MH1402 Algorithms & Computing II

Lecture 6 Arrays, Random Numbers

Wu Hongjun

Overview

- **Arrays**
  - **Declaration**
  - **Access array elements**
  - **Multidimensional arrays**
  - **Passing arrays to function**
- **Random Numbers**
  - **C programming: rand( ), srand( )**
  - **C++11 random engine (recommended)**

Array

- **Array is one of the basic data structures**
  - **It consists of a sequence of elements of the same type;**
- **Arrays in C++ are similar to that in MATLAB**
  - **But arrays in C++ should be declared before being used.**
  - **Different syntax**
- **Why do we use arrays in programming?**
  - **Convenient for accessing data when the data are in array;**
  - **Otherwise, how many variables do we need to declare if there are 1000 integers to process? And how to write the program?**

Array

- **Array declaration in C++**

  type array_name[array_size];

```
Example: int  bar[10];
         // declare an array
         // 10 elements in the array
         // the data type of elements is int
```

  - ***type*** **is the data type of the elements in an array**
    - **All the elements in an array are of the same type**
  - ***array_name*** **is an identifier**
  - ***array_size*** **specifies the number of elements in an array**

Array

- **array_size should be a constant. It should not be a variable (as specified in the C++ standard)**

Example:

```
int x = 5;
int  bar[x];
// it is wrong!
// but some compilers accept it.
// our GCC compiler in Code::Blocks accept it. But don't use it.
```

Array

- **Array Declaration in C++ (cont.)**
  - **When we declare `int bar[4];`**
    the array elements are not initialized, the array elements' values are random
  - **Array can be declared and initialized in this way:**

```
int bar[4] = {30,40,50,60};
```

  - **Alternatively, array can be declared and initialized as follows:**

```
int bar[ ] = {30,40,50,60};
```

    - **Note that the array size can be omitted here in bar[ ]**
    - **The size of this array is 4**
      (Here, the array size is determined by the compiler, according to the number of elements being provided.)
    - **Those four elements in the array are 30, 40, 50, 60**

Array

- **Array declaration in C++ (cont.)**
  - **Array can be declared and initialized as follows:**

```
int bar[4] = {30, 40};
```

    - **The array size is 4, the elements are initialized as 30, 40, 0, 0**
      The last two elements are initialized to 0 automatically
      (even though not specified in the code)
  - **Wrong declaration and initialization:**

```
int bar[4] = {30, 40, 50, 60, 70};
//compilation error,
//too many initializer for array bar
```

Array

- **Array elements**

  Example: For the array with four elements:

```
double bar[4];
its first element is          bar[0];
its second element is         bar[1];
its third element is          bar[2];
its fourth (last) element is  bar[3];
```

  0, 1, 2, 3 above are called index values.

- **In C/C++, the array elements are numbered using the zero-based indexing:**
  - **The i-th element with index i-1 (the first element always with index 0)**
  - **Advantage: convenient for the computer to allocate memory address for array elements as the memory address starts from 0.**

Array

- **Array elements (cont.)**
  - **We can assign a value to an array element after declaration**

```
int bar[4];
bar[1] = 12345;
```

  - **Then we can use the value of an element:**

```
int y;
y = bar[1] + 6;
```

Array

- **Array elements (cont.)**
  - **If an array element has not been initialized, its value is likely random**
  - **If accessing an element out of array bounds: (risky for computing and security)**

```
int bar[4];
bar[4] = 34567;  //  wrong! Since the last element of bar is bar[3]
                 //  but the compiler does not check this error for you;
                 //  and the computer does not check this error for you;
                 //  The program may crash during run-time
int y;
y = bar[7] + 1;  //  wrong! bar[7] is out of the bound of array.
                 //  likely a random value is assigned to y.
                 //  the compiler and computer do not check this error for you
```

Array

- **Array elements (cont.)**
  - **Array cannot be assigned:**

```
int bar[4] = {6, 7, 8, 9};
int foo[4];
foo = bar;        // error: cannot assign one array to another in C++
```

