CMPSC 122 Lab 2 Report (revised Mar 30, 2014)

Code

//Programmer: Yizhou Wang

//Section: 2

//Lab: 2

//Date: Jan 31, 2014

//Description: A small program to display histogram based on a array data

#include <iostream>

#include <iomanip> //To enable setw

#include <stdlib.h> //To enable rand()

using namespace std;

const int MAX\_DATASIZE = 500; //Maximum number of input data

const int MAX\_CATEGORIES = 100; //maximum number of categories

void displayHistogram (int numofCategories, int numofElements[],

double lowerBound[], double upperBound[])

//PRE: numofCategories is initialized, 0 < numofCategories <= MAX\_CATEGORIES,

// numofElements[0..(numofCategories-1)] is initialized,

// lowerBound[0..(numofCategories-1)] is initialized,

// upperBound[0..(numofCategories-1)] is initialized;

// for i from 0 to numofCategories-1, the (i+1)th interval has lowerBound[i] as lower bound

// and upperBound[i] as upper bound; they form parallel arrays.

//POST: The function plots a histogram. The histogram has numofCategories number of

// categories. The histogram is displayed in a way that each interval is listed in a

// separate line. In the (i+1)th interval, there are numofElements[i]

// number of "\*" listed right after the interval range, each of which represents

// an Element in the input data.

{

cout << fixed; //Ensure the output numbers have two

cout.precision(2); // decimal digits.

cout << "HISTOGRAM:" << "\n"; //Display the title of the histogram

//DISPLAY OUTPUT

//The following outer for loop print the range (lowerBound to upperBound) for each

// category.

for (int i = 0; i < numofCategories; i++)

{

cout << setw(6) << lowerBound[i] << " to ";

cout << setw(6) << upperBound[i] << ":";

// The inner loop prints "\*"s representing elements in every category. And the

// string of "\*"s are followed by the braced numbers of elements.

for (int j = 0; j < numofElements[i]; j++)

{

cout << "\*";

}

cout << "(" << numofElements[i] << ")" << "\n"; //The number of elements in the category

};

cout << "\n\n"; //White spaces between every run

};

void generateHistogram (double data[], double lowest,

double highest, int numofCategories,

int logicalsize)

//PRE: 0 < logicalsize < MAX\_DATASIZE

// data[0..(size-1)] is initialized

// lowest <= highest

// 0 < numofCategories < MAX\_CATEGORIES

//POST: A histogram of data[0..logicalsize-1] is generated. The histogram has numofCategories

// categories, with lowest as the lower bound, and highest as the higher bound. For the

// interface of the histogram, each row begins with the bounds for the category, followed

// by visualization of the values in data[0..(size-1)]. Each element in the input data

// is represented as a single "\*" in the histogram.

{

//DATA DICTIONARY

double sizeofInterval; //the size of intervals

double lowerBound[MAX\_CATEGORIES]; //the array of all the lower

// bounds of categories

double upperBound[MAX\_CATEGORIES]; //the array of all the upper

// bounds of categories

int numofElements[numofCategories]; //the array of the number of elements

// falling into each category

//for i from 0 to numofCategories-1, the (i+1)th interval has lowerBound[i] as lower bound

// and upperBound[i] as upper bound; they form parallel arrays.

//INITIALIZATION & ASSIGNMENT

sizeofInterval = (highest - lowest) / numofCategories; //Determine the size of intervals

//PROCCESSING OF DATA

//This loop establishes the two arrays: 1) the array of lower bounds, and 2)the upper

// bound array. It determines the lower and upper bounds for every category and form the

// parallel arrays.

for (int i = 0; i < numofCategories; i++)

{

lowerBound[i] = lowest + sizeofInterval \* i;

upperBound[i] = lowerBound[i] + sizeofInterval;

numofElements[i] = 0; //Initialize the number of elements in

// each category to be 0

}

// The outer loop inspects every element in the input data.

for (int i = 0; i < logicalsize; i ++)

{

//The inner loop counts the number of elements falling into the range of a corresponding

// category.

for (int j = 0; j < numofCategories; j ++)

{

//If the value of the element is greater than the lower bound and smaller than the

// upper bound of a category, it belongs to the category

if (data[i] >= lowerBound[j] && data[i] < upperBound[j]) //Check the value of the

// data

{

numofElements[j] ++; //If it suffices the range,

// count once

}

}

}

//VISUALIZATION

//Call the displayHistogram function to display the histogram

displayHistogram (numofCategories, numofElements, lowerBound, upperBound);

};

int main ()

{

//INPUT

//DATA DICTIONARAY

double test1[11] =

{ 0, 2, 3.8, 5, 9, 16, 16.2, 17, 18, 19, 19.5}; //11 hardcoded numbers for test1

double test2[30]; //30 numbers for test2

// The loop generates 30 pseudo-random integer numbers in the range [0, 101)

for (int i = 0; i<30; i++)

{

test2[i] = rand() % 101;

}

//PROCESS & OUTPUT

//TEST1

//Call the function to generate a histogram with 5 categories, with the range from 0 to 20,

//using data test1, which contains totally 11 elements.

generateHistogram(test1, 0, 20, 5, 11);

//TEST2

//Call the function to generate a histogram with 10 categories, with the range from 0 to 100,

//using data test2, which contains totally 30 elements.

generateHistogram(test2, 0, 100, 10, 30);

//TEST3

//Call the function to generate a histogram with 5 categories, with the range from 0 to 100,

//using data test2, which contains totally 30 elements.

generateHistogram(test2, 0, 100, 5, 30);

return 0;

}

Sample Runs

HISTOGRAM:

0.00 to 4.00:\*\*\*(3)

4.00 to 8.00:\*(1)

8.00 to 12.00:\*(1)

12.00 to 16.00:(0)

16.00 to 20.00:\*\*\*\*\*\*(6)

HISTOGRAM:

0.00 to 10.00:(0)

10.00 to 20.00:(0)

20.00 to 30.00:\*\*\*(3)

30.00 to 40.00:\*\*(2)

40.00 to 50.00:\*\*\*(3)

50.00 to 60.00:\*\*(2)

60.00 to 70.00:\*\*\*\*\*\*\*\*\*\*(10)

70.00 to 80.00:\*\*(2)

80.00 to 90.00:\*\*\*\*\*(5)

90.00 to 100.00:\*\*\*(3)

HISTOGRAM:

0.00 to 20.00:(0)

20.00 to 40.00:\*\*\*\*\*(5)

40.00 to 60.00:\*\*\*\*\*(5)

60.00 to 80.00:\*\*\*\*\*\*\*\*\*\*\*\*(12)

80.00 to 100.00:\*\*\*\*\*\*\*\*(8)

Discussion

It is very important to abide with programming guidelines. As good programming manners can enhance the understandability of code, bad manners cause troubles in debugging and maintaining of the code for both the programmer and others.