Programming Assignment_1: Random Number Generation and Analysis

Overview

This assignment focuses on generating random numbers using various distributions, analyzing their statistics, and visualizing the results. You will work on three scenarios with predefined parameters and complete the specified tasks for each scenario.

Objectives

- Understand and implement random number generation using uniform and normal distributions.
- Analyze the statistical properties of generated data.
- Visualize data using histograms.

Assignment Specification

Scenarios

For each of the **three scenarios** with the combinations of parameters μ, σ, m, M, N provided in the table below, perform the following **three tasks.**

Scenario	μ (Mean)	σ (Std. Dev.)	m (Min)	M (Max)	N (Sample Size)
1	5	1	1	8	20
2	2^{10}	2^8	1	2000	200,000
3	2^{12}	1.3×2^{10}	1	8100	2,000,000

Tasks

Task 1 (60 Points): Random Number Generation

Write a C program that generates six sequences of random numbers for each scenario. The sequences must include:

1. Uniform integers in $[m, M]$.
2. Uniform real numbers in $[m, M]$.
3. Normally distributed integers with mean μ and standard deviation σ.
4. Normally distributed real numbers with mean μ and standard deviation σ.
5. Truncated normal integers within $[m, M]$.
6. Truncated normal real numbers within $[m, M]$.

Output:

- Save each sequence to separate `.txt` files in subfolders named after the scenarios (e.g., `DATA/Scenario1/uniform_integers.txt`).
- Look at the `Deliverables` folder for the subfolder hierarchy to save the generated data.

Task 2 (20 Points): Statistical Analysis

For each sequence generated in Task 1, compute the following:

1. **Sample Mean** using : $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$

   Where:

   - (N) is the number of random numbers in the sequence.
   - (x_i) represents each random number in the sequence.

2. **Sample Standard Deviation** using: $s = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2}$

   Where:

   - ($N - 1$) accounts for the degrees of freedom in the sample.
   - $(x_i - \bar{x})^2)$ is the squared difference between each data point and the mean.

Output:

- Create a summary table given below showing the **sample mean** and **sample standard deviation** *for each sequence under each of the three scenarios*.
- Include this table as `statistics.pdf` in your final submission.

	Sample Mean	Sample Standard Deviation
Scenario-1		
Uniform integers		
Uniform real numbers		
Normally distributed integers		
Normally distributed real numbers		
Truncated normal integers		
Truncated normal real numbers		
Scenario-2		
Uniform integers		
Uniform real numbers		
Normally distributed integers		
Normally distributed real numbers		

	Sample Mean μ	Sample Standard Deviation σ
Truncated normal integers		
Truncated normal real numbers		
Scenario-3		
Uniform integers		
Uniform real numbers		
Normally distributed integers		
Normally distributed real numbers		
Truncated normal integers		
Truncated normal real numbers		

Task 3 (20 Points): Histogram Generation

For **Scenario 3**, plot histograms with **50 bins** for **all 6 distributions**. *Please include the file name at the bottom and the corresponding team member's name at the top for all six histograms.* Refer to the sample file, `histogram.pdf`, where the "file names" are displayed at the bottom and the instructor's name, "Bhargav Bhatkalkar," is included in the top for all the 6 histogram plots. *You may use Microsoft Excel, other applications, or the provided Python script to plot the histograms.*

Output:

- Save the histograms as `.txt` files in the `Histogram/Scenario3/` folder.
- Include the output as `histogram.pdf` in your final submission.
- Look at the `Deliverables` folder for the subfolder hierarchy to save the generated data.

Sample Codes for Reference

In the `SAMPLE` folder, you can refer to the code examples to learn the following:

- Generating Histogram of **50 bins** for N random numbers from a normal distribution using the **Box-Muller transform**
- Generating different types of random numbers. *You need to implement additional random number generators to complete the assignment.*
- Writing and reading the output of a C program to and from a `.txt` file
- The subfolder `Python_code_for_histogram_plot` provides a Python script to automatically plot histograms and save them in a PDF file. Please read the instruction file **Install Python on Linux and Plot Histogram.pdf** carefully to follow the steps. This automated script can save you time when plotting histograms during experimentation!