  - **We need to assign the values of the elements one by one:**

```
for (int i = 0; i < 4; i++)
    foo[i] = bar[i];
```

Array: Passing an array to a function

Example:

```
#include <iostream>
using namespace std;

void increase(int bar[ ], int length)
{
    for (int i = 0; i < length; i++)
        bar[i]++;
}
```

Continued on the next page …

```
void increase(int [ ], int);

int main()
{
    int foo[4] = {32, 45, 67, 89};

    increase(foo, 4);

    cout <<"The array after modification is ";
    for (int i = 0; i < 4; i++)
        cout << foo[i] << " ";
    cout << endl;

    return 0;
}
```

```
The array after modification is 33 46 68 90
```

Array: Passing array to a function

- **Note that passing array to a function is passing by reference**
  - **The elements of the array can be modified in that function; and the modified values retain outside the function**
- **When passing an array to a function, the function does not know the size of that array**
  - **If that function needs to know the size of the array, we should pass the value of size as another parameter to the function**

Array: Multidimensional array

- **Example: A two dimensional array is declared as:**

```
int bar[2][4];   // two dimensional array
```

This array has 2*4=8 elements of the type int.

- **Example: A two dimensional array is declared and initialized as:**

```
int foo[2][4] = { {2, 3, 4, 5}, {7, 8, 9, 0} };
```

// Those eight elements are :

// foo[0][0] = 2; foo[0][1] = 3; foo[0][2] = 4; foo[0][3] = 5;

// foo[1][0] = 7; foo[1][1] = 8; foo[1][2] = 9; foo[1][3] = 0;

Array: Multidimensional array

- **Example: A two dimensional array may be declared and initialized as:**

```
int qux[2][4] = {2, 3, 4, 5, 7, 8, 9, 0};
```

// Those eight elements are :

// qux[0][0] = 2; qux[0][1] = 3; qux[0][2] = 4; qux[0][3] = 5;

// qux[1][0] = 7; qux[1][1] = 8; qux[1][2] = 9; qux[1][3] = 0;

Array: Multidimensional Array

- **Example: declare three dimensional array as:**

```
int bar[2][4][5];     // 2*4*5 elements
double foo[3][4][5];
```

- **Example: declare four dimensional array as:**

```
int bar[3][2][5][7]; // 3*2*5*7 elements
```

Array: Multidimensional Array

- **Passing multidimensional array to function**
  - **It is also passing by reference: the values of the array elements being modified within a function can get retained outside the function**
  - **In function definition and declaration, *the size of the first dimension is left blank (as for one dimensional array), the sizes of other dimensions must be specified.***
    - **Examples: in the next four pages, one example for 2d array, another for 3d array**

Example of 2D array

```
#include <iostream>
using namespace std;

void increase(int bar[][4])
{
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 4; j++)
        {
            if (i == 0) bar[i][j]++;
            if (i == 1) bar[i][j] += 3;
        }
    }
}
```

Continued on the next page …

```
void increase(int [ ][4]);

int main()
{
    int foo[2][4] = {2, 3, 4, 5, 6, 7, 8, 9};

    increase(foo);

    cout <<"The array after modification is ";
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 4; j++)
            cout << foo[i][j] << " ";
    return 0;
}
```

```
The array after modification is 3 4 5 6 9 10 11 12
```

Example of 3D array

```
#include <iostream>
using namespace std;

void increase(int bar[][2][3])
{
    for (int i = 0; i < 2; i++)
    for (int j = 0; j < 2; j++)
    for (int k = 0; k < 3; k++)
    {
        if (i == 0)
            bar[i][j][k] = 1;
    }
}
```

