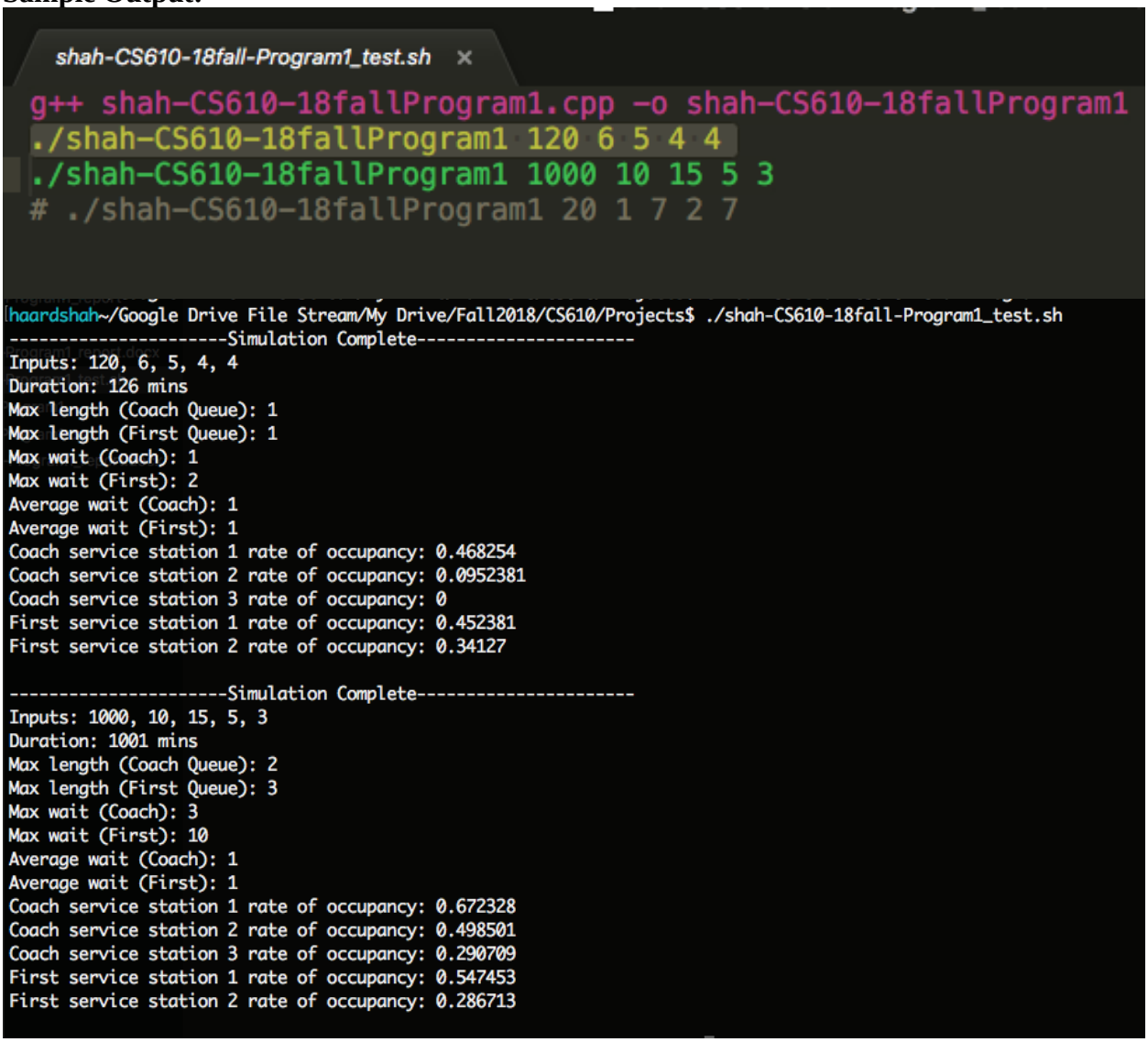
Haard Shah
CS610851
Programming Assignment 1

Programming Assignment 1 Report

Files included:

shah-CS610-18fallProgram1.cpp - source code (also pasted below)
shah-CS610-18fall-Program1_test.sh - shell script builds and runs the above script with sample inputs provided by TA (also contains the build instructions for any linux/unix system)
shah-CS610-18fallProgram1 - executable

Sample Output:



Haard Shah
CS610851
Programming Assignment 1

Source Code:

```
#include <iostream>
#include <random>

// #define DEBUG_PRINT
// #define PROGRESS_PRINT
#define PRINT_STATISTICS

#define COACH false
#define FIRST true
#define COACH_SS 3          //service station
#define FIRST_SS 2

using namespace std;

int g_id = 0;
int g_time = 0;

class Passenger {
public:
        int starttime;    // time of enqueue
        int wait;                 // time of dequeue - starttime - initial_serviceTime
        bool isFirstClass;
        int serviceTime;
        int initial_serviceTime; // to be used to calculate wait
        Passenger *next;
        int PassengerId;
        Passenger(int serviceTime, bool isFirstClass){
                this->next = NULL;
                wait = 0;
                this->serviceTime = serviceTime;
                this->initial_serviceTime = serviceTime;
                this->isFirstClass = isFirstClass;
                PassengerId = g_id++;
                this->starttime = g_time;
        }

        ~Passenger() {
                // this->wait = g_time - this->starttime - this->initial_serviceTime + 1;
                #ifdef DEBUG_PRINT
                cout << "Summary\t\t"; printSummary();
                #endif
        }
```

```
        void decrementServiceTime() {
                this->serviceTime--;
        }
        bool isServiced(){
                return (this->serviceTime==0);
        }
        void printSummary() {
                cout << PassengerId << "\t";
                if (isFirstClass) {
                        cout << "First\t";
                } else {
                        cout << "Coach\t";
                }
                cout << this->serviceTime << "\t" << this->wait << endl;
        }
};
int g_i = 0;
class Queue {
        Passenger *head; // for dequeue
