```
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 handleAllocation(){
   // point cpu to the proc named in the allocation event
   cpuHead->pLink = eHead->pLink;

   if( schedulerType == 2 ||      // FCFS
      schedulerType == 3 ){     // HRRN
      // find the corresponding process in readyQ and move
      // it to top of readyQ if it's not already there
      readyQueue* rIt = rHead->rNext;
      readyQueue* rItPrev = rHead;
      if( rItPrev->pLink->arrivalTime != eHead->pLink->arrivalTime ){
         while( rIt != 0 ){
        if( rIt->pLink->arrivalTime ==
           eHead->pLink->arrivalTime ){
               rItPrev->rNext = rIt->rNext;
               rIt->rNext = rHead;
               rHead = rIt;
               break;
            }
            rIt = rIt->rNext;
            rItPrev = rItPrev->rNext;
         }
      }
   }

   // pop the readyQ and eventQ records
   popReadyQHead();
   popEventQHead();

   // set the busy flag to show the cpu is now busy
   cpuHead->cpuBusy = true;

   // update sim clock
   if( cpuHead->clock < cpuHead->pLink->arrivalTime ){
      // if clock < arrival time, then clock = arrival time
      cpuHead->clock = cpuHead->pLink->arrivalTime;
   }

   // update start/restart time as needed
   if( cpuHead->pLink->startTime == 0 ){
      cpuHead->pLink->startTime = cpuHead->clock;
   }
   else{
      cpuHead->pLink->reStartTime = cpuHead->clock;
   }
}
```

```cpp


void handleDeparture(){
   // update cpu data
   cpuHead->pLink->finishTime = eHead->time;
    cpuHead->pLink->remainingTime = 0.0;
   cpuHead->pLink = 0;
   cpuHead->clock = eHead->time;
   cpuHead->cpuBusy = false;

   // pop the departure from the eventQ
   popEventQHead();
}

void handlePreemption(){
   // create a temp ptr to hold the current cpu pLink
   process* preemptedProcPtr = cpuHead->pLink;

   // update the remaining time
   cpuHead->pLink->remainingTime =
      cpuEstFinishTime() - eHead->time;

   // point cpu to preempting process and update data as needed
   cpuHead->pLink = eHead->pLink;
   cpuHead->clock = eHead->time;
   if( cpuHead->pLink->reStartTime == 0.0  ){
      cpuHead->pLink->startTime = eHead->time;
   }
   else{
      cpuHead->pLink->reStartTime = eHead->time;
   }

   // schedule an arrival event for the preempted proc
   eventQueue* preemptedProcArrival = new eventQueue;
   preemptedProcArrival->time = eHead->time;
   preemptedProcArrival->type = 1;
   preemptedProcArrival->eNext = 0;
   preemptedProcArrival->pLink = preemptedProcPtr;

   // pop the preemption event from the eventQ
   popEventQHead();

   // insert new event into eventQ
   insertIntoEventQ( preemptedProcArrival );
}

void popEventQHead(){
   eventQueue* tempPtr = eHead;
   eHead = eHead->eNext;
   delete tempPtr;
}

void popReadyQHead(){
   readyQueue* tempPtr = rHead;
   rHead = rHead->rNext;
   delete tempPtr;
}
```