Программирование и алгоритмизация

Санкт-Петербургский государственный политехнический университет

15 октября 2014

Массивы

Массив — это совокупность упорядоченных элементов. Каждый массив имеет имя.
Имя массива — база, относительно которой можно обратиться к любому элементу массива. Имя массива эквивалентно указателю, тип которого зависит от типа элементов и размерности массива.
Можно задать одномерный, двумерный, трехмерный и т.д. массивы, указав требуемое число размерностей.
Все элементы массива:

- одного и того же типа;
- занимают непрерывную область памяти;
- пронумерованы начиная с 0.

Доступ к элементам массива:

- с помощью оператора [];
- посредством оператора разыменования.

Объявление массива

В общем случае объявление массива совмещено с его определением, таким образом программисту необходимо указать размерность (количество элементов) массива, которая должна быть известна на этапе компиляции.
Исходя из контекста объявления массив может быть глобальным, локальным, статическим или заключенным в пространство имен.

```
char cArr [10];
int iArr [10];
float fArr [10];
```

Задание размерностей одномерного массива

```
const int N = 5;
char cArr [N]; //???

int M = 5;
int iArr [M]; //???

#define R 5
float fArr [R]; //???
```

Обращение к элементу массива

- Индекс может быть задан любым выражением, которое приводит к целому.
- Индексы нумеруются с НУЛЯ!
- Для повышения эффективности кода, компилятор НЕ КОНТРОЛИРУЕТ выход за ГРАНИЦЫ массива!

Обращение к элементу массива

```
int arr[10];
ar[5] = 55;
ar[9] = 70;
ar[10] = 90; //???
ar[-1] = 100; //???
ar[2.5] = 7; //???

int n;
std::cin >> n;
std::cin >> ar[n]; //???
```

Обращение к элементу многомерного массива

```
int arr[10][5];

arr[0][0]; //???
arr[5][3]; //???
arr[10][5]; //???
```

Инициализация массива

Бывает явная и неявная.

Неявная инициализация массива:

```
int arr1[10][5]; //???
namespace A { char arr1[10][5]; } //???

int main()
{
    static int arr2[100]; //???
    float arr4[3][4][5]; //???
    double arr5[]; //???
}
```

Явная инициализация массива

```
int myArr[3] = {1, 2, 3};
int myArr2[3] = {1, 2, 3, 4}; //???
int myArr3[3] = {1, 2}; //???

int myArr4[3] = {0}; //???

char chArr[5] = {'A', 'B', 'C'}; //???
char chArr[5] = {"ABC"}; //???
char chArr[5] = "ABC"; //???

char chArr[3] = "ABC"; //???

char chArr[10] = "ABC"; //???
```

Явная инициализация многомерного массива

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

int nArr2[2][3] = { {1,2,3}, {4,5,6} };

int nArr2[2][3] = { {1}, {4,6} };

char cStrings [3][10] = { "aaaa", "bbb" };
```

Явная инициализация многомерного массива

При наличии списка инициализации старшую размерность можно опустить.

```
char cString[] = "QWERTY"; //???

char cString2[] = { 'Q', 'W', 'E', 'R', 'T', 'Y' };

int nArr[][3] = {
  {0,1},
  {10},
  {20, 30, 40}
                 };
```

Особенности инициализации массивов

```
int ar[5];
ar = {1, 2, 3}; //???

int n1 = 4;
int iArr[4] = { 7, n1 }; //???

const int iArr2[5]; //???
const int iArr3[5] = {1, 2, 3}; //???
const int iArr4[] = {1, 2, 3}; //???
iArr4[0] = 7; //???
```

Вычисление размеров встроенных массивов

Размер массива можно вычислить используя оператор sizeof.
Либо задав размерность массива константой.

