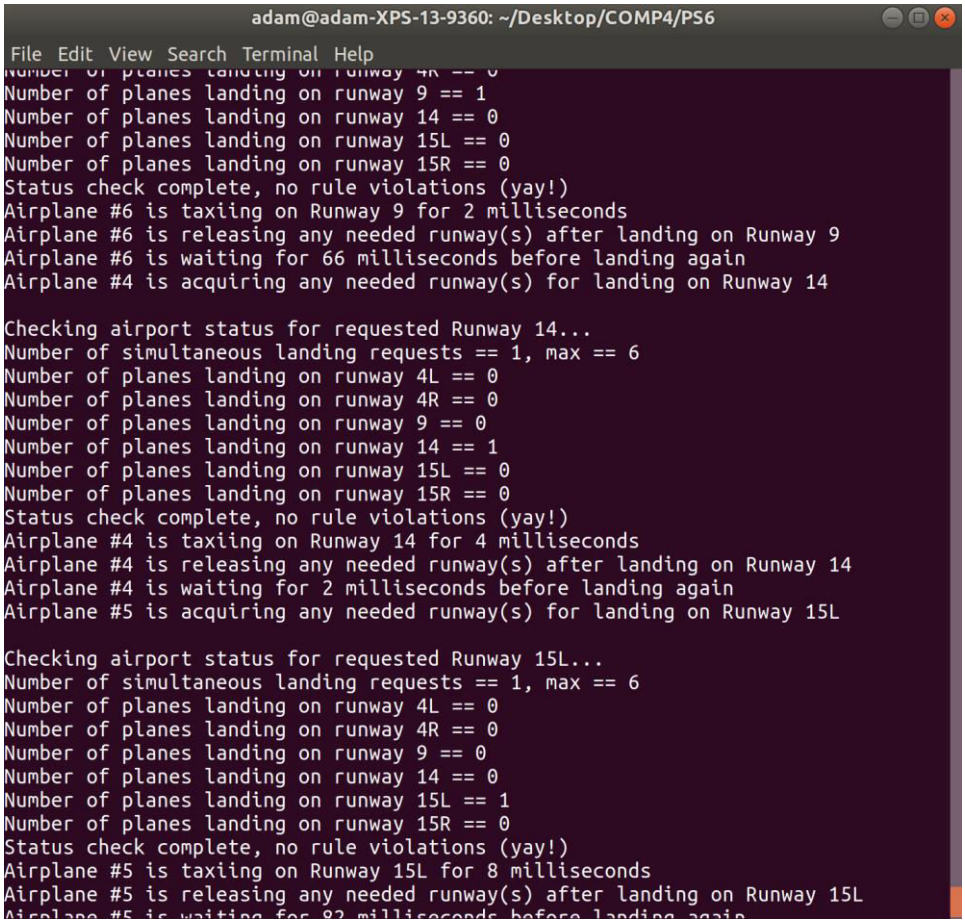
PS6 Airport Simulation (C++ Concurrency)

In this assignment, I had to simulate landings of multiple airplanes on multiple runways at Logan Airport. Use thread and mutex to run multiple threads of code at the same time, which simulated multiple landing lanes landing at the same time. OO design was mutex, and using switch statements to lock and unlock different runways when they were in use. In this assignment I implemented switch statement to check if lanes are being used by different planes.

I learned how to use thread and mutex libraries to run multiple different threads at the same time. One issue I had was that the first time I ran Airport_Sync the planes crashed after a few seconds, but after that it ran fine, for 15 min. I do not know what caused this, so if this happens when you test it for the first time, try again and it might fix itself. Another problem I had was that my terminal only saves 10000 lines so my output.txt only has 10000 output lines.