Deliverables

Submit a single `Deliverables.zip` file containing:

1. A folder `CODE` containing the following:
    - all `.c` files and any optional `.h` header files used in your project.
    - a `Makefile` supporting the following commands:
        - `make`: compiles the project and generates a single executable file named `ass1`
        - `make clean`: removes all compiled files, object files, executables, and any temporary files generated during the build process.
2. A folder `DATA` with subfolders `Scenario1`, `Scenario2`, `Scenario3` containing all 6 random sequences `.txt` files generated for each of the 3 scenarios. Look at the `Deliverables` folder for the subfolder hierarchy to save the generated data.
3. A folder `HISTOGRAM` containing the `.txt` files for histograms of all 6 random sequences generated for 3^{rd} scenario. Look at the `Deliverables` folder for the subfolder hierarchy to save the generated data.
4. A `README.txt` file that includes team details, corresponding team member information, instructions on how to compile and run the programs, and any specific guidelines or notes about the project. *Refer to the course syllabus to know more on the assignment submission guidelines.*
5. A `histogram.pdf` file containing histograms with **50 bins** for **all 6 distributions** for **Scenario 3. See the give sample `histogram.pdf` file under the folder `Sample`.
6. A `statistic.pdf` file showing a table for the **sample mean** (μ) and **sample standard deviation** (σ) for each sequence under each of the three scenarios.

Deliverables Folder Structure

You are required to strictly adhere to the following structure for the `Deliverables.zip` file when submitting the assignment.

Deliverables.zip
|
|-- **CODE/**
| |-- *.c # All .c source files
| |-- *.h # Any optional header files
| |-- Makefile # Makefile with commands: make and make clean
| |-- executable # **Statically linked executable file** of your implementation
| | #Try running it on multiple computers to check its portability
|-- **DATA/**
| |-- **Scenario1/**
| | |-- uniform_integers.txt
| | |-- uniform_real.txt
| | |-- normal_real.txt
| | |-- normal_integer.txt
| | |-- truncated_normal_real.txt
| | |-- truncated_normal_integer.txt
| |

```
|   |-- Scenario2/
|   |   |-- uniform_integers.txt
|   |   |-- uniform_real.txt
|   |   |-- normal_real.txt
|   |   |-- normal_integer.txt
|   |   |-- truncated_normal_real.txt
|   |   |-- truncated_normal_integer.txt
|   |
|   |-- Scenario3/
|       |-- uniform_integers.txt
|       |-- uniform_real.txt
|       |-- normal_real.txt
|       |-- normal_integer.txt
|       |-- truncated_normal_real.txt
|       |-- truncated_normal_integer.txt
|
|-- HISTOGRAM/
|   |-- uniform_integers_histogram.txt
|   |-- uniform_real_histogram.txt
|   |-- normal_real_histogram.txt
|   |-- normal_integer_histogram.txt
|   |-- truncated_normal_real_histogram.txt
|   |-- truncated_normal_integer_histogram.txt
|
|-- README.txt        # Includes team details, instructions, and guidelines
|-- histogram.pdf     # Histograms with 50 bins for all 6 distributions for Scenario 3
|-- statistics.pdf    # Sample μ and σ for each sequence under the three scenarios
```

Important Note

- ***All submissions will be tested on a Linux platform.***
- ***You must implement, debug, and thoroughly test your assignment using the GCC compiler on a Linux platform before submission.***
- ***If your Makefile fails to compile the project or generate the executable, the entire team will receive a grade of ZERO.***
- ***The instructor/grader will not be responsible for debugging your code.***
- ***Carefully review the assignment specifications and strictly adhere to the submission guidelines to avoid penalties.***
- ***The submission due date is final, and no late submissions will be accepted under any circumstances.***
- ***Go through the course syllabus to know more on the assignment submission requirements and guidelines on the formation of assignment teams.***

Rubric

Task	Full Marks	Partial Marks (based on the level of implementation and adhering to the submission requirements)	No makefile submission, not adhering to the submission guidelines, No submission before due date
Task 1	60 pts - 10 points for each generator	0 to 40 pts	0
Task 2	20 pts - 10 points for Sample Mean -10 points for Sample Standard Deviation	0 to 15 pts	0
Task 3	20 pts - 10 points for generating Histogram - 10 points for plotting the Histogram	0 to 15 pts	0

