02 - String and Vector Class

Post-Lab Exercises

Exercise 1

Write a programme that uses an integer vector and member functions from the `vector` class to store random numbers in to the vector. The user should first choose the vector size and range of random numbers to be filled (e.g. numbers between 5 and 10) and then a histogram is displayed on the screen showing the frequency of each of the numbers stored.

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

int main()
{
    //seeding random number generator
    srand(time(NULL));

    //totalNumbers - number of values, lower/upper - min and max values
    int totalNumbers, lowerLimit, upperLimit;

    //vectors for values and their frequencies
    vector<int> randomNumbers, frequency;

    //Get input from user
    cout << "How many numbers would you like to generate?" << endl;
    cin >> totalNumbers;

    cout << "Lower limit for random numbers?" << endl;
    cin >> lowerLimit;

    cout << "Upper limit for random numbers?" << endl;
    cin >> upperLimit;

    //one index per random value in the frequency vector
    frequency.resize(upperLimit - lowerLimit + 1);

    //entering 10 random numbers into vector
    for (int i = 0; i < totalNumbers; i++)
        randomNumbers.push_back(lowerLimit + rand()%upperLimit);

    //echoing random numbers generated
    cout << "\nGenerated " << totalNumbers << " random numbers "
    << "between " << lowerLimit << " and " << upperLimit << endl;
```

```
    for (int i = 0; i < randomNumbers.size(); i++)
    {
        cout << randomNumbers[i] << "\t";
        if ((i + 1) % 10 == 0)      //newline every 10 values
            cout << endl;
    }
    cout << endl;

    //THE MAGIC
    //first element of frequency - lowerLimit's frequency
    //second element of frequency - (lowerLimit + 1)'s frequency...
    //nth element of frequency - (lowerLimit + n)'s frequency
    for (int i = 0; i < randomNumbers.size(); i++)
        for (int j = 0; j < frequency.size(); j++)
            if (randomNumbers[i] == lowerLimit + j)
                frequency[j]++;

    //echoing frequency distribution
    cout << "Frequency Distribution" << endl;
    for (int i = 0; i < frequency.size(); i++)
        cout << i + 1 << "\t" << frequency[i] << endl;

    cout << "\nEND OF PROGRAMME" << endl;
    return 0;
}//end main
```

```
How many numbers would you like to generate?
30
Lower limit for random numbers?
1
Upper limit for random numbers?
5

Generated 30 random numbers between 1 and 5
2   4     3     5     5     5     5     4     5     4
1   5     4     5     3     4     2     2     1     3
3   3     2     3     5     1     2     4     3     4

Frequency Distribution
1   3
2   5
3   7
4   7
5   8

END OF PROGRAMME
Program ended with exit code: 0
```

Exercise 2

Use the `string` class to declare a `string` array and store the names of all the course in SE Electrical this semester. After the course names are stored, use the `sort()` function from the <algorithm> header to sort the string array and display the results. You must describe the method for using the sort( ) function and also the algorithm that works when it used to sort the string variables.

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

void printArray(string array[], int arraySize);

int main()
{
    string courses[] = {"MT-226 Multivariable Calculus",
        "MT-272 Linear Algebra and Analytical Geometry",
        "EL-240 Electronic Devices and Circuits",
        "HS-214 Academic Writing",
        "HS-205 Islamic Studies",
        "EE-264 Data Structures and Algorithms",
        "NED-101 Avoiding Security Guards with God Complexes"
    };
    int SIZE = 7;   //7 courses

    cout << "Before sorting, array of SE courses is: " << endl;
    printArray(courses, SIZE);

    sort(courses, courses + SIZE);
    cout << "\nAfter sorting, array of SE courses is: " << endl;
    printArray(courses, SIZE);

    cout << "END OF PROGRAMME" << endl;
    return 0;
}


void printArray(string array[], int arraySize)
{
    for (int i = 0; i < arraySize; i++)
        cout << array[i] << endl;
    cout << endl;
}
```

```
Before sorting, array of SE courses is:
MT-226 Multivariable Calculus
MT-272 Linear Algebra and Analytical Geometry
EL-240 Electronic Devices and Circuits
HS-214 Academic Writing
HS-205 Islamic Studies
EE-264 Data Structures and Algorithms
NED-101 Avoiding Security Guards with God Complexes


After sorting, array of SE courses is:
EE-264 Data Structures and Algorithms
EL-240 Electronic Devices and Circuits
HS-205 Islamic Studies
HS-214 Academic Writing
MT-226 Multivariable Calculus
MT-272 Linear Algebra and Analytical Geometry
NED-101 Avoiding Security Guards with God Complexes

END OF PROGRAMME
Program ended with exit code: 0
```

Explanation - Using `sort()`

`sort()` is a function in the C++ `algorithm` library, which can be used to sort pointer-based data structures such as arrays or vectors.

The `sort` function takes three arguments
1. an iterator or pointer to the beginning of the data structure
2. an iterator or pointer to the end of the data structure.
3. (optional) a method to define how the data structure should be sorted.

in that order.

The first argument defines where the first element of the data structure is stored in memory. This element will be sorted by the `sort()` function. However, the second argument (which defines the last item in the data structure) will NOT be sorted by the `sort()` function. This is why the code on the previous page specifies `SIZE` and not `SIZE - 1` as the last memory location of the array, as the latter would exclude the last value of the array from being sorted.

When no argument is provided for the third parameter, the `sort()` function arranges the data structure's elements in ascending order by default. This can be changed by including a function (or rather a pointer to a function) which specifies an alternative sorting method such as descending order or in order of a custom key for a user-defined data type.

Explanation - sort( ) Implementation

There is no standard implementation of the sort( ) function, with different standard libraries defining the algorithm in different ways. For instance, the GNU Standard C++ library's implementation of sort uses a hybrid algorithm based on quicksort and insertion sort (collectively called introsort) followed by heapsort (Gcc.gnu.org, 2009).