```
1  // A simple trivial Discrete Event Simulator to illustrate DES concepts
2  // This one uses typesafe callbacks for the event handlers.
3
4  // George F. Riley, Georgia Tech, Fall 2011 ECE8893
5
6  #include <math.h> // For random numbers
7  #include <sys/time.h>
8  #include <iostream>
9  #include <vector>
10 #include <map>
11 #include <set>
12 using namespace std;
13
14 // Keep trace of time with a double-precision float
15 typedef double Time_t;
16
17 // Create an "EventBase" class that is an abstract base class.
18 // Actual events are templated subclasses.
19 // This might be subclassed  if the various event types need additional
20 // data
21 class EventBase
22 {
23 public:
24   EventBase(Time_t t) : time(t) {}
25 public:
26   Time_t time;                    // Timestamp for the vent
27   virtual void CallHandler() = 0; // All subclasses must implement this
28 };
29
30 // Define four event subclasses, templated, such that the events
31 // can have 0, 1, 2, or 3 members that become arguments to the
32 // event handler callback
33
34 template<typename T, typename OBJ>
35 class Event0 : public EventBase
36 { // Event class with no arguments
37   // Type T is the object type for the event handler object
38   // Type OBJ is the actual event handler object
39 public:
40   Event0(double t, void (T::*f)(void), OBJ* obj0)
41     : EventBase(t), handler(f), obj(obj0){}
42   void (T::*handler)(void);
43   OBJ*      obj;
44 public:
45   void CallHandler();
46 };
47
48 template <typename T, typename OBJ>
49 void Event0<T, OBJ>::CallHandler()
50 {
51   (obj->*handler)();
52 }
53
54 // Event with one parameter on the callback function
55 template<typename T, typename OBJ, typename U1, typename T1>
56 class Event1 : public EventBase
```

Program des-simple3.cc

```
{
public:
  Event1(double t, void (T::*f)(U1), OBJ* obj0, T1 t1_0)
    : EventBase(t), handler(f), obj(obj0), t1(t1_0){}
  void (T::*handler)(U1);
  OBJ*      obj;
  T1        t1;
public:
  void CallHandler();
};

template <typename T, typename OBJ, typename U1, typename T1>
void Event1<T, OBJ, U1, T1>::CallHandler()
{
  (obj->*handler)(t1);
}

template<typename T, typename OBJ,
         typename U1, typename T1,
         typename U2, typename T2>
class Event2 : public EventBase
{
public:
  Event2(double t, void (T::*f)(U1, U2), OBJ* obj0, T1 t1_0, T2 t2_0)
    : EventBase(t), handler(f), obj(obj0), t1(t1_0), t2(t2_0) {}
  void (T::*handler)(U1, U2);
  OBJ*      obj;
  T1        t1;
  T2        t2;
public:
  void CallHandler();
};

template <typename T, typename OBJ,
          typename U1, typename T1,
          typename U2, typename T2>
void Event2<T, OBJ, U1, T1, U2, T2>::CallHandler()
{
  (obj->*handler)(t1, t2);
}

template <typename T,  typename OBJ,
          typename U1, typename T1,
          typename U2, typename T2,
          typename U3, typename T3>
class Event3 : public EventBase {
public:
   Event3(double t, void (T::*f)(U1, U2, U3), OBJ *obj0, T1 t1_0, T2 t2_0, T3 t3_0)
     : EventBase(t), handler(f), obj(obj0), t1(t1_0), t2(t2_0), t3(t3_0) {}
   void (T::*handler)(U1, U2, U3);
   OBJ* obj;
   T1 t1;
   T2 t2;
   T3 t3;

public:
```

