**Jacob Rook**

**Data Structure and Algorithms II**

**Project 2**

**Functional Decomposition**

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

/\*

\* @brief Customer\_t Customer structure to hold the various info needed to

\* be stored for the customer

\*

\* @var type Variable to indicate whether the customer is an arrival

\* or a departure. "A" for arrival "D" for departure.

\*

\* @var arrivalTime Variable to store the time that the customer arrived

\*

\* @var startOfServiceTime Time in which the customer begins the service

\*

\* @var departureTime Variable to store the time when the customer will leave

\*

\* @var nextCust Variable for the FIFO Queue, this variable will point

\* to the next customer in line after the current customer

\*/

**typedef** **struct** Cust

{

**char** type;

**float** arrivalTime;

**float** startOfServiceTime;

**float** departureTime;

**struct** Cust \*nextCust; //For FIFO Queue

}Customer\_t;

//Pointers to the Front and Rear of the FIFO Queue

Customer\_t \*FIFOFront;

Customer\_t \*FIFORear;

/\*

\* The Customer Priority Queue (PQ) is implemented using an array based heap of

\* customer pointers. The priority is based off of the customers arrival time or

\* departure time. The customers with the lower arrival or departure time will be

\* of higher priority and will be higher in the heap. The children of the heap

\* elements can be determined by multiplying the element's index by 2 and adding 1.

\* The index\*2 and index\*2 + 1 are the children of the customer at the index. The

\* size of the heap will be determined by the variable PQSize.

\*/

**#define** PQMAXSIZE 200

Customer\_t \*PQ[PQMAXSIZE+1];

**int** PQSize; //Current size of the PQ throughout the simulation

**Files and Functions in the Program**

/\*

\* @file main.c File contains the main function which calls the functions

\* of Analytical.h. The main function first asks the user for the statistic information

\* needed for the simulation and sets the values using the Analytical.h's set

\* functions. Then the main function calls the printAllCal function to print the

\* expected calculations. After, that the runSimulation function is called to run

\* the simulation. And, then the printComparison function is called to compare the

\* expected and simulated values.

\*

\* @author Jacob Rook

\* @date 10/12/2017

\* @info Course COP 4534

\* Project 2

\*/

**main.c**

/\*

\* @file Analytical.h File contains all of the function prototypes and function

\* descriptions for each function. The functions in Analytical.h are used to simulate

\* customers passing through a service system and calculating the simulation statistics,

\* namely the percent idle time, average time a customer spends in the system, the

\* average time a customer spends waiting in the queue, and the utilization factor

\* for the system. The Analytical functions use FIFO queues and priority queues to

\* simulate customers arriving, departing, and waiting in line. More specifically,

\* the FIFO queue is used to simulate the line and the priority queue is used to

\* simulate when a customer arrives or departs. The queue data structures are

\* provided by the Queue.h file.

\*

\* @author Jacob Rook

\* @date 10/12/2017

\* @info Course COP 4534

\* Project 2

\*/

**Analytical.c Analytical.h**

/\*

\* Absolute clock for simulation. The clock will progress whenever an event occurs.

\* The clock will be set to the events time and all other new events will be based

\* off the clock. Therefore, the simulation will be monotonically increasing.

\*/

**float** clock;

//Initial statistic values provided by the user

**int** numArrivals;

**float** lambda;

**float** mu;

**int** numService;

//Variables for the current numService and numArrivals left in the simulation

**int** serverAvailable;

**int** numArrivalsLeft;

//Variables to keep track of the statistics data through out the simulation

**float** PoSim; //Simulated time between last customer being served until next arrival

**float** WSim; //Simulated average Time a customer spends in the system

**float** WqSim; //Simulated average time a customer spends waiting in queue

**float** rhoSim; //Simulated utilization factor for system

**int** numWait; //Number of customers added to the FIFO Queue

**float** waitProb; //Simulated wait probability

/\*

\* @brief setN This function sets the value for numArrivals to the passed

\* value

\*

\* @param N Value to set numArrivals

\*/

**void** **setN**(**int** N);

/\*

\* @brief setLambda This function sets the value for lambda to the passed value

\*

\* @param L Value to set lambda

\*/

**void** **setLambda**(**float** L);

/\*

\* @brief setMu This function sets the value for mu to the passed value

\*

\* @param M Value to set mu

\*/

**void** **setMu**(**float** M);

/\*

\* @brief setM This function sets the value for numService to the passed

\* value

\*

\* @param M Value to set numService

\*/

**void** **setM**(**float** M);

/\*

\* @brief calIdle Function calculates the expected percent idle time (Po),

\* the percent of time that no one is in the system, of a long simulation

\*

\* @return float Returns the value of the expected percent idle time

\*/

**float** **calIdle**(**void**);

/\*

\* @brief calAvePeopleSys Function calculates the expected average number of

\* people in the system (L) of a long simulation

\*

\* @return float Returns the value of the expected average number

\* of people in the simulation

\*/

**float** **calAvePeopleSys**(**void**);

/\*

\* @brief calAveTimeSys Function calculates the expected average time a

\* customer spends in the system (W), the time the customer spent in line plus the

