```
/**********************************************************************
Authors : Manuella,Evan      , Box 4065, <manuella@grinnell.edu>
        : Albert Owusu-Asare , Box 4497, <owusuasa@grinnell.edu>

Date    : Fri Sep 19 22:02:11 CDT 2014

This contains the scheduler simulation for sjn.

http://www.cs.grinnell.edu/~weinman/courses/CSC213/2014F/labs/scheduling.html

**********************************************************************/


/* Citations:
 * Code cited here is part of scheduler_fcfs.c, which was written by Jerod
 * Weinman. We copied this file, and modified it. Jeord Weinman's sources
 * are listed below.
 *
 * Our additions to this code are marked.
 */




/* Implements a fastest job next scheduler.
 *
 * Created by Henry Walker, 27 September 2004
 * Last modified by Janet Davis, 25 September 2010
 * Revised by Jerod Weinman, 10 August 2012
 */

#include <stdlib.h>
#include <stdio.h>
#include "scheduler.h"

/* The ready queue */
job_queue_t ready;

/* Initializes the ready queue.  Call before any other functions. */
void ready_queue_init(void) {
  ready.first = NULL;
  ready.last  = NULL;
}

/* Returns true or false, according to whether any jobs are waiting
 * in the ready queue.
 */
int ready_queue_empty(void) {
  return (ready.first == NULL);
}

/* Adds the specified job to the ready queue.
 *
 * Preconditions:
 *   job != NULL
 * Postconditions:
 *   Creates a new node for the job
 *   job is inserted at the end of the queue
 */
void ready_queue_insert(job_t* job) {

  job_queue_node_t* node
    = (job_queue_node_t *)malloc(sizeof(job_queue_node_t));

  /* check if memory allocated */
  if (!node) {
    perror("Unable to allocate job node");
    exit(EXIT_FAILURE);
  }

  /* Addition: pointers to keep track of previous and current nodes*/
  job_queue_node_t* prev
    = (job_queue_node_t *)malloc(sizeof(job_queue_node_t));

  /* check if memory allocated */
  if (!prev) {
    perror("Unable to allocate job node");
    exit(EXIT_FAILURE);
  }

  job_queue_node_t* current
    = (job_queue_node_t *)malloc(sizeof(job_queue_node_t));

  /* check if memory allocated */
  if (!current) {
    perror("Unable to allocate job node");
    exit(EXIT_FAILURE);
  }

  /* copy event data to new node */
  node->job = job;
  node->next = NULL;
  /* Implementation of queue
   * Addition: sort queue by the amount of cpu time for each job
   */
  //if queue is empty
  if (ready_queue_empty()) {
    ready.first = node;
    ready.last  = node;
  } else //queue is not empty
    {
      //if cpu_time for new node is shortest on queue
      if(((ready.first)->job->cpu_time_left) >= node->job->cpu_time_left){

        node->next = ready.first;
        ready.first = node;
      } else
        // cpu_time for new node not shortest
        {
          prev = ready.first;
          current  = ready.first;
          // find position of insert
          while(current->next != NULL) {

            if((node->job)->cpu_time > current->job->cpu_time) {
            prev = current;
            current = current->next;
            }
          }
          /* insert node in designated position*/
          //if designated position at end of the queue
          if( prev->next == NULL){
            prev->next = node;
            node->next = NULL;
            ready.last = node; // modify end of ready queue
          } else
            //if designated position not at the end of the queue
            {
              prev->next = node;
              node->next = current;
            }
        } // else cpu time not the shortest

    } //else  queue not empty

} // end of modifications
```

```c
/* Removes and returns the job at the head of the ready queue.
 *
 * Postconditions:
 *   If ready_queue_empty(), returns NULL
 *   Otherwise, returns head job and frees the associated node
 */
job_t* ready_queue_select(void) {
  job_t* job;
  job_queue_node_t* old_node;

  /* if no jobs are ready, return NULL */
  if (ready_queue_empty())
    return NULL;

  /* next job is at front of queue */
  job = ready.first->job;

  /* record node at front of queue */
  old_node = ready.first;
  ready.first = ready.first->next;

  /* check if queue is -now- empty */
  if (ready_queue_empty()) {
    ready.last = NULL; /* make last pointer consistent */
  }

  /* return old front of queue to memory pool */
  free(old_node);

  return job;
}
```