Матрицы

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

8 апреля 2017 г.

Версия: 0.10

Аннотация

Показаны различные традиционные способы задания матриц, как двумерных, так и многомерных.

Цель. Научится работать с матрицами.

Предварительный вариант!

1 Матрицы

В первом подразделе будет показано как создавать многомерный массив статического размера, т.е. заданного до компиляции. Во втором и третьем будет показаны различные способы задания матрицы динамического размера. В четвертом подведем итог.

1.1 Матрица статического размера

Объявление Напомню как происходит взаимодействие с одномерным массивом.

```
1 double arr[10];//Объявлен одномерный 10 элементный массив
2 //Каждый элемент имеет тип плавающая точка -- double.
3 //Во всех arr[0],...,arr[9] мусор.
4 arr[1] = 3;//Теперь есть значение у элемента arr[1].
5 //arr[1]~3.
6 arr[4] = arr[1] + 5;//Выполняем арифметические операции.
7 //arr[4]~8.
```

По аналогии с одномерным массивом можно объявить и двумерный массив, т.е. матрицу:

```
1 double arr2[6][5];//Матрица размера 6 на 5.
```

Размер очередной размерности указывается в квадратных скобках. В данном случае `arr2` – это двумерная матрицы размера 6 на 5 элементов. Как эти размерности интерпретировать с точки зрения математики зависит от задачи, т.е. можно считать что, например, это матрица из 6 строк и 5 столбцов, а можно наоборот. По аналогии можно задавать массивы и большей размерности:

```
1 double arr3[3][6][5];//3 матрицы размера 6 на 5.
```

С точки зрения языка Си правильная интерпретация повышения размерности (т.е. добавления числа в квадратных скобках) гласит, что, например, `arr3` – это массив из трех элементов, каждый из которых является двумерным массивом (матрицей) размера 6 на 5. Точнее даже так: `arr3` – это массив из трех элементов, каждый из которых является массивом из 6 элементов, каждый из которых является массивом из 5 элементов, каждый из которых имеет тип плавающая точка (`double`).

Элементы массива `arr2` задаются двумя индексами – неотрицательными целыми числами. Каждый из индексов меняется в соответствующем диапазоне, а именно – от 0 до размера минус 1. Тогда элементами массива `arr2` являются следующие элементы: `arr2[0][0]`, ..., `arr[0][4]`, `arr2[1][0]`, ..., `arr[1][4]`, ..., `arr2[5][0]`, ..., `arr[5][4]`. Отмечу, что после объявления матрицы во всех её элементах естественно мусор.

Вычисления Зная последнее далее можно выполнить вычисления над её элементами:

```
1 //По аналогии с одномерным случаем
2 //во всех arr2[0][0],...,arr2[5][4] мусор.
3 arr2[1][3] = 5;//Присвоили 5 элементу с индексом 1,3.
4 arr2[3][1] = 3+arr2[1][3];//Выполнили вычисление.
5 //arr2[3][1]~8.
```

При выходе индексов за пределы возможных значений возникнет ошибка.

```
1 //Ошибка, при выходе за границу какого-либо из индексов:
2 arr2[3][5] = 1;//Ошибка, выход за границу второго индекса
3 arr2[6][4] = 4;//Ошибка, выход за границу первого индекса
```

Снизу индекс ограничен 0, а сверху строго меньше размера.

Инициализация В одномерном случае было возможно выполнить инициализацию массива:

```
1 double arr[5]={1,5,-1,7,4};
```

По аналогии в двумерном (и соответственно в многомерном) случае

```
1 double arr3[3][2]={1,5,-1,2,7,4};
```

Инициализация элементов матрицы выполняется в ранее отмеченном порядке. Так, например, элемент `arr3[0][0]` ~1, `arr3[0][1]` ~5, `arr3[1][1]` ~2, а `arr3[2][0]` ~7. Если массив больше указанного списка, то хвостовые элементы не будут проинициализированы – в них будет мусор. Если список содержит большее количество чисел, то это вызовет ошибку компиляции.