```
 1: CC = g++
 2: CFLAGS = -c -g -Og -std=c++11
 3: OBJ = Airplane.o Airport.o AirportRunways.o AirportServer.o
 4: DEPS =
 5: LIBS = -pthread
 6: EXE = Airport-Sync
 7:
 8: all: $(OBJ)
 9:         $(CC) $(OBJ) -o $(EXE) $(LIBS)
10:
11: %.o: %.cpp $(DEPS)
12:         $(CC) $(CFLAGS) -o $@ $<
13:
14: clean:
15:         rm -f $(OBJ) $(EXE)
```

```
    1: /**
    2: *  Airplane.h
    3: *  Definition of the Airplane class
    4: */
    5:
    6: #ifndef AIRPLANE_H
    7: #define AIRPLANE_H
    8:
    9: #include "AirportRunways.hpp"
   10: #include "AirportServer.hpp"
   11:
   12:
   13: class Airplane
   14: {
   15: public:
   16:
   17:         int airplaneNum;
   18:         AirportServer* apServ;
   19:
   20:         // Value constructor for the Airplane class
   21:         Airplane(int num, AirportServer* s)
   22:         {
   23:                 airplaneNum = num;
   24:                 apServ = s;
   25:         }
   26:
   27:
   28:         // Setter method for requestedRunway
   29:         void setRequestedRunway(AirportRunways::RunwayNumber runway)
   30:         {
   31:                 requestedRunway = runway;
   32:         }
   33:
   34:
   35:         // The run() function for Airplane threads in Airport will call this
 function
   36:         void land();
   37:
   38:
   39: private:
   40:
   41:         AirportRunways::RunwayNumber requestedRunway; // Picked at random
   42:
   43: }; // end class Airplane
   44:
   45: #endif
   46:
```

```
    1: #include <random>
    2: #include <thread>
    3: #include <chrono>
    4:
    5: #include "Airplane.hpp"
    6:
    7: // The run() function in Airport will call this function
    8: void Airplane::land()
    9: {
   10:         // obtain a seed from the system clock:
   11:         unsigned seed = std::chrono::system_clock::now().time_since_epoch().
count();
   12:
   13:         std::default_random_engine generator(seed);
   14:         std::uniform_int_distribution<int> runwayNumberDistribution(AirportR
unways::RUNWAY_4L, AirportRunways::RUNWAY_15R);
   15:
   16:         while (true)
   17:         {
   18:                 // Get ready to land
   19:                 requestedRunway = AirportRunways::RunwayNumber(runwayNumberD
istribution(generator));
   20:
   21:                 apServ->reserveRunway(airplaneNum, requestedRunway);
   22:
   23:                 // Landing complete
   24:                 apServ->releaseRunway(airplaneNum, requestedRunway);
   25:
   26:                 // Wait on the ground for a while (to prevent starvation of
other airplanes)
   27:                 std::this_thread::sleep_for(std::chrono::milliseconds(1000))
;
   28:
   29:         } // end while
   30:
   31: } // end Airplane::land
```

```
    1: /**
    2:  *  AirportServer.h
    3:  *  This class defines the methods called by the Airplanes
    4:  */
    5: #ifndef AIRPORT_SERVER_H
    6: #define AIRPORT_SERVER_H
    7:
    8: #include <mutex>
    9:
   10: #include <random>
   11:
   12: #include <condition_variable>
   13:
   14: #include "AirportRunways.hpp"
   15:
   16:
   17: class AirportServer {
   18:     public:
   19:
   20:         /**
   21:          * Default constructor for AirportServer class
   22:          */
   23:         AirportServer() {
   24:             // ***** Initialize any Locks and/or Condition Variables here as
 necessary *****
   25:             lck15L = std::unique_lock < std::mutex > (run15L);
   26:             lck15R = std::unique_lock < std::mutex > (run15R);
   27:             lck4L = std::unique_lock < std::mutex > (run4L);
   28:             lck4R = std::unique_lock < std::mutex > (run4R);
   29:             lck14 = std::unique_lock < std::mutex > (run14);
   30:             lck9 = std::unique_lock < std::mutex > (run9);
   31:
   32:         } // end AirportServer default constructor
   33:
   34:     /**
   35:      * Called by an Airplane when it wishes to land on a runway
   36:      */
   37:     void reserveRunway(int airplaneNum, AirportRunways::RunwayNumber runway)
;
   38:     /**
   39:      * Called by an Airplane when it is finished landing
   40:      */
   41:     void releaseRunway(int airplaneNum, AirportRunways::RunwayNumber runway)
;
   42:
   43:     private:
   44:
   45:         // Constants and Random number generator for use in Thread sleep cal
ls
   46:         static
   47:     const int MAX_TAXI_TIME = 10; // Maximum time the airplane will occupy t
he requested runway after landing, in milliseconds
   48:     static
   49:     const int MAX_WAIT_TIME = 100; // Maximum time between landings, in mill
iseconds
   50:
   51:     /**
   52:     AirportServer.h Tue Apr 23 19:36:55 2019 2
   53:     * Declarations of mutexes and condition variables
   54:     */
   55:     mutex runwaysMutex; // Used to enforce mutual exclusion for acquiring &
```

releasing runways

```
   56:     /**
