ШАГ компьютерная академия

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Урок №7

Програмиране на език C

Съдържание

1. Генератор на случайни числа

В предишния урок се запознахме с понятието масив и се научихме да запълваме неговите значения. Но всичките значение сме ги запълвали с ами, тоест сме знаели предварително какви ще бъдат. Такъв метод на запълване на променливите и масивите ограничава възможностите на програмата. Не е възможно да се създаде нещо притежаващо изкуствен интелект, ако няма метод за получаване на данни независещи от потребителя. Какво да правим ако ни е необходимо значение избрано на случаен принцип?

Използване на функцията rand.

В език С съществува възможност за генериране на случайно число. За тази операция се използва функцията **rand()**. Тази функция се намира в библиотечен файл – stblib.h, следователно за нейната работа е необходимо този файл да бъде включен с директивата #include. На мястото на **rand()** в програмата ще се представи случайно число от диапазона 0 до 32767. Да разгледаме елементарен пример:

```
#include<iostream>
#include<stdlib.h>// в този файл се съдържа функцията rand

using namespace std;
void main()
{
        int a;
   //генерация на случайно число и да се запише в променливата a
   a=rand();
   cout<<a<<"\n";

 /* повторна генерация на случайно число и записване в променливата a */
   a=rand();
   cout<<a<<"\n";
}
```

Ако сте създали проект и сте написали и пуснали нашия пример за изпълнение, ще забележите, че програмата последователно е генерирала две така наречени случайни числа:

Пуснете още един път и отново – същата картина. Излиза, че случайните числа не са случайни, освен това те и се повтарят на всяко пускане. Не зависимо че несправедливо това утвърждение – то е така, и е прекрасно ясно защо. Ако отидете към случаен човек на улицата и го помолите да каже произволно цяло число, този човек несъмнено ще назове именно случайно число. И пак, надали. Вероятно минаващият човек ще погледне на някоя табела или часовник и ще извлече това число от видяното. Компютъра за разлика от живите същества, не е способен на асоциативно мислене, затова функцията **rand()** не получава числото от въздуха, а работи използвайки в качеството на начална точка – точка

определена при написването на алгоритъма генератор на случайни числа, тоест някое постоянно число. С други думи, изхождайки от тази точка, при различни команди на програмата тази функция ще генерира едно и също число, в което вече успяхме да се убедим. От това може да се направи извод: За да може rand() при различни повиквания от програмата да изкарва различни числа е необходимо да се промени началната точка на генерация.

Използване на функцията srand.

Местоположение на функцията — библиотека stdlib.h.

Функция **srand** установява начална точка за генерация на случайни числа и обладава следния синтаксис:

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

Параметъра start, който приема функцията е нова точка за генерация на случайно число. Нека да помислим, какво число да напишем на мястото на този параметър. Целочислен литерал или променлива, значението на която да е определено в момента на написването на програмата – не подхождат. Предавайки ги към мястото на отправната точка ние непременно ще я променим, но няма да я направим динамична. Числата които ще се генерират след това, ще са различни, но пак еднакви при всяко стартиране.

Пример с *литерал* в качеството на отправна точка:

```
#include<iostream>
//в този файл се съдържат функциите rand и srand
#include<stdlib.h>

using namespace std;
void main()
{
        srand(5);
        int a;
        //генерация на случайно число да се запише в променливата
        a=rand();
        cout<<a<<"\n";
}
```

Пример с променлива в качеството на отправна точка:

```
include<iostream>
//в този файл се съдържат функциите rand и srand
#include<stdlib.h>

using namespace std;
void main()
{
        int start=25;
        srand(start);
        int a;
        // генерация на случайно число да се запише в променливата
        a=rand();
        cout<<a<<"\n";
}
```

На нас с вас ни е необходимо нещо, което се променя постоянно и е извън зависимостта на каквито и да е външни фактори. Такава величина е времето. И именно времето ние ще използваме в качеството на отправна точка.

Използване на функцията time.

Местоположение на функцията — библиотека time.h.

