/\*

\* File Name : treeNqueue\_functions.cpp

\* Primary Author : Hein Htet Zaw

\* Contributing Author(s) :

\* Date Created : 28 April 2016

\* Date Last Modified : 9 May 2016

\*

\* Description : This file defines all the functions needed for the storage

\* tree and queues required to make this project work.

\*

\*/

#include <iostream>

#include <array>

#include "simulation\_header.h"

using namespace std;

bool tree::add(job \*theJob) {

// Receives – The job to insert

// Task - Insert the job into the tree \*\*sorted by Job Number\*\*

// Returns - Whether or not success

//reserve vrequired variables

bool inserted = false;

job \*currentPointer;

//set left and right pointers to NULL for new node

theJob->left = NULL;

theJob->right = NULL;

//the first ever node is root

currentPointer = root;

//loop if not inserted yet

while (inserted == false) {

//put the node in root if the root is empty

if (currentPointer == NULL) {

root = theJob;

inserted = true;

}

//reject the job if the same job information is already in the tree

else if (theJob->num == currentPointer->num) return false;

//branch to left if the new node is less than current node

else if (theJob->num < currentPointer->num) {

//extend branch from the free side

if (currentPointer->left != NULL) currentPointer = currentPointer->left;

else {

currentPointer->left = theJob;

inserted = true;

}

}

//branch to right side if the new node is higher than current one

else {

//extend the branch from the free side

if (currentPointer->right != NULL) currentPointer = currentPointer->right;

else {

currentPointer->right = theJob;

inserted = true;

}

}

}

jobCount++;

return true;

}

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

bool tree::add\_jobLength(job \*theJob) {

// Receives – The job to insert

// Task - Insert the job into the tree \*\*sorted by Job Length\*\*

// Returns - Whether or not success

//reserve vrequired variables

bool inserted = false;

job \*currentPointer;

//set left and right pointers to NULL for new node

theJob->left = NULL;

theJob->right = NULL;

//the first ever node is root

currentPointer = root;

//loop if not inserted yet

while (inserted == false) {

//put the node in root if the root is empty

if (currentPointer == NULL) {

root = theJob;

inserted = true;

}

//go to left if same length -- jobs can have the same length

//branch to left if the new node is less than current node

else if (theJob->length <= currentPointer->length) {

//extend branch from the free side

if (currentPointer->left != NULL) currentPointer = currentPointer->left;

else {

currentPointer->left = theJob;

inserted = true;

}

}

//branch to right side if the new node is higher than current one

else {

//extend the branch from the free side

if (currentPointer->right != NULL) currentPointer = currentPointer->right;

else {

currentPointer->right = theJob;

inserted = true;

}

}

}

jobCount++;

return true;

}

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

job \* tree::getJob(int jobNum) {

// Receives - The job number

// Task - Find the job in the tree

// Returns - The pointer of the job if found, NULL pointer if not found

//begin at root

job \*current = root;

//go through the whole tree

while (current != NULL) {

//return the node if found

if (jobNum == current->num) return current;

//keep searching till found or the end

else {

if (jobNum < current->num) current = current->left;

else current = current->right;

}

}

//return NULL pointer if not found

return NULL;

}

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

bool longQueue::add(job \*theJob){

// Receives – The pointer of the job to add

// Task - Adds the pointer of the job provided to \*this\* queue

// Returns - Whether fail or success adding

//return false when the queue is full

if (isFull()) return false;

//start filling from the front if the queue is empty

if (isEmpty()) front = 0;

//increase one space to rear and add the job

rear++;

//loop the places when the line is filled in rear but line is not full

if (rear == long\_max && size != long\_max) rear = 0;

//add the job into the queue

theQ[rear] = theJob;

//keep track of number of jobs in the queue

size++;

return true;

};

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

job \* longQueue::getNext(){

// Receives – Nothing

// Task - Take out the first job from the queue, and move the next job to front

// Returns - The pointer of the job that leaves the queue

//return NULL if the queue is empty

if (isEmpty()) return NULL;

job \*temp;

if (front < 0) front = 0;

if (front == long\_max) front = 0;

