**Data Structure\_2071035 Lee Somin**

**Technical Report – Simulation.cpp**

*Theorical Explanation of Functions in ‘Simulation.cpp’*

**#define MAX\_QUEUE\_SIZE 100**

This definition is for setting the max size of the queue.

**typedef struct element**

This structure is made to represent a customer’s information. It contains the customer’s id(int id), arrival time,(int arrival\_time), and the time that takes to complete the service(int service\_time).

**typedef struct QueueType**

This structure contains the information on position of the front and rear, and the array of the elements in the queue.

**Global Variables:**

<State variables needed for simulation>

int duration: simulation time

double arrival\_prob: average number of customers arriving in one time unit

int max\_serv\_time: maximum service time for one customer

int clock: the clock count for unit time that has passed

<Results of the simulation>

int customers: total number of customers

int served\_customers: number of customers served

int waited\_time: time the customers waited

**random**

Input: non

Return: non

This function generates a real number in range of 0 to 1 by dividing a random number by the constant number RAND\_MAX which represents the max number that the ‘rand()’ function can generate.

**is\_empty**

Input: QueueType \*q

Return: int (q->front == q->rear) //0 or 1

This function returns 1 when the queue is empty and 0 when the queue is not empty. It determines whether the queue is empty by comparing the front position and the rear position of the queue. If the front and rear is identical, the queue is empty.

**is\_full**

input: QueueType \*q

return: int ((q->rear+1)%MAX\_QUEUE\_SIZE==q->front) //0 or 1

This function returns 1 when the queue is full and 0 when the queue is not full. It determines whether the queue is full by comparing the front position and the next position of the rear. If the two are same, the queue is full.

**enqueue**

input: QueueType \*q, element item

return: non

This function performs the enqueue operation. It first determines whether the queue is full by calling ‘is\_full()’. If the queue is full, it prints the error message “Queue is full”. If the queue is not full, it moves the position of the rear to the next position of the circular queue by assigning (q->rear+1)%Max\_QUEUE\_SIZE to itself. Then, it assigns the item given by input ‘element item’ to the updated rear of the queue.

**dequeue**

input: QueueType \*q

return: element q->queue[ q->front ]

This function performs the dequeue operation. It first determines whether the queue is empty by calling ‘is\_empty()’. If the queue is empty, it prints the error message “Queue is empty”. If the queue is not empty, it moves the position of the front to the next position of the circular queue by assigning (q->front+1)%Max\_QUEUE\_SIZE to itself. Then, returns the first element that is in the updated ‘q->queue[q->front]’ of the queue.

**is\_customer\_arrived**

input: non

return: int ture / int false

The function returns true or false by the generated random number according to the probability set(double arrival\_prob). If the number generated by ‘random()’ function is smaller than the arrival probability, it returns true and otherwise, it returns false. The probability of arrival(double arrival\_prob) **should** be in range of 0~1.

**insert\_customer**

input: int arrival time

return: non

This function performs simulation to insertion of customer to the queue using the enqueue operation. This function is called when the customer arrives. It assigns the number of customers to customer.id as the id of new customer, and it increases the global variable ‘int customers’ by 1 because the number of customer that lined up in the queue has increased. The function initializes the customer’s information by assigning values to ‘element customer’. When the initialization is finished, the function adds the customer to the queue by calling function ‘enqueue()’. Then, it prints new customer’s id and service time.

**remove\_customer**

input: int banker

return: int service\_time

This function performs simulation to removement of customer in the queue using the dequeue operation. This function is called when there is no person receiving the service from a banker. It returns the service time that the customer requires. This return is used in main function and is decreased as the clock increases. So, if the queue is empty, it returns 0. If the queue is not empty, the function sets ‘element customer’ to the first customer in the queue through function ‘dequeue()’. It assigns ‘customer.service\_time-1’ to ‘int service\_time’ since the customer has already started the service at the clock. Then, add 1 to the global variable ‘int served\_customers’ and add current customer’s (customer that is starting the service) waited time(clock-customer.arrival\_time) to the global variable ‘int waited\_time’.

**print\_stat**

Input: non

Return: non

This function prints out the number of customers served by global variable ‘int served\_customers’, total wait time by ‘int waited\_time’, average wait time per person by dividing the total wait time by number of served customers, and number of customers still wating in the queue at the time when this function is called by subtracting the number of customers served from the number of total customers that waited in the queue.

**main()**

input: non

Return: non

In main function, the service time of two banker and the clock time is initialized as 0. The while loops until 'int clock' meets 'int duration'. In the loop, it prints current time and determine wether the customer has arrived by calling function ‘is\_customer\_arrived’. If the return of the function called is 1, it adds customer to the queue through ‘insert\_customer’ function. Then, using if statement, the program checks if each banker is finished with the service. If the service time is 0, the banker is considered as free to get a new customer. So, in this case, the banker gets the customer at the very front of the queue by ‘remove\_customer()’. When service time is not 0, the program just reduces the service time by 1. When 'clock' meets 'duration', while loop ends and current statistics are printed through 'print\_stat()'function.

*Execution Result is continued.*

**<Execution Result>**

텍스트이(가) 표시된 사진

자동 생성된 설명

*Result continued in the next page*

텍스트이(가) 표시된 사진

자동 생성된 설명