//main.c

/\*

Mehmet Fatih Çelik 2385268

Hocam for some reason that I dont know from where but in the main, Sometimes I compile it gives run-time error, and sometimes

it does not give error, compiles but last function(displaying) is not fully correct, so I want from you to please

note that.

I actually have a problem in the main, the other functions are works perfectly fine. You can try hocam.

But I couldnt have time to fix the error which is in the main. I hope you evaluate the assignment one by one

the functions, thanks.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "queue.h"

int main(int argc, char \*argv[]){

srand(time(NULL));

int noOfCustomers, noOfCouriers, maxPurchaseTime, maxDeliveryTime;

parseInput(argc, argv, &noOfCustomers, &noOfCouriers, &maxPurchaseTime, &maxDeliveryTime);

List L;

L = createCustomerList(noOfCustomers, noOfCouriers, maxPurchaseTime, maxDeliveryTime);

Queue q;

int couriers[noOfCouriers];

q = initialiseSimulator(couriers, noOfCouriers);

//printing couriers

int i,j;

for(i=0;i<noOfCouriers;i++)

printf("%d. courier = %d\n",i+1,couriers[i]);

int clock=0, available = 0, countAvailable, temp;

int purchaseTime, deliveryTime, waitingTime, deliveryStartTime, courierID;

char deliveryType, amountOfPurchase;

int servedCustomer=0, numE = 0, numS = 0, numF = 0, counterCourier[noOfCouriers] ,totalWaitingTime = 0, numA = 0, numB = 0, maxWaitingTime = 0;

struct Node \*traversal = L->head->next;

for(i=0;i<noOfCustomers;i++){

countAvailable = 0;

// for checking if any of the couriers is available or not

for(j=0;j<noOfCouriers;j++){

if(couriers[i] == 1){

available = 1;

countAvailable++;

}

}

// if there is no customer in the queue, and at least one of the couriers is available

if(traversal->purchaseTime == clock && available){

temp = rand()%countAvailable;

int temp\_counter = 0;

for(j=0; j<noOfCouriers;j++){

if(couriers[j] && temp\_counter == temp){// if courier = 1 and counter = randomValue

couriers[j] = 0;

traversal->courierID = j+1;

break;

}

else if(couriers[j] && temp\_counter != temp)

temp\_counter++; //if courier = 1 and counter != randomValue

}

//printing couriers

for(j=0;j<noOfCouriers;j++)

printf("%d. courier = %d\n",j+1,couriers[j]);

purchaseTime = traversal->purchaseTime;

deliveryTime = traversal->deliveryTime;

traversal->deliveryStartTime = clock;

waitingTime = traversal->deliveryStartTime - purchaseTime;

if(waitingTime > maxWaitingTime)

maxWaitingTime = waitingTime;

L->head->next = traversal->next;//putting the first node to the last

L->tail->next = traversal;

traversal = L->tail;

traversal->next = NULL;

}

// Couriers are not available, we are putting customer into the queue

else if(traversal->purchaseTime == clock && !available){

newCustomer(q, traversal);

L->head->next = traversal->next;//putting the first node to the last

L->tail->next = traversal;

traversal = L->tail;

traversal->next = NULL;

}

else if(purchaseTime+deliveryTime < clock && !isEmptyQueue(q) && available){

struct Node \*customer;

couriers[traversal->courierID-1] = 1;

customer = serveCustomer(q);

purchaseTime = customer->purchaseTime;

deliveryTime = customer->deliveryTime;

deliveryStartTime = customer->deliveryStartTime;

customer->deliveryStartTime = clock;

deliveryType = customer->deliveryType;

amountOfPurchase = customer->amountOfPurchase;

courierID = customer->courierID;

waitingTime = customer->deliveryStartTime - purchaseTime;

if(waitingTime > maxWaitingTime)

maxWaitingTime = waitingTime;

}

//when robot finishes at deliveryTime + deliveryStartTime

else if(deliveryStartTime+ deliveryTime == clock && !available){

couriers[traversal->courierID-1] = 1;

servedCustomer++;

if(deliveryType == 'E')

numE++;

if(deliveryType == 'S')

numS++;

if(deliveryType == 'F')

numF++;

counterCourier[courierID-1]++;

totalWaitingTime += waitingTime;

if(amountOfPurchase == 'A')

numA++;

if(amountOfPurchase == 'B')

numB++;

clock--;

}

clock++;

}

reportStatistics(noOfCouriers, noOfCustomers, numE, numS, numF, counterCourier, clock, totalWaitingTime, maxWaitingTime, numA, numB);

return 0;

}

//queue.h

struct Node{

char deliveryType;

int purchaseTime;

int deliveryTime;

int deliveryStartTime;

int courierID;

char amountOfPurchase;

struct Node \*next;

};

struct QueueRecord{

struct Node \*front;

struct Node \*rear;

int size;

};

struct ListRecord{

struct Node \*head;

struct Node \*tail;

int size;

};

typedef struct QueueRecord \*Queue;

typedef struct ListRecord \*List;

int isEmptyQueue(Queue);

void parseInput(int, char \*[], int \*, int \*, int \*, int \*);

List createCustomerList(int, int, int, int);

Queue initialiseSimulator(int [], int);

void newCustomer(Queue, struct Node \*);

struct Node\* serveCustomer(Queue);

void reportStatistics(int, int, int, int, int, int [], int, int, int, int, int);

//deliverySimulator.c

#include <stdio.h>

#include <stdlib.h>

#include "queue.h"

void parseInput(int argc, char \*argv[], int \*noOfCustomers, int \*noOfCouriers, int \*maxPurchaseTime, int \*maxDeliveryTime){

if (argc < 2)

printf("No argument has been passed through the command line!\n");