\* time spent getting service, of a long simulation

\*

\* @return float Returns the expected average time a customer spends

\* in the system

\*/

**float** **calAveTimeSys**(**void**);

/\*

\* @brief calAveCustQ Function calculates the expected average number

\* of customers in the queue (Lq) of a long simulation

\*

\* @return float Returns expected average number of customers in queue

\*/

**float** **calAveCustQ**(**void**);

/\*

\* @brief calAveTimeWait Function calculates the expected average time a

\* customer spends wait in the FIFO queue (Wq), or in line, of a long simulation

\*

\* @return float Returns the expected average time a customer spends

\* waiting in the FIFO queue

\*/

**float** **calAveTimeWait**(**void**);

/\*

\* @brief calUtiliFactor Function calculates the expected utilization factor

\* for the system (rho), that is, the proportion of the system's resources which

\* is used by the traffic which arrives of a long simulation

\*

\* @return float Returns the expected utilization factor for the system

\*/

**float** **calUtiliFactor**(**void**);

/\*

\* @breif printAllCal Function prints all of the expected calculations:

\* Po, L, W, Lq, Wq, and rho. The function calculates the values of the expected

\* calculations itself rather than call the individual functions to save multiple

\* calcualtions of the same variables.

\*/

**void** **printAllCal**(**void**);

/\*

\* @brief fact Function calculates the factorial of the integer n

\*

\* @return float Returns the factorial of n as float to simplify

\* calculations in the cal functions

\*/

**float** **fact**(**int** n);

/\*

\* @brief power Function calculates x^y

\*

\* @return float Returns x^y

\*/

**float** **power**(**float** x, **int** y);

/\*

\* @brief runSimulation Function acts as the main function for the Analytical

\* file. All of the simulation code is ran or called inside of runSimulation. The user

\* will call runSimulation and the simulation will run and print out the simulation

\* results. runSimulation first initialize all of the important variables the simulation

\* keeps track of throughout the simulation. Then, the first couple of customers will

\* be added to the PQ to start off the simulation. After the intital customers are

\* added, a while loop is activated to keep running until the PQ is empty. In the while

\* loop the next event will be processed and more arrivals will be added to the PQ

\* when needed. After this the function will do the last bit of calculations to finish

\* the calculations and then call the printSimulation function to print the calculations.

\*/

**void** **runSimulation**(**void**);

/\*

\* @brief getInterval Function calculates a random interval around the given

\* average using the rand() function and negative exponential distribution

\*

\* @param avg The average value to calculate a random interval around

\*

\* @return float Returns interval

\*/

**float** **getInterval**(**float** avg);

/\*

\* @brief PlaceFirstArrivals Function places M or numService number of arrivals into

\* PQ.

\*/

**void** **PlaceFirstArrivals**(**void**);

/\*

\* @brief createNewArrivl Function creates a new customer variable dynamically

\* in memory, initializes the nextCust, type, and arrivalTime of the variable and

\* returns the pointer to the new customer variable

\*

\* @return Customer\_t\* Returns pointer to the newly created customer variable

\*/

Customer\_t \***createNewArrival**(**void**);

/\*

\* @brief moreArrivals Function check to see if there are more arrivals to be

\* added to the simulation.

\*

\* @return int Returns 1 if more arrivals can be added 0 if not

\*/

**int** **moreArrivals**(**void**);

/\*

\* @brief generateNextSet Function fills the PQ with new arrivals

\*/

**void** **generateNextSet**(**void**);

/\*

\* @brief ProcessNextEvent Function takes the next customer off of the PQ,

\* determines if the customer is an arrival or departure. If the customer is an arrival

\* and their is service available the clock is updated with the customer's arrival time

\* and customer goes straight to the service. If all of the service providers are

\* busy the customer is put in the FIFO queue. If the customer is a departure, the

\* clock is updated with the customers departure time. If the FIFO queue is not empty

\* the customer at the front of the FIFO queue is then served. The customer is switched

\* from an arrival customer to a departure customer, and the departure time is calculated

\* and set for the customer. If there is not a person in the FIFO queue, the function

\* checks to see if there are any customers being served at that moment. If there are

\* no customers being served at that moment then the PoSim is updated because there

\* will be a period of time where there will be no customers in the simulation. This

\* can be calculated to be the difference in time of the next arrival and the last

\* departure. The function then frees the memory of the customer that was processed.