Continued on the next page …

```
void increase(int [ ][2][3]);
int main()
{
    int foo[2][2][3] = {2, 3, 4,  5,  6,  7,
                        8, 9, 10, 11, 12, 13};
    increase(foo);
    cout <<"The array after modification is ";
    for (int i = 0; i < 2; i++)
    for (int j = 0; j < 2; j++)
    for (int k = 0; k < 3; k++)
        cout << foo[i][j][k] << " ";
    cout << endl;
    return 0;
}
```

```
The array after modification is 1 1 1 1 1 1 8 9 10 11 12 13
```

Random Numbers

Random Numbers

- **Random numbers are useful for statistical analysis, simulation …**
- **The rand( ) function (`#include <cstdlib> , it is a C function`) generates a random integer in the range [0, RAND_MAX]**
- **RAND_MAX is a constant**
  - **Its value is 32,767 in Code::Blocks using the default Mingw-GCC compiler**
  - **But its value would be different on different compiler (the value is at least 32,767)**
- **rand() function is not standardized, so the rand() function may be different for different compilers**

Random Numbers

- **rand( ) function is not a truly random function**
  - **The name is misleading**
  - **There are horrible flaws of using rand( ) in security applications**
- **The random numbers generated from rand( ) are pseudo-random, weak**
  - **With period 2^{32} (the random numbers start to repeat after generating 2^{32} random numbers).**
  - **Avoid using the rand( ) function if your simulation requires a large number of random numbers**
  - **Anyway, rand( ) function provides a convenient way to generate random numbers suitable for some computer simulations (but risky)**

Random Numbers

- **To generate a random integer in the range [a, b] use**

```
a + rand()%(b-a+1);
```

  - **Example: Simulate the roll of a die (ranging from 1 to 6):**

```
1 + rand()%6
```

  - **Example: Generate a random even integer between 2 and 10.**

```
2*(1 + rand()%5)
```

Random Numbers

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    for (int i = 0; i < 10; i++)
    {
        int rand_num = rand();
        cout << rand_num << endl;
    }
    return 0;
}
```

Run this program for many times, the same sequence of random numbers would get generated …. **Why?**

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
```

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
```

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
```

Random Numbers

- **To explain the results in the previous slides, we need to know that the rand( ) function generates a sequence of random numbers from a seed (an integer) through a deterministic algorithm**
  - **The value of seed set to 1 by default when we execute a program**

Random Numbers

- **To avoid generating the same sequence of random numbers for every execution of the program, we need to set the seed to different values, called the seeding of rand( )**
  - **We can manually set seed value for each program**

    Example: `srand(237);`
  - **We may set the seed as the current time**

    Example: `srand(time(NULL));` **or** `srand(time(0));`

    // #include <ctime> in order to call time();

    // time(NULL) or time(0) gives the current time in seconds

    // (the number of seconds since 00:00 hours, Jan 1, 1970 UTC)

Random Numbers

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
    srand(time(0));
    for (int i = 0; i < 10; i++)
    {
        int rand_num = rand();
        cout << rand_num << endl;
    }
    return 0;
}
```

Run this program several times, different sequences of random numbers are generated ….

```
32078
3823
27100
8984
27920
29458
9851
10336
18693
17419
```

```
32228
6730
29652
1784
10870
13480
21952
6753
24562
29520
```

```
32392
19862
5356
25302
2311
13208
13734
24229
30942
17029
```

Random Numbers

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
    for (int i = 0; i < 10; i++)
    {
        srand(time(0));
        cout << rand() << endl;
    }
    return 0;
}
```

Run this program, most of the time, 10 identical numbers are generated.

Why?

