top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

Урок № 7

Многомерные массивы

Содержание

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

1. Генератор случайных чисел

В прошлом уроке мы с вами познакомились с таким понятием как массив и научились заполнять его значениями. Все наши значения мы вписывали сами, то есть заранее знали какими они будут. Такой способ заполнения переменных и массивов ограничивает возможности программы. Невозможно создать нечто, обладающее искусственным интеллектом, если нет способа получать данные не зависящие от пользователя. Что же делать, если нам необходимы значения, выбранные случайным образом?

Использование функции rand

В языке C++ существует возможность генерировать случайное число. Для этой операции используется функция под названием rand(). Эта функция находиться в библиотечном файле — stdlib.h, следовательно для ее работы необходимо этот файл подключить с помощью директивы #include. На место вызова rand() в программе, подставится случайное число в диапазоне от 0 до 32767. Рассмотрим простой пример:

```
#include<iostream>
#include<stdlib.h> // в этом файле содержится
                   // функция rand
using namespace std;
int main()
{
```

```
    int a;

    // генерация случайного числа и запись его
    // в переменную a
    a = rand();
    cout << a << "\n";

    /* повторная генерация случайного числа и запись
       его впеременную a */
    a = rand();
    cout << a << "\n";
    return 0;
}
```

Если вы создали проект, набрали и запустили наш пример на выполнение, то обнаружили, что программа последовательно сгенерировала два так называемых случайных числа. Это могут быть числа на изображении ниже или пара любых других:

Рисунок 1

Запустите еще раз. И, снова — та же картина, пара снова и снова повторяется. Выходит, случайные числа — не случайны, кроме того они еще и повторяются от запуска к запуску. Несмотря на несправедливость этого утверждения — это так, и, вполне понятно, почему. Если вы подойдете к любому человеку и на улице и попросите назвать произвольное целое число, этот человек несомненно назовет именно случайное число. И то — не факт.

Возможно, прохожий посмотрит на вывеску или на часы и извлечет это число из увиденного. Компьютер, в отличии от живого существа, не способен на ассоциативное мышление, поэтому функция rand() не получает число из воздуха, а работает, используя в качестве начальной точки — точку определенную при написании алгоритма генератора случайного числа, то есть некое постоянное число. Другими словами, опираясь на эту точку, при разных вызовах программы эта функция генерирует одно и то же число, в чём мы уже успели убедиться. Из этого можно сделать вывод: Для того, чтобы rand() при разных вызовах программы выдавал разные числа необходимо изменить начальную точку генерации.

Использование функции srand

Местоположение функции — библиотека *stdlib.h*. Функция srand устанавливает начальную точку для генерации случайных чисел и обладает следующим синтаксисом:

```
void srand(unsigned int start)
```

Параметр start, который принимает функция, и есть — новая точка для генерации случайного числа. Давайте подумаем, какое же число вписать на место этого параметра. Целочисленный литерал или переменная, значение которой определено в момент написания программы — не подходят. Передав их на место отправной точки мы непременно изменим ее, но не сделаем динамической. И, числа будут генерироваться другие, но все еще одинаковые при каждом запуске.

Пример с литералом, в качестве отправной точки

```
#include<iostream>
// в этом файле содержатся функции rand и srand
#include<stdlib.h>

using namespace std;
int main()
{
    srand(5);
    int a;
    /* генерация случайного числа и запись его
       в переменную */
    a=rand();
    cout<<a<<"\n";
    return 0;
}
```

Пример с переменной, в качестве отправной точки

```
#include<iostream>
// в этом файле содержатся функции rand и srand
#include<stdlib.h>

using namespace std;
int main()
{
    int start=25;
    srand(start);
    int a;
    // генерация случайного числа и запись его
    // в переменную
    a=rand();
    cout<<a<<"\n";
    return 0;
}
```

Нам с вами необходима некая величина, которая меняется постоянно вне зависимости от каких-либо внешних факторов. Согласитесь, такой величиной является — время. И, именно, время мы используем в качестве отправной точки.

Использование функции time

Местоположение функции — библиотека *time.h*.

