//Riya Sawant-CS1-COP

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//You must create a struct that stores information about a customer (name, number of tickets, line

//number, time entering line).

typedef struct customer{

char name;

int ntickets;

int linenum;

int timeenter;

int checkout;

} customer

//You must create a node struct for a linked list of customers. This struct should have a pointer

//to a customer struct, and a pointer to a node struct.

typedef struct node{

node\* customer;

//make que using linked list

struct node \*next;

}node

//You must create a struct to store a queue of customers. This struct should have two pointers –

//one to the front of the queue and one to the back.

typedef struct storeQue{

}storeQue

//----------------------------------------------

//Also, you must need to create a function that can create a customer

//using dynamic memory allocation, fill out the customer and then and return the customer. You

//have to use this function whenever you need to create a customer.

void createCustomer(){

struct customer = malloc(sizeof(customer));

return customer;

}

// Your queue must support the following operations: Enqueue,Dequeue,Peek,Empty

//(returns 1 if the queue is empty, 0 if it is not)

void enqueue()

{

}

void dequeue()

{

}

void peek()

//Return the front of the queue WITHOUT dequeuing

{

}

void empty(){

//returns 1 if the queue is empty, 0 if it is not)

if(){

//if EMPTY

return 1

}

else(){

//if NOT empty

return 0

}

}

int main(){

}

//You must free memory appropriately. Namely, when you dequeue, you’ll free memory for a

//node, but you will NOT free memory for the customer. You will free this memory a bit later right

//after you calculate when that customer will finish checking out

void freememory()

{

free();

free();

free();

}

Note that the storage of the line number is redundant, but is designed

to ease implementation.

4. You must implement all of the lines that form as an array of size 12 (stored as a constant) of

queues.

5. You must dynamically allocate memory as appropriate for linked lists.

8. You must use the memory leak detector like PA1 and as shown in earlier labs.

9. Due to the nature of the problem, when you process the input, you can add everyone into their

appropriate lines right at the beginning, before checking anyone out. This wouldn’t work in all

simulations (some of which you have to do in time order), but because there is ONLY one check

out line, you can get away with it. The only thing you have to be cognizant about is that when you

select a line, if the current time is 100 for example, and three lines have customers who arrived

before time 100 and the other lines have customers in the front who arrived AFTER time 100, you

have to ignore the customers in those lines who arrived after time 100. In the case that all the lines

have customers who arrived after time 100, you would take the line which has a customer who

arrived first. You are guaranteed no ties for arrival time so this would be unique.

//Riya Sawant-CS1-COP

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//define int checkout;

//You must use the memory leak detector like PA1 and as shown in earlier labs.

#define MAXNAMESIZE 20

#define

#define

#define

// You must dynamically allocate memory as appropriate for linked lists.

typedef struct customer{

char name[MAXNAMESIZE+1];

int ntickets;

int linenum;

int timeenter;

} customer;

//You must create a node struct for a linked list of customers. This struct should have a pointer

//to a customer struct, and a pointer to a node struct.

typedef struct node{

customer\* customerPtr;

//node\* customer;

//make que using linked list

struct node \*next;

}node;

//You must create a struct to store a queue of customers. This struct should have two pointers –

//one to the front of the queue and one to the back.

typedef struct storeQue{

node\* front;

node\* back;

}storeQue;

//----------------------------------------------

//using dynamic memory allocation, fill out the customer and then and return the customer. You

//have to use this function whenever you need to create a customer.

void createCustomer(){

struct customer = malloc(sizeof(customer));

return customer;

}

// Your queue must support the following operations: Enqueue,Dequeue,Peek,Empty

//(returns 1 if the queue is empty, 0 if it is not)

void enqueue()

{

//no malloc in

}

void dequeue()

//save where its at from front

{

}

void peek()

//Return the front of the queue WITHOUT dequeuing

//

{

}

void empty(){

//returns 1 if the queue is empty, 0 if it is not)

if(){

//if EMPTY

return 1

}

else(){

//if NOT empty

return 0

}

}

int main(){

}

//You must free memory appropriately. Namely, when you dequeue, you’ll free memory for a

//node, but you will NOT free memory for the customer. You will free this memory a bit later right

//after you calculate when that customer will finish checking out

void freememory()

{

free();

free();

}

Note that the storage of the line number is redundant, but is designed

to ease implementation.

4. You must implement all of the lines that form as an array of size 12 (stored as a constant) of

queues.

9. Due to the nature of the problem, when you process the input, you can add everyone into their

appropriate lines right at the beginning, before checking anyone out. This wouldn’t work in all

simulations (some of which you have to do in time order), but because there is ONLY one check

out line, you can get away with it.

-The only thing you have to be cognizant about is that when you

select a line, if the current time is 100 for example, and three lines have customers who arrived

before time 100 and the other lines have customers in the front who arrived AFTER time 100, you

have to ignore the customers in those lines who arrived after time 100.

-In the case that all the lines

have customers who arrived after time 100, you would take the line which has a customer who

arrived first. You are guaranteed no ties for arrival time so this would be unique.