Total Points: **100**

Project Code Skeleton *(For Reference Only)*

The **skeletal code** for your project, which you may use to complete your project by filling in the missing code, is provided. *Please note that it is just for your reference. You are free to use your own modular approach to implement your project, and you are not required to strictly adhere to the skeletal code*

```
/*****************************************************************************
 File: random_number_statistics.c

Purpose:
=========
```

```
This program generates random numbers using uniform and normal
distributions,writes the generated sequences to text files, and calculates the
sample mean and standard deviation for each sequence. It also generates
histograms for the generated sequences.

Features:
=========
1.**Uniform Integer Distribution**: Generates random integers within a specified
range [m, M], where all integers have equal probability.
2.**Uniform Real Distribution**: Generates random real numbers uniformly
distributed in [m, M].
3.**Normal Distribution (Real)**: Generates random real numbers from a normal
distribution with specified mean and standard deviation.
4.**Normal Distribution (Integer)**: Generates random integer numbers from a
normal distribution with specified mean and standard deviation.
5.**Truncated Normal Distribution (Real)**: Generates random real numbers from a
normal distribution truncated to lie within [m, M].
6.**Truncated Normal Distribution (Integer)**: Generates random integers from a
normal distribution truncated to lie within [m, M].
7.To compute the sample mean and standard deviation of the generated sequences.
8.Generates histograms for all the 6 distributions in Scenario 3.

Scenarios:
==========
1.Scenario 1: mu=5, sigma=1, m=1, M=8, N=20
2.Scenario 2: mu=2^10, sigma=2^8, m=1, M=2000, N=200,000
3.Scenario 3: mu=2^12, sigma=1.3*(2^10), m=1, M=8100, N=2,000,000

Output:
==========
- Separate .txt files are created for each generator in the respective subfolders
(Scenario1, Scenario2, Scenario3) under the DATA directory.
- The program calculates the sample mean and standard deviation for all the six
generators of the three given scenarios.
- Histograms are generated for all the generators in Scenario 3 and written to
the HISTOGRAM folder.

*******************************************************************************
/

#include <stdio.h>    // For standard I/O operations
#include <math.h>     // For mathematical functions
#include <stdlib.h>   // For memory allocation and exit() function
#include <time.h>     // For seeding the random number generator
#include <string.h>   // For string manipulation
#include <sys/stat.h> // For mkdir()
#include <errno.h>    // For error handling

// Macros for generating random numbers
#define frand() (rand() / (double)RAND_MAX) // Uniform random number in [0, 1)
#define nrand() (sqrt(-2 * log(frand())) * cos(2 * M_PI * frand())) // Normal
random number

// Number of bins for histograms
#define HISTOGRAM_BINS 50
```

```c
// Function prototypes
void generate_random_numbers_to_file(const char* filename, int type, double m,
double M, double mu, double sigma, int N);

void calculate_statistics_from_file(const char* filename, int N);

void generate_histogram_from_file(const char* input_filename, const char*
output_filename, int bins, double min, double max);

int generate_uniform_integer(double m, double M);

double generate_uniform_real(double m, double M);

double generate_normal_real(double mu, double sigma);

int generate_normal_integer(double mu, double sigma);

double generate_truncated_normal_real(double m, double M, double mu, double
sigma);

int generate_truncated_normal_integer(double m, double M, double mu, double
sigma);

double calculate_mean(double* data, int n);

double calculate_std_dev(double* data, int n, double mean);

int create_directory(const char* path);

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

    // Define scenarios: {mu, sigma, m, M, N}
    double scenarios[3][5] = {
        {5, 1, 1, 8, 20},
        {pow(2, 10), pow(2, 8), 1, 2000, 200000},
        {pow(2, 12), 1.3 * pow(2, 10), 1, 8100, 2000000}
    };

    // Paths for directories and files
    char* subfolders[3] = {"DATA/Scenario1", "DATA/Scenario2", "DATA/Scenario3"};
    char* histogram_folder = "HISTOGRAM";

    // Directory creation (e.g., DATA, HISTOGRAM, subfolders)
    // Ensure to use `create_directory` function and validate each directory
creation.

    // Loop through each scenario to process random numbers
    for (int i = 0; i < 3; i++) {
        double mu = scenarios[i][0];
        double sigma = scenarios[i][1];
        double m = scenarios[i][2];
        double M = scenarios[i][3];
        int N = (int)scenarios[i][4];

        // Process random number generation, statistics, and histograms
```

```
        // Utilize appropriate function calls (detailed below) for each step.
    }

    return 0;
}


/*
Function: create_directory
==========================
Creates a directory if it does not already exist.

