```
    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()
```