(it is a common error in the past year exam)

Generating Random Numbers in C++11

Random Numbers in C++

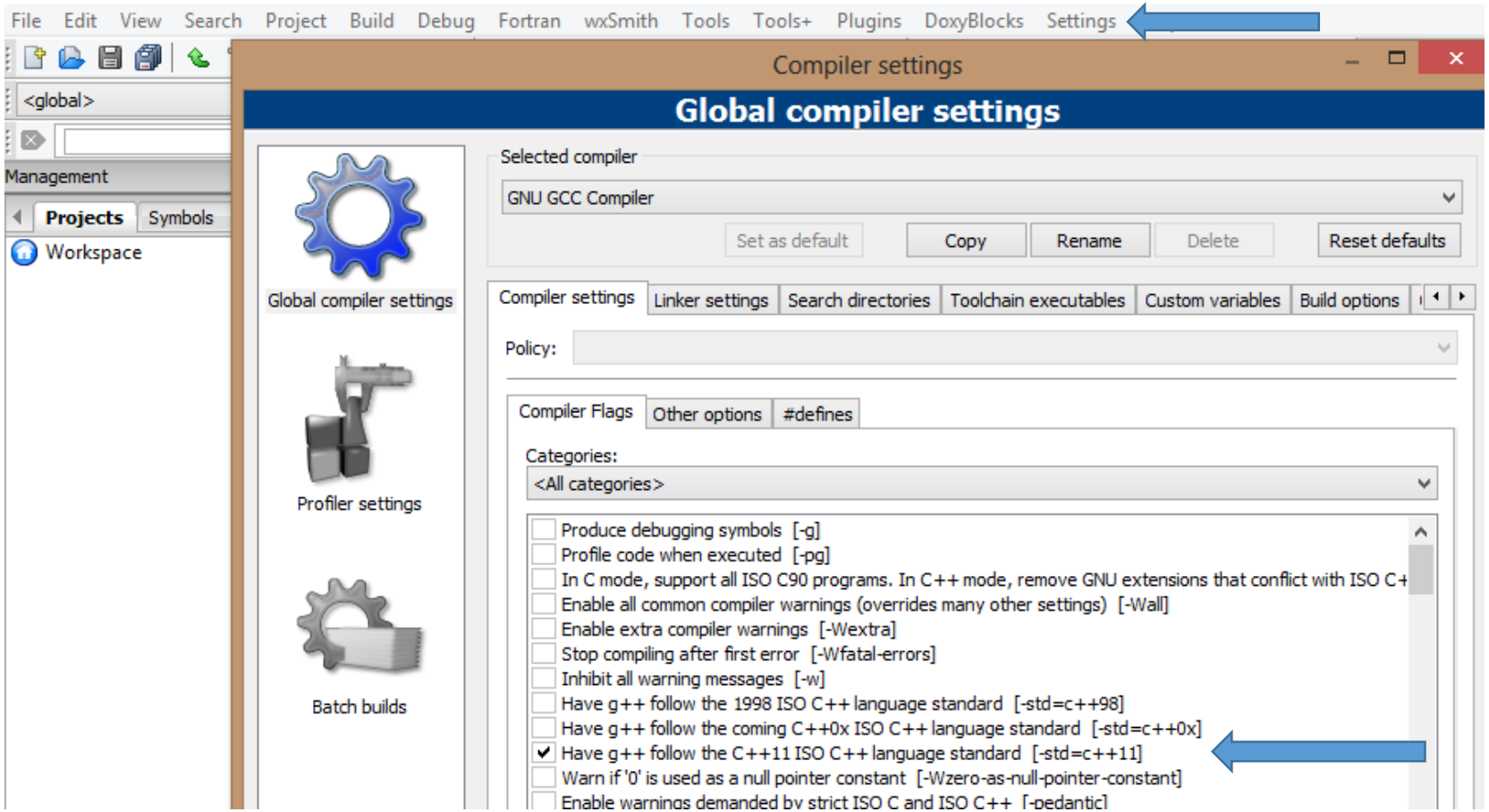
- **In C++11 standard (approved in 2011), there is a better way to generate random numbers (with many features)**
- **We recommend the use of C++ mt19937 random engine**
  - **It is a version of Mersenne Twister pseudo-random number generator**
  - **Period: 2^{19937} -1 (too large, suitable for all the simulations)**
  - **Slow: about 0.2 millseconds to generator a number on the latest desktop computer**
    - **We get this speed when compiling using the default Code::Blocks compiler, selecting "Release" compilation option**

