Design Doc
Small Group Project 01 CSC 340
Group 02

NOTE: *The main function provides a menu interface to try all the three questions of the assignment with different values.*

Question 1:

Generating a histogram of randomly generated floating-point numbers according to a normal distribution with a mean and standard deviation specified by the user.

Variables:

- **(double) mean**: The mean of the normal distribution.
- **(double) standardDeviation**: The standard deviation of the normal distribution.
- **(int) numSamples**: Number of samples to generate for the histogram.
- **(int) numBins**: Number of bins for the histogram.
- **vector<pair<int, int>> points**: A vector of pairs that will hold the bin value (as floating-point number) and the frequency of the random number.

Algorithm:

1) **Preparation of bins:**
    - A vector of pairs is declared to hold the bin center and the frequency of the number.

   1.1) Calculating bin centers:
    - Bin centers are calculated using the user-specified mean, standard deviation, and the number of bins.
    - The calculation for the starting and ending bin centers depends on whether the number of bins is even or odd.
        - If numBins is even: startingPoint = -1 * (numBins/2) + 1
        - If numBins is odd: startingPoint = -1 * ((numBins-1)/2)
        - endingPoint = startingPoint + numBins - 1
    - For each bin, the center is determined by adding multiples of the standardDeviation to the mean.
    - These centers are then rounded off to integers using the round function.
    - The first value of each pair in points is set to this calculated bin center, and the second value (representing the frequency for that bin) is initialized to zero.

   1.2) The vector points are populated with these rounded bin centers.
   Example: **Points [{1,0},{2,0},{3,0} …….]**

2) Random Number Generation:
- Using the normal distribution random number generation engine of c++
- Random numbers are generated following a normal distribution with the user-specified mean and standard deviation.

- These generated numbers are then rounded off to integers using the round function.

3) Fill the histogram bins:

- For each rounded number, find its corresponding bin in the points vector.
- If the bin is found, increment the frequency (second value in the pair) of that bin.
- If the number doesn't match any bin center, do nothing.

4) Print the histogram:

- We use a self created makeHistogram function to print the histogram in an upright manner. (Explained later)

5) Time Complexity:

- Preparing bins for histogram: O(n); n = number of bins
- Generating random numbers and filling bins: O(n); n = number of samples

Question 2:

Generate a histogram of randomly generated floating-point numbers according to a uniform distribution with a user-specified minimum and maximum.

Variables:

- **(double) a, b**: Range of numbers for the uniform distribution.
- **(int) samples**: Number of samples to generate for the histogram.
- **(int) bins**: Number of bins for the histogram.
- **vector<pair<double, int>> points**: A vector of pairs that will hold the bin value (as floating-point number) and the frequency of the random number.

Algorithm:

1) **Preparation of bins:**

1.1) Calculating bin centers:

- Find bin width (center of the bin) using the formula (b-a)/(bins-1).
- For each bin, the center is calculated by adding multiples of the binCenter to the minimum value a.
- We use the user defined roundToTwo function to round the binCenters to floating points of 1 decimal place.
- The first value of each pair in points is set to this rounded bin center, and the second value is initialized to zero for storing the frequency of random numbers.

1.2) The vector points are populated with these values.

Example: **Points [{1,0},{2,0},{3,0} ……..]**

2) **Random Number Generation:**
    - Using the uniform distribution random number generation engine of c++
    - Random numbers are generated following a uniform distribution between a and b.
    - These generated numbers are then rounded to one decimal point using the user defined roundToTwo function.

3) **Fill the histogram bins:**
    - For each rounded number, find its corresponding bin in the points vector.
    - If the bin is found, increment the frequency of that bin.
    - If the number doesn't match any bin center, do nothing.

4) **Display the histogram:**
    - We use a self created makeHistogram function to print the histogram in an upright manner. (Explained later)

5) **Time Complexity:**

- Preparing bins for histogram: O(n); n = number of bins
- Generating random numbers and filling bins: O(n); n = number of samples

Function: makeHistogram

It is a user defined function which will take in a vector of pairs as input and print a histogram upright.

Input:

A vector of pairs where each pair consists of a bin value and the frequency of that bin.

