```cpp
/*
Title: Memo P7
Author: Jaco du Toit
Date: 9 April 2022
*/

#include "libP07.h"
#include <iostream>

using namespace std;
using namespace HistogramSpace;

int main(int argc, char** argv)
{
    /* This program will accomplish the following

    1. It will initialise a fixed 1D array of integers to zero values.
    2. It will create a set of random numbers and populate the 1D array with random numbers given the lower and upper bound limits
    3. It will create a horizontal histogram of the array of values.
    4. It will create a vertical histogram of the array of values.
    5. In both frequency diagrams, values within the three bands are drawn using specific characters
       a) The lower 33% values are represented by '!' character.
       b) The middle 33% values are represented by '@' characters.
       c) The upper 33% values are represented by '#' characters.
    6. The program will run until the user quits

    This code makes use of the setw-function.  This was not required for students to use, but is used to show how the screen
    can be organised to make things look a bit better.  For more information about setw see https://www.cplusplus.com/reference/iomanip/setw/

    */

    if(argc!=2)
    {
        cerr << "Incorrect command line arguments." << endl;
        exit(ERR_ARGC);
    }

    int intCount = GetInt(argv[1]);
    if(intCount>MAX_COUNT)
    {
        cerr << "Please ensure the number of random number are not more than:" << MAX_COUNT << endl;
        exit(ERR_RANGE);
    }

    arrType arrNums = InitArray(intCount);                          //Declare an array called arrNums of type arrType.

    bool blnContinue = true;
    char chOption = '\0';
    //Seed the random number generator to the current time value.  This is required to help generate
    //pseudorandom numbers.
    srand(time(0));

    //This week we implement the main loop using a while loop instead of a do-while loop.
    //The effect is the same for this type of implementation since blnContinue starts off being true.
    while(blnContinue)
    {
        //Get some of the menu ready for output
        system("cls");
        cout << "1. Set values of the array to zero values." << endl
             << "2. Fill the values with a set of random numbers" << endl
             << "3. Output a horizontal histogram for the values in the array." << endl
             << "4. Output a vertical histogram for the values in the array." << endl
             << "5. Quit this application" << endl;
        //For debugging purposes the array is also displayed.  NOT necessary for student submissions
        OutputArray(arrNums,intCount);
        //Handle the input from the user and run the various functions.
        cin >> chOption;
        switch(chOption)
        {
        case '1':
            ZeroArray(arrNums,intCount);
            break;
        case '2':
            RandomArray(arrNums,intCount);
            break;
```

```
76            case '3':
77                OutputHorizontal(arrNums,intCount);
78                break;
79            case '4':
80                OutputVertical(arrNums,intCount);
81                break;
82            case '5':
83                blnContinue = false;
84                break;
85            default:
86                cerr << "Please enter a valid menu option" << endl;
87                system("pause");
88            }
89        }
90
91        return SUCCESS;
92    }
93
```

```
1   //The following two lines and the line at the bottom of the header file is used to ensure that
2   //functions are not declared multiple times when functions are used in different files.
3   #ifndef LIBP04_H_INCLUDED
4   #define LIBP04_H_INCLUDED
5
6   #include<cstdlib>               //c-standard library required for srand, rand, system and exit
7   #include<ctime>                 //Library allowing us to make use of the time function
8   #include<iomanip>               //Library that stores the implementation of setw
9   #include<iostream>              //Library used by cin, cout and cerr
10
11  //We declare the const variables and typedef in the header file so that we can use in the code
    in main file as well as
12  //implementation file.
13
14  using namespace std;
15
16  namespace HistogramSpace
17  {
18      typedef int* arrType;                   //Using typedef to create a new type to help
    declare fixed size array
19
20      const char RANGE_LOWER = '!';           //Characters for bottom third values histogram bars
21      const char RANGE_MIDDLE = '@';          //Characters for middle third values histogram bars
22      const char RANGE_UPPER = '#';           //Characters for top third values histogram bars
23
24      const int ERR_ARGC = -1;
25      const int ERR_CONV = -2;
26      const int ERR_RANGE = -3;
27
28      const int MAX_COUNT = 25;
29      const int MAX_RAND = 25;
30
31      const int SPACING = 2;                  //Used with setw to help spacing some of the output
32      const int SUCCESS = 0;                  //Return code for the main function
33
34      int GetRandom(int intLower, int intUpper);  //Value returning function that generates a
    random number between intLower and intUpper
35      void OutputHorizontal(arrType arrNums, int intCount);     //Output horizontal histogram
36      void ZeroArray(arrType arrNums, int intCount);            //Put zero values in all array
    items
37      void RandomArray(arrType arrNums, int intCount);          //Add random numbers to the
    arrNums fixed size array
38      void OutputVertical(arrType arrNums, int intCount);       //Output vertical historigram
39      void OutputArray(arrType arrNums, int intCount);          //Output the array
40      int GetInt(string strNum);
41      arrType InitArray(int intCount);
42  }
43
44  #endif // LIBP04_H_INCLUDED
45
```

```cpp
#include "libP07.h"

namespace HistogramSpace
{
    //Function returns a random whole number between intLower and intUpper
    int GetRandom(int intLower, int intUpper)
    {
        int intRange = intUpper - intLower + 1;      //Calculate the range of potential numbers, counting from zero
        return rand()%intRange + intLower;           //Use the rand() function to get a random number.  Use modulus to enforce the range
                                                     //Shift the random number into the lower and upper bound range.
