ECE 223: Airport Simulation II
Due Date in Canvas

Airport Simulation Program

In the second stage of this project we are going to focus on setting up all of the simulation queues, and programming all of the simulation events. There are also a few tweeks to state 1's code to clean a few things up. There will be a sequence of queues. Each queue has a simulated server (does not actually appear in the code). There are two events that will simulated the processing from one queue to another. It is important to remember that when a passenger is in a queue, there is not an event for that passenger. She must wait until all of the passengers before her are serviced before she exists the queue. There will be an event for the previous passenger to get her to the head of the queue, and an event to time how long she spends with the server. Special care must be taken when a passenger is the first into an empty queue to make sure the right events happen to begin the server. An even with 0.0 time can be used sometimes to cause code to run, but only in these circumstances.

PROJECT 2

For project 2 you are modify the original code as described (and shown in the code below). Then you are to implement the remaining queues and events, as well as statistics as needed.

- Modify event_cause() to set the global time (time_set()) to the newly retrieved event's event_time. Remove the code from the top of main()'s while loop that sets the global time to the event_time.
- Update event_init with a size argument if you don't already have it. Pass the size to priority_init, adding a size argument to it if needed. Use the size to malloc the heap array to the given size and put the pointer to it in the priority_t.
- Move all time functions from randsim.c to time.c and add all remaining time functions show. Randsim.c should export any functions needed by time.c

You should test your code to make sure that the simulation works then you are to adjust the mean times to produce a reasonably steady state flow.

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 */
```

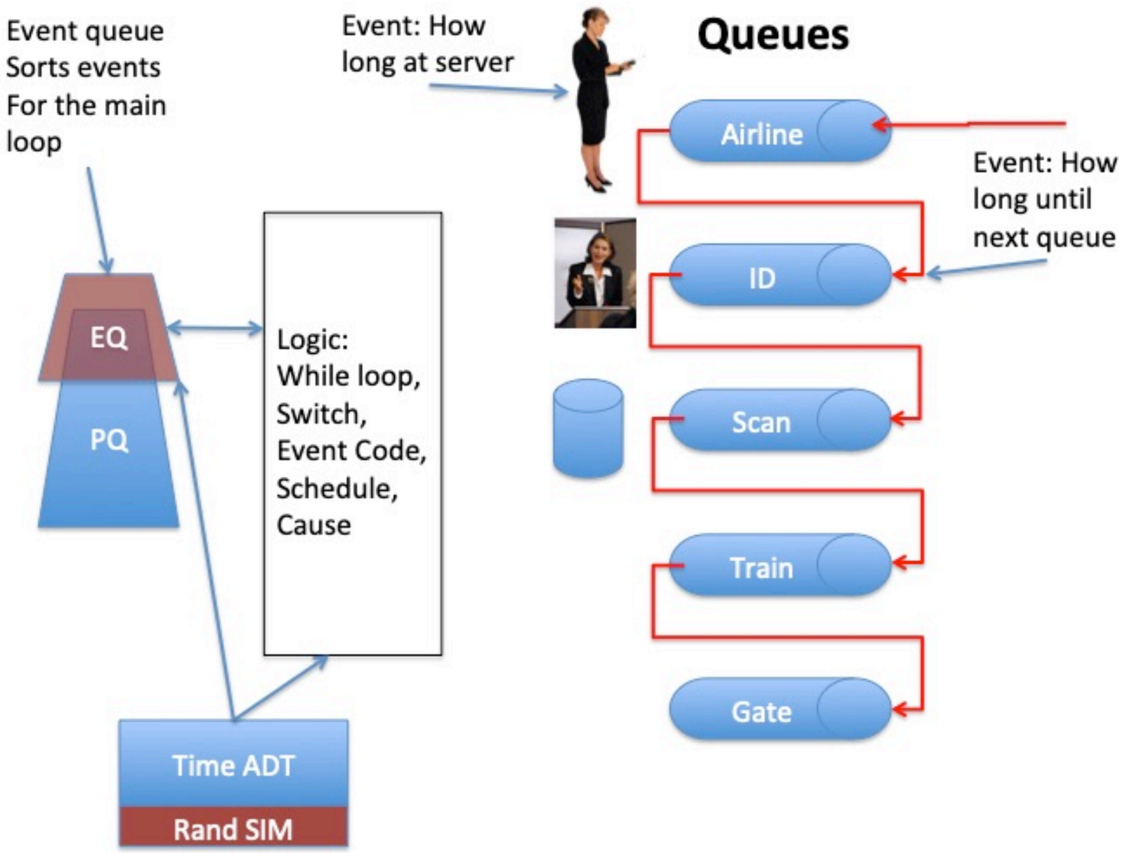
```
randsim.c  /* provided to you – calls to get */
randsim.h  /* times using random variables */
```

Below is the starting template and interface for various modules that may have changed since Phase I. A complete set of these are provided on canvas in C file form. Items in bold may have changed since the original specification and you should check and make sure you implement the changes.