   57:     * ***** Add declarations of your own Locks and Condition Variables
   58:      here *****
   59:     */
   60:     std::mutex run15L;
   61:     std::mutex run15R;
   62:     std::mutex run4L;
   63:     std::mutex run4R;
   64:     std::mutex run14;
   65:     std::mutex run9;
   66:
   67:     std::unique_lock < std::mutex > lck15L;
   68:     std::unique_lock < std::mutex > lck15R;
   69:     std::unique_lock < std::mutex > lck4L;
   70:     std::unique_lock < std::mutex > lck4R;
   71:     std::unique_lock < std::mutex > lck14;
   72:     std::unique_lock < std::mutex > lck9;
   73:
   74:     std::condition_variable cv;
   75:
   76: }; // end class AirportServer
   77:
   78: #endif
```

```
    1: #include <iostream>
    2: #include <thread>
    3: #include <condition_variable>
    4:
    5: #include "AirportServer.hpp"
    6:
    7:
    8: /**
    9:  * Called by an Airplane when it wishes to land on a runway
   10:  */
   11: void AirportServer::reserveRunway(int airplaneNum, AirportRunways::RunwayNum
ber runway) {
   12:     // Acquire runway(s)
   13:     { // Begin critical region
   14:
   15:         //unique_lock<mutex> runwaysLock(runwaysMutex);
   16:
   17:         {
   18:             unique_lock < mutex > lk(AirportRunways::checkMutex);
   19:             cv.wait(lk, [] {
   20:                 return !(AirportRunways::getNumLandingRequests() >= 6);
   21:             });
   22:
   23:             cout << "Airplane #" << airplaneNum << " is acquiring any needed
 runway(s) for landing on Runway " <<
   24:                 AirportRunways::runwayName(runway) << endl;
   25:             AirportRunways::incNumLandingRequests();
   26:
   27:         }
   28:
   29:         /**
   30:          * ***** Add your synchronization here! *****
   31:          */
   32:         switch (runway) {
   33:         case AirportRunways::RUNWAY_4L:
   34:             cv.wait(lck4L, [ = ] {
   35:                 bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4L] == 0);
   36:                 bool av15L = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15L] == 0);
   37:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
   38:                 if (av4L && av15L && av15R)
   39:                     return true;
   40:                 else
   41:                     return false;
   42:             });
   43:             cv.wait(lck15L, [ = ] {
   44:                 bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4L] == 0);
   45:                 bool av15L = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15L] == 0);
   46:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
   47:                 if (av4L && av15L && av15R)
   48:                     return true;
   49:                 else
   50:                     return false;
   51:             });
   52:             cv.wait(lck15R, [ = ] {
   53:                 bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
```

```
WAY_4L] == 0);
   54:                 bool av15L = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15L] == 0);
   55:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
   56:                 if (av4L && av15L && av15R)
   57:                     return true;
   58:                 else
   59:                     return false;
   60:             });
   61:             break;
   62:         case AirportRunways::RUNWAY_4R:
   63:             cv.wait(lck4R, [ = ] {
   64:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
   65:                 bool av15L = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15L] == 0);
   66:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
   67:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
   68:                 if (av4R && av15L && av15R && av9)
   69:                     return true;
   70:                 else
   71:                     return false;
   72:             });
   73:             cv.wait(lck15L, [ = ] {
   74:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
   75:                 bool av15L = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15L] == 0);
   76:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
   77:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
   78:                 if (av4R && av15L && av15R && av9)
   79:                     return true;
   80:                 else
   81:                     return false;
   82:             });
   83:             cv.wait(lck15R, [ = ] {
   84:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
   85:                 bool av15L = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15L] == 0);
   86:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
   87:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
   88:                 if (av4R && av15L && av15R && av9)
   89:                     return true;
   90:                 else
   91:                     return false;
   92:             });