Функцията time има няколко предназначения и подробно

Няма да я разглеждаме. Ще вземем само това, което ни е необходимо за работа. А именно ако функцията time я извикваме с параметъра NULL, то на мястото на извикване в програмата, тази функция ще върне количеството милисекунди изминали от 1 януари 1970 година. Тази величина всеки път ще бъде различна. А това е точно това, което търсихме. Ако съберем получената информация в едно цяло ще се получи:

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

Функцията srand установява в качеството на стартова точка число, представящо количеството милисекунди изминали от 1 януари 1970 година. Да пробваме:

```
#include<iostream>
#include<stdlib.h>// в този файл се съдържа rand и srand
#include<time.h>// в този файл се съдържа функция time

using namespace std;
void main()
{
        srand(time(NULL));
        int a;
        // генерация на случайно число да се запише в променливата
        a=rand();
        cout<<a<<"\n";
}
```

Набирайки поправения пример, вие се убеждавате, че сега при всяко пускане програмата генерира различни числа, но и с това не се изчерпват възможностите на генератора.

Установяване на диапазона за генератора.
Числата, които се получават чрез функцията rand са в диапазона от 0 до 32767. Но на нас не винаги ни трябва такъв голям диапазон. Какво да правим, ако ни е необходимо да генерираме число от 0 до 10 или от 0 до 100 и така нататък?! На помощ в такива случаи идва старото добро модулно деление
Да вземем за пример произволно число – 23. Каквото и число да не разделим модулно на 23 ще получим или 0 или остатък в диапазона от 1 до 22. От това свойство ние ще се възползваме, разделяйки генерираното случайно число :

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

На основанието на това правило може да се изкара формула:

```
Число с диапазон от нула  ДО X:

       rand() % X
```

Но, диапазона не винаги започва от нула. Нека да ни е необходим диапазон от 11 до 16. Много е лесно. Необходимо е да се генерират числа от 0 до 5 (разликата между 16 и 11), а след това да преместим получения резултат с 11 единици.

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

И на основание на вече модифицираното правило можем да изкараме формула:

```
Число с диапазон от Y ДО X:

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

И така ние се запознахме с генератора на случайни числа и сега можем да се отпуснем с работата с масиви. Как? Следващия раздел от урока ще ви разкаже за това.

2.Използване на генератора за случайни числа

Нека да разгледаме пример за използване на генератор за случайни числа, а именно запълването на масиви със случайни числа:

```
#include<iostream>
// в този файл се съдържа функция rand() и srand()
#include<stdlib.h>
#include<time.h> // в този файл се съдържа функциятime()
using namespace std;

void main()
{
        srand(time(NULL));
        int array[10];
        for (int i=0;i<10;i++)
        {
// генерация на случайно число да се запише в текущия елемент на масива
                array[i]=rand()%100;

                // показване значението на елемента на екрана
                cout<<array[i]<<"\n";
        }
}
```

1. В изкарания горе пример на екрана ще бъде изкаран масив от 10 елемента запълнен със случайни числа.
2. На всяка итерация от цикъла се генерира ново случайно число.
3. При всяко пускане на програмата масива ще бъде запълнен по различен начин благодарение на srand(time(NULL));
4. Числата разполагащи се в масива ще варират

В диапазона от нула до 100, след като резултата на генерацията се дели модулно на 100.

Както виждате, генератора на случайни числа е елементарен за обръщение и сега имате оръжие в ръцете си, което ще ви позволи не само да тествате програмите без да въвеждате данни с клавиатурата, но и да създавате примитивен изкуствен интелект.

3. Двумерните масиви като частен случай на многомерните

С вас вече имаме понятие от масиви, в предишния урок сме разглеждали така наречения едномерен масив. Едномерен масив – масив от данни, където всяко значение има само една характеристика – с порядков номер (индекс). Именно чрез този индекс ние се обръщаме към конкретен елемент.

Днес ще поговорим за многомерните масиви, тоест за масивите, където всеки елемент се описва от няколко характеристики. Пример за значението на многомерен масив може да бъде каквото и да е

1. Шахматна дъска – всяка клетка има две размерности E2 (буква и цифра)

С вас ще се спрем на двумерните масиви, познати още като матрица.

Двумерен масив.

Обявяване и разполагане в паметта.

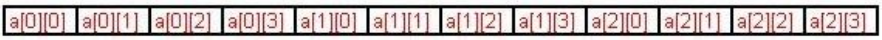
Двумерния масив представя съвкупност от редове и стълбове, в пресечната точка на които се намира конкретно значение. Да се обяви двумерен масив не е сложно, необходимо е да се укаже количеството редове и стълбове.

Тук също действат същите правила, които и при обявяването на едномерен масив. Тоест, не трябва в качеството на количество редове и стълбове да се указват константни и не целочислени значения.

