```
void FCFS(){
   int departureCount = 0;
   while( departureCount < batchSize ){
      // CASE 1: cpu is not busy ------------------
      if( cpuHead->cpuBusy == false ){
         scheduleArrival();
         if( rHead != 0 ){
         scheduleAllocation();
         }
        }
      // CASE 2: cpu is busy ----------------------
      else scheduleDeparture();

      // ANY CASE: handle next event --------------
      if( eHead->type == 1 ) handleArrival();
      else if( eHead->type == 2 ){
         handleDeparture();
         departureCount++;
      }
      else if( eHead->type == 3 ) handleAllocation();
   } // end while
}
void scheduleArrival(){
   // add a process to the process list
   process* pIt = pHead;
   while( pIt->pNext !=0 ){
      pIt = pIt->pNext;
   }
   pIt->pNext = new process;
   pIt->pNext->arrivalTime = pIt->arrivalTime + genExp((float)lambda);
   pIt->pNext->startTime = 0.0;
   pIt->pNext->reStartTime = 0.0;
   pIt->pNext->finishTime = 0.0;
   pIt->pNext->serviceTime = genExp(mu);
   pIt->pNext->remainingTime = pIt->pNext->serviceTime;
   pIt->pNext->pNext = 0;

   // create a corresponding arrival event
   eventQueue* nuArrival = new eventQueue;
   nuArrival->time = pIt->pNext->arrivalTime;
   nuArrival->type = 1;
   nuArrival->pLink = pIt->pNext;
   nuArrival->eNext = 0;

   // insert into eventQ in asc time order
   insertIntoEventQ(nuArrival);
}
void scheduleAllocation(){
   // create a new event queue node
   eventQueue* nuAllocation = new eventQueue;

   // identify the next process to be allocated to the cpu:
   process* nextProc;
   if( schedulerType == 1 ) nextProc = rHead->pLink;          // FCFS
   else if( schedulerType == 2 ){                  // SRTF
      if( cpuHead->clock > rHead->pLink->arrivalTime ){
         nextProc = getSRTFProcess();
      }
      else{
         nextProc = rHead->pLink;
      }
   }
   else if( schedulerType == 3 ){                             // HRRN
      nextProc = getHRRNProcess();
   }

```

```
    // set the time of the allocation event
    if( cpuHead->clock < nextProc->arrivalTime ){
       nuAllocation->time = nextProc->arrivalTime;
    }
    else{
       nuAllocation->time = cpuHead->clock;
    }

    // set the values for type, next, and pLink
    nuAllocation->type = 3;
    nuAllocation->eNext = 0;
    nuAllocation->pLink = nextProc;

    // insert new event into eventQ
    insertIntoEventQ( nuAllocation );
}
void scheduleDeparture(){
    // create a new event node for the departure event
    eventQueue* nuDeparture = new eventQueue;
    nuDeparture->type = 2;
    nuDeparture->eNext = 0;
    nuDeparture->pLink = cpuHead->pLink;

    // set the departure time for the event
    if( schedulerType == 1 ||                  // FCFS
        schedulerType == 3 ){                  // HRRN
           nuDeparture->time =
          cpuHead->pLink->startTime +
              cpuHead->pLink->remainingTime;
    }
    else if( schedulerType == 2 ){             // SRTF
       if( cpuHead->pLink->reStartTime == 0 ){
          nuDeparture->time =
             cpuHead->pLink->startTime +
             cpuHead->pLink->remainingTime;
       }
       else{
       nuDeparture->time =
          cpuHead->pLink->reStartTime +
          cpuHead->pLink->remainingTime;
       }
    }

    // insert the new event into eventQ in asc time order
    insertIntoEventQ(nuDeparture);
}
void handleArrival(){
    // create a new readyQ node based on proc in eHead
    readyQueue* nuReady = new readyQueue;
    nuReady->pLink = eHead->pLink;
    nuReady->rNext = 0;

    // push the new node into the readyQ
    if( rHead == 0 ) rHead = nuReady;
    else{
       readyQueue* rIt = rHead;
       while( rIt->rNext != 0 ){
          rIt = rIt->rNext;
       }
       rIt->rNext = nuReady;
    }

    // pop the arrival from the eventQ
    popEventQHead();
}
void handleDeparture(){
```

```
133     // update cpu data
134     cpuHead->pLink->finishTime = eHead->time;
135      cpuHead->pLink->remainingTime = 0.0;
136     cpuHead->pLink = 0;
137     cpuHead->clock = eHead->time;
138     cpuHead->cpuBusy = false;
139
140     // pop the departure from the eventQ
141     popEventQHead();
142  }
143  void handleAllocation(){
144     // point cpu to the proc named in the allocation event
145     cpuHead->pLink = eHead->pLink;
146
147     if( schedulerType == 2 ||      // FCFS
148        schedulerType == 3 ){       // HRRN
149        // find the corresponding process in readyQ and move
150        // it to top of readyQ if it's not already there
151        readyQueue* rIt = rHead->rNext;
152        readyQueue* rItPrev = rHead;
153        if( rItPrev->pLink->arrivalTime != eHead->pLink->arrivalTime ){
154           while( rIt != 0 ){
155          if( rIt->pLink->arrivalTime ==
156             eHead->pLink->arrivalTime ){
157                 rItPrev->rNext = rIt->rNext;
158                 rIt->rNext = rHead;
159                 rHead = rIt;
160                 break;
161              }
162              rIt = rIt->rNext;
163              rItPrev = rItPrev->rNext;
164           }
165        }
166     }
167
168     // pop the readyQ and eventQ records
169     popReadyQHead();
170     popEventQHead();
171
172     // set the busy flag to show the cpu is now busy
173     cpuHead->cpuBusy = true;
174
175     // update sim clock
176     if( cpuHead->clock < cpuHead->pLink->arrivalTime ){
177        // if clock < arrival time, then clock = arrival time
178        cpuHead->clock = cpuHead->pLink->arrivalTime;
179     }
180
181     // update start/restart time as needed
182     if( cpuHead->pLink->startTime == 0 ){
183        cpuHead->pLink->startTime = cpuHead->clock;
184     }
185     else{
186        cpuHead->pLink->reStartTime = cpuHead->clock;
187     }
188  }
189
190
191  void insertIntoEventQ( eventQueue* nuEvent ){
192     // put the new event in the readyQ, sorted by time
193     if( eHead == 0 ) eHead = nuEvent;
194     else if( eHead->time > nuEvent->time ){
195        nuEvent->eNext = eHead;
196        eHead = nuEvent;
197     }
198     else{
```

```
199         eventQueue* eIt = eHead;
200         while( eIt != 0 ){
201            if( (eIt->time < nuEvent->time) && (eIt->eNext == 0) ){
202               eIt->eNext = nuEvent;
203           break;
204            }
205            else if( (eIt->time < nuEvent->time) &&
206                     (eIt->eNext->time > nuEvent->time)){
207               nuEvent->eNext = eIt->eNext;
208               eIt->eNext = nuEvent;
209               break;
210            }
211            else{
212               eIt = eIt->eNext;
213            }
214         }
215      }
216   }
```