/\*

\* File Name : simulation\_header.h

\* Primary Author : Hein Htet Zaw

\* Contributing Author(s) : Katie Schaffer

\* Date Created : 26 April 2016

\* Date Last Modified : 11 May 2016

\*

\* Description : This is the header file where all the global variables and

\* data structures are declared and defined.

\*

\*/

#ifndef \_SIMULATION\_HEADER\_H\_

#define \_SIMULATION\_HEADER\_H\_

#include <string>

#include <array>

#define long\_max 60

#define short\_max 30

#define io\_max 30

#define cpu\_burst\_max 25

using namespace std;

// IO device structure

struct IOdevice {

bool available; // Signals that the IO device is available

bool complete; // Signals the completion of an IO burst

int timer; // Indicates the current IO burst

int burst\_length; // Length of current burst

bool job\_finished; // Signals that a job is finished

job\* process; // Pointer to the process in the IO device

job\* entering\_process; // Pointer to the process entering the IO device

};

// CPU structure

struct CPU {

int timer; // Keeps track of the current CPU burst

bool complete; // Signals the completion of a CPU burst

bool ready; // Signals that the CPU is available

bool processing\_stopped; // Signals to stop CPU job processing

bool suspended; // Signals context switch to handle interrupt

int suspend\_timer; // Keeps track of current interrupt time

int total\_wait; // Total time spent waiting (in suspension)

job\* susp\_process; // Pointer to suspended process

job\* process; // Pointer to which job has the CPU

};

// Flag container structure

struct FlagContainer {

int jobs\_in\_system; // Number of jobs currently in the system

bool incoming\_job; // Signals that a job has arrived

bool interrupt; // Signals that an interrupt is in progress

bool io\_interrupt;

};

// Computer part management function prototypes

void manage\_ltq(longQueue&, job\*, FlagContainer&);

void manage\_stq(shortQueue&, longQueue&, IOdevice\*, FlagContainer&);

void manage\_ioq(ioQueue&, CPU\*);

void manage\_cpu(CPU\*, shortQueue&, FlagContainer&);

void manage\_iodevice(IOdevice\*, ioQueue&, FlagContainer&);

// function declarations for auxillary functions

double avg\_ltq(int, double);

double avg\_stq(int, double);

double avg\_ioq(int, double);

double avg\_response\_time(int, double);

double avg\_turnaround\_time(int, double);

double cpu\_utilization(int,double);

void print\_output(string, int, int, double, int, int, double, double, double, double, ofstream&);

void print\_header(std::ofstream&);

void print\_footer(std::ofstream&);

//declare and instantiate global variables and arrays

//create spaces for the followings

static job \*processor = NULL; //the (only) processor

static job \*IO = NULL; //the I/O device

//declare variables for the followings

extern int sys\_clock; //clock to keep track of time

static int LTQ\_time = 0; //total long term queue wait time for all jobs

static int STQ\_time = 0; //total short term queue wait time for all jobs

static int IOQ\_time = 0; //total I/O Queue wait time for all jobs

static job \*temp = NULL; //temporary space

extern int total\_jobs\_run; // Total jobs run

extern double total\_response\_time; // Total response time

extern double total\_productive\_time; // Total productive time

extern double total\_turnaround\_time; // Total turnaround time

extern double total\_switch\_time; // Total time spent context switching

extern double total\_stq\_wait; // Total time spent waiting in shortterm queue

extern double total\_ltq\_wait; // Total time spent waiting in longterm queue

extern double total\_ioq\_wait; // Total time spent waiting in io queue

//set up a structure for jobs

struct job {

//include the following information in the job

int num; //job number

int length; //(CPU bursts + I/O bursts) time

int inter\_arrival; //interarrival time

int arrival; //arriaval time

int io\_burst; //the length of time this job requires an I/O device

int cpu\_burst[cpu\_burst\_max]; //the time this process requires the CPU

int burst\_num; //current burst

int burst\_count = 0; //total number of bursts (it's 0 initially)

