SAGARIKA M CHAVAN

PES2UG19CS347

SEC L

PROBLEM STATEMENT: You need to simulate the working of an airport.

Let us consider a small but busy airport with two runways, one always used for landings and the other always used for takeoffs. Planes arrive ready to land or ready to take off at random times, so at any given unit of time, the runways may be idle or a plane may be landing or taking off and there may be several planes waiting either to land or take off. We therefore need two queues, called landing and takeoff, to hold these planes. It is better to keep a plane waiting on the ground than in the air, so a small airport allows a plane to take off only if there are no planes waiting to land. Hence, after receiving requests from new planes to land or take off, our simulation will first service the head of the queue of planes waiting to land, and only if the landing queue is empty, will it allow a plane to take off. The simulation should run through many units of time

//HEADER FILE

//flight.h

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<ctype.h>
#include<math.h>
#include<time.h>
#include<limits.h>
#define MAX 3
#define ARRIVE 0
#define DEPART 1
//typedef enum action{ARRIVE,DEPART}Action;
struct plane
{
int id;  /*identification number of airplane*/
```

```
int tm;   /*Time arrival in queue*/
};
struct queue
{
int count;
int front;
int rear;
struct plane p[MAX];
};
void initqueue(struct queue *);
void addqueue(struct queue *,struct plane);
struct plane delqueue(struct queue *);
int size(struct queue);
int empty(struct queue);
int full(struct queue);
struct airport
{
        struct queue landing;
        struct queue takeoff;
        struct queue *pl;
        struct queue *pt;
        int idletime;   /*Number of units when runway is idle*/
        int landwait;   /*Total waiting time for planes landed*/
        int takeoffwait; /*Total waiting time for take off*/
        int nland;      /*Number of planes landed*/
        int nplanes;    /*Number of planes processed so far*/
        int nrefuse;    /*Number of planes refused of airport*/
        int ntakeoff;   /*Number of phones taken off*/
        struct plane pln;

};
```

```
void initairport(struct airport *);
void start(int *,double *,double *);
void newplane(struct airport *,int,int);
void refuse(struct airport *,int);
void land(struct airport *,struct plane,int);
void fly(struct airport *,struct plane,int);
void idle(struct airport *,int);
void conclude(struct airport *,int);
int randomnumber(double);
void apaddqueue(struct airport *,char);
struct plane apdelqueue(struct airport *,char);
int apsize(struct airport,char);
int apfull(struct airport,char);
int apempty(struct airport,char);
void myrandomsize();
```

//c file with all functions

//flight1.c

```
#include<stdio.h>
#include "flight.h"
void initqueue(struct queue *pq)
{
pq->count=0;
pq->front=0;
pq->rear=-1;
}
void addqueue(struct queue *pq,struct plane item)
{
```

```
if(pq->count>=MAX)
{
printf("\nQueue is full.\n");
return;
}
(pq->count)++;
pq->rear=(pq->rear+1)%MAX;
pq->p[pq->rear]=item;
}
struct plane delqueue(struct queue *pq)
{
struct plane pl;
if(pq->count<=0)
{
printf("\nQueue is empty.\n");
pl.id=0;
pl.tm=0;
}
else
{
(pq->count)--;
pl=pq->p[pq->front];
pq->front=(pq->front+1)%MAX;
}
return pl;
}
int size(struct queue q)
{
        return q.count;
}
int empty(struct queue q)
```

```
{
        return (q.count<=0);
}
int full(struct queue q)
{
        return(q.count>=MAX);
}
void initairport(struct airport *ap)
{
        initqueue(&(ap->landing));
        initqueue(&(ap->takeoff));
        ap->pl=&(ap->landing);
        ap->pt=&(ap->takeoff);
        ap->nplanes=ap->nland=ap->ntakeoff=ap->nrefuse=0;
        ap->landwait=ap->takeoffwait=ap->idletime=0;
}
void start(int *endtime,double *expectarrive,double *expectdepart)
{
        int flag=0;
        char wish;
        printf("\nProgram that simulates an airport with two runways. \n");
   /*Instruct user*/
   printf("One plane can land or depart in each unit of time.\n");
   printf("Up to %d planes can be waiting to land or take off at any time.\n",MAX);
   printf("How many units of time will the simulation run?");
   /*Input parameter*/
   scanf("%d",endtime);
   myrandomize();
   do
   {
        printf("\nExpected number of arrivals per unit time"
```