    }

    arrType InitArray(int intCount)
    {
        arrType arrNums;
        arrNums = new int[intCount];
        for(int i=0;i<intCount;i++)
            arrNums[i] = 0;

        return arrNums;
    }

    int GetInt(string strNum)
    {
        int intNum = 0;
        stringstream ss {strNum};
        ss >> intNum;
        if(ss.fail())
        {
            cerr << "Could not convert string to int" << endl;
            exit(ERR_CONV);
        }
        return intNum;
    }

    void OutputArray(arrType arrNums, int intCount)
    {
        cout << "Array:";
        //For each of the array values, output the value, seperated by a SPACE character.
        for(int i=0;i<intCount;i++)
        {
            cout << arrNums[i] << " ";
        }
        cout << endl;
    }

    void ZeroArray(arrType arrNums, int intCount)
    {
        //Give each item in the array a value of ZERO.  This could have been implemented during the declaration of the array.
        for(int n=0;n<intCount;n++)
        {
            arrNums[n] = 0;
        }
    }

    //This is a local function, not used anywhere else, but helps to enforce the user to type integer characters.
    //Because GetInt() is used only in this file, the prototype does not have to be declared in the header file.
    int GetInt()
    {
        int intNum = 0;
        //Read in a character from std input stream and try to convert it to int.
        cin >> intNum;
        //If converstion to int fails then try again.
        while(cin.fail())
        {
            cin.clear();            //Clear the fail flag
            string strJunk;         //Try to get rid of any characters up to the first space that are still stuck on the input stream.
            cin >> strJunk;
            cin >> intNum;          //Prompt to read in the integer again.
        }
        return intNum;
    }

    void RandomArray(arrType arrNums, int intCount)
    {
        int intLower = 0;
        int intUpper = 0;
```

```
78          //Get the upper and lower bound numbers from the users
79          cout << "Lower bound number (>=0):";
80          intLower = GetInt();
81          //Make sure the number is 0 or greater than zero.
82          while(intLower<0)
83          {
84              cerr << "Please make sure the lower bound number is 0 or greater than zero" << endl;
85              intLower = GetInt();
86          }
87          cout << "Upper bound number:";
88          intUpper = GetInt();
89          //Make sure the number is greater than the lower bound number.
90          while(intUpper <= intLower || intUpper > MAX_RAND)
91          {
92              cerr << "Please make sure the upper bound number is greater than the lower bound
    number and less than " << MAX_RAND << endl;
93              intUpper = GetInt();
94          }
95
96          //Get the random numbers and add them to the array
97          for(int n=0;n<intCount;n++)
98          {
99              arrNums[n] = GetRandom(intLower, intUpper);
100         }
101     }
102
103     //Get the maximum value in the array.  This is also a local function and is not used by the
    main function.
104     int GetMax(arrType arrNums, int intCount)
105     {
106         int intMax = 0;                 //Assume the highest number is zero.
107         for(int n=0;n<intCount;n++)   //Search through each array element.
108         {
109             if(arrNums[n]>intMax)       //If the array element is greater than the current
    maximum value
110                 intMax = arrNums[n];    //then make the current element our new maximum value.
111         }
112         return intMax;                  //Return the answer
113     }
114
115     void OutputHorizontal(arrType arrNums, int intCount)
116     {
117
118         int intMax = GetMax(arrNums, intCount);               //Stores the highest number in
    the array.  Required to calcuate the
119                                                 //three ranges of values
120         /*
121         If our range is not nicely devidable by three, then one or two of the ranges must be
122         one value greater than the other
123         We modify the high boundary value to help with this shift.
124         We use the modulo 3 operation to determine how well the range of numbers divide by
    three.  If it divides very poorly
125         then we modify the middle range to include one extra value.
126         Examples:
127         If we can divide the range equally with three then it will automtically have even ranges
    such as.  Example if our maximum
128         value is three then we have the following situation.
129         3: Upper range
130         2: Middle range
131         1: Lowest range
132         If however our maximum value is 4, then if we divide the range the we get a remainder of 1
133         The solution wil automatically add one to the top range.
134         4: Upper range
135         3: Upper range
136         2: Middle range
137         1: Lowest range
138         If however our maximum value is 5 then if we divide the range by three we will get a
    remainder of 2
139         Both the top and middle range will get one extra value.  This is achieved by
    incrementing the
140         step value after the lower boundary has been applied.
