**ECE 223: Airport Simulation**

**Due Date in Canvas**

Airport Simulation Program

You will have several new ADTs for this program: a FIFO queue ADT named queue.c (which may be based on YOUR existing linked list ADT - must be brought up to spec if it is not already) ,and a priority queue ADT named priority.c (also can be YOUR reused code). You will provide several different queues. The first is a priority queue we call the *event queue* which is used to keep up with program events. The rest of the queues will be regular FIFO queues that will represent passengers in line for at one place or another. The priority queue has the interface shown in priority.h You will implement this priority queue as a heap implemented in an array of fixed size as discussed in class. The second queue is a FIFO queue built in a module called queue.c with the interface shown in queue.h implemented with a linked list. There is a time ADT that keeps up with simulated time and provides random interval times for various parts of the code.

Simulation

You will write a program that simulates passengers in an airport using an event driven simulation engine. You will create events represented by a struct that includes an event type (an int), an event time (a float), event data (a passenger\_t \*) and/or a queue pointer. These events will be placed in the event queue using the event ADTs event\_schedule() method. The event queue will sort the events by time so that the event with the smallest time returns first via the event\_cause() method. A while loop will remove events from the event queue as long as there are events in the queue, and then use a switch statement to decide what to do based on the event type. Some events will cause new events to be created and added to the event queue, and others will free old events. If the event queue becomes full, it is an error and the simulation must be re-run with a larger event queue. A command line argument –e should allow the size of the event queue to be set or override a default. When there are no more events in the event queue, the simulation is over.

Events and Passengers

We will use random variables to control when each passenger arrives at the airport. We will give you code for this purpose (see below). When a passenger arrives, we first set the global simulation time using the time\_set() method. Then the passenger goes through a series of queues and after each queue will do something (check bags, have ID checked, etc.) for a period of time. We will also use random variables for each of those times. When the passenger has reached his gate we will check the total time to see if she missed her plane or not. We will be recording times all along the way so we can check the system’s behavior. When using the security scanners there are several scanners available, each with their own queue. To start with the passenger always picks the shortest line, but later we will have helpful TSA agents send them to random lines. The passenger gets in line. When the passenger gets to the scanner, we mark down the time, and then use another random variable to determine how long scanning takes – does she get stopped and searched, etc. or go straight through? Finally the passenger will ride a train out to the concourse before heading for her gate.

We will keep statistics as we go and at the end of the simulation we will produce a report showing average in the airport time, average wait time, average overall time, etc.

For each of these event types you will write a bit of code that carries out the needed effect – marking time, finding a queue, using a random variable, etc. These bits of code will be in a switch statement so that each time you take an event out of the event queue, you will use that event’s event\_type in the switch to select the right bit of code. The event queue will make sure that everything will happen in the correct order.

The main routine should be structured something like the code in main.cn (see below):

PROJECT 1

For project 1 you are to implement the basic event engine, construct the arrival loop, and code definitions for all of the events and major data structures. You do not need to implement all of the event code yet (this will come in a subsequent project). You will implement the priority queue, the event system, the time system, and the main function related functions. This main function will be the real main, not a test driver, though you might use it to run some tests.

You should test your code to make sure that the event\_queue works, sorting event times, that the main loop correctly sets it up and runs the arrival loop, and the while loop can handle different “dummy” events. The code fragments outline the interfaces and major code structure. Note there may be typos or minor errors. I will post fixes when needed.

Your program should consist of the following files:

queue.c /\* FIFO queues \*/

queue.h

priority.c /\* PQ implemented as a heap \*/

priority.h /\* using sequential (array) storage \*/

event.c /\* events and event queue (PQ) \*/

event.h

time.c /\* various time related functions \*/

time.h

sim.h /\* defs for events, passengers, etc. \*/

main.c /\* the main simulation code \*/

randsim.c /\* provided to you – calls to get \*/

randsim.h /\* times using random variables \*/

============================================================

/\* event.h \*/

typedef struct event\_s event\_t;

struct event\_s

{

int event\_type; /\* type of event – see below \*/

queue\_t \*queue /\* queue passenger is waiting in \*/

double event\_time; /\* sim time when event occurs \*/

passenger\_t \*passenger;/\* passenger related to this event \*/

};

/\* initializes events, creates a priority queue \*/

void event\_init();

/\* frees up all event space, including space in the priority

queue \*/

void event\_fini();

/\* allocate a fresh event with empty fields \*/

event\_t \*event\_create();

/\* free an event \*/

void event\_destroy(event\_t \*e);

/\* insert the event into the priority queue. The key

value is the current sim time plus the event\_time in

event. Update the event time to the key value. \*/

void event\_schedule(event\_t \*e);

/\* remove the next event from the priority and return

it to the program for execution \*/

event\_t \*event\_cause();

============================================================

/\* time.h \*/

/\* creates a double for using as the global sim time \*/

void time\_init();

/\* cleans up the time system as needed \*/

void time\_fini();

/\* returns the current sim time \*/

double time\_get();