```
int myArr[10];
unsigned int size = sizeof(myArr); //???
unsigned int realSize = sizeof(myArr) / ?;


char ar[] = "qwerty"; //??? Как вывести строку
    наоборот: ytrewq

char ar[][10] = { "qwerty", "asdfg", "zzzz" }; //
    ??? Как вывести все строки наоборот?
```

Связь одномерных массивов и указателей

Имя одномерного массива является КОНСТАНТНЫМ указателем на его первый элемент.

```
int myArr[5];
int *p = myArr;

myArr[3] = 7; //???
p[3] = 8; //???

*(p + 3) = 100;
*(myArr + 3) = 100;

p++; //???
myArr++; //???
std::cout << *p; //???
std::cout << *myArr; //???
```

Коммутативность в массивах

```
int x;
int arr[10];

x = arr[1]; //???

x = *(arr + 1); //???

x = *(1 + arr); //???

x = [1]arr; //???
```

Вычисление адреса элемента массива

Адрес i-го элемента = база(имя массива) + i*sizeof(тип массива).

```
int *p;
int arr[10] = {1, 2, 0};

p = arr + 3; //адрес 3-го элемента (счет с нуля).
    Умножение на sizeof выполняет компилятор
    основываясь на типе данных массива.
```

Вычисление адреса в двумерных массивах

Двумерный массив, как и одномерный является непрерывной последовательностью элементов в области памяти.
Задавая двумерный массив, программист определяет правила вычисления компилятором адреса [i][j] элемента.

```
int dArr[2][3] = { {1,2,3}, {4,5,6} };
int (*p)[3] = dArr;

dArr[1][2] = 6;
p[1][2] = 8;
*( *(dArr+1) + 2) = 1;

p++; //???
dArr++; //???

std::cout << *p;
std::cout << **p;
std::cout << *dArr; //???
std::cout << **dArr; //???
```

Вычисление адреса в двумерных массивах

```
int dArr[2][3] = { {1,2,3}, {4,5,6} };
int (*p)[3];

p = &dArr[0];

std::cout << p[0] == &dArr[0][0]; //???
std::cout << p[1] == &dArr[1][0]; //???

p = &dArr[1];

std::cout << p[0][1]; //???
std::cout << p[1][1]; //???

//Задание: пользователь вводит 5 строк, введенные
    строки сохраняются в массиве. Вывести на печать
     только строки начинающиеся с буквы 'A'.
```

Сумма элементов и другие примеры

Пример: сумма элементов одномерного и двумерного массива.

```
int arr[6] = {1, 3, 5, 6, 7};
int sum = 0;
for(int i = 0; i < 6; i++) //Исправить, чтобы
    работало на любых массивах
{
  sum = sum + arr[i];
}

int arrD[2][3] = {1, 3, 5, 6, 7};
//Написать вычисление суммы массива. Как сделать
    эффективное вычисление суммы элементов массива.
int *p = arrD[0]; //одномерный указатель на первый
    элемент двумерного массива
for(int i = 0; i < 6; i++)
{
  sum = sum + *p; // sum += *p++;
  p++;
}
```

Трехмерный массив

```
int arr3D[2][3][4] = {
                      { {1, 2, 3, 4}, {5, 6, 7, 8},
                        {9, 10, 11, 12} },
                      { {-1,-2,-3,-4} }

int (*p)[3][4] = arr3D; //требуется указать младшие
    размености

p++;

//???
std::cout << p;
std::cout << *p;
std::cout << **p;
std::cout << ***p;
```

Адрес для 3-х мерного массива вычисляется следующим образом:

адрес элемента i-j-k = база(имя) + i*3*4*sizeof(int) +

Массивы указателей

Используются для хранения строк текста, а также динамически создаваемых объектов. Могут быть проинициализированы при определении.