У функции time есть несколько предназначений и подробно разбирать мы ее сейчас не будем. Возьмем только то, что необходимо нам для работы. А, именно, если функцию time вызвать с параметром NULL, то на место своего вызова в программе, эта функция вернет количество милисекунд прошедших с 1 января 1970 года. Согласитесь, что эта величина каждый раз будет разной. Это, как раз то, что мы искали. «Соберем» полученную информацию в единое целое и получим:

```
srand(time(NULL));
```

Функция srand устанавливает в качестве стартовой точки число, представляющее собой количество, милисекунд прошедших с 1 января 1970 года. Попробуем:

```
#include<iostream>
/* в этом файле содержатся rand и srand */
#include<stdlib.h>
// в этом файле содержится функция time
#include<time.h>

using namespace std;
int main()
{
```

```
    srand(time(NULL));
    int a;

    // генерация случайного числа и запись его
    // в переменную
    a=rand();
    cout<<a<<"\n";
    return 0;

}
```

Набрав исправленный пример, вы, конечно, убедились, что теперь при разных запусках программы генерируются разные числа, но и этом возможности генератора не исчерпываются.

Установка диапазона для генератора

Числа, которые получаются путем вызова функции rand, находятся в диапазоне от 0 до 32767. Но, ведь нам не всегда требуется такой масштабный разброс данных. Что делать, если необходимо генерировать числа от 0 до 10 или от 0 до 100 и так далее?! На помощь, в таких случаях, приходит старое, доброе — деление по модулю.

Возьмем для примера произвольное число — 23. Согласитесь, что какое бы число вы не разделили на 23 по модулю, вы получите либо 0 (если остатка нет), либо остаток в диапазоне от 1 до 22. Этим свойством мы и воспользуемся, разделив сгенерированное случайное число по модулю:

```
int a=rand()%23;
```

На основании этого правила можно вывести формулу: ЧИСЛО В ДИАПАЗОНЕ ОТ НУЛЯ ДО X.

```
rand() % X
```

Но, диапазон не всегда начинается с нуля. Пусть нам необходим диапазон от 11 до 16. Все просто. Необходимо генерировать числа от 0 до 5 (разница между 16 и 11), а потом «сдвинуть» полученный результат на 11 единиц.

```
int a=rand()%5+11;
```

И, на основании уже модифицированного правила можно вывести формулу: ЧИСЛО В ДИАПАЗОНЕ ОТ Y ДО X.

```
rand() % (X-Y) + Y
```

Итак, мы с вами познакомились с генератором случайных чисел и теперь можем облегчить себе работу с массивами. Как? Следующий раздел урока расскажет об этом.

2. Использование генератора случайных чисел

Давайте рассмотрим пример использования генератора случайных чисел, а именно заполнение массива случайными числами:

```
#include<iostream>
// в этом файле содержатся функции rand() и srand()
#include<stdlib.h>
// в этом файле содержится функция time()
#include<time.h>
using namespace std;

int main()
{

    int array[10];
    srand(time(NULL));
    for (int i=0;i<10;i++)
    {
        // генерация случайного числа и запись его
        // в текущий элемент массива
        array[i]=rand()%100;
        // показ значения элемента на экран
        cout<<array[i]<<"\n";
    }
    return 0;
}
```

1. В приведенном выше примере на экран будет выведен массив из 10 элементов, заполненный случайными числами.

2. На каждой итерации цикла генерируется новое случайное число.
3. При каждом запуске программы массив будет заполнен по разному, благодаря строке srand(time(NULL)).
4. Числа располагающиеся, в массиве будут варьироваться в диапазоне от нуля до 100, так как результат генерации делится на 100 по модулю.

Как видите, генератор случайных чисел прост в обращении, и теперь у вас в руках есть оружие, которое позволит вам не только тестировать программы, не вводя данные с клавиатуры, но и создавать примитивный искусственный интеллект.

3. Двумерные массивы, как частный случай многомерных массивов

Мы с вами уже имеем понятие о том, что такое массивы, в прошлом уроке мы разобрали так называемый — одномерный массив. Одномерный массив — массив данных, где каждое значение обладает только одной характеристикой — порядковым номером (индексом). Именно по этому индексу мы и обращаемся к конкретному элементу.