```
                "(real number)?");
        /*Error checking*/
        scanf("%lf",expectarrive);
        printf("\nExpected number of departures per unit time?");
        scanf("%lf",expectdepart);
        if(*expectarrive<0.0||*expectdepart<0.0)
        {
                printf("These numbers must be nonnegative.\n");
                flag=0;
                }
                else
                        {
                                if(*expectarrive+*expectdepart>1.0)   //this program runs properly
when (expectarrive+expectdepart)>1
                                {
                                        printf("The airport will become saturated. Read new
numbers?");
                                        fflush(stdin);
                                        scanf("%c",&wish);
                                        if(tolower(wish)=='y')
                                           flag=0;
                                        else
                                          flag=1;
                                }
                                else
                                  flag=1;

                }
        }while(flag==0);
}
void newplane(struct airport *ap,int curtime,int action)
        /*Newplane:make a new recored for a plane.update nplanes*/
```

```
{  //printf("hi");
        (ap->nplanes)++;
        ap->pln.id=ap->nplanes;
        ap->pln.tm=curtime;
        switch(action)
        {
                case ARRIVE:
                        printf("\n");
                        printf("Plane %d ready to land in Runway 1.\n ",ap->nplanes);
                        break;
                case DEPART:
                        printf("\n");
                        printf("Plane %d ready to takeoff in Runway 2.\n ",ap->nplanes);
                        break;
   }
}
void refuse(struct airport *ap,int action)
/*Rfuses processes the plane when queue is full*/
{
        switch(action)
        {
          case ARRIVE:
                        printf("\tPlane %d directed to another airport.\n ",ap->pln.id);
                        break;
          case DEPART:
                        printf("\tPlane %d told to try later.\n ",ap->pln.id);
                        break;
        }
(ap->nrefuse)++;
}
void land(struct airport *ap,struct plane pl,int curtime)
```

```
/*Land :process a plane p that is actually lading in runway 1*/
{
        int wait;
        wait=curtime-pl.tm;
        printf("%d: plane %d landed in Runway 1 ",curtime,pl.id);
        printf("in queue %d units \n",wait);
        (ap->nland++);
        (ap->landwait)+=wait;
}
void fly(struct airport *ap,struct plane pl,int curtime)
/*Fly:process a plane p that is actually landing*/
{
        int wait;
        wait=curtime-pl.tm;
        printf("%d: Plane %d took off",curtime,pl.id);
        printf("in queue %d units \n",wait);
        (ap->ntakeoff)++;
        (ap->takeoffwait)+=wait;
}
void idle(struct airport *ap,int curtime)
/*idle:updates variables for idle runway*/
{
        printf("%d: Runway 1 and 2 is idle.\n",curtime);
        ap->idletime++;
        //printf("idle time : %d",ap->idletime);
}
void conclude(struct airport *ap,int endtime)
/*Conclude:write out statistics and conclude simulation*/
{
        printf("\tSimulation has concluded after %d units.\n",endtime);
        printf("\tTotal number of planes processed:%d\n",ap->nplanes);
```

```
        printf("\tNumber of planes landed in Runway 1: %d\n",ap->nland);
        printf("\tNumber of planes taken off in Runway 2: %d\n",ap->ntakeoff);
        printf("\tNumber of planes refused use:%d\n",ap->nrefuse);
        printf("\tNumber left ready to land in Runway 1: %d\n",apsize(*ap,'l'));
        printf("\tNumber left ready to take off in Runway 2: %d\n",apsize(*ap,'t'));
   if(endtime>0)
     printf("\tPercentage of time 2 runways idle: %lf\n",((double)ap->idletime/endtime)*100.0);
   if(ap->nland>0)
     printf("\tAverage wait time to land in Runway 1: %lf\n",((double)ap->landwait/ap->nland));
    if(ap->ntakeoff>0)
     printf("\tAverage wait time to takeoff in Runway 2: %lf\n",((double)ap->takeoffwait/ap-
>ntakeoff));
}
int randomnumber(double expectedvalue)
/*Randomnumber:generates a pseudorandom integer according to the Poisson distribution*/
{  //printf("x");
        int n=0;  /*Counter of iterations*/
        double em;
        double x;   /*Psuedorandom number*/
        em=exp(-expectedvalue);
        x=rand()/(double)INT_MAX;
        //printf("x: %f / em: %f |",x,em);
        while(x>em)
        {
                n++;
                x*=rand()/(double)INT_MAX;
        }
        //printf("# n:%d #",n);
        return n;
}
void apaddqueue(struct airport *ap,char type)
```