\*noOfCustomers = atoi(argv[1]);

\*noOfCouriers = atoi(argv[2]);

\*maxPurchaseTime = atoi(argv[3]);

\*maxDeliveryTime = atoi(argv[4]);

}

List createCustomerList(int noOfCustomers, int noOfCouriers, int maxPurchaseTime, int maxDeliveryTime){

List L;

L = (struct ListRecord\*)malloc(sizeof(struct ListRecord));

if (L == NULL){

printf("Out of memory!");

exit(-1);

}

L->size = 0;

L->head = (struct Node\*)malloc(sizeof(struct Node));

if (L->head == NULL){

printf("Out of memory!");

exit(-1);

}

L->head->next = NULL;

L->tail = L->head;

int i, temp;

struct Node \*t;

for(i=0;i<noOfCustomers;i++){

t = (struct Node\*)malloc(sizeof(struct Node));

t->next = NULL;

temp = 1+rand()%3;

if (temp == 1)

t->deliveryType = 'E';

else if (temp == 2)

t->deliveryType = 'S';

else if (temp == 3)

t->deliveryType = 'F';

printf("\n %c->",t->deliveryType);

t->purchaseTime = 1 + rand()%maxPurchaseTime;

printf(" purchase: %d->",t->purchaseTime);

t->deliveryTime = 1 + rand()%maxDeliveryTime;

printf(" delivery: %d->",t->deliveryTime);

temp = 1 + rand()%1000;

if (temp>= 500)

t->amountOfPurchase = 'A';

else

t->amountOfPurchase = 'B';

printf(" amount: %c",t->amountOfPurchase);

L->tail->next = t;

L->tail = t;

L->size++;

}

//sorting part

int swapped;

t = NULL;

struct Node \*t2;

do{

swapped = 0;

t2 = L->head->next;

while (t2->next != t){

if (t2->purchaseTime > t2->next->purchaseTime){

temp = t2->purchaseTime; // swapping purchaseTime

t2->purchaseTime = t2->next->purchaseTime;

t2->next->purchaseTime = temp;

temp = t2->deliveryType; // swapping deliveryType

t2->deliveryType = t2->next->deliveryType;

t2->next->deliveryType = temp;

temp = t2->amountOfPurchase; // swapping amountOfPurchase

t2->amountOfPurchase = t2->next->amountOfPurchase;

t2->next->amountOfPurchase = temp;

temp = t2->deliveryTime; // swapping deliveryTime

t2->deliveryTime = t2->next->deliveryTime;

t2->next->deliveryTime = temp;

swapped = 1;

}

t2 = t2->next;

}

t = t2;

}while(swapped);

//for printing

printf("\nAfter swapping:");

struct Node \*traversal = L->head->next;

while(traversal){

printf("\n %c->",traversal->deliveryType);

printf(" purchase: %d->",traversal->purchaseTime);

printf(" delivery: %d->",traversal->deliveryTime);

printf(" amount: %c\n",traversal->amountOfPurchase);

traversal = traversal->next;

}

return L;

}

Queue initialiseSimulator(int couriers[], int noOfCouriers){

Queue q;

q = (struct QueueRecord\*)malloc(sizeof(struct QueueRecord));

if (q == NULL){

printf("Out of memory!");

exit(-1);

}

q->size = 0;

q->front = (struct Node\*)malloc(sizeof(struct Node));

if (q->front == NULL){

printf("Out of memory!");

exit(-1);

}

q->front->next = NULL;

q->rear = q->front;

int i;

for(i=0;i<noOfCouriers;i++)

couriers[i] = 1;

return q;

}

int isEmptyQueue(Queue q){

if (q->size == 0)

return 1;

else

return 0;

}

void newCustomer(Queue q, struct Node \*traversal){

struct Node \*t;

if(isEmptyQueue){

q->front->next = traversal;

}

else{

t = q->front;

if (traversal->deliveryType == 'E'){

if (t->next->deliveryType!= 'E')

t = t->next;

else{

traversal->next = t->next;

t->next = traversal;

if(traversal->next == NULL)//if last element

q->rear = traversal;

}

}

else if(traversal->deliveryType == 'S'){

if (t->next->deliveryType!= 'E' && t->next->deliveryType!= 'S')

t = t->next;

else{

traversal->next = t->next;

t->next = traversal;

}

if(traversal->next == NULL)//if last element

q->rear = traversal;

}

else if(traversal->deliveryType == 'F'){

q->rear->next = traversal;

q->rear = traversal;

traversal->next = NULL;

}

}

}

struct Node\* serveCustomer(Queue q){

struct Node \*removal;

removal = q->front->next;

q->front->next = removal->next;

removal->next = NULL;

return removal;

}

void reportStatistics(int noOfCouriers, int noOfCustomers, int numE, int numS, int numF, int counterCourier[], int clock, int totalWaitingTime, int maxWaitingTime, int numA, int numB){

int i;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Delivery Statistics\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("The number of couriers: %d\n",noOfCouriers);

printf("The number of customers: %d\n",noOfCustomers);

printf("Number of customers for each delivery type:\n");

printf(" Express: %d\n",numE);

printf(" Standard: %d\n",numS);

printf(" Free: %d\n",numF);

printf("Number of customers for each courier:\n");

for(i=0;i<noOfCouriers;i++)

printf(" Courier %d:%d\n",i+1,counterCourier[i]);

printf("Completion time: %d\n",clock);

printf("Average time spent in the queue: %f\n",totalWaitingTime/noOfCustomers);

printf("Maximum waiting time: %d\n",maxWaitingTime);

printf("Popular purchase: ");

if(numA > numB)

printf("A");

else if(numB > numA)

printf("B");

else

printf("Both are the same!");

}