Assignment 2 - Random Distributions

Due Sep 20 by 11:59pm **Points** 56 **Submitting** a file upload **File Types** cpp and txt
Available Sep 7 at 12am - Sep 22 at 11:59pm 16 days

This assignment was locked Sep 22 at 11:59pm.

The purpose of this assignment is to give you another "gentle" introduction to writing C++ programs. In spite of it being a relatively small program, there are several big new components. The first is the use of CMake to generate platform specific project build files. The second is the use of Clang Format to automatically format code. If the first two aren't enough, you'll be using the C++ random library to generate random numbers according to different distributions.

Assignment

Write a program that generates and plots three different random distributions: ***Uniform***, ***Normal***, and ***Poisson***. Write three functions, each generating an `std::vector` of data. Write a single plot function that takes the generated data and creates a plot of the distribution.

Use the following class definition for the `DistributionPair` (place it at the top of your .cpp file, rather than making a separate .hpp file for it...one time only allowance):

```
class DistributionPair
{
  public:
    DistributionPair(std::uint32_t minValue, std::uint32_t maxValue) :
        minValue(minValue),
        maxValue(maxValue),
        count(0)
    {
    }

    std::uint32_t minValue;
    std::uint32_t maxValue;
    std::uint32_t count;
};
```

A `DistributionPair` is a lower and upper bound (`minValue`/`maxValue`) along with a count of how many numbers are in that range, or "bin".

The three distribution generation functions must have exactly the following signatures:

- `std::vector<DistributionPair> generateUniformDistribution(std::uint32_t howMany, std::uint32_t min, std::uint32_t max, std::uint8_t numberBins);`
- `std::vector<DistributionPair> generateNormalDistribution(std::uint32_t howMany, float mean, float stdev, std::uint8_t numberBins);`
- `std::vector<DistributionPair> generatePoissonDistribution(std::uint32_t howMany, std::uint8_t howOften,`

`std::uint8_t numberBins);`

Each of these functions generates `howMany` random numbers and then bins them based on their value. The `numberBins` parameter to each generation function is how many `DistributionPairs` (or bins) to make, the trick is in correctly setting the upper/lower values for each bin. You may assume a 'nice' number of bins is passed to these functions. In other words, the `minValue`/`maxValue` for each bin will fall on integer boundaries, no need to worry about fractional values.

Here are the guidelines to use for each distribution type:

- Uniform
  - min/max define the range of numbers to generate [min, max]
  - Generate only integer random numbers.
- Normal
  - The minimum value to bin is the mean minus four times the standard deviation. The maximum value to bin is the mean plus four times the standard deviation.
    - Overall min: mean - 4 * stdev
    - Overall max: mean + 4 * stdev. You may find it necessary to have the max be 'mean + 4 * stdev - 1', in order to have the correct number of bins. The number of bins in the most important part of this, get the right number of bins and the min/max in those bins will fall out correctly.
- Poisson
  - The minimum value to bin is 0, Poisson distributions do not generate negative numbers. The maximum value to bin is `numberBins - 1`

If a value falls below the smallest bin range, count it in that bin. If a value falls above the larger bin range, count it in that bin. This shouldn't happen with the Uniform distribution, mostly on the Normal distribution, and possibly on the upper range of the Poisson distribution.

To bin a fractional value, when the fractional value falls between the `maxValue` for one bin and the `minValue` for the next bin, place it in the bin belonging to the `maxValue`.

The plot function must have exactly the following signature:

- `void plotDistribution(std::string title, const std::vector<DistributionPair>& distribution, const std::uint8_t maxPlotLineSize);`
  - `maxPlotLineSize` is the number of characters to use for the bin that has the most values. You'll need to find the bin with the largest count and then scale each bin according to the largest one, using `maxPlotLineSize` as the max number of characters to display on a line.

Implementation Notes

- All of the C++ random number generation code is located in the `<random>` header file.
- To format output for `std::cout`, use the `<iomanip>` header file. Here is a tutorial on it (don't do '`using namespace std;`' like the author, however: **link** **(https://www.cprogramming.com/tutorial/iomanip.html)**

Testing Code

The grader for the class (Cameron) wrote some testing code you should use to verify you've written your generateX functions correctly. Follow the instructions at **this page** to incorporate it into your assignment.

Submission Notes

- Turn in two files, **Assignment2.cpp** and **CMakeLists.txt**.
- Use ***EXACTLY*** `RandDistributions` as the name for the `add_executable` target in your CMakeLists.txt file. The reason for this is to allow the script the grader uses to compile and run the programs to know the name of the executable to, well, execute.
- Your code must compile without any warnings or compiler errors. See syllabus regarding code that has compiler errors.
- Your code must adhere to the CS 3460 coding standard: **link**
- Your code must be formatted through the use of the following clang-format configuration file: **link**

