//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 15

#define CONSTANT int

#define MAXITEMS 100

#define timeperitem 5

#define timepercustomer 30

#define maxtime 100,000,000

// 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));

//strcpy () 0

temp \* Ptr

temp = maxtime,line #, ntickets

//return temp0

return customer;

}

void init queue (){

front-> NULL

back-> NULL

}

void front cuntomer queue ()

{

}

void load queue ()

{

}

// 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

if (queue Ptr = NULL){

newnode \* front= front;

newnode \* back = back;

else

{

back->next;

back\* newnode;

}

}

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()

{

}

void freememory()

{

free(customerPtr);

free(Ptr);

}

// Riya Sawant-CS1-COP

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "leak\_detector\_c.h"

// define int checkout;

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

#define MAXNAMESIZE 15

#define CONSTANT int

#define MAXITEMS 100

#define timeperitem 5

#define timepercustomer 30

#define maxtime 100,000,000

// 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;

struct node \*next;

// node\* customer;

// make que using linked list

} 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 {

// struct before

node \*front;

node \*back;

} storeQue;

customer\* createCustomer(char\* name,int timeenter, int linenum,int ntickets);

void init queue ;

void front cuntomerqueue(customerPtr);

void loadqueue();

void enqueue();

void timeused();

void dequeue();

void getNext ();

void empty();

void math();

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

// 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.

customer\* createCustomer(char\* name,int timeenter, int linenum,int ntickets)

{

struct customer = malloc(sizeof(customer));

// strcpy () $

temp \*Ptr temp = maxtime, linenum,

ntickets

// return temp$

return customer;

}

void front customer queue(customerPtr)

// param? customerPtr $

{

if (customerPtr = NULL) {

return NULL;

} else {

return customerPtr;

}

}

void load queue()

{

// Your queue must support the following operations:

// Enqueue,Dequeue,Peek,Empty

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

}

void init queue() {

// set queuePtr(front & back) = NULL

front->NULL back->NULL

}

void enqueue() {

// no malloc in $

if (queue Ptr = NULL) {

newnode \*front = front;

newnode \*back = back;

} else {

back->next;

back \*newnode;

}

// enqueue:

// if (queuePtr = empty):

// queuePTr(front & back) = newNode

// else:

// queuePtr -> next = newNode

// queuePtr ->back = newNode

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

// void enqueue(queue \*q, int value)

//{

// node \*tmp;

// tmp = malloc(sizeof(node));

// tmp->data = value;

// tmp->next = NULL;

// if(!isempty(q))

//{

// q->rear->next = tmp;

// q->rear = tmp;

//}

// else

// {

// q->front = q->rear = tmp;

//}

// q->count++;

}

void dequeue() {

// save where its at from front

//$

if (= NULL) {

return NULL;

ptr = prt->front;

ptr->front = ptr->front->next;

}

if (front = NULL &&back = NULL)

return ptr->front;

}

void empty() {

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

// return Ptr-> front = NULL $

if () {

// if EMPTY

return 1

} else

() {

// if NOT empty

return 0

}

}

void time used() {

//$

return timepercustomer + nTickets \* TimePerItem;

}

void getNext(int currentTime) {

res = -1;

ntickets->MAXITEMS + 1

// loop for # lines

if (linenum[i] = Empty) continue tmp = frontCust[lines];

temp->timeinline > curTime continue

// timeenter?

if (temp->ntickets < minTickets) {

res = i;

minTickets = tmp->ntickets;

}

if (res != -1) {

return res;

}

int bestTime = nextTime + 1

// besttime = maxTime +1 ?$

// loop for # lines

// if (lines(empty)): continue

if (linenum[i] = NULL) {

continue;

tmp = frontCustomer(linenum[i]);

}

// timeenter

if (timeinLine < bestTime) {

res = i;

// bestTime = TimeInLine

bestTime = TimeInLine

}

return res;

}

// void peek()

// Return the front of the queue WITHOUT dequeuing

//

//{

//}

int main() {

// memory leak check

// atexit(report\_mem\_leak);

customers \*createCustomer;

int ntickets, linenum, timeenter;

char name[MAXNAMESIZE + 1];

// struct

int num\_cases scanf("%d", num\_cases) {

for (int i = 0; i < num\_cases; i++) {

int num\_customer;

scanf("d", &num\_customers);

for (int j = 0; j < num\_customers; j++) {

scanf("%d %d %s %d", timeenter, lineNum name, nticket);

}

}

}

for (i = 0; i < num\_customers, i++)

{

printf("%s from line %d checks out at time %d", name, lineNum, maxtime);

printf("%c from line %d checks out at time %d", name, lineNum, currectTime);

}

for (i = 0; i < )

{

printf("%c from line %d checks out at time %d", character, lineNum, maxtime);

}

// scan # cases

// call queue // queue lines

// loop and init queue //loop # lines// - initqueue (lines)

// loadQueues

// loop and call getnext, dequque, endpoint using ~ pointer and temp node

// increment time used

//--------

// store time in int variable

// loops for # customers

// - use setNext & dequeue functions on lines

// - set tmpPtr to customerPtr

// -if (tmptime < time in Line)