        Passenger *tail; // for enqueue
        int arrivalRate;
        int serviceRate;
        int size;
        bool isFirstClass;
        std::default_random_engine generator;
        std::bernoulli_distribution arrival_rand;
        std::uniform_int_distribution<int> service_rand;
        int max;
        int maxWait;
        int totalPassengers; // to calculate averageWait
        int totalWait; // to calculate averageWait
public:
        Queue(int arrival_r, int service_r, bool isFirstClass) {
                head = tail = NULL;
                arrivalRate = arrival_r;
                serviceRate = service_r;
                max = size = totalPassengers = 0;
                maxWait = 0;
                totalWait = 0;
                // unsigned prev_seeds[2] = {3292402367, 3292402430};
                // generator.seed(prev_seeds[g_i++]);
                unsigned s = std::chrono::system_clock::now().time_since_epoch().count();
                generator.seed(s);
                #ifdef DEBUG_PRINT
                cout << isFirstClass << "\t" << s << endl;
```

```
		#endif
		// random yes with probability 1/arrivalRate and no with probability 1-yes
		arrival_rand = bernoulli_distribution((double)1.0/arrivalRate); // not sure if this is
the right distribution. If it is, would this be the p?
		// produce range of numbers from [1,serviceRate*2]
		// uniform_int_distribution<int> service_rand(1, serviceRate*2);
		service_rand = uniform_int_distribution<int>(1, serviceRate*2);
		this->isFirstClass = isFirstClass;
	}
	bool isEmpty() {
		if (head == NULL)
			return 1;
		else
			return 0;
	}
	void enqueue_passenger() {
		if (arrival_rand(generator)) {
			// append passenger to queue
			size++; totalPassengers++;
			if (size > max) max = size;
			int rand_service_time = service_rand(generator);
			Passenger *temp = new Passenger(rand_service_time, isFirstClass);
			if (head == NULL && tail == NULL) {
				head = temp;
				tail = temp;
			}
			else {
				tail->next = temp;
				tail = temp;
			}
			#ifdef DEBUG_PRINT
			if (head == NULL) {
				cout << "HOWWW" << endl;
			}
			cout << "Enqueue:\t"; tail->printSummary();
			#endif
		}
	}
	Passenger *dequeue_passenger(){
		Passenger *dequeued = head;
		head = head->next;
		// dequeued->next = NULL; // should this be done?