При инициализации можно прервать инициализацию элементов текущей строки:

```
1 double arr3[3][2]={{1,5},{-1},{7,4}};
```

Для перехода на следующую строчку (т.е. в начало следующего массива) используется запятая. В данном случае, все элементы за исключением элемента с индексом 1,1 будут проинициализированы как и ранее. Отмеченный элемент будет содержать мусор.

Объект любого типа Естественно, что вместо типа `double` можно применить любой другой тип данных: `int`, а то и частные типы (структуры).

Связь с указателем Напомню какая есть связь массива с указателем:

```
1  double *a;//Объявили указатель на тип double.
2  a=arr;//Присвоили одномерный массив указателю, т.е.
3  //Напомню, что по определению *a ~ a[0] ~ arr[0].
4  //a указывает на 0-ой элемент массива, равносильно
5  a=&arr[0];
6  //Тогда, например, a[2] есть суть arr[2], т.е.
7  a[2] = 3;//a[2] ~ *(a+2) ~ *(arr+2) ~ arr[2] ~ 3.
8  //Более того, справедливо:
9  a=arr+3;//a указывает на третий элемент массива arr, т.е.
10 a[2]=7;//a[2] ~ *(a+2) ~ *(arr+3+2) ~ arr[5] ~ 7.
```

По аналогии с этим

```
a=&arr2[2][1];//a указывает на элемент arr2[2][3].
//Тогда
a[0]=2;//Элементу arr2[2][3] присвоено значение 2.
*(a+1)=3;//Элементу arr2[2][4] присвоено значение 3.
```

Но можно пойти и дальше:

```
a=arr2[1];//Указывает на начало первой строки
a=&arr2[1][0];//Равносильно этой строке.
```

Последние выражения вычисляются на этапе компиляции, т.е. нельзя написать:

```
arr2[1]=a;//Ошибка. Такого элемента на самом деле нет.
```

1.2 Массив массивов

Создание Зададимся целью создания двумерного (а также и многомерного) массива имеющий динамический размер, т.е. размер который задается при запуске программы, а не при её компиляции. По аналогии с выше описанным, нам нужно создать массив массивов. Если последние имеют тип `double*`, то тогда массив, элементы которого имеют тип `double*`, будет иметь тип `double**`. Напишем соответствующий код:

```
//Сначала считаем размер требуемой матрицы
int n, m;
scanf("%d%d", &n, &m);//Пусть n - строчки, m - столбцы.
//Будем считать, что размер считан успешно.
//Объявим переменную являющуюся массивом указателей:
double **arr4;
//Элементы arr4[0], ..., arr4[3], ... имеют тип double*.
//Выделим память под него:
arr4 = (double **)malloc( n * sizeof(double*) );
//Отмечу его размер - n и тип элементов -- double*.
```

Каждый из элементов созданного массива `arr4` имеет тип `double*`, и содержит мусор. т.е. для корректной работы им нужно присвоить массивы соответствующего размера.

```
int i;
for(i = 0; i<n; i++)
  arr4[i] = (double*)malloc( m*sizeof(double) );
```

Всё, матрица создана. Отмечу, что размер каждого из массивов соответствует количеству столбцов, т.е. равно m. Тип элементов массива выбран `double`.

Важно отметить о порядке освобождения памяти. Сначала нужно освободить память выделенную под каждую из строчек матрицы:

```
1 for(i = 0; i<n; i++)
2   free( arr4[i] );
```

а уже далее память выделенную под массив массивов:

```
1 free( arr4 );
```

В противоположном порядке ничего не получится, так как если сначала освободить память массива arr4, то после этого любое обращение к его элементам будет считаться ошибкой.