   93:             cv.wait(lck9, [ = ] {
   94:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
   95:                 bool av15L = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15L] == 0);
   96:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
```

```
   97:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
   98:                 if (av4R && av15L && av15R && av9)
   99:                     return true;
  100:                 else
  101:                     return false;
  102:             });
  103:             break;
  104:         case AirportRunways::RUNWAY_15R:
  105:             cv.wait(lck4L, [ = ] {
  106:                 bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4L] == 0);
  107:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
  108:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
  109:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
  110:                 if (av4L && av4R && av15R && av9)
  111:                     return true;
  112:                 else
  113:                     return false;
  114:             });
  115:             cv.wait(lck4R, [ = ] {
  116:                 bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4L] == 0);
  117:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
  118:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
  119:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
  120:                 if (av4L && av4R && av15R && av9)
  121:                     return true;
  122:                 else
  123:                     return false;
  124:             });
  125:             cv.wait(lck15R, [ = ] {
  126:                 bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4L] == 0);
  127:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
  128:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
  129:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
  130:                 if (av4L && av4R && av15R && av9) return true;
  131:                 else return false;
  132:             });
  133:             cv.wait(lck9, [ = ] {
  134:                 bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4L] == 0);
  135:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
  136:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
  137:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
  138:                 if (av4L && av4R && av15R && av9)
  139:                     return true;
  140:                 else
```

```
  141:                     return false;
  142:             });
  143:             break;
  144:         case AirportRunways::RUNWAY_15L:
  145:             cv.wait(lck4L, [ = ] {
  146:                 bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4L] == 0);
  147:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
  148:                 bool av15L = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15L] == 0);
  149:                 if (av4L && av4R && av15L)
  150:                     return true;
  151:                 else
  152:                     return false;
  153:             });
  154:             cv.wait(lck4R, [ = ] {
  155:                 bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4L] == 0);
  156:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
  157:                 bool av15L = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15L] == 0);
  158:                 if (av4L && av4R && av15L)
  159:                     return true;
  160:                 else
  161:                     return false;
  162:             });
  163:             cv.wait(lck15L, [ = ] {
  164:                 bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4L] == 0);
  165:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
  166:                 bool av15L = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15L] == 0);
  167:                 if (av4L && av4R && av15L)
  168:                     return true;
  169:                 else
  170:                     return false;
  171:             });
  172:             break;
  173:         case AirportRunways::RUNWAY_9:
  174:             cv.wait(lck4R, [ = ] {
  175:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
  176:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
  177:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
  178:                 if (av4R && av15R && av9)
  179:                     return true;
  180:                 else
  181:                     return false;
  182:             });
  183:             cv.wait(lck15R, [ = ] {
  184:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
  185:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
  186:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
```

```
  187:                 if (av4R && av15R && av9)
  188:                     return true;
  189:                 else
  190:                     return false;
  191:             });