```
char * ar[] = { "One", "Two", "Three" }; //FIXME
char arr[][6] = { "One", "Two", "Three" };

ar[0][0] = 'F'; //???
arr[0][0] = 'F'; //???
```

Данные массивы различаются расположением в памяти, а следовательно и их использованием.

Динамическая память

Предоставляет программисту возможность самому выделять и очищать участки памяти заданного размера. Такие участки памяти располагаются в области называемой кучей (heap). Выделение динамической памяти выполняется на этапе выполнения программы, следовательно не обязательно знать размер выделяемой области на этапе компиляции.

Операторы управления памятью

Выделение и освобождение памяти осуществляется с помощью конструкий языка или низкоуровневых функций:

- Для языка Си: функции malloc() и free().
- Для языка C++: конструкции new и delete.

Операторы выделения и освобождения памяти: Предоставляет программисту возможность самому выделять и очищать участки памяти заданного размера. Такие участки памяти располагаются в области называемой кучей (heap). Выделение динамической памяти выполняется на этапе выполнения программы, следовательно не обязательно знать размер выделяемой области на этапе компиляции.

Динамическая память

```
int n;
std::cin >> n;

int arr[n]; //ошибка при компиляции

int *pArr = (int*)malloc(sizeof(int) * n); //malloc
    требует размер выделяемой памяти в байтах
pArr[0] = 4; //обращение к динамической памяти
free(pArr); //очищает память выделенную
    пользователя
pArr = 0;

int *pArr = new int[n]; //malloc требует размер
    выделяемой памяти в байтах
pArr[0] = 4;
delete pArr; //очищает память пользователя,
    автоматически обнуляет указатель
```

Минусы использования динамической памяти

- Для небольших участков памяти служебная информация существенно увеличит количество занимаемой ими памяти.
- На процесс выделения и освобождения динамической памяти тратятся дополнительные ресурсы коспьютера.
- Постоянно выделение и освобождение памяти может привести к фрагментированию кучи.
- Программист должен быть внимательным и не зыбывать очищать занятую им память.

Пример небрежного манипулирования кучей

```
char *p = new char;
*p = 'A';
p = "QWERTY";
delete p; //???
```

Создание одномерного динамического массива

```
int n;
std::cin >>n;

int ar[n];
int *pn = new int[n];

for(int i =0; i < n; i++)
    pn[i] = i; //*(pn + i) = i;


delete [] pn; //Освобождение памяти

pn = nullptr; //новый тип обозначающий 0 указатель
```

Создание 3-х мерного массива

```
int n;
std::cin >>n;

float (*p) [2][3] = new float [n][2][3];

p[0][1][2] = 77;

delete [] p;
```

Создание двумерного массива как одномерного

```
int n, m;
std::cin >> n >>m;

int *arr = new int[n * m]; //Размерность массива
    вычисляется как n * m

arr[i*m + j] = 6; //Обращение к [i][j] элементу в
    одномерном массиве

delete [] arr;
```

Создание двумерного массива как массив указателей

Создание матриц.

```
int n, m;
std::cin >> n >>m;

int **arr = new int*[n]; //Создаем массив
    указателей размерностью n

for(int i = 0; i < n; i++)
    arr[i] = new int[m]; //для каждой строки
        указателей выделяется массив размера m

arr[i][j] = 6; //Обращение к [i][j] элементу в
    двумерном массиве

for(int i = 0; i < n; i++)
    delete [] arr[i]; //требуется удалить память от
        каждой выделенной строки!!!

delete [] arr;
```

Изменение размера существующего массива

```
int n;
std::cin >> n;
int *arr = new int[n];

for(int i = 0; i < n; i++)
    arr[i] = i * i;

int *temp = new int[n+10]; //Новый размер массива n
    +10
for(int i = 0; i < n; i++)
    temp[i] = arr[i];
delete [] arr;

arr = temp; //Обязательно ОЧИСТИТЬ память перед
    операцией присваивания указателя иначе
    получатся УТЕЧКИ ПАМЯТИ
arr[n+1] = 100;
delete [] arr; //после использования необходимо
    очистить динамическую память
```