```
{
        switch(tolower(type))
        {
          case 'l':
            addqueue(ap->pl,ap->pln);
            break;
           case 't':
            addqueue(ap->pt,ap->pln);
            break;
        }
}
struct plane apdelqueue(struct airport *ap,char type)
{
        struct plane pl;
        switch(tolower(type))
        {
                case 'l':
                  pl=delqueue(ap->pl);
                  break;
                case 't':
                        pl=delqueue(ap->pl);
                        break;
        }
        return pl;
}
int apsize(struct airport ap,char type)
{
        switch(tolower(type))
        {
                case 'l':
                        return(size(*(ap.pl)));
```

```
                case 't':
                    return (size(*(ap.pt)));
        }
        return 0;
}
int apfull(struct airport ap,char type)
{
        switch(tolower(type))
        {
                case 'l':
                        return(full(*(ap.pl)));
                case 't':
                    return(full(*(ap.pt)));
        }
        return 0;
}
int apempty(struct airport ap,char type)
{
        switch(tolower(type))
        {
                case 'l':
                        return(empty(*(ap.pl)));
                case 't':
                        return(empty(*(ap.pt)));
        }
        return 0;
}
void myrandomize()
/*Sets starting point for pseudorandam numbers*/
{
        srand((unsigned int)(time(NULL)%10000));
```

```
}
```

//c file with the main function in it

//mains.c

```
#include<stdio.h>
#include "flight.h"
void main()
{
	struct airport a;
	int i;	/*increment*/
	int pri;  /*Pseudo random integer*/
	int curtime; /*Current time:one unit=time taken for take off in runway 1 and landing in runway 2*/
	int endtime;  /*Total number of time units to run*/
	double expectarrive; /*Number of planes arriving in one unit in runway 1*/
	double expectdepart; /*Number of planes newly ready to take off from runway 2*/
	struct plane temp;
	initairport(&a);
	start(&endtime,&expectarrive,&expectdepart);
	printf("Endtime: %d\n",endtime);//not necessary just for checking
	idle(&a,curtime);
	for(curtime=1;curtime<=endtime;curtime++)
	{  //printf("%d",expectarrive);
		pri=(rand() % endtime);
		//printf("%d",pri);
		for(i=1;i<=pri;i++)  /*Add to landing queue*/
		{  //printf("hi");
			newplane(&a,curtime,ARRIVE);
			if(apfull(a,'l'))
```

```
                refuse(&a,ARRIVE);
            else
                apaddqueue(&a,'l');
        }
        pri=(rand() % endtime);
        for(i=1;i<=pri;i++)  /*Add to takeoff queue*/
        {
            newplane(&a,curtime,DEPART);
            if(apfull(a,'t'))
                refuse(&a,DEPART);
            else
                apaddqueue(&a,'t');
        }
        if(!(apempty(a,'l')))  /*Bring plane to land*/
        {
            temp=apdelqueue(&a,'l');
            land(&a,temp,curtime);
        }
        else
        {
            if(!(apempty(a,'t')))/*Allow plane to take off*/
            {
                temp=apdelqueue(&a,'t');
                fly(&a,temp,curtime);
            }
            else
                idle(&a,curtime);   /*Runway idle*/
        }
    }
conclude(&a,endtime);  /*Finish simulation*/
getch();
```

}

//makefile to simplify or to organize code for compilation

//makefile.txt

```
all: simulate

stimulate: flight1.o mains.o
    gcc -Wall -o flight1.o mains.o

main.o: mains.c
    gcc -c -Wall mains.c

misc.o:flight1.c
    gcc -c -Wall flight1.c

clean:
     rm mains.o
     rm flight1.o
```