**Jacob Rook**

**Data Structure and Algorithms II**

**Project 4**

**Functional Decomposition**

**User-defined data structures used as parameters in the functions**

/\*

\* @brief DataSet This structure is used to store all the information for

\* the Monte Carlo algorithm for a given simulation.

\*

\* @var numBatches Number of Batches in the simulation

\*

\* @var numItems Number of items in a batch

\*

\* @var perBadBath Percent of bad batches in the simulation

\*

\* @var perBadItems Percent of bad items in a bad batch

\*

\* @var sampledItems Number of items sampled in the simulation

\*

\* @var simBadBatches Number of bad batches made in the simulation

\*

\* @var simBadBatchesDet Number of detected bad batches in the simulation

\*/

**typedef** **struct** Data {

**int** numBatches;

**int** numItems;

**int** perBadBatch;

**int** perBadItem;

**int** sampledItems;

**int** simBadBatches;

**int** simBadBatchesDet;

}DataSet;

**Files and Functions in the Program**

/\*

\* @file rook-j-p4.c This file contains the main file for the Monte Carlo to call]

\* the functions for the Monte Carlo simulation and algorithm. The main function will

\* ask the user if they would like to convert the SimParameters.txt to a binary file

\* if so, the program will call txt2Binary to do just that. Then the program will ask

\* if the user will want to do the Monte Carlo simulation or algorithm continuously

\* until the exit program option is chosen.

\*

\* @author Jacob Rook

\* @date 11/16/2017

\* @info Course COP 4534

\* Project 4

\*/

**Rook-j-p4.c**

/\*

\* @file monteCarlo.h This file contains all of the functions to execute the

\* Monte Carlo simulation and algorithm. The user just has to call either MCSimulation

\* or MCAlgorithm depending on what they want to do and the functions will handle

\* the rest. Further detail of the process can be found above the function prototypes.

\*

\* @author Jacob Rook

\* @date 11/16/2017

\* @info Course COP 4534

\* Project 4

\*/

**monteCarlo.c monteCarlo.h**

/\*

\* @brief DataSet This structure is used to store all the information for

\* the Monte Carlo algorithm for a given simulation.

\*

\* @var numBatches Number of Batches in the simulation

\*

\* @var numItems Number of items in a batch

\*

\* @var perBadBath Percent of bad batches in the simulation

\*

\* @var perBadItems Percent of bad items in a bad batch

\*

\* @var sampledItems Number of items sampled in the simulation

\*

\* @var simBadBatches Number of bad batches made in the simulation

\*

\* @var simBadBatchesDet Number of detected bad batches in the simulation

\*/

**typedef** **struct** Data {

**int** numBatches;

**int** numItems;

**int** perBadBatch;

**int** perBadItem;

**int** sampledItems;

**int** simBadBatches;

**int** simBadBatchesDet;

}DataSet;

/\*

\* @brief txt2Binary This function converts the SimParameters.txt file to a

\* binary file, namely SimParameters.dat

\*

\*/

**void** **txt2Binary**(**void**);

/\*

\* @brief MCSimulation This function runs the Monte Carlo simulation calculating

\* the expectation of the given data. The data is read from the SimParameters.dat

\* file. The file stores only integers and in the order of the number of simulations,

\* the number of categories in the given simulation, the frequencies for each

\* category, and the number of events to simulate. The function reads all of this

\* information and passes the necessary information to runMCSim to do the actual

\* simulation. The frequency list is store using dynamically allocated memory.

\*/

**void** **MCSimulation**(**void**);

/\*

\* @brief runMCSim This function actually runs the Monte Carlo simulation. The

\* function first calls calExpectation to calculate the expect expectation. This

\* must be done first because the frequency list is modified for the generateRandEvent

\* function. The frequency list is turned into an interval list by adding all of

\* the previous frequencies to get the total up to that point. generateRandEvent

\* then can use that interval list to generate a random event. After all of the

\* random events happen the function calls printSimResults to print out the

\* results of the simulation.

\*

\* @var numCategories The number of categories for the given simulation

\*

\* @var frequenyList List of all of the frequencies for all the categories

\* of the simulation.

\*

\* @var numEvents Number of events to generate for the simulation

\*/

**void** **runMCSim**(**int** numCategories, **int** frequencyList[], **int** numEvents);

/\*

\* @brief calExpecation This function calculates the expected expectation for

\* a given data set.

\*

\* @var numCategories The number of categories in the data set

\*

\* @var frequencyList The frequency list for the given categories

\*

\* return Returns the expectation of the data set

\*/

**double** **calExpectation**(**int** numCategories, **int** frequencyList[]);

/\*

\* @brief generateRandEvent This function generates a random event based of

\* the passed interval list. The function uses the rand() function to generate a

\* random number in between the interval an depending on which indexes the random

\* number is, determines the Event. The lesser event is returned.

\*

\* @var numCategories The number of categories in the data set

\*

\* @var intervalList List for all the intervals for the given categories

\*/

**int** **generateRandEvent**(**int** numCategories, **int** intervalList[]);

/\*

\* @brief printSimResults This function prints the Simulation results to

\* the stdout.

\*

\* @var numEvents Number of events in the simulation

\*

\* @var simResults Simulated expectation

\*

\* @var expectedResults Expected Expectation

\*/

**void** **printSimResults**(**int** numEvents, **double** simResult, **double** expectedResult);

/\*

\* @brief MCAlgorithm This function runs the Monte Carlo algorithm. Data is

\* read from four text files, namely: c1.txt, c2.txt, c3.txt, and c4.txt. These

\* files contain the number of Batches, number of items in a batch, percentage of

\* batches containing bad items, percentage of items that are bad in a bad set,

\* and items sampled from each set in that order. The function reads each file one

\* by one and then calls readConfig, generateDataSets, and analyzeDataSets in that

\* order to execute the Monte Carlo algorithm. The names are self explanatory.

\* After all four simulations are processed a summary is printed with printSummary.

\* All of the data for the simulations are stored in an array of DataSet.

\*/

**void** **MCAlgorithm**(**void**);

/\*

\* @brief readConfig This function reads the file pointed to by fp into the

\* DataSet structure data and prints the data to stdout.

\*

\* @var data Structure to store the data read from the file

\*

\* @var fp File pointer to the file that needs to be read

\*/

**void** **readConfig**(DataSet \*data, FILE \*fp);

/\*

\* @brief generateDataSets This function creates the physical data sets

\* for the algorithm. It will create a file in the ds\_File folder for each batch.

\* A batch is determine to be either good or bad. If a batch is good, g is printed

\* into the file for all of the items in the batch. If a batch is bad a mixture of

\* g and bs are printed into the file based on the percentage of bad items in a

\* bad batch. The function also keeps track of how many bad batches are created

\* and stores it into the data structure.

\*

\* @var data Structure to read and store the data for the simulation

\*

\*/

**void** **generateDataSets**(DataSet \*data);

/\*

\* @brief analyzeDataSets This function reads all of the files created

\* by the generateDataSets function and randomly selects items from the batch to

\* test if the batch is bad or not. If a bad item is found the simBadBatchesDet

\* variable in the data structure is incremented and the batch number is printed

\* to the stdout.

\*

\* @var data Structure to read and store the data for the simulation

\*

\*/

**void** **analyzeDataSets**(DataSet \*data);

/\*

\* @brief printSummary This function prints all of the data for an array

\* of DataSet variable.

\*

\* @var data Array of data structures used to print its

\* contents to the stdout

\*

\* @var size Size of the array to read data from

\*/

**void** **printSummary**(DataSet \*data, **int** size );