/\* sets the current sim time \*/

void time\_set(double t);

/\* these return a random time which is used to set the

time each passenger takes to make each of these steps

these times are set into the corresponding event

before it is scheduled. \*/

double time\_arrive();

double time\_airline();

double time\_id();

double time\_scan();

double time\_train();

double time\_gate();

============================================================

/\* queue.h \*/

typedef struct queue\_s queue\_t;

/\* create and initialize a new queue

must be able to hold at least size items

return pointer to the new queue, NULL if error \*/

queue\_t \*queue\_init(int size);

/\* insert an item into the queue

return 0 if successful, -1 otherwise \*/

int queue\_insert(queue\_t \*q, passenger\_t \*c);

/\* return the next item in the queue but do not remove it

return NULL if the queue is empty or on error \*/

passenger\_t \*queue\_peek(queue\_t \*q);

/\* remove the next item from the queue

and return it, return NULL if there is an error \*/

passenger\_t \*queue\_remove(queue\_t \*q);

/\* return the number of items in the queue

You can see if queue is empty with this \*/

int queue\_size(queue\_t \*q);

/\* return nono-zero if the queue is full

This may be trivial using a linked implementation \*/

int queue\_full(queue\_t \*q);

/\* free all resourced used by the queue then free

the queue itself \*/

void queue\_finalize(queue\_t \*q);

============================================================

/\* priority.h \*/

typedef struct priority\_s priority\_t;

/\* create and initialize a new priority queue

must be able to hold at least size items

return pointer to the new priority queue, NULL if error \*/

priority\_t \*priority\_init(int size);

/\* insert an item into the priority queue

return 0 if successful, -1 otherwise \*/

int priority\_insert(priority\_t \*q, event\_t \*ev);

/\* remove the highest priority item from the queue

and return it, return NULL if there is an error \*/

event\_t \*priority\_remove(priority\_t \*q);

/\* return non-zero if the priority queue us empty \*/

int priority\_empty(priority\_t \*q);

/\* return nono-zero if the priority queue is full

This may be trivial using a linked implementation \*/

int priority\_full(priority\_t \*q);

/\* free all resourced used by the priority queue then free

the queue itself \*/

void priority\_finalize(priority\_t \*q);

============================================================

<sim.h>

/\* The *event types* are as follows: \*/

#define EV\_ARRIVE 0 /\* this is when passenger arrives

at the airport \*/

#define EV\_AIRLINEQ 1 /\* this is when she selects an airline

and gets in line to check bags\*/

#define EV\_AIRLINE 2 /\* this is when she gets to the agent

to check bags \*/

#define EV\_IDQ 3 /\* this is when she enters the line

to check her ID \*/

#define EV\_ID 4 /\* this is when she gets her ID check \*/

#define EV\_SCANQ 5 /\* this is when she gets in line for the scan \*/

#define EV\_SCAN 6 /\* this is when she gets to the scanner \*/

#define EV\_TRAINQ 7 /\* this is when she is finished being

scanned and heads for the train \*/

#define EV\_TRAIN 8 /\* this is when she gets off the train

and heads for her gate \*/

#define EV\_GATE 9 /\* this is when she arrives at the gate \*/

typedef struct passenger\_s passenger\_t;

struct passenger\_s

{

double arrival\_time; /\* gets to the airport \*/

double airlineQ\_time;

double airline\_time;

double idQ\_time;

double id\_time;

double scanQ\_time;

double scan\_time;

double trainQ\_time;

double train\_time;

double gate\_time; /\* gets to gate \*/

};

#define MAX\_PASS 1000000

int num\_passengers 0;

============================================================

/\* main.c \*/

/\* initialize queues \*/

/\* malloc new EV\_ARRIVE event and passenger \*/

event\_t \*start\_ev;

start\_ev = event\_create();

start\_ev.passenger = (passenger\_t \*)malloc(sizeof(passenger\_t));

/\* schedule EV\_ARRIVE event at t=0 and put in event queue \*/

start\_ev.event\_time = 0.0;

event\_schedule(start\_ev);

/\* run main loop \*/

while(!event\_empty(eq))

{

new\_ev = event\_cause();

time\_set(new\_ev.event\_time);

switch (new\_ev.event\_type)

{

case (EV\_ARRIVE) :

new\_ev.passenger.arrive\_time = time\_get();

/\* create EV\_ENQUEUE event for this passenger \*/

/\* schedule EV\_ENQUEUE event \*/

if (MAX\_PASS > num\_passengers++)

{

/\* create new EV\_ARRIVE event and passsenger \*/

/\* schedule EV\_ARRIVE event \*/

}

break;

case (EV\_AIRLINEQ) :

break;

case (EV\_AIRLINE) :

break;

case (EV\_IDQ) :

break;

/\* remaining event types \*/

default :

/\* error \*/

break;

}

/\* free event \*/

}

/\* Print overall stats \*/

Generic diagram of the project – this will need to be filled in a bit with Project 2.