Random Numbers in C++

- **Do not use C and C++ random number generators for security applications (they are not random enough)**
- **There are several C++ random number generator algorithms**
  - **Some are not good**
  - **Do not use the C++ default random engine default_random_engine**
  - **In the Code::Blocks default compiler, the default_random_engine turns out to be the weak random engine minstd_rand0 (not better than rand())**

- **Currently, on Windows Computer, to compile the C++ random number generator using Code::Blocks, you need to change the compiler setting so that Code::Blocks would accept C++11.**

  Settings → Compiler → tick "Have g++ follow the C++11 ISO C++ language standard"



Random Numbers in C++

- **The program on the next page generates random numbers in the range [0, 999] using the C++ random engine**
  - **It generates random integers uniformly in the range [0, 999]**
  - **Fixed seeding is used**

```
#include <iostream>
#include <random>      // must be included

using namespace std;

int main( )
{
   mt19937 rand_generator;    // declare a random_engine rand_generator
   rand_generator.seed(135);   // fixed seeding 135 is used here.

   // to generate uniformly distributed integers in range [0,999]. This range can be changed.
   uniform_int_distribution<int>  rand_distribution(0, 999);

   for (int i = 0; i < 10; i++)
   {
      int rand_num = rand_distribution(rand_generator);
      cout << rand_num << endl;
   }

   return 0;
}
```

Random Numbers in C++

- **The program on the next page generates random numbers in the range [1.2, 7.8] using the C++ random engine**
  - **It generates random real numbers uniformly the range [1.2, 7.8]**
  - **Fixed seeding is used**

```
#include <iostream>
#include <random>

using namespace std;

int main( )
{
   mt19937 rand_generator;

   rand_generator.seed(36); //fixed seeding 36 is used here.

   // to generate uniformly distributed double in range [1.2, 7.8]. This range can be changed.
   uniform_real_distribution<double>  rand_distribution(1.2, 7.8);

   for (int i = 0; i < 10; i++)
   {
      double rand_num = rand_distribution(rand_generator);
      cout << rand_num << endl;
   }

   return 0;
}
```

Random Numbers in C++

- **Random seeding: if we want to use random seeding for the C++ random engine, we need to provide a random seed.**
  - **C++ random_device is supposed to generate a truly random number, so good for seeding.**
  - **C++ random_device is fully supported in Microsoft Visual Studio C++ 2012, so it can be used to provide seed (but such code may not be compiled using other compilers, or may not run on other computers, so not portable)**
  - **But C++ random_device is not supported in the current version of Windows Code::Blocks**
  - **So we have to use some portable random seeding for the C++ random engine**
    - **In this course, we simply use the portable *time(0)* function**

C++ Random seeding (supported in Code::Blocks on Windows Computer)

```
#include <iostream>
#include <random>
#include <ctime>      // time() function

using namespace std;

int main()
{
   mt19937 rand_generator;

   rand_generator.seed(time(0));  //random seeding using the current time, recommended!

   uniform_int_distribution<int>  rand_distribution(0, 999);

   for (int i = 0; i < 10; i++)
   {
      int rand_num = rand_distribution(rand_generator);
      cout << rand_num << endl;
   }

   return 0;
}
```

C++ Random seeding (NOT supported in Code::Blocks on Windows Computer)

```
#include <iostream>
#include <random>

using namespace std;

int main()
{
   random_device rdev;  // seeding more random, but the C++ random_device is not that portable
   mt19937 rand_generator;

   rand_generator.seed(rdev()); //random seeding

   uniform_int_distribution<int>  rand_distribution(0, 999);

   for (int i = 0; i < 10; i++)
   {
      int rand_num = rand_distribution(rand_generator);
      cout << rand_num << endl;
   }

   return 0;
}
```