int lastEnterTime = 0; //the time that a process last entered a device/queue

//create variables for the following

int time\_in\_cpu = 0; //total time spent in the CPU

int time\_in\_longQ = 0; //total time spent in the Long Term Queue

int time\_in\_shortQ = 0; //total time spent in the Short Term Queue

int time\_in\_ioQ = 0; //total time spent in the I/O Queue

int time\_in\_system = 0; //total time spent in the system

int turnaround = 0; //the turnaround time

int response = 0; //the response time

int switching = 0; //time spent in context switching

job \*left;

job \*right;

};

//declare and define STORAGE \*This is a binary tree\*

class tree {

private:

job \*root;

int jobCount;

public:

tree() { root = NULL; jobCount = 0;}; //create constructor function

bool add(job \* ); //add the items in the tree, sorted by \*\*JOB NUMBER\*\*

bool add\_jobLength(job \* ); //add the job in the tree, sorted by \*\*JOB LENGTH\*\*

job \* getJob(int JobNum); //get the pointer of a certain job by its job number

job \* getRoot() { return root; }; //get the root of "this" tree

int getJobCount() { return jobCount; }; //get the number of jobs in "this" tree

};

//declare and define the long queue type and its required functions

class longQueue {

private:

int front, rear, size;

job \*theQ[long\_max];

public:

longQueue() { front = -1; rear = -1; size = 0; }; //construct the object

bool isEmpty() { if (size == 0) return true; else return false; }; //returns empty or not

bool isFull() { if (size == long\_max) return true; else return false; };//returns full or not

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Warning! If a new job is added while queue is full,

// the job will be dropped without any recovery option!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool add(job \* ); //adds the given job

job \* getNext(); //get the pointer of the next job in the queue

int getRear() { return rear; }; //returns the array number of lsatmost node

job \* getFront() { return theQ[front]; }; //returns the array number of frontmost node

int getSize() { return size; }; //returns the current number of jobs in the queue

bool incrementAll(); //increment all the jobs inside the queue

};

//declare and define short queue type and its required functions

class shortQueue {

private:

int front, rear, size;

job \*theQ[short\_max];

public:

shortQueue() { front = -1; rear = -1; size = 0; }; //constructs the object

bool isEmpty() { if (size == 0) return true; else return false; }; //returns empty or not

bool isNearlyFull() { if (size == short\_max-1) return true; else return false; };

bool isFull() { if (size == short\_max) return true; else return false; }; //returns full or not

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Warning! If a new job is added while queue is full,

// the job will be dropped without any recovery option!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool add(job \* ); //adds the given job

job \* getNext(); //get the pointer of the next job in the queue

int getRear() { return rear; }; //returns the array number of lsatmost node

job \* getFront() { return theQ[front]; }; //returns the array number of frontmost node

int getSize() { return size; }; //returns the current number of jobs in the queue

bool incrementAll(); //increment all the jobs inside the queue

};

//declare and define the required functions for I/O queue type

class ioQueue {

private:

int front, rear, size;

job \*theQ[io\_max];

public:

ioQueue() { front = -1; rear = -1; size = 0; }; //constructs the object

bool isEmpty() { if (size == 0) return true; else return false; }; //returns empty or not

bool isFull() { if (size == io\_max) return true; else return false; }; //returns full or not

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Warning! If a new job is added while queue is full,

// the job will be dropped without any recovery option!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool add(job \*); //adds the given job

job \* getNext(); //get the pointer of the next job in the queue

int getRear() { return rear; }; //returns the array number of lsatmost node

job \* getFront() { return theQ[front]; }; //returns the array number of frontmost node

int getSize() { return size; }; //returns the current number of jobs in the queue

bool incrementAll(); //increment all the jobs inside the queue

};

#endif // !\_SIMULATION\_HEADER\_H\_