  192:             cv.wait(lck9, [ = ] {
  193:                 bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4R] == 0);
  194:                 bool av15R = (AirportRunways::runwayInUse[AirportRunways::RU
NWAY_15R] == 0);
  195:                 bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNW
AY_9] == 0);
  196:                 if (av4R && av15R && av9)
  197:                     return true;
  198:                 else
  199:                     return false;
  200:             });
  201:             break;
  202:         case AirportRunways::RUNWAY_14:
  203:             cv.wait(lck4L, [ = ] {
  204:                 bool av14 = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_14] == 0);
  205:                 if (av14)
  206:                     return true;
  207:                 else
  208:                     return false;
  209:             });
  210:             break;
  211:         }
  212:         // Check status of the airport for any rule violations
  213:         AirportRunways::checkAirportStatus(runway);
  214:
  215:     } // End critical region
  216:
  217:     // obtain a seed from the system clock:
  218:     unsigned seed = std::chrono::system_clock::now().time_since_epoch().coun
t();
  219:     std::default_random_engine generator(seed);
  220:
  221:     // Taxi for a random number of milliseconds
  222:     std::uniform_int_distribution < int > taxiTimeDistribution(1, MAX_TAXI_T
IME);
  223:     int taxiTime = taxiTimeDistribution(generator);
  224:
  225:     {
  226:         lock_guard < mutex > lk(AirportRunways::checkMutex);
  227:
  228:         cout << "Airplane #" << airplaneNum << " is taxiing on Runway " << A
irportRunways::runwayName(runway) << " for " << taxiTime << " milliseconds\n";
  229:     }
  230:
  231:     std::this_thread::sleep_for(std::chrono::milliseconds(taxiTime));
  232:
  233: } // end AirportServer::reserveRunway()
  234:
  235: /**
  236:  * Called by an Airplane when it is finished landing
  237:  */
  238: void AirportServer::releaseRunway(int airplaneNum, AirportRunways::RunwayNum
ber runway) {
  239:     // Release the landing runway and any other needed runways
```

```
  240:     { // Begin critical region
  241:         lock_guard < mutex > lk(AirportRunways::checkMutex);
  242:         cout << "Airplane #" << airplaneNum << " is releasing any needed run
way(s) after landing on Runway " << AirportRunways::runwayName(runway) << endl;
  243:
  244:         /**
  245:          * ***** Add your synchronization here! *****
  246:          */
  247:         switch (runway) {
  248:         case AirportRunways::RUNWAY_4L:
  249:             run4L.unlock();
  250:             run15L.unlock();
  251:             run15R.unlock();
  252:             break;
  253:         case AirportRunways::RUNWAY_4R:
  254:             run4R.unlock();
  255:             run15L.unlock();
  256:             run15R.unlock();
  257:             run9.unlock();
  258:             break;
  259:         case AirportRunways::RUNWAY_15R:
  260:             run4L.unlock();
  261:             run4R.unlock();
  262:             run15R.unlock();
  263:             run9.unlock();
  264:             break;
  265:         case AirportRunways::RUNWAY_15L:
  266:             run4L.unlock();
  267:             run4R.unlock();
  268:             run15L.unlock();
  269:             break;
  270:         case AirportRunways::RUNWAY_9:
  271:             run4R.unlock();
  272:             run15R.unlock();
  273:             run9.unlock();
  274:             break;
  275:         case AirportRunways::RUNWAY_14:
  276:             run14.unlock();
  277:             break;
  278:         }
  279:         AirportRunways::decNumLandingRequests();
  280:         cv.notify_one();
  281:         // Update the status of the airport to indicate that the landing is
comp
  282:         AirportRunways::finishedWithRunway(runway);
  283:         //runwaysLock.unlock();
  284:     } // End critical region
  285:
  286:     // obtain a seed from the system clock:
  287:     unsigned seed = std::chrono::system_clock::now().time_since_epoch().coun
t();
  288:     std::default_random_engine generator(seed);
  289:     // Wait for a random number of milliseconds before requesting the next l
anding for this Airplane
  290:     std::uniform_int_distribution < int > waitTimeDistribution(1, MAX_WAIT_T
IME);
  291:     int waitTime = waitTimeDistribution(generator); {
  292:         lock_guard < mutex > lk(AirportRunways::checkMutex);
  293:         cout << "Airplane #" << airplaneNum << " is waiting for " << waitTim
e << " milliseconds before landing again\n";
  294:     }
```

```
  295:     std::this_thread::sleep_for(std::chrono::milliseconds(waitTime));
  296: } // end AirportServer::releaseRunway()
```

```
    1: /**
    2: *  Class AirportRunways provides definitions of constants and helper methods
 for the Airport simulation.
