализация массивов

Объявление массива:

Плохо!!!
В элементах массивов лежит «мусор»!

```
int c[12];
double b[100], x[27];
```

Плохо!!!
Размер массива указан как число

Объявление и инициализация:

Во втором случае количество элементов будет рассчитано автоматически

```
int n[5] = {5,4,3,2};
int n[ ] = {5,4,3,2,1};
```

Плохо!!!
Размер массива указан как число

Инициализация нулями:

Хорошо!!!
Все элементы массива будут заполнены нулями

```
const int SIZE = 10;
int s[SIZE]={0};
```

Хорошо!!!
Размер массива указан как константная переменная

Инициализация символьного массива (строки)

Два одинаковых варианта объявления и инициализации (строки):

```
char myString1 [ ] = "first";
```

```
char myString2 [ ] =
      {'f', 'i', 'r', 's', 't', '\0'};
```

Для работы со строками в C++ лучше использовать класс **std::string** (библиотека **<string>**):

```
std::string myString3 = "first";
```

Обращение к элементу массива

Объявление массива:

```
int arr[8] = {1,2,3,4,5,6,7,8};
```

Вывод на экран пятого элемента:

```
std::cout << arr[4] << std::endl;
```

Изменение значения элемента с индексом 5 и последующий вывод на экран:

```
arr[5] = 21;
std::cout << arr[5] << std::endl;
```

Пятый элемент массива имеет индекс 4
Элемент с индексом 5 на самом деле шестой

Обращение к элементу массива

- В качестве индекса может выступать выражение, результатом которого является целое число

```
int a = 5, b = 2;
const int SIZE = 10;
int arr [SIZE] = {0};
arr [a + b] = 15;
std::cout << arr [a + b] << std::endl;
```

- Элемент одного массива может быть индексом для другого

```
const int SIZE = 36000, N_STATISTIC = 11;
int diceThrows[SIZE] = {0};
int statistic [N_STATISTIC] = {0};
…
statistic [diceThrows [i] ] ++;
```

Печать массива

Для работы с каждым элементом массива нужен цикл

```
const int SIZE = 5;
float arr[SIZE] = { 1.2, 2.1, 3.0, 4.15, 5.3 };
for (int i = 0; i < size; i++)
{
    std::cout << arr[i] << std::endl;
}
```

Для печати строки цикл не нужен

```
char myString[] = "Hello, world!";
std::cout << myString << std::endl;
```

```
std::string myString = "Hello, world!";
std::cout << myString << std::endl;
```

Пузырьковая сортировка

- Данная сортировка получила название пузырьковая сортировка или сортировка погружением, потому что наименьшее значение постепенно «всплывает», продвигаясь к вершине (началу) массива, подобно пузырьку воздуха в воде, тогда как наибольшее значение погружается на дно (конец массива).
- Одна из простейших сортировок
- Эффективна лишь для небольших массивов.
- Сложность алгоритма $O(n^2)$
- Лежит в основе некоторых более совершенных алгоритмов, таких как шейкерная сортировка, пирамидальная сортировка и быстрая сортировка

Пузырьковая сортировка

- Алгоритм состоит из повторяющихся проходов по сортируемому массиву.
- За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов.
- Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.
- При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива

Пузырьковая сортировка

```
const int SIZE = 10;
int arr[SIZE] = { 2,6,4,8,10,12,89,68,45,37 };
for (int pass = SIZE - 1; pass > 0; --pass) {
    bool isSorted = true;
    for (int i = 0; i < pass; i++) {
        if (arr[i] > arr[i + 1]) {
            int hold = arr[i];
            arr[i] = arr[i + 1];
            arr[i + 1] = hold;
            isSorted = false;
        }
    }
    if (isSorted)
        break;
}
```

Линейный поиск

- Линейный поиск последовательно сравнивает каждый элемент массива с ключом поиска.
- Поскольку массив может быть неупорядочен, вполне вероятно, что отыскиваемое значение окажется первым же элементом массива. Но в среднем программа должна сравнить с ключом поиска половину элементов массива
- Метод линейного поиска хорошо работает для небольших массивов или в случае одиночного поиска в неупорядоченном массиве
- Однако для больших массивов линейный поиск неэффективен. В этом случае массив нужно отсортировать, а затем использовать высокоэффективный метод двоичного поиска.