		size--;
		if (!size) head = tail = NULL;
```

```
                //statistics
                dequeued->wait = g_time - dequeued->starttime + 1;
                totalWait += dequeued->wait;
                if (dequeued->wait > maxWait) maxWait = dequeued->wait;

                #ifdef DEBUG_PRINT
                cout << "Dequeue:\t"; dequeued->printSummary();
                #endif
                return dequeued;
        }
        int getMaxSize() {
                return max;
        }
        int getMaxWait() {
                return maxWait;
        }
        int getAverageWait() {
                return ((double)totalWait/totalPassengers);
        }
        int getSize(){
                return size;
        }

        void printQueue() {
                if (isFirstClass)
                        cout << "First_Queue:\t";
                else
                        cout << "Coach_Queue:\t";
                Passenger *temp = head;
                while (temp != NULL){
                        cout << temp->PassengerId << "  ";
                        temp = temp->next;
                }
                cout << endl;
        }
};

int g_stationId = 0;

class ServiceStation {
        Passenger *current;
        int id;
        int totalOccupancy;
public:
        ServiceStation(){
```

```
                current = NULL;
                id = g_stationId++;
                totalOccupancy = 0;
        }

        bool isOccupied() {
                if (current) return true;
                else return false;
        }

        int getPassengerId() { return current->PassengerId; }

        void service() {
                current->decrementServiceTime();
                #ifdef DEBUG_PRINT
                cout << id << "Service:\t"; current->printSummary();
                #endif
                if (current->isServiced()) {
                        #ifdef DEBUG_PRINT
                        cout << id << "Serviced:\t"; current->printSummary();
                        #endif
                        delete current;
                        current = NULL;
                }
        }

        void incrementTotalOccpancy() {totalOccupancy++;}
        int getTotalOccupancy() {return totalOccupancy; }

        void takePassenger(Passenger *passenger) { current = passenger; }
};

void printServiceStations(ServiceStation *coachclass_stations, ServiceStation
*firstclass_stations) {
        cout << "SS:\t C\t C\t C\t F\t F\n\t";
        for (int i = 0; i < COACH_SS; i++) {
                if (coachclass_stations[i].isOccupied())
                        cout << coachclass_stations[i].getPassengerId() << "\t";
                else
                        cout << "  \t";
        }
        for (int i = 0; i < FIRST_SS; i++) {
                if (firstclass_stations[i].isOccupied())
                        cout << firstclass_stations[i].getPassengerId() << "\t";
                else
```

```
                        cout << "  \t";
        }
        cout << endl;
}

int g_checkinDuration;

bool onGoingSimulation(Queue *q1, Queue *q2, ServiceStation *coach, ServiceStation *first) {
        if (g_time > g_checkinDuration && q1->isEmpty() && q2->isEmpty()) {
                for (int i = 0; i < COACH_SS; i++) {
                        if (coach[i].isOccupied())
                                return true;
                }
                for (int i = 0; i < FIRST_SS; i++) {
                        if (first[i].isOccupied())
                                return true;
                }
                return false; // done
        }
        return true;
}

int main(int argc, char *argv[]) {
        /* read arguments
                0: executable name
                1: checkin_duration
                2: coach_arrivalRate
                3: coach_serviceRate
                4: firstclass_arrivalRate
                5: firstclass_serviceRate
                */
        if (argc != 6) {
                printf("Usage: %s <checkin_duration> <coach_arrivalRate>
<coach_serviceRate> <firstclass_arrivalRate> <firstclass_arrivalRate>
<firstclass_serviceRate>\n\n", argv[0]);
                exit(0);
        }

        g_checkinDuration = atoi(argv[1]);
        int coach_arrivalRate = atoi(argv[2]);
        int coach_serviceRate = atoi(argv[3]);