    3: */
    4:
    5: #ifndef AIRPORT_RUNWAYS_H
    6: #define AIRPORT_RUNWAYS_H
    7:
    8: #include <iostream>
    9: #include <string>
   10: #include <mutex>
   11:
   12: using namespace std;
   13:
   14:
   15: class AirportRunways
   16: {
   17: public:
   18:
   19:         static const int NUM_RUNWAYS = 6;     // Number of runways in this s
imulation
   20:         static const int NUM_AIRPLANES = 7;   // Number of airplanes in this
 simulation
   21:         static const int MAX_LANDING_REQUESTS = 6; // Maximum number of simu
ltaneous landing requests that Air Traffic Control can handle
   22:
   23:         enum RunwayNumber { RUNWAY_4L, RUNWAY_4R, RUNWAY_9, RUNWAY_14, RUNWA
Y_15L, RUNWAY_15R };
   24:
   25:         static mutex checkMutex; // enforce mutual exclusion on checkAirport
Status
   26:
   27:         static string runwayName(RunwayNumber rn);
   28:
   29:         /**
   30:         *  Check the status of the aiport with respect to any violation of t
he rules.
   31:         */
   32:         static void checkAirportStatus(RunwayNumber requestedRunway);
   33:
   34:         /**
   35:         *  requestRunway() and finishedWithRunway() are helper methods for k
eeping track of the airport status
   36:         */
   37:
   38:         static void requestRunway(RunwayNumber rn)
   39:         {
   40:                 runwayInUse[rn]++;
   41:
   42:         } // end useRunway()
   43:
   44:
   45:         static void finishedWithRunway(RunwayNumber rn)
   46:         {
   47:                 runwayInUse[rn]--;
   48:
   49:         } // end finishedWithRunway()
   50:
   51:
   52:         static int getNumLandingRequests()
   53:         {
```

```
   54:                     return numLandingRequests;
   55:             }
   56:
   57:
   58:             static void incNumLandingRequests()
   59:             {
   60:                     numLandingRequests++;
   61:                     if (numLandingRequests > maxNumLandingRequests)
   62:                             maxNumLandingRequests = numLandingRequests;
   63:             }
   64:
   65:
   66:             static void decNumLandingRequests()
   67:             {
   68:                     numLandingRequests--;
   69:             }
   70:
   71:             static int runwayInUse[NUM_RUNWAYS]; // Keeps track of how many airp
lanes are attempting to land on a given runway
   72:
   73:             static int numLandingRequests; // Keeps track of the number of simul
taneous landing requests
   74:
   75:             static int maxNumLandingRequests; // Keeps track of the max number o
f simultaneous landing requests
   76:
   77:
   78: private:
   79:
   80:             /**
   81:              *  The following variables and methods are used to detect violation
s of the rules of this simulation.
   82:              */
   83:
   84: }; // end class AirportRunways
   85:
   86: #endif
   87:
```

```
    1: #include "AirportRunways.hpp"
    2:
    3: int AirportRunways::runwayInUse[AirportRunways::NUM_RUNWAYS];
    4:
    5: int AirportRunways::numLandingRequests = 0;
    6:
    7: int AirportRunways::maxNumLandingRequests = 0;
    8:
    9: mutex AirportRunways::checkMutex;
   10:
   11:
   12: string AirportRunways::runwayName(RunwayNumber rn)
   13: {
   14:         switch (rn)
   15:         {
   16:         case RUNWAY_4L:
   17:                 return "4L";
   18:         case RUNWAY_4R:
   19:                 return "4R";
   20:         case RUNWAY_9:
   21:                 return "9";
   22:         case RUNWAY_14:
   23:                 return "14";
   24:         case RUNWAY_15L:
   25:                 return "15L";
   26:         case RUNWAY_15R:
   27:                 return "15R";
   28:         default:
   29:                 return "Unknown runway " + rn;
   30:         } // end switch
   31:
   32: } // end AirportRunways::runwayName()
   33:
   34:
   35:  /**
   36:   *  Check the status of the aiport with respect to any violation of the rul
es.