Вычисления над элементами Все вычисления соответствуют ранее рассмотренному массиву статического размера. Так,

```
1 //Присваивает элементу с индексом 3,2 значение 9:
2 arr4[3][2] = 9;
3 arr4[2][1] = arr4[3][2]-10;//Выполняется вычисление.
```

Последнее естественно справедливо в случае, если размер указанного массива при запуске программы будет превосходить рассмотренные индексы.

Взаимодействие с указателями также справедливо:

```
1 double *b;
2 b=arr4[7];//В b запишется указатель 7го массива.
3 b[3]=6;//Запишет 6 в 3ий элемент 7 строчки.
```

Замечу, что в обще говоря ничто не мешает для каждой строчки выделить массив своего размера. Такое может быть полезным, например, для эффективного хранения треугольной матрицы.

Многомерный случай Естественно, что сами строчки (в данном случае, элементы массива arr4) могут быть многомерными массивами, в частности, двумерными. Тогда объединяя получим трехмерный массив.

```
1 //Сначала считаем размер требуемой матрицы
2 int n, m, k;
3 //Пусть n - матрицы, m - строчки, k - столбцы.
4 scanf("%d%d%d", &n, &m, &k);
```

```
5  //Будем считать, что размер считан.
6  //Объявим переменную соответствующего типа:
7  double ***arr5;
8  //Элемент arr5[0], ..., arr5[3], ... имеют тип double**.
9  //Выделим память под него:
10 arr5 = (double ***)malloc( n * sizeof(double**) );
11 //Отмечу его размер - n и тип элементов -- double**.
```

Каждый из элементов созданного массива `arr5` имеет тип `double**`, т.е. необходимо выполнить аналогичные действия для двумерного массива.

```
1  int i;
2  for(i = 0; i<n; i++)
3  {
4     arr5[i] = (double**)malloc( m*sizeof(double*) );
5     int j;
6     for(j=0; j<m; j++)
7        arr5[i][j]=(double*)malloc( k*sizeof(double) );
8  }
```

Упражнение. Напишите код освобождающий память.

1.3 Через одномерный массив

В данном подходе выделяется одномерный массив по размеру соответствующий двумерному (многомерному).

```
1  //Сначала считаем размер требуемой матрицы
2  int n, m;
3  scanf("%d%d", &n, &m);//Пусть n - строчки, m - столбцы.
4  double *arr6 = (double*)malloc( n*m * sizeof( double ) );
```

При обращении к элементу выполняется вычисление соответствие индексов. Существуют два традиционных способа обхода элементов: по строкам и по столбцам.

По строкам В данном подходе элементы матрицы обходятся строка за строкой. Последнее означает, что соседние элементы в строке являются соседними в одномерном массиве. А также, последний элемент строки является соседним к первому элементу следующей строки. Формула будет такой: $i * m + j$, где i, j – это индекс элемента матрицы. Исходя из последнего:

```
1  arr6[2*m+1]=7;//Элементу с индексом 2,1 присвоено 7.
```

Работа с указателями следует из способа отображения индексов.

```
1 double *q;
2 q=arr6+5*m;//q указывает на 5ю строчку.
3 q[7]=3;//Элементу с индексом 5,7 присвоено 3.
```

По столбцам В полной аналогии с предыдущим, в данном подходе элементы матрицы обходятся столбец за столбцом. Формула будет: $i * m + j$, где i, j – это индекс элемента матрицы. Все остальное тоже по полной аналогии, но только для столбцов.

Освобождение памяти Память освобождается крайне просто.

```
1 free( arr6 );
```

1.4 Итог

Показано как выделять память под многомерные массивы. Тип естественно можно заменить на произвольный, например на структуры.

У каждого подхода есть свои преимущества. Например, в представлении матрицы как массив массивов операция перестановки двух строк очень быстрая, тогда как в представлении одномерного массива тяжелая.

В представлении массива массивов очень высокая нагрузка на систему выделения памяти, а в случае представления одномерным массивом такой проблемы нет.