```
/*
 *     File Name                    : driver.cpp
 *     Author(s)                    : Francesco Polizzi, Katie Schaffer, Jeremy Viner, Hein Htet Zaw
 *     Date Created                 : 26 April 2016
 *     Date Last Modified           : 6 May 2016
 *
 *     Description          :   Main routine of the program; reads the data files and runs
 *                                  the simulation.
 *
 */

    //libraries to include
#include <iostream>
#include <fstream>
#include <iomanip>
#include "simulation_header.h"

using namespace std;

    // Declare tracking variables
int total_jobs_run;                  // Total jobs run
double total_response_time;          // Total response time
double total_productive_time;        // Total productive time
double total_turnaround_time;        // Total turnaround time
double total_switch_time;            // Total time spent context switching
double total_ltq_wait;               // Total time spent waiting in longterm queue
double total_stq_wait;               // Total time spent waiting in shortterm queue
double total_ioq_wait;               // Total time spent waiting in the IO queue
int sys_clock;                       // Current system time (in clock ticks)

/* main
 * Author(s): Francesco Polizzi, Katie Schaffer, Jeremy Viner, Hein Htet Zaw
 * Date Created: 28 April 2016
 * Last revised: 10 May 2016
 *
 * Description: Primary simulation routine; initializes counters and variables, reads input file,
 *                  calls all functions to managed parts of the computer, and prepares output
 *                  data for printing
 */
int main() {
    /////////////////////////////////////////////////////////////////////////////////////////
    /// STEP 1 - Initialize
    /////////////////////////////////////////////////////////////////////////////////////////

        // Initialize tracker variables to 0
    total_stq_wait = 0;
    total_jobs_run = 0;
    total_response_time = 0;
    total_productive_time = 0;
    total_turnaround_time = 0;
    total_switch_time = 0;
    total_stq_wait = 0;
    total_ltq_wait = 0;
    total_ioq_wait = 0;
    sys_clock = 0;

        // Declare counter variables
    int jobs_admitted = 0;      // Counts number of jobs admitted so far
    int job_timer = 0;          // Keeps track of the time between job arrivals

        // Simulation devices
    longQueue longterm_queue;   // Longterm queue
    shortQueue shortterm_queue; // Shortterm queue
    ioQueue io_queue;           // IO queue
    IOdevice io_device;         // IO device
    CPU cpu;                    // CPU

        // Initialize flags and flag container
    FlagContainer flags;
    flags.jobs_in_system = 0;
    flags.incoming_job = false;
    flags.interrupt = false;

        // Initialize IO device values
    io_device.available = true;
    io_device.complete = false;
io_device.job_finished = false;
io_device.timer = 0;

    // Initialize CPU values
cpu.ready = true;
cpu.timer = 0;
cpu.complete = false;
cpu.processing_stopped = false;
cpu.suspended = false;

     // Initialize our job and jobs list
job tempJob;
job* current_job;
job job_list[150];

     // Initialize data files
ifstream infile("SIM_DATA.txt", ios::in);   // Onput file
ofstream outfile("Output.txt", ios::out);   // Output file

     //initialize our reading flag and job count
bool reading = true;
int job_count = 0;
int jobs_entering_system=0;


//////////////////////////////////////////////////////////////////////////////////////////
/// STEP 2 - Get data from input file
//////////////////////////////////////////////////////////////////////////////////////////

     // Read and process data from our file
while (reading) {
        // Create a new job
    tempJob = *new job();
        // Read in job information
    infile >> tempJob.num;
    infile >> tempJob.length;
    infile >> tempJob.inter_arrival;
    infile >> tempJob.io_burst;

        // Initialize other job variables
    tempJob.burst_num = 0;
    tempJob.response = -1;

        // Initialize burst list to all -1
    for (int burst_num = 0; burst_num < cpu_burst_max; burst_num++) {
        tempJob.cpu_burst[burst_num] = -1;
    }

        // Next value to read could be burst or sentinel
    int temp_input;
    infile >> temp_input;

        // Continue to read until sentinel
    while (temp_input > 0){
            // Add CPU burst to temp job cpu burst array
        tempJob.cpu_burst[tempJob.burst_count]=temp_input;
        tempJob.burst_count++;
        infile >> temp_input;
    }

        // Add new job to job array
    job_list[job_count] = tempJob;
    job_count++;

        // Confirm we've reached the sentinel and finish reading
    if (temp_input == -1) {
        reading = false;
    }
}


//////////////////////////////////////////////////////////////////////////////////////////
/// STEP 3 - Get first job into the system
//////////////////////////////////////////////////////////////////////////////////////////

    // Update job timer
```

```
job_timer++;

    // When a job enters the system
if (job_list[jobs_admitted].inter_arrival == job_timer) {
        // Set job flag to true
    flags.incoming_job = true;
        // Get reference to job
    current_job = &job_list[total_jobs_run];
        // Record time of arrival
    current_job->arrival = sys_clock;
        // Reset job_timer to zero
    job_timer = 0;
        // Update counter of jobs admitted
    jobs_admitted++;

        // Increment number of jobs currently int the system
    jobs_entering_system++;
    flags.jobs_in_system++;
}

//////////////////////////////////////////////////////////////////////////////////////////
/// STEP 4 - Process incoming jobs until all are processed
//////////////////////////////////////////////////////////////////////////////////////////

    // Process while there are jobs to process
while(total_jobs_run < job_count) {
        // Manage all parts of the computer
    manage_ltq(longterm_queue, current_job, flags);
    manage_stq(shortterm_queue, longterm_queue, &io_device, flags);
    manage_cpu(&cpu, shortterm_queue, flags);
    manage_ioq(io_queue, &cpu);
    manage_iodevice(&io_device, io_queue, flags);

        // Increment clock
    sys_clock++;

        // Check for incoming processes.
        //  When a job enters the system...
    if (job_list[jobs_admitted].inter_arrival <= job_timer && !longterm_queue.isFull()) {
            // Set job flag to true
        flags.incoming_job = true;
            // Get reference to job
        current_job = &job_list[jobs_entering_system];
            // Record time of arrival
        current_job->arrival = sys_clock;
            // Reset job_timer to zero
        job_timer = 0;
            // Increment admitted job count
        jobs_admitted++;
            // Increment more_jobs
        jobs_entering_system++;
        flags.jobs_in_system++;
    }

        // Update job timer
    job_timer++;
}


//////////////////////////////////////////////////////////////////////////////////////////
/// STEP 5 - Compile results and print to output file
//////////////////////////////////////////////////////////////////////////////////////////

    // Process accumulated data
double total_time = total_switch_time + sys_clock;
double avgLTQ = avg_ltq(total_jobs_run, total_ltq_wait);
double avgSTQ = avg_stq(total_jobs_run, total_stq_wait);
double avgIOQ = avg_ioq(total_jobs_run, total_ioq_wait);
double avgResponse = avg_response_time(total_jobs_run, total_response_time);
double avgTurnaround = avg_turnaround_time(total_jobs_run, total_turnaround_time);
double cpuUtilization = cpu_utilization(total_productive_time, sys_clock);
double contextSwitchTime = total_switch_time;
double systemThroughput = ((double)total_jobs_run) / ((double)total_time);

    // Print header before printing anything
print_header(outfile);
```

```
        // Print "First in First Out" results
    print_output("First in First Out", total_time, contextSwitchTime,
                 cpuUtilization, avgResponse, avgTurnaround, systemThroughput, avgLTQ,
                 avgSTQ, avgIOQ, outfile);
        // Indicate end of output at the end
    print_footer(outfile);

    return 0;
}
```