Parameters:
    path - Path of the directory to create

Returns:
    0 if the directory is successfully created or already exists,
    -1 if an error occurs.
*/

int create_directory(const char* path)
{

}


/*
Function: generate_random_numbers_to_file
=========================================
Generates random numbers based on the 6 generator types (e.g., uniform, normal)
and writes them to a file.

Parameters:
    filename - Name of the file to write the random numbers
    type - Type of random number generator (1=Uniform Int, 2=Uniform Real, etc.)
    m, M - Range for uniform generators or truncation
    mu, sigma - Mean and standard deviation for normal generators
    N - Number of random numbers to generate
*/

void generate_random_numbers_to_file(const char* filename, int type, double m,
double M, double mu, double sigma, int N)
{

}


/*
Function: calculate_statistics_from_file
========================================
Reads random numbers from a file, computes mean and standard deviation, and
prints results.

Parameters:
    filename - Name of the file to read random numbers from
    N - Number of random numbers in the file
```

```
*/

void calculate_statistics_from_file(const char* filename, int N)
{

}


/*
Function: generate_histogram_from_file
=======================================
Reads numbers from a file, generates a histogram, and saves the histogram to a
new file.

Parameters:
    input_filename  - File containing the numbers to process
    output_filename - File to write the histogram to
    bins            - Number of bins in the histogram
    min, max        - Range of values to be included in the histogram
*/

void generate_histogram_from_file(const char* input_filename, const char*
output_filename, int bins, double min, double max)
{

}


/*
Function: generate_uniform_integer
==================================
Generates a random integer uniformly distributed in [m, M].

Parameters:
    m - Lower bound of the range
    M - Upper bound of the range

Returns:
    A random integer in the range [m, M].
*/

int generate_uniform_integer(double m, double M)
{

}


/*
Function: generate_uniform_real
===============================
Generates a random real number uniformly distributed in [m, M].

Parameters:
    m - Lower bound of the range
    M - Upper bound of the range
```

```
Returns:
    A random real number in the range [m, M].

*/
double generate_uniform_real(double m, double M)
{

}


/*
Function: generate_normal_integer
=================================
Generates a random integer from a normal distribution with the specified mean and
standard deviation.

Parameters:
    mu - Mean of the normal distribution
    sigma - Standard deviation of the normal distribution

Returns:
    A random integer from the normal distribution.
*/

int generate_normal_integer(double mu, double sigma)
{

}


/*
Function: generate_normal_real
==============================
Generates a random real number from a normal distribution with the specified mean
and standard deviation.

Parameters:
    mu - Mean of the normal distribution
    sigma - Standard deviation of the normal distribution

Returns:
    A random real number from the normal distribution.
*/

double generate_normal_real(double mu, double sigma)
{

}


/*
Function: generate_truncated_normal_integer
===========================================
Generates a random integer from a normal distribution, truncated to lie within
the range [m, M].
```

```
Parameters:
    m - Lower bound of the range
    M - Upper bound of the range
    mu - Mean of the normal distribution
    sigma - Standard deviation of the normal distribution

Returns:
    A random integer from the truncated normal distribution.
*/

int generate_truncated_normal_integer(double m, double M, double mu, double
sigma)
{

}


/*
Function: generate_truncated_normal_real
========================================
Generates a random real number from a normal distribution, truncated to lie
within the range [m, M].

Parameters:
    m - Lower bound of the range
    M - Upper bound of the range
    mu - Mean of the normal distribution
    sigma - Standard deviation of the normal distribution

Returns:
    A random real number from the truncated normal distribution.
*/

double generate_truncated_normal_real(double m, double M, double mu, double
sigma)
{

}


/*
Function: calculate_mean
========================
Calculates the sample mean of a sequence of numbers.

Parameters:
    data - Pointer to an array of numbers
    n - Number of elements in the array

Returns:
    The sample mean of the sequence.
*/

double calculate_mean(double* data, int n)
{
```

```
}


/*
Function: calculate_std_dev
==========================
Calculates the sample standard deviation of a sequence of numbers.

Parameters:
    data - Pointer to an array of numbers
    n - Number of elements in the array
    mean - The sample mean of the sequence

Returns:
    The sample standard deviation of the sequence.
*/

double calculate_std_dev(double* data, int n, double mean)
{

}
```