Program des-simple3.cc (continued)

```
113     void CallHandler();
114 };
115 
116 template <typename T,  typename OBJ,
117           typename U1, typename T1,
118           typename U2, typename T2,
119           typename U3, typename T3>
120 void Event3<T,OBJ,U1,T1,U2,T2,U3,T3>::CallHandler() {
121      (obj->*handler)(t1,t2,t3);
122 }
123 
124 
125 // Define a class modeling an  "Airport" that is an event handler.
126 // In this example, we have three different event handlers in
127 // class airport. (1) AircraftLanded, (2) AircraftAtGate,
128 // (3) Aircraft Departed.
129 // time
130 
131 class Airport
132 {
133 public:
134   Airport(int id0) : id(id0), totalArrivals(0), totalOnGround(0),
135                      totalPassengers(0) {};
136   void SetPeer(Airport* n);  // Give the node a pointer to its peer
137   // Event Handlers.  Use different arguments for each for illustratino
138   void AircraftLanded(int aircraftType);
139   void AircraftAtGate(int aircraftType, int numberPassengers);
140   void AircraftDeparted(int aircraftType, int numberPassengers,
141                         double fuelLoad);
142 public:
143   int   id;
144   int   totalArrivals;
145   int   totalOnGround;
146   int   totalPassengers;
147   vector<Airport*> peers;
148 };
149 
150 // Now define the sorted set of events and the event comparator
151 class event_less
152 {
153 public:
154   event_less() { }
155   inline bool operator()(EventBase* const & l, const EventBase* const & r) const {
156     if(l->time < r->time) return true;
157     return false;
158   }
159 };
160 
161 // Define the type for the sorted event list
162 typedef std::multiset<EventBase*, event_less> EventSet_t;
163 
164 
165 // Define the Simulator class that maintains the event list
166 // and runs the simulation.  Note the simulator object is also
167 // a handler, as it needs to process the "Stop" event.
168 class Simulator
```

Program des-simple3.cc (continued)

```
169 {
170 public:
171   Simulator();
172   void TimeToStop();               // Stop time has been reached
173
174   // Define the templated schedule functions
175
176   // This one has zero arguments on the handler callback
177   template <typename T, typename OBJ>
178     static void Schedule(double t, void(T::*handler)(void), OBJ* obj)
179   {
180     EventBase* ev = new Event0<T, OBJ>(t + Simulator::Now(), handler, obj);
181     events.insert(ev);
182   }
183
184   template <typename T, typename OBJ,
185     typename U1, typename T1>
186     static void Schedule(double t, void(T::*handler)(U1), OBJ* obj, T1 t1)
187   {
188     EventBase* ev = new Event1<T, OBJ, U1, T1>(t + Simulator::Now(), handler, obj, t1);
189     events.insert(ev);
190   }
191
192   template <typename T, typename OBJ,
193     typename U1, typename T1,
194     typename U2, typename T2>
195     static void Schedule(double t, void(T::*handler)(U1, U2), OBJ* obj, T1 t1, T2 t2)
196   {
197     EventBase* ev = new Event2<T, OBJ, U1, T1, U2, T2>(t + Simulator::Now(), handler, obj, t
198     events.insert(ev);
199   }
200
201   template <typename T, typename OBJ,
202     typename U1, typename T1,
203     typename U2, typename T2,
204     typename U3, typename T3>
205     static void Schedule(double t, void(T::*handler)(U1, U2, U3), OBJ* obj, T1 t1, T2 t2, T3
206   {
207     EventBase* ev = new Event3<T, OBJ, U1, T1, U2, T2, U3, T3>(t + Simulator::Now(), handler
208     events.insert(ev);
209   }
210
211   static void Run(); // Run the simulation until the stop time is reached
212   static void StopAt(Time_t t);     // Stop the simulation at time "t"
213   static Time_t Now();              // Return current simulation tiome
214   static Simulator* instance;       // Points to singleton simulator object
215 private:
216   static EventSet_t  events;
217   static bool        stopped;
218   static Time_t      now;
219   };
220
221 // Implementations for Simulator object
222
223 Simulator::Simulator()
224 {
```

Program des-simple3.cc (continued)

```
225   instance = this;
226 }
227
228 void Simulator::TimeToStop()
229 {
230   stopped = true;
231 }
232
233 void Simulator::Run()
234 {
235   while (!stopped && !events.empty())
236     {
237       // Get the next event
238       EventBase* currentEvent = *events.begin();
239       // Remove from queue
240       events.erase(events.begin());
241       // Advance Simulation Time
242       now = currentEvent->time;  // Advance simulation time to time of event
243       // call the event
244       currentEvent->CallHandler();
245     }
246 }
247
248 void Simulator::StopAt(Time_t t)
249 {
250   Simulator::Schedule(t - Simulator::Now(),
251                       &Simulator::TimeToStop, Simulator::instance);
252 }
253
254 Time_t Simulator::Now()
255 {
256   return now;
257 }
258
259 // Simulator static objects
260 EventSet_t  Simulator::events;
261 bool        Simulator::stopped = false;
262 Time_t      Simulator::now = 0;
263 Simulator*  Simulator::instance = 0;
264
265 // Implementations for the Airport object
266 // First the handlers
267 void Airport::AircraftLanded(int aircraftType)
268 { // Here the event is an arrival
269   cout << "Airport " << id << " got arrival event at time " << Simulator::Now()
270        << " aircraft type " << aircraftType
271        << endl;
272   totalArrivals++;
273   totalOnGround++;
274   // Choose a random time to taxi to gate, 0 .. 10 mins
275   double timeToTaxi = drand48() * 10.0 / 60.0;
276   int    nPassengers = 100 + (int)(drand48() * 100);
277   Simulator::Schedule(timeToTaxi, &Airport::AircraftAtGate,
278                       this, aircraftType, nPassengers);
279 }
280
```