```
============================================================
/* 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
   including the size of the queue */
void event_init(int size);

/* 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);


int event_empty();

/* remove the next event from the priority queue, set the global
time to the event’s time, and then return it to the caller for
processing */
event_t *event_cause();

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

============================================================
============================================================
============================================================
/* main.c */
#define MAX_PASS 1000000
#define MAX_SCAN 4
#define QSZ 100
int num_passengers 0; /* counts the number of passengers */

queue_t airlineQ;
queue_t idQ;
queue_t scanQ[MAX_SCAN];
queue_t trainQ;

/* initialize modules */
event_init(QSZ);
time_init();

/* initialize queues */
airlineQ = queue_init(QSZ);
idQ = queue_init(QSZ);
for (i = 0; i < MAX_SCAN; i++)
    scanQ[i] = queue_init(QSZ);
trainQ = queue_init(100);

event_t *start_ev;
start_ev = event_create();
start_ev.passenger = (passenger_t *)malloc(sizeof(passenger_t));
start_ev.event_time = 0.0;
start_ev.event_type = EV_ARRIVE;
event_schedule(start_ev);
/* run main loop */
while(!event_empty(eq))
{
    event_t new_ev;
    new_ev = event_cause();
    switch (new_ev.event_type)
    {
    case (EV_ARRIVE) :
    {
        event_t airline_ev;
        airline_ev = event_create();
        airline_ev.passenger = new_ev.passenger;
        airline_ev.passenger.arrive_time = time_get();
        airline_ev.event_time = time_airlineQ();
        airline_ev.event_type = EV_AIRLINEQ;
        event_schedule(airline_ev);
        if (MAX_PASS > num_passengers++)
        {
            event_t arrive_ev;
            arrive_ev = event_create();
            arrive_evjkk.passenger =
                    (passenger_t *)malloc(sizeof(passenger_t));
            arrive_ev.event_time = time_arrive();
            arrive_ev.event_type = EV_ARRIVE;
            event_schedule(arrive_ev);
        }
    }
        break;
    case (EV_AIRLINEQ) :
        break;
    case (EV_AIRLINE) :
        break;
    case (EV_IDQ) :
        break;
    case (EV_ID) :
        break;
    case (EV_SCANQ) :
        break;
    case (EV_SCAN) :
        break;
    case (EV_TRAINQ) :
        break;
    case (EV_TRAIN) :
        break;
    case (EV_GATE) :
        break;

    default :
        /* error */
        break;
    }
    /* free event */
    event_destroy(new_ev);
}
/* Print overall stats */
time_fini();
event_fini();
```

Generic diagram of the project – this will need to be filled in

Queues



Event queue
Sorts events
For the main
loop

Event: How
long at server

Event: How
long until
next queue

Logic:
While loop,
Switch,
Event Code,
Schedule,
Cause