141         5: Upper range
142         4: Upper range
143         3: Middle range
144         2: Middle range
145         1: Lowest range
146         */
147         //Calculate the number of values that will divide the range into three equal parts
148         int intStep = intMax / 3;                   //Divide the max number by three, but stick
    with a whole number answer.
149         int intLBoundary = intStep;                 //Set the boundary for the lowest third
    equal to the third range.
150
151         if(intMax%3==2)
```

```
152             intStep++;
153         int intHBoundary = intLBoundary + intStep;
154 
155         //For debugging purposes the values are output.
156         cout << "Maximum value:" << intMax << endl;
157         cout << "Step value:" << intStep << endl;
158 
159         //Iterate through each item in the array.
160         for(int r=0;r<intCount;r++)
161         {
162             //Calculate the num of normalised items
163             int intNumItems = arrNums[r];
164             //The values of the array is output for debugging purposes and gives the bars a
    nice label.
165             cout << setw(SPACING);
166             cout << arrNums[r] << ":";
167 
168             //Output the number of characters equal to the normalised value
169             for(int c=0;c<intNumItems;c++)
170             {
171                 cout << setw(SPACING);
172                 //Determine which character to output
173                 if(intNumItems <= intLBoundary)
174                     cout << RANGE_LOWER;
175                 else if(intNumItems>intLBoundary && intNumItems<=intHBoundary)
176                     cout << RANGE_MIDDLE;
177                 else
178                     cout << RANGE_UPPER;
179             }
180             //Add an end of line character after all the characters have been generated.
181             cout << endl;
182         }
183         //Draw the X-axis
184         cout << " ";
185         for(int n=0;n<=intMax;n++)
186         {
187             cout << setw(SPACING);
188             cout << '=';
189         }
190         cout << endl;
191         cout << " ";
192         for(int n=0;n<=intMax;n++)
193         {
194             cout << setw(SPACING);
195             cout << n;
196         }
197         cout << endl;
198         //Pause the input (The first option is not operating system friendly as it may not work
    on all operating systems)
199         system("pause");
200     }
201 
202     void OutputVertical(arrType arrNums, int intCount)
203     {
204         //Most of the initial steps are the same as in the Horizontal function.
205         int intMax = GetMax(arrNums,intCount);                          //Store the max value
206         int intStep = intMax / 3;                                 //Work out range for the lowest range
207         int intLBoundary = intStep;
208         //Calculate whether we need to increase the middle range with one extra value to
    compensate
209         //ranges that do not divide equally into 3
210         if(intMax%3==2)
211             intStep++;
212         int intHBoundary = intLBoundary + intStep;
213 
214         //Start counting from the biggest value in the array down to smallest
215         //because those are the amount of rows that we are going to have to potentially output
216         //so that we can fit the tallest histogram bar on the screen.
217         for(int r=intMax;r>0;r--)
218         {
219             //Start with the top row
220             cout << setw(SPACING);
221             cout << r << "|";
222             //Go through each value in the array.
223             for(int c=0;c<intCount;c++)
224             {
225                 //Get the value of the array element.
226                 int intNormItem = arrNums[c];
227                 cout << setw(SPACING);
228 
229                 //To help visualise the solution consider the following hypothetical histogram
230                 /*
231                 4  #
232                 3  #     @
```

```
                2  #  !  @
                1  #  !  @
                   ==========
                   4  2  3  0

                Remember that cout outputs row by row, starting at the top row.
                The values 4,3,2,1 on the Y-axis depicts the bigest value to the smallest value
                The values 4,2,3,0 on the X-axis depicts the four items in the array.
                Imagine we start at row-4 and work downwards to row-1 then we take the case of row-4
                When we iterate thorugh the array items we evaluate each item.  Only items whose value is
                greater than or equal to the row value should get a bar.
                The first item in the array has a value of 4, that is why we output the corresponding character.
                The second item in the array has a value of 2.  Two is not greater or equal to 4 that is why we only output
                a ' '-character.

                Lets use the last row as an example.
                The last row is row-1.  When we look at each array item's value then the first three array items are greater
                than or equal to 1, except the last item, which is a 0, that is why each one gets their corresponding character,
                except the last one, which gets a ' '-character.
                */

                //If the value of the current element is less than the counter then just output a space.
                if(arrNums[c]<r)
                {
                    cout << " ";
                }
                //else output the character representation of the bar
                else
                {
                    if(intNormItem <= intLBoundary)
                        cout << RANGE_LOWER;
                    else if(intNormItem>intLBoundary && intNormItem<=intHBoundary)
                        cout << RANGE_MIDDLE;
                    else
                        cout << RANGE_UPPER;
                }
            }
            cout << endl;
        }
        //Last but not least we output a set of labels for each histogram column.
        cout << "   ";
        for(int n=0;n<intCount;n++)
        {
            cout << setw(SPACING);
            cout << '=';
        }
        cout << endl;
        cout << "   ";
        for(int n=0;n<intCount;n++)
        {
            cout << setw(SPACING);
            cout << arrNums[n];
        }
        cout << endl;
        system("pause");
    }
}
```