\*/

**void** **ProcessNextEvent**(**void**);

/\*

\* @brief processStats Function adds the appropriate times to the WSim,

\* WqSim, and rhoSim variables to keep track of how much time has elapsed for each

\* simulated calculation.

\*

\* @param Departure Departure to process stats

\*/

**void** **processStats**(Customer\_t \*Departure);

/\*

\* @brief printSimulation Function prints the results of the simulation to the

\* output.

\*/

**void** **printSimulation**(**void**);

/\*

\* @brief printComparison Function prints out the percent error of each simulated

\* calculation to the expected calculation. To save the same calculations being made

\* multiple times the function does all of the calculations inside the function.

\*/

**void** **printComparison**(**void**);

/\*

\* @file Queue.h File contains the data structures for a the Analytical files.

\* The data structures defined in this file are a customer structure, a FIFO queue,

\* and a priority queue. The customer structure stores all of the information needed

\* for the Analytical files and a customer pointer that is used in the FIFO queue

\* data structure. The FIFO queue uses the pointer in the customer structure to

\* create a linked list of customers. When a customer is added to the FIFO queue, the

\* customer is placed at the rear of the linked list and removals from the FIFO

\* queue are taken from the front of the linked list. The front and rear of the linked

\* list are tracked using a FIFOFront and FIFORear pointer. The priority queue data

\* structure is implemented using an array based heap of pointers to customer data

\* structures. The higher priority customers are those which have the lower arrival

\* time or departure time.

\*

\* @author Jacob Rook

\* @date 10/12/2017

\* @info Course COP 4534

\* Project 2

\*/

**Queue.c Queue.h**

/\*

\* @brief FIFOEnque Function adds a customer to the end of the FIFO Queue.

\*

\* @param Customer Customer to be added to the FIFO Queue

\*/

**void** **FIFOEnque**(Customer\_t \*Customer);

/\*

\* @brief FIFODeque Function removes the customer at the head of the queue

\* and moves the head pointer to the next customer in line. The original head of

\* the queue is returned.

\*

\* @return Customer\_t\* Head of FIFO queue is returned.

\*/

Customer\_t \***FIFODeque**(**void**);

/\*

\* @brief isFIFOEmpty Checks to see if the FIFO queue is empty.

\*

\* @return int Returns 1 if the FIFO queue is empty, 0 if not

\*/

**int** **isFIFOEmpty**(**void**);

/\*

\*@brief PQEnque Function will add the passed customer to the end of the

\*PQ and use an insert method comparing the parent to the customer to be added.

\*If the parent is of lower priority, the parent will be moved to the current

\*location of the customer to be added and this process will continue to recur

\*until the parent has a higher priority or the customer to be added is a the top

\*of the PQ. When the parent is of higher priority the customer will stay at the

\*current location in the PQ.

\*

\*@param Customer Customer to be added to the PQ

\*/

**void** **PQEnque**(Customer\_t \*Customer);

/\*

\*@brief PQDeque Function will take the customer off the head of the PQ,

\*take the lowest customer on the PQ and place them at the top of the PQ, use

\*percolateDown to place the lowest customer to rightful spot, and then return

\*the original customer at the top of the PQ

\*

\*@return Customer\_t Function will return the customer at the top of the PQ

\*that is being returned

\*/

Customer\_t \***PQDeque**(**void**);

/\*

\*@brief isPQEmpty Function will check to see if the PQ is empty or not

\*

\*@return int Function will return 1 if the PQ is empty, 0 if not empty

\*/

**int** **isPQEmpty**(**void**);

/\*

\* @brief percolateDown Function will use an insert method of percolating down

\* the customer at the index passed in the parameters. Typically the percolateDown

\* function will be used when the lowest priority customer has just been moved to

\* the highest priority position (1) and is needed to be percolated down. But,

\* the function can be used at any position in the PQ. However, the function will

\* compare the customers "children" in the heap at indices 2 times the customer's

\* index and 2 time the customer's index plus 1. The customer that needs to be

\* percolated down will be swapped with the higher priority child, if either child

\* is of higher priority. If neither child is of higher priority the algorithm stops

\* and the customer stays at the current index.

\*

\*@param index Index to start the percolateDown function

\*/

**void** **percolateDown**(**int** index);

/\*

\* @brief findTime Function determines which time to use (arrival or departure)

\* and returns that value.

\*

\* @param Customer Customer to find the time to return

\*

\* @return float Returns the appropriate time (arrival or departure)

\*/

**float** **findTime**(Customer\_t \*Customer);