Working:

1) **Find Maximum Frequency:**
   Iterate through the points vector to find the highest frequency (maxBarLength).

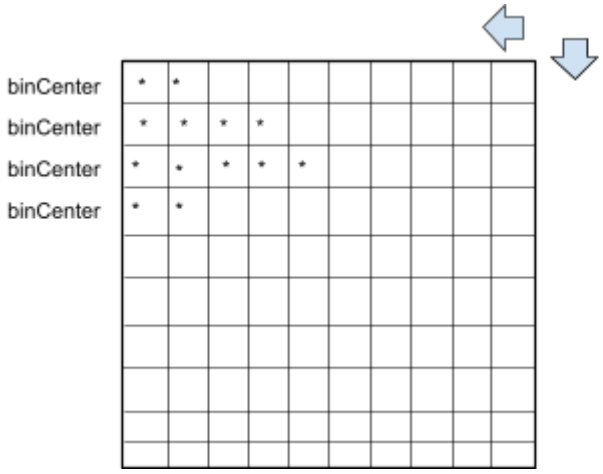
2) **Scale-down (by 5): As 20,000 is a lot of numbers**
    - Calculate unit length for the bars of the histogram.
    - This is achieved by dividing the maximum bar length by 5.

3) **Scale-down each Bin Frequencies:**
   For each bin in points, adjust its frequency by dividing it by the unit length.

4) **Printing it upright:**
   IDEA!!: Instead of starting from top left and going right, we start from top right and go down and back.

- Start from the maximum frequency.
- For each level from unitLength down to 0:
    - For each bin in points:
        - If the adjusted frequency of the bin is greater than or equal to the current level, print a '*'.
        - Otherwise, print a space.
    - Move to the next line.
- After printing all levels of the histogram bars, print the bin centers.

5)



Question 3:

Randomly assign students who haven't found a group to an existing group or create a new group, ensuring they belong to the same section.

1) classOfStudents:

- This is a vector of tuples. Each tuple has a student's name, their section number, and their group number.
- ***A group number of '0' indicates the student is not part of any group.***
- ***The teacher can add, remove, and modify student information using this vector***

2) Classify Students:

- Use the classifyStudents function to create an initial categorization of students.
- students from classOfStudents get organized into a structure of sections and groups.
- Return: objectGroups, which is organized as:
    - Sections: Each element represents a different section.
    - Groups: Within each section, each element is a group.
    - Students: Each group contains a vector of student tuples.
- objectGroups is **vector<vector<vector<tuple<string, int, int>>>>**

- ***The first group (group '0') within each section contains students not yet assigned to any group.***

3) Randomly Assign Students to Groups:

- ***Using the function groupingStudents***
- For each section, process students in group '0'. The unassigned students.
- For each section:
    - Calculate the number of unassigned students using the size of the first group of the section. Assign it as x.
    - Identify the groups with available space and record these groups along with their free spaces by iterating over the groups ***1 to size() -1*** of the particular section.
    - Also save a variable of total free space.
    - Compare the number of unassigned students (x) to the total free spaces in existing groups (freeSpaces):
        - **If freeSpaces >= x:**
            - While there are unassigned students:
                - Randomly select an unassigned student.
                - Randomly pick a group with available space.
                - Add the selected student to the picked group.
                - Remove the student from the list of unassigned students.
                - Update the available space for the chosen group.
                - If the chosen group is now full, remove it from the freeGroups list.
        - **else:**
            - Create new groups from the unassigned students until the number of unassigned students (x) is less than the available free spaces in the existing groups. Each new group will contain a number of students up to the specified group size ‘groupSize’.
            - After forming the necessary new groups, assign the remaining unassigned students to groups that have available space.

4) Display the Initial & Final Groups:

- Use the **printClass function** to display the groups and sections in a structured manner.
- Initial and final print will let you see how students have been assigned.
- In the final print group 0 of each section will be empty, which means no unassigned students after our algorithm is completed.

5) Time Complexity:

- Initial Grouping of Students: O(n); n is the number of students;
- Final Grouping in to groups: O(n*m); n is numberOfSections and m is numberOfGroups