// - cartime = timeinLine

// - curTime += timeused

// - freeNode

return 0

}

void math() {

currentTime += ((temp->ntickets) \* timeperitem) + timepercustomer

return currentTime;

}

void freememory() {

free(customerPtr);

free(Ptr);

//$

}

// Riya Sawant-CS1-COP

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//#include "leak\_detector\_c.h"

// define int checkout;

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

#define MAXNAMESIZE 15

#define CONSTANT int

#define MAXITEMS 100

#define timeperitem 5

#define timepercustomer 30

#define maxtime 100,000,000

// 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;

struct node \*next;

// node\* customer;

// make que using linked list

} 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 {

// struct before

node \*front;

node \*back;

} storeQue;

customer\* createCustomer(char\* name,int timeenter, int linenum,int ntickets);

void init queue();

void frontcustomer queue(\*customerPtr);

//void load queue();

void enqueue();

void timeused();

void dequeue();

void getNext ();

void empty();

void math();

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

// 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.

customer\* createCustomer(char\* name,int timeenter, int linenum,int ntickets)

{

struct\* customer = malloc(sizeof(customer));

// strcpy () $

//int temp;

temp \*Ptr temp = maxtime, linenum,ntickets;

// return temp$

return customer;

}

void front customer queue(\*customerPtr)

{

// param? customerPtr $

if (\*customerPtr = NULL)

{

printf("NULL");

//return NULL;

}

else

{

return struct customerPtr;

}

}

//void loadqueue()

//{

// Your queue must support the following operations:

// Enqueue,Dequeue,Peek,Empty

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

//}

void init queue()

{

// set queuePtr(front & back) = NULL

//front->NULL

//back->NULL

qPTR->front = NULL

qPTr->back = NULL

}

void enqueue() {

// no malloc in $

if (queue Ptr = NULL) {

newnode\* front = front;

newnode\* back = back;

} else {

back->next;

back \*newnode;

}

// enqueue:

// if (queuePtr = empty):

// queuePTr(front & back) = newNode

// else:

// queuePtr -> next = newNode

// queuePtr ->back = newNode

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

// void enqueue(queue \*q, int value)

//{

// node \*tmp;

// tmp = malloc(sizeof(node));

// tmp->data = value;

// tmp->next = NULL;

// if(!isempty(q))

//{

// q->rear->next = tmp;

// q->rear = tmp;

//}

// else

// {

// q->front = q->rear = tmp;

//}

// q->count++;

}

void dequeue() {

// save where its at from front

//$

if (= NULL) {

return NULL;

ptr = prt->front;

ptr->front = ptr->front->next;

}

if (front = NULL &&back = NULL)

return ptr->front;

}

void empty() {

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

// return Ptr-> front = NULL $

if () {

// if EMPTY

return 1

} else

() {

// if NOT empty

return 0

}

}

void time used() {

//$

return timepercustomer + nTickets \* TimePerItem;

}

void getNext(int currentTime) {

res = -1;

ntickets->MAXITEMS + 1

// loop for # lines

if (linenum[i] = Empty) continue tmp = frontCustomer[lines];

temp->timeinline > curTime continue

// timeenter?

if (temp->ntickets < minTickets) {

res = i;

minTickets = tmp->ntickets;

}

if (res != -1) {

return res;

}

int bestTime = nextTime + 1

// besttime = maxTime +1 ?$

// loop for # lines

// if (lines(empty)): continue

if (linenum[i] = NULL) {

continue;

tmp = frontCustomer(linenum[i]);

}

// timeenter

if (timeinLine < bestTime) {

res = i;

// bestTime = TimeInLine

bestTime = TimeInLine

}

return res;

}

// void peek()

// Return the front of the queue WITHOUT dequeuing

//

//{

//}

int main() {

// memory leak check

// atexit(report\_mem\_leak);

customers \*createCustomer;

int ntickets, linenum, timeenter;

char name[MAXNAMESIZE + 1];

// struct

int num\_cases scanf("%d", num\_cases) {

for (int i = 0; i < num\_cases; i++) {

int num\_customer;

scanf("d", &num\_customers);

for (int j = 0; j < num\_customers; j++) {

scanf("%d %d %s %d", timeenter, lineNum name, nticket);

}

}

}

for (i = 0; i < num\_customers, i++)

{

printf("%s from line %d checks out at time %d", name, lineNum, currentTime);

}

for (i = 0; i < )

{

printf("%c from line %d checks out at time %d", character, lineNum, currentTime);

//printf("%c from line %d checks out at time %d", name, lineNum, currentTime);

}

// scan # cases

// call queue // queue lines

// loop and init queue //loop # lines// - initqueue (lines)

// loadQueues

// loop and call getnext, dequque, endpoint using ~ pointer and temp node

// increment time used

//--------

// store time in int variable

// loops for # customers

// - use setNext & dequeue functions on lines

// - set tmpPtr to customerPtr

// -if (tmptime < time in Line)

// - cartime = timeinLine

// - curTime += timeused

// - freeNode

return 0

}

void math() {

currentTime += ((temp->ntickets) \* timeperitem) + timepercustomer

return currentTime;

}

void freememory() {

free(customerPtr);

free(Ptr);

//$

}