        int firstclass_arrivalRate = atoi(argv[4]);
        int firstclass_serviceRate = atoi(argv[5]);

        g_time = 1;     // beginning
```

```
	// create queues
	Queue *coachclass_queue = new Queue(coach_arrivalRate, coach_serviceRate,
COACH);
	Queue *firstclass_queue = new Queue(firstclass_arrivalRate, firstclass_serviceRate,
FIRST);

	// create service stations
	// 3 coachclasses
	ServiceStation coachclass_stations [COACH_SS];
	// 2 firstclasses
	ServiceStation firstclass_stations [FIRST_SS];

	while(onGoingSimulation(coachclass_queue, firstclass_queue, coachclass_stations,
firstclass_stations)){
		#ifdef PROGRESS_PRINT
		cout << "Time: " << g_time << endl;
		#endif
		if (g_time <= g_checkinDuration) {
			// customers arriving (add passengers to queue)
			coachclass_queue->enqueue_passenger();
			firstclass_queue->enqueue_passenger();
		}

		// service coach passengers
		for (int i = 0; i < COACH_SS; i++) {
			if (coachclass_stations[i].isOccupied()) {
				coachclass_stations[i].service();
				coachclass_stations[i].incrementTotalOccpancy();
			}
			else {
				if (!coachclass_queue->isEmpty()) {
					// coachclass_stations[i].assignPassenger()
					coachclass_stations[i].takePassenger(coachclass_queue-
>dequeue_passenger());
					coachclass_stations[i].service();
					coachclass_stations[i].incrementTotalOccpancy();
				}
			}
		}

		// service first class
		for (int i = 0; i < FIRST_SS; i++) {
			if (firstclass_stations[i].isOccupied()) {
				firstclass_stations[i].service();
```

```
                        firstclass_stations[i].incrementTotalOccpancy();
                    }
                    else {
                        if (!firstclass_queue->isEmpty()) {
                            firstclass_stations[i].takePassenger(firstclass_queue-
>dequeue_passenger());
                            firstclass_stations[i].service();
                            firstclass_stations[i].incrementTotalOccpancy();
                        }
                        // if nobody is firstclass queue then take passenger from coach
queue
                        else if(!coachclass_queue->isEmpty()) {
                            firstclass_stations[i].takePassenger(coachclass_queue-
>dequeue_passenger());
                            firstclass_stations[i].service();
                            firstclass_stations[i].incrementTotalOccpancy();
                        }
                    }
                }

                #ifdef PROGRESS_PRINT
                    coachclass_queue->printQueue();
                    firstclass_queue->printQueue();
                    printServiceStations(coachclass_stations, firstclass_stations);
                #endif
                g_time++;
        }

        // calculate rate of occupancy for service stations
        double coach_ss_occupancyRate[COACH_SS];
        double first_ss_occupancyRate[FIRST_SS];
        for (int i = 0; i < COACH_SS; i++) {
            coach_ss_occupancyRate[i] =  (double)
coachclass_stations[i].getTotalOccupancy() / g_time;
        }
        for (int i = 0; i < FIRST_SS; i++) {
            first_ss_occupancyRate[i] = (double) firstclass_stations[i].getTotalOccupancy() /
g_time;
        }

#ifdef PRINT_STATISTICS
        cout << "----------------------Simulation Complete----------------------" << endl;
        printf("Inputs: %s, %s, %s, %s, %s\n", argv[1], argv[2], argv[3], argv[4], argv[5]);
        cout << "Duration: " << g_time << " mins" << endl;
        cout << "Max length (Coach Queue): " << coachclass_queue->getMaxSize() << endl;
```

```
	cout << "Max length (First Queue): " << firstclass_queue->getMaxSize() << endl;
	cout << "Max wait (Coach): " << coachclass_queue->getMaxWait() << endl;
	cout << "Max wait (First): " << firstclass_queue->getMaxWait() << endl;
	cout << "Average wait (Coach): " << coachclass_queue->getAverageWait() << endl;
	cout << "Average wait (First): " << firstclass_queue->getAverageWait() << endl;
	for (int i = 0; i < COACH_SS; i++)
		cout << "Coach service station "<< i+1 << " rate of occupancy: " <<
coach_ss_occupancyRate[i] << endl;
	for (int i = 0; i < FIRST_SS; i++)
		cout << "First service station "<< i+1 <<" rate of occupancy: " <<
first_ss_occupancyRate[i] << endl;
#endif
	printf("\n");
	return 0;
}
```