   37:   */
   38: void AirportRunways::checkAirportStatus(RunwayNumber requestedRunway)
   39: {
   40:         lock_guard<mutex> checkLock(checkMutex);
   41:
   42:         bool crash = false; // Set to true if any rule is violated
   43:
   44:         cout << "\nChecking airport status for requested Runway " << runwayN
ame(requestedRunway) << "..." << endl;
   45:
   46:         requestRunway(requestedRunway);
   47:
   48:         // Check the number of landing requests
   49:         cout << "Number of simultaneous landing requests == " << numLandingR
equests
   50:                  << ", max == " << maxNumLandingRequests << endl;
   51:
   52:         if (numLandingRequests > MAX_LANDING_REQUESTS)
   53:         {
   54:                 cout << "***** The number of simultaneous landing requests e
xceeds Air Traffic Control limit of " << MAX_LANDING_REQUESTS << "!\n";
   55:                 crash = true;
   56:         }
   57:
```

```
   58:         // Check the occupancy of each runway
   59:         for (int i = RUNWAY_4L; i <= RUNWAY_15R; i++)
   60:         {
   61:                 cout << "Number of planes landing on runway " << runwayName(
RunwayNumber(i)) << " == " << runwayInUse[i] << endl;
   62:
   63:                 if (runwayInUse[i] > 1)
   64:                 {
   65:                         cout << "***** The number of planes landing on runwa
y " << runwayName(RunwayNumber(i)) << " is greater than 1!\n";
   66:                         crash = true;
   67:                 }
   68:         }
   69:
   70:         // Check individual restrictions on each runway
   71:         if ((runwayInUse[RUNWAY_9] > 0)
   72:                 && ((runwayInUse[RUNWAY_4R] > 0) || (runwayInUse[RUNWAY_15R]
 > 0)))
   73:         {
   74:                 cout << "***** Runways 9, 4R, and/or 15R may not be used sim
ultaneously!\n";
   75:                 crash = true;
   76:         }
   77:
   78:         if (((runwayInUse[RUNWAY_15L] > 0) || (runwayInUse[RUNWAY_15R] > 0))
   79:                 && ((runwayInUse[RUNWAY_4L] > 0) || (runwayInUse[RUNWAY_4R]
> 0)))
   80:         {
   81:                 cout << "***** Runways 15L or 15R may not be used simultaneo
usly with Runways 4L or 4R!\n";
   82:                 crash = true;
   83:         }
   84:
   85:         // If any of the rules have been violated, terminate the simulation
   86:         if (crash)
   87:         {
   88:                 cout << "***** CRASH! One or more rules have been violated.
Due to the crash, the airport is closed!\n";
   89:                 exit(-1); // Abnormal program termination
   90:         }
   91:
   92:         // Status check is normal
   93:         cout << "Status check complete, no rule violations (yay!)\n";
   94:
   95: } // end AirportRunways::checkAirportStatus()
```

```
    1: /**
    2: *  Airport driver program
    3: */
    4:
    5: #include <iostream>
    6: #include <thread>
    7: #include <vector>
    8:
    9: #include "AirportServer.hpp"
   10: #include "AirportRunways.hpp"
   11: #include "Airplane.hpp"
   12:
   13: using namespace std;
   14:
   15:
   16: void run(Airplane* ap)
   17: {
   18:         ap->land();
   19:
   20: } // end run
   21:
   22:
   23: int main(void)
   24: {
   25:         AirportServer as;
   26:
   27:         vector<thread> apths; // Airplane threads
   28:
   29:                                                 // Create and launch the i
ndividual Airplane threads
   30:         for (int i = 1; i <= AirportRunways::NUM_AIRPLANES; i++)
   31:         {
   32:                 Airplane* ap = new Airplane(i, &as);
   33:
   34:                 apths.push_back(thread([] (Airplane* ap){
   35:                         ap->land();
   36:                 }, ap));
   37:         }
   38:
   39:         // Wait for all Airplane threads to terminate (shouldn't happen!)
   40:         for (auto& th : apths)
   41:         {
   42:                 th.join();
   43:         }
   44:
   45:         return 0;
   46:
   47: } // end main
```