Сегодня мы поговорим о многомерных массивах, т.е. о массивах, где каждый элемент описывается несколькими характеристиками. Примером значения многомерного массива может являться, что угодно:

1. Шахматная доска — каждая клетка имеет две размерности E2 (буква и цифра).
2. Оценка КВН — три размерности ЧЛЕН_ЖЮРИ, КОНКУРС, КОМАНДА.

Мы с вами остановимся на двумерном массиве, второе название которого матрица (так его обычно называют математики).

Двумерный массив. Объявление и расположение в памяти

Двумерный массив представляет собой совокупность строк и столбцов, на пересечении которых находится конкретное значение.

Объявить двумерный массив несложно, необходимо указать количество строк и столбцов.

При этом, здесь действуют все те же правила, что и при объявлении одномерного массива. Т.е. нельзя в качестве количества строк и столбцов указывать не константные и не целочисленные значения.

Общий синтаксис:

```
тип_данных имя_массива [число_строк]
[число_столбцов];
```

Пример:

```
const int row=3; // строки
const int col=4; // столбцы
int array[row][col]; // массив размером row
                     // на col(3x4)
```

Рисунок 2

Несмотря на то, что мы представляем массив в виде матрицы, на самом деле — любой двумерный массив располагается в памяти построчно: сначала нулевая строка, затем первая и так далее. Об этом следует помнить,

т.к. выход за пределы массива может повлечь за собой некорректную работу программы, при этом не выдав ошибки.

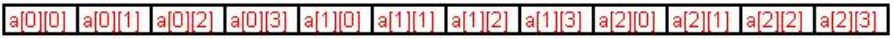

Рисунок 3

Инициализация

Инициализация двумерного массива аналогична инициализации одномерного:

1. Инициализация при создании

Каждая строка заключается в отдельные фигурные скобки:

```
int array[2][2]={{1,2},{7,8}};
```

Значения указываются подряд и построчно вписываются в массив:

```
int array[2][2]={7,8,10,3};
```

Если значение пропущено, оно будет инициализировано нулем:

```
int array[3][3]={{7,8},{10,3,5}};
```

2. Инициализация с помощью цикла

Откроем один секрет — двумерный массив можно рассматривать как совокупность, не просто строк, а одномерных массивов. То есть, один одномерный массив, мы заполняем простым циклом, перебирая конкретные

элементы, а при совокупности, нам необходимо перебирать еще и отдельные массивы.

> **Примечание:** *Обращение к конкретному элементу массива производится по номеру строки и номеру столбца, например — **mr[2][1]** — значение, лежащее на пересечении второй строки и первого столбца.*

Работа с двумерным массивом не намного сложнее, чем с одномерным — докажем это на практике.

```
#include<iostream>
// в этом файле содержатся rand и srand
#include<stdlib.h>
// в этом файле содержится функция time
#include<time.h>

using namespace std;
int main()
{
    const int row=3; // строки
    const int col=3; // столбцы
    int mr[row][col]; // массив размером row на col

    /* перебираем отдельные строки
      (одномерные массивы в совокупности) */
    for(int i=0; i<row; i++)
    {
        // перебираем отдельные элементы каждой
        // строки
        for(int j=0; j<col;j++)
        {
            /* инициализация элементов значениями
               в диапазоне от 0 до 100 */

            mr[i][j]=rand()%100;
```

```
            // показ значений на экран
            cout<<mr[i][j]<<" ";
        }
        /* переход на другую строку матрицы */
        cout<<"\n\n";
    }
    return 0;
}
```

4. Практический пример

Постановка задачи

Написать программу, которая в двумерном массиве находит максимальный элемент каждой строки.