Program des-simple3.cc (continued)

```
281  void Airport::AircraftAtGate(int aircraftType, int numberPassengers)
282  {
283    cout << "Aircraft arrived at gate time " << Simulator::Now()
284         << " type " << aircraftType
285         << " passengers " << numberPassengers
286         << endl;
287    totalPassengers += numberPassengers;
288    // Choose random time for time at gate.  30 mins plus a random factor
289    Time_t timeAtGate = 0.5 + drand48() * 1.0;
290    // Also choose random fuel load (pounds)
291    double fuelLoad = 1000 + drand48() * 1000.0;
292    // Departs with different passenger count than arrival
293    int nPass = 100 + (int)(drand48() * 100);
294    // Schedule the departure event
295    Simulator::Schedule(timeAtGate, &Airport::AircraftDeparted,
296                        this, aircraftType, nPass,
297                        fuelLoad);
298  }
299
300  void Airport::AircraftDeparted(int aircraftType,
301                                 int numberPassengers,
302                                 double fuelLoad)
303  {
304    cout << "Airport " << id
305         << " got departure event at time " << Simulator::Now()
306         << " type " << aircraftType
307         << " nPass " << numberPassengers
308         << " with fuel " << fuelLoad << " pounds"
309         << endl;
310    // Schedule arrival at another airport
311    Time_t flightTime = drand48() * 10;
312    int    peer = drand48() * peers.size();
313    // The event will be handled "flightTime" in the future by the
314    // randomly selected peer.
315    Simulator::Schedule(flightTime,
316                        &Airport::AircraftLanded,
317                        peers[peer],
318                        aircraftType);
319    // And decrement "on ground" count
320    totalOnGround--;
321  }
322
323  void Airport::SetPeer(Airport* p)
324  {
325    peers.push_back(p);
326  }
327
328  int main()
329  {
330    // We need an object of class Simulator to handle simulator events
331    Simulator simulator;
332    // Seed the random number generator to get different random values
333    // on each run
334    struct timeval tv;
335    gettimeofday(&tv, 0);
336    srand48(tv.tv_usec);
```

Program des-simple3.cc (continued)

```
337
338   // First create the airports
339   vector<Airport*> airports;
340   for (int i = 0; i < 10; ++i)
341     {
342       Airport* ap = new Airport(i);
343       airports.push_back(ap);
344     }
345   // Now set connecting airports (randomly)
346   for (unsigned i = 0; i < airports.size(); ++i)
347     {
348       for (unsigned j = 0; j < airports.size(); ++j)
349         {
350           double rand = drand48();
351           if (rand < 0.5)
352             { // set peer with 50% probability
353               airports[i]->SetPeer(airports[j]);
354             }
355         }
356     }
357   // now schedule some  arrivals at each airport
358   for (unsigned i = 0; i < airports.size(); ++i)
359     {
360       for (unsigned j = 0; j  < 15; ++j)
361         {
362           double rand = drand48();
363           if (rand < 0.3)
364             { // Schedule an arrival with 30% probability
365               int aircraftType = (int)(drand48() * 10);
366               Simulator::Schedule(drand48() * 8,
367                                   &Airport::AircraftLanded,
368                                   airports[i],
369                                   aircraftType);
370             }
371         }
372     }
373   // Set the stop time
374   Simulator::StopAt(500.0); // Model 500 hours of activity
375   Simulator::Run();         // Run the simulation
376   cout << "Simulation Complete" << endl;
377   // Print some arrival statistics for each airport
378   for (unsigned i = 0; i < airports.size(); ++i)
379     {
380       cout << "Airport " << i
381            << " totalArrivals " << airports[i]->totalArrivals
382            << " totalPassengers " << airports[i]->totalPassengers
383            << " total on ground " << airports[i]->totalOnGround
384            << endl;
385     }
386 }
387
388
389
390
391
392
```

Program des-simple3.cc (continued)

Program des-simple3.cc (continued)