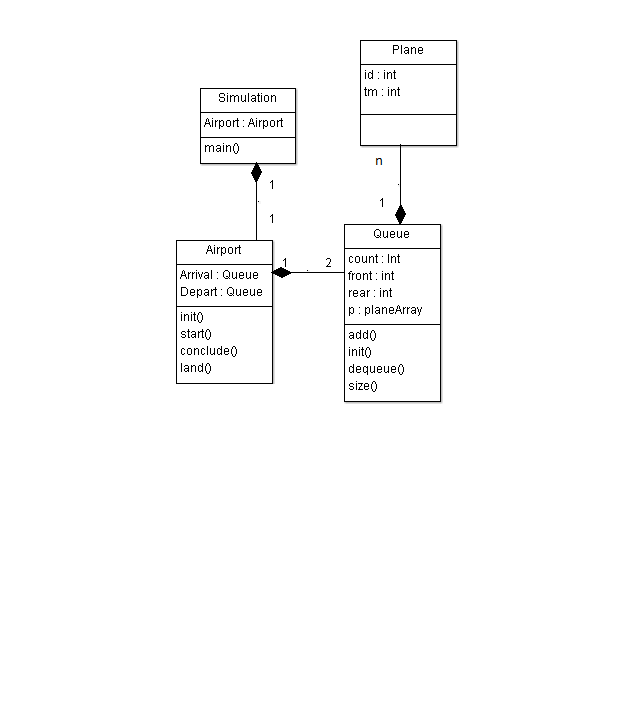
Project: Airport

Author: Anonymous

Code Lines: 341

Files: Main.c //It was really hard to separate in different files without changing its //functioning

UML

Main.c

/\* Airport simulation \*/

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <ctype.h>

#include <math.h>

#include <time.h>

#include <limits.h>

#include <windows.h>

#define MAX 3

#define ARRIVE 0

#define DEPART 1

struct plane

{

int id ;

int tm ;

} ;

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 ) ;

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 p1 ;

if ( pq -> count <= 0 )

{

printf ( "\nQueue is empty.\n" ) ;

p1.id = 0 ;

p1.tm = 0 ;

}

else

{

( pq -> count )-- ;

p1 = pq -> p[pq -> front] ;

pq -> front = ( pq -> front + 1 ) % MAX ;

}

return p1 ;

}

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 ) ;

}

struct airport

{

struct queue landing ;

struct queue takeoff ;

struct queue \*pl ;

struct queue \*pt ;

int idletime ;

int landwait, takeoffwait ;

int nland, nplanes, nrefuse, ntakeoff ;

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 myrandomize ( ) ;

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 only one runway.\n" ) ;

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?" ) ;

scanf ( "%d", endtime ) ;

myrandomize( ) ;

do

{

printf ( "\nExpected number of arrivals per unit time? " ) ;

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 )

{

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 )

{

( ap -> nplanes )++ ;

ap -> pln.id = ap -> nplanes ;

ap -> pln.tm = curtime ;

switch ( action )

{

case ARRIVE :

printf ( "\n" ) ;

printf ( "Plane %d ready to land.\n", ap -> nplanes ) ;

break ;

case DEPART :

printf ( "\nPlane %d ready to take off.\n", ap -> nplanes ) ;

break ;

}

}

void refuse ( struct airport \*ap, int action )

{

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 )

{

int wait ;

wait = curtime - pl.tm ;

printf ( "%d: Plane %d landed ", 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 )

{

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 )

{

printf ( "%d: Runway is idle.\n", curtime ) ;

ap -> idletime++ ;

}

void conclude ( struct airport \*ap, int endtime )

{

printf ( "\tSimulation has concluded after %d units.\n", endtime ) ;

printf ( "\tTotal number of planes processed: %d\n", ap -> nplanes ) ;

printf ( "\tNumber of planes landed: %d\n", ap -> nland ) ;

printf ( "\tNumber of planes taken off: %d\n", ap -> ntakeoff ) ;

printf ( "\tNumber of planes refused use: %d\n", ap -> nrefuse ) ;

printf ( "\tNumber left ready to land: %d\n", apsize ( \*ap, 'l' ) ) ;

printf ( "\tNumber left ready to take off: %d\n", apsize ( \*ap, 't' ) ) ;

if ( endtime > 0 )

printf ( "\tPercentage of time runway idle: %lf \n", ( ( double ) ap -> idletime / endtime ) \* 100.0 ) ;

if ( ap -> nland > 0 )

printf ( "\tAverage wait time to land: %lf \n", ( ( double ) ap -> landwait / ap -> nland ) ) ;

if ( ap -> ntakeoff > 0 )

printf ( "\tAverage wait time to take off: %lf \n", ( ( double ) ap -> takeoffwait / ap -> ntakeoff ) ) ;

}

int randomnumber ( double expectedvalue )

{

int n = 0 ;

double em ;

double x ;

em = exp ( -expectedvalue ) ;

x = rand( ) / ( double ) INT\_MAX ;

while ( x > em )

{

n++ ;

x \*= rand( ) / ( double ) INT\_MAX ;

}

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 p1 ;

switch ( tolower ( type ) )

{

case 'l' :

p1 = delqueue ( ap -> pl ) ;

break ;

case 't' :

p1 = delqueue ( ap -> pl ) ;

break ;

}

return p1 ;

}

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( )

{

srand ( ( unsigned int ) ( time ( NULL ) % 10000 ) ) ;

}

int main( )

{

struct airport a ;

int i, pri, curtime, endtime ;

double expectarrive, expectdepart ;

struct plane temp ;

system ( "cls" ) ;

initairport ( &a );

start ( &endtime, &expectarrive, &expectdepart ) ;

for ( curtime = 1 ; curtime <= endtime ; curtime++ )

{

pri = randomnumber ( expectarrive ) ;

for ( i = 1 ; i <= pri ; i++ )

{

newplane ( &a, curtime, ARRIVE ) ;

if ( apfull ( a, 'l' ) )

refuse ( &a, ARRIVE ) ;

else

apaddqueue( &a, 'l' ) ;

}

pri = randomnumber ( expectdepart ) ;

for ( i = 1 ; i <= pri ; i++ )

{

newplane ( &a, curtime, DEPART ) ;

if ( apfull ( a, 't' ) )

refuse ( &a, DEPART ) ;

else

apaddqueue ( &a, 't' ) ;

}

if ( ! ( apempty ( a, 'l' ) ) )

{

temp = apdelqueue ( &a, 'l' ) ;

land ( &a, temp, curtime ) ;

}

else

{

if ( ! ( apempty ( a, 't' ) ) )

{

temp = apdelqueue ( &a, 't' ) ;

fly ( &a, temp, curtime ) ;

}

else

idle ( &a, curtime ) ;

}

}

conclude ( &a, endtime ) ;

return 0 ;

}