Код реализации

```
// в этом файле содержатся rand и srand
#include<stdlib.h>
#include<time.h> // в этом файле содержится функция time
using namespace std;

int main()
{
    // задаем размерность массива
    const int m = 3;
    const int n = 2;
    int A[m][n]; // объявляем двумерный массив

    // заполнение массива случайными числами
    // и показ на экран
    // перебираем отдельные строки
    for(int i=0; i<m; i++)
    {
        /* перебираем отдельные элементы каждой строки */
        for(int j=0; j<n;j++)
        {
            // инициализация элементов значениями
            // в диапазоне от 0 до 100
            A[i][j]=rand()%100;
            // показ значений на экран
            cout<<A[i][j]<<" ";
        }
```

```
        // переход на другую строку матрицы
        cout<<"\n\n";
    }
    cout << "\n\n";
    // поиск в строках максимального элемента
    // перебираем отдельные строки
    for (int i=0; i<m; i++)
    {
        // предполагаем, что максимальный -
        // нулевой элемент строки
        int max = A[i][0];
        // поиск максимального элемента
        // в текущей строке

        // изменение индекса столбца
        // для текущей строки
        for (int j=0; j<n; j++)
        {
            if (A[i][j] > max)
                max = A[i][j];
        }
        cout << "Max element "<< i << " row=" <<
                max << endl;
    }
    return 0;
}
```

Обратите внимание!

1. На каждой итерации цикла в качестве максимума выбирается нулевой элемент текущей строки.
2. После анализа конкретной строки найденный максимум выводится на экран.

5. Понятие статического выделения памяти

Настало время немного упомянуть о работе с памятью. Пока мы всерьёз не задумывались о том, как происходит выделение памяти для той или иной объявленной переменной.

Пока мы используем только один вид выделения памяти: статическое выделение памяти.

Что это значит? Механизм статического выделения памяти вычисляет нужное количество байт для переменной или массива на этапе компиляции программы. Например:

```
// под переменную a будет выделено 4 байта
int a;

// под массив будет выделено 40 байт
int   arr[10];
```

Когда выполнение программы дойдет до места объявления переменной a, будет выделен блок памяти размером 4 байта. Тот же механизм работает и для массива ar.

Важно отметить, что при статическом выделении памяти невозможно увеличить или уменьшить размер массива ar на этапе выполнения программы. Отсюда можно сделать вывод, что размер массива будет фиксированным и равным той величине, которая была указана в момент компиляции.

Это значит, что статическое выделение памяти не подходит для задач, где размер массива должен меняться в процессе работы программы.

Как решить эту проблему мы покажем в ближайшем будущем.

6. Домашнее задание

1. Дан двухмерный массив размерностью 3×4. Необходимо найти количество элементов значение которых равно нулю.
2. Дана квадратная матрица порядка n (n строк, n столбцов). Найти наибольшее из значений элементов, расположенных в тёмно-синих частях матриц.
3. Все массивы в данном домашнем задании заполняются случайным образом.

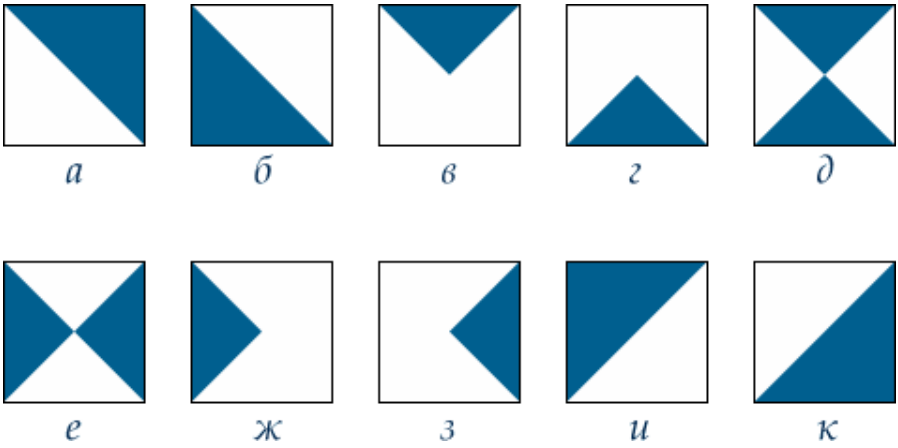

Рисунок 4