```
Общ синтаксис:
        Тип на данни име на масива [число редове][число колони ];

пример:
        const int row=3; // редове
        const int col=4; // колони
        int array[row][col]; // масив с размер row на col(3x4)
```

Независимо от това, че представяме масива във вид на матрица, всъщност – всеки двумерен масив се разполага в паметта по редово: първо е нулев ред, след това първи и така нататък . За това трябва да се помни, защото излизането извън пределите на масива може да доведе до некоректна работа на програмата, без да издаде грешка.



Инициализация.

Инициализация на двумерен масив е аналогична на инициализацията на едномерния:

1. Инициализация при създаване.

```
Всеки ред се заключва в отделни фигурни скоби:

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

значенията се указват подред и по редово се записват в масива:

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

Ако значението е пропуснато, то ще бъде инициализирано с нула:

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

2. Инициализация с помощта на цикъл.

Ще разкрием една тайна – двумерния масив може да се разглежда като съвкупност от не просто редове, а едномерни масиви. Тоест, един едномерен масив ние запълваме с елементарен цикъл, избирайки конкретни елементи, а при съвкупности на нас ни е необходимо да изберем още и отделни масиви.

Забележка: Обръщението към конкретен елемент на масива протича по номера на реда и номера на колоната — mr[2][1] — значение, лежащо на пресечените втори ред и първа колона.

Работата с двумерните масиви не е по-сложна отколкото с едномерните, ще го докажем на практика.

```
#include<iostream>
#include<stdlib.h> //в този файл се съдържа rand и srand
#include<time.h> // в този файл се съдържа функция time
using namespace std;
void main()
{
        const int row=3; // редове
        const int col=3; // колони
        int mr[row][col]; // масив с размер row на col

/* избираме отделни редове (едномерни масиви и съвкупности) */
        for(int i=0; i<row; i++)
        {
                // избираме отделни елементи от всеки ред
                for(int j=0; j<col;j++)
                {
/* инициализация на елементите със значения в диапазона от 0 до 100 */
                        mr[i][j]=rand()%100;

                        // показване на значенията на екрана
                        cout<<mr[i][j]<<" ";
                }

                // преминаване на друг ред от матрицата
                cout<<"\n\n";
        }
}
```

4. Практически пример

Условие на задачата.

Да се напише програма, която в двумерен масив да намира максималния елемент на всеки ред.

Код за реализация:

```
include<stdlib.h> // в този файл се съдържа  rand и srand
#include<time.h> // в този файл се съдържа функция time

using namespace std;

void main()
{
      // задаваме размерност на масива
      const int m = 3;
      const int n = 2;
      int A[m][n]; // обявяваме двумерен масив

      // запълване на масива със случайни числа и извеждане на екрана

      // избираме отделни редове
      for(int i=0; i<m; i++)
      {
           // избираме отделни елементи от всеки ред
           for(int j=0; j<n;j++)
           {
 // инициализация на елементите със значение в диапазона от 0 до 100
                A[i][j]=rand()%100;

                // показване на значенията на екрана
                cout<<A[i][j]<<" ";
            }

            // преход на друг ред от матрицата
            cout<<"\n\n";
        }
        cout << "\n\n";

     // търсене в редовете на максималния елемент

        // избираме отделни редове
        for (int i=0; i<m; i++){

     // предполагаме, че максималния е нулев елемент от реда
```

```
                int max = A[i][0];

               // търсене на максималния елемент в реда

               // промяна на индекса на колоната за текущия ред
                for (int j=0; j<n; j++)
                {
                        if (A[i][j] > max)
                                max = A[i][j];
                }
                cout << "максимален елемент " << i
                         << "- = " << max << endl;
        }
}
```

Обърнете внимание!

1. На всяка итерация на цикъла в качеството на максимум се измира нулев елемент на текущия ред.
2. След анализа на конкретния ред намерения максимум се изкарва на екрана.

5. Домашна работа

1. Даден е двумерен масив с размери 3X4. Необходимо е да се намери количеството елементи, значението на които да е равно на нула.

2. Дадена е квадратна матрица от типа n (n редове, n колони). Да се намери най-голямото от значенията на елементите разположени в тъмно сините части на матрицата.

Всичките масиви в даденото домашно се запълват по случаен образ.