Sample Run

The following code produces the output below:

```
    auto uniform = generateUniformDistribution(100000, 0, 79, 40);
    plotDistribution("--- Uniform ---", uniform, 80);

    auto normal = generateNormalDistribution(100000, 50, 5, 40);
    plotDistribution("--- Normal ---", normal, 80);

    auto poisson = generatePoissonDistribution(100000, 6, 40);
    plotDistribution("--- Poisson ---", poisson, 80);
```

```
--- Uniform ---
[  0,   1] : ********************************************************************************
[  2,   3] : ********************************************************************
[  4,   5] : ********************************************************************************
[  6,   7] : ********************************************************************************
[  8,   9] : ********************************************************************
[ 10,  11] : ********************************************************************************
[ 12,  13] : ********************************************************************************
[ 14,  15] : ********************************************************************************
[ 16,  17] : ********************************************************************************
[ 18,  19] : ********************************************************************
[ 20,  21] : ********************************************************************************
[ 22,  23] : ********************************************************************************
[ 24,  25] : ************************************************************************
[ 26,  27] : ********************************************************************************
[ 28,  29] : ********************************************************************************
[ 30,  31] : ********************************************************************
[ 32,  33] : ************************************************************************
[ 34,  35] : ********************************************************************************
[ 36,  37] : ********************************************************************************
[ 38,  39] : ********************************************************************
[ 40,  41] : ********************************************************************************
```

```
[ 42,  43] : ******************************************************************************
[ 44,  45] : ******************************************************************************
[ 46,  47] : ******************************************************************************
[ 48,  49] : *****************************************************************************
[ 50,  51] : ********************************************************************************
[ 52,  53] : ****************************************************************************
[ 54,  55] : **************************************************************************
[ 56,  57] : ****************************************************************************
[ 58,  59] : ****************************************************************************
[ 60,  61] : *****************************************************************************
[ 62,  63] : ****************************************************************************
[ 64,  65] : *******************************************************************************
[ 66,  67] : **************************************************************************
[ 68,  69] : *****************************************************************************
[ 70,  71] : ****************************************************************************
[ 72,  73] : ******************************************************************************
[ 74,  75] : ****************************************************************************
[ 76,  77] : ******************************************************************************
[ 78,  79] : ****************************************************************************

--- Normal ---
[ 30,  30] :
[ 31,  31] :
[ 32,  32] :
[ 33,  33] :
[ 34,  34] :
[ 35,  35] : *
[ 36,  36] : **
[ 37,  37] : ***
[ 38,  38] : ******
[ 39,  39] : ********
[ 40,  40] : *************
[ 41,  41] : *******************
[ 42,  42] : *************************
[ 43,  43] : *********************************
[ 44,  44] : *******************************************
[ 45,  45] : ******************************************************
[ 46,  46] : **************************************************************
[ 47,  47] : **********************************************************************
[ 48,  48] : ******************************************************************************
[ 49,  49] : ********************************************************************************
[ 50,  50] : *******************************************************************************
[ 51,  51] : ****************************************************************************
[ 52,  52] : **********************************************************************
[ 53,  53] : ***************************************************************
[ 54,  54] : ******************************************************
[ 55,  55] : *******************************************
[ 56,  56] : *********************************
[ 57,  57] : **************************
[ 58,  58] : *******************
[ 59,  59] : *************
[ 60,  60] : *********
[ 61,  61] : *****
[ 62,  62] : ***
[ 63,  63] : **
[ 64,  64] : *
```

```
[ 65,  65] :
[ 66,  66] :
[ 67,  67] :
[ 68,  68] :
[ 69,  69] :

--- Poisson ---
[  0,   0] : *
[  1,   1] : *******
[  2,   2] : **********************
[  3,   3] : ******************************************
[  4,   4] : ************************************************************
[  5,   5] : ********************************************************************************
[  6,   6] : ************************************************************************
[  7,   7] : ********************************************************************
[  8,   8] : ************************************************
[  9,   9] : **********************************
[ 10,  10] : ********************
[ 11,  11] : ***********
[ 12,  12] : *****
[ 13,  13] : **
[ 14,  14] : *
[ 15,  15] :
[ 16,  16] :
[ 17,  17] :
[ 18,  18] :
[ 19,  19] :
[ 20,  20] :
[ 21,  21] :
[ 22,  22] :
[ 23,  23] :
[ 24,  24] :
[ 25,  25] :
[ 26,  26] :
[ 27,  27] :
[ 28,  28] :
[ 29,  29] :
[ 30,  30] :
[ 31,  31] :
[ 32,  32] :
[ 33,  33] :
[ 34,  34] :
[ 35,  35] :
[ 36,  36] :
[ 37,  37] :
[ 38,  38] :
[ 39,  39] :
```

Assignment 2 - Random Distributions

Criteria	Ratings		Pts
CMake generates valid build projects for both MSVC and g++ on LInux	**6.0 pts** **Full Marks**	**0.0 pts** **No Marks**	6.0 pts
Compiles without warnings or errors on Windows using MSVC No middle ground, no credit or full credit	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**	5.0 pts
Compiles without warnings or errors on Linux using g++ No middle ground, no credit or full credit	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**	5.0 pts
Follows required course code style. Must also be formatted using clang-format.	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**	5.0 pts
Correct 'generateUniformDistribution' implementation	**10.0 pts** **Full Marks**	**0.0 pts** **No Marks**	10.0 pts
Correct 'generateNormalDistribution' implementation	**10.0 pts** **Full Marks**	**0.0 pts** **No Marks**	10.0 pts
Correct 'generatePoissonDistribution' implementation.	**10.0 pts** **Full Marks**	**0.0 pts** **No Marks**	10.0 pts
Correct 'plotDistribution' implementation (Produces the correct display).	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**	5.0 pts
			Total Points: 56.0