Линейный поиск

```
const int SIZE = 10;
int arr[SIZE] = {13,6,4,8,10,12,89,68,45,37};
int searchKey = 0, element = -1;
std::cout << "Enter search key: ";
std::cin >> searchKey;
for (int i = 0; i< SIZE; i++) {
   if (arr[i] == searchKey) {
       element = i;
       break;
   }
}
```

Генератор псевдослучайных чисел

- В C++ есть стандартный генератор псевдослучайных чисел (ГПСЧ)
- ГПСЧ реализован в виде функции `rand()` библиотеки `<stdlib.h>`
- Функция `rand()` возвращает целые неотрицательные числа в диапазоне от `0` до `RAND_MAX`
- Константа `RAND_MAX` определена в файле `<stdlib.h>` и равна `65355`

Задание диапазона ГПСЧ

- Для задания нужного диапазона целых чисел необходимо выполнить следующее:
  - Сосчитать количество чисел в диапазоне
  - Результат функции `rand()` поделить по модулю на количество чисел в диапазоне
  - Если диапазон начинается не с нуля, сдвинуть начало диапазона на стартовое число

Диапазон	Формула
[-100, 100]	`rand() % 201 - 100;`
[0, 100)	`rand() % 100;`
[100, 200]	`rand() % 101 + 100;`

Сдвиг генератора псевдослучайных чисел

- На одном компьютере ГПСЧ всегда будет выдавать одну и ту же последовательность чисел
- Для получения разных последовательностей нужно использовать функцию, сдвигающую ГПСЧ, - **`srand()`** той же библиотеки **`<stdlib.h>`**
- В качестве значения сдвига используется системное время, получаемое в функции **`time()`** библиотеки **`<time.h>`**
- Сдвиг ГПСЧ должен выполняться в программе один раз

Пример работы с ГПСЧ

```
const int SIZE = 10000;
int arr[SIZE] = { 0 };
srand (time ( 0 ) );
for (int i = 0; i < SIZE; i++) {
    arr[i] = rand() % 201 - 100;
}
```

Многомерные массивы

- Массивы в C++ могут иметь много индексов
- Обычным представлением многомерных массивов являются таблицы значений, содержащие информацию в строках и столбцах
- Чтобы определить отдельный табличный элемент, нужно указать два индекса:
  - первый (по соглашению) показывает номер строки
  - второй (по соглашению) - номер столбца
- Таблицы или массивы, которые требуют двух индексов для указания отдельного элемента, называются двумерными
- Фактически двумерный массив – это массив, элементами которого являются одномерные массивы

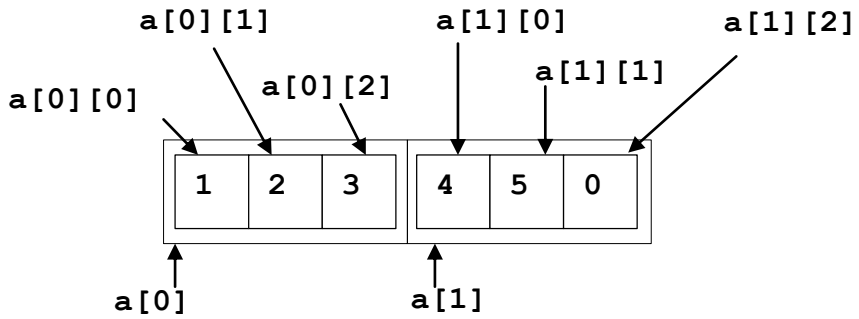
Многомерные массивы

```
int a [2][3] = {1,2,3,4,5};
```

a

1	2	3
4	5	0

Элементы, которым «не хватило» начальных значений, обнуляются



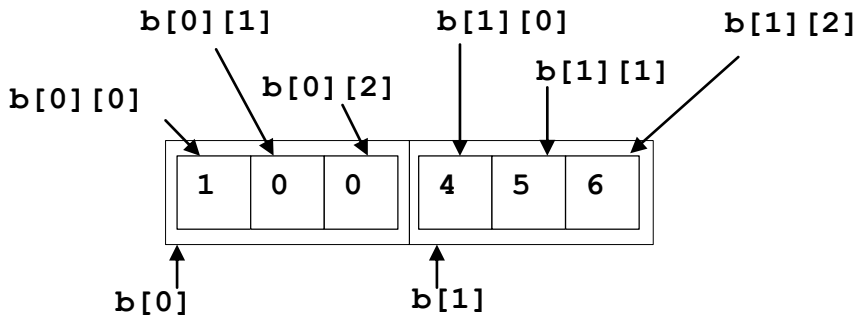
Многомерные массивы

```
int b [2][3] = { {1}, {4,5,6} };
```

b

1	0	0
4	5	6

В первой строке инициализирован только один элемент



Работа с двумерным массивом

- Для работы с двумерным массивом необходимо 2 цикла – один вложенный в другой
- Количество строк и столбцов в общем случае может не совпадать
- Если количество строк и столбцов совпадает, то массив представляет собой квадратную матрицу
- Обычно внешний цикл перебирает строки, а внутренний - столбцы

Пример работы с многомерным массивом

```
const int ROW = 5, COLUMN = 4;
int arr[ROW][COLUMN] = { 0 };
srand (time ( 0 ) );
for (int i = 0; i < ROW; i++) {
    for (int j = 0; j < COLUMN; j++) {
        arr[i][j] = rand() % 201 - 100;
        std::cout << arr[i][j] << "\t";
    }
    std::cout << std::endl;
}
```

```
-65     -46     8       89
-83     100     24      -35
-100    -52     93      7
82      -50     -88     -37
-17     72      20      -87
```

Конец