//move the next job to the front of the queue

temp = theQ[front];

for (int i = 0; i < size - 1; i++) {

theQ[i] = theQ[i + 1];

theQ[i + 1] = NULL;

}

if (size == 1)

theQ[0] = NULL;

rear--;

//keep track of the number of jobs in the queue

if (size > 0) size--;

return temp;

};

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

bool longQueue::incrementAll() {

// Receives – Nothing

// Task - Increment the time in long queue of all the processes

// Returns - Incremented everything or queue is empty

//return false if the queue is empty

if (isEmpty()) return false;

//if queue is not empty, increment all the processes in the queue

else {

int counter = 0, temp = front;

while (counter != size) {

if (temp == long\_max) temp = 0;

theQ[temp]->time\_in\_longQ++;

counter++; temp++;

}

}

return true;

}

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

bool shortQueue::add(job \*theJob){

// Receives – The pointer of the job to add

// Task - Adds the pointer of the job provided to \*this\* queue

// Returns - Whether fail or success adding

//return false when the queue is full

if (isFull()) return false;

//start filling from the front if the queue is empty

if (isEmpty()) front = 0;

//increase one space to rear and add the job

rear++;

//loop the places when the line is filled in rear but line is not full

if (rear == short\_max && size != short\_max) rear = 0;

//add the job into the queue

theQ[rear] = theJob;

//keep track of number of jobs in the queue

size++;

return true;

};

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

job \* shortQueue::getNext(){

// Receives – Nothing

// Task - Take out the first job from the queue, and move the next job to front

// Returns - The pointer of the job that leaves the queue

//return NULL if the queue is empty

if (isEmpty()) return NULL;

job \*temp;

if (front < 0) front = 0;

if (front == short\_max) front = 0;

//move the next job to the front of the queue

temp = theQ[front];

for (int i = 0; i < size - 1; i++) {

theQ[i] = theQ[i + 1];

theQ[i + 1] = NULL;

}

if (size == 1)

theQ[0] = NULL;

rear--;

//keep track of the number of jobs in the queue

if (size > 0) size--;

return temp;

};

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

bool shortQueue::incrementAll() {

// Receives – Nothing

// Task - Increment the time in short queue of all the processes

// Returns - Incremented everything or queue is empty

//return false if the queue is empty

if (isEmpty()) return false;

//if queue is not empty, increment all the processes in the queue

else {

int counter = 0, temp = front;

while (counter != size) {

if (temp == short\_max) temp = 0;

theQ[temp]->time\_in\_shortQ++;

counter++; temp++;

}

}

return true;

}

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

bool ioQueue::add(job \*theJob){

// Receives – The pointer of the job to add

// Task - Adds the pointer of the job provided to \*this\* queue

// Returns - Whether fail or success adding

//return false when the queue is full

if (isFull()) return false;

//start filling from the front if the queue is empty

if (isEmpty()) front = 0;

//increase one space to rear and add the job

rear++;

//loop the places when the line is filled in rear but line is not full

if (rear == io\_max && size != io\_max) rear = 0;

//add the job in the queue

theQ[rear] = theJob;

//keep track of number of jobs in the queue

size++;

return true;

};

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

job \* ioQueue::getNext(){

// Receives – Nothing

// Task - Take out the first job from the queue, and move the next job to front

// Returns - The pointer of the job that leaves the queue

//return NULL if the queue is empty

if (isEmpty()) return NULL;

job \*temp;

if (front < 0) front = 0;

if (front == io\_max) front = 0;

//move the next job to the front of the queue

temp = theQ[front];

for (int i = 0; i < size - 1; i++) {

theQ[i] = theQ[i + 1];

theQ[i + 1] = NULL;

}

if (size == 1)

theQ[0] = NULL;

rear--;

//keep track of the number of jobs in the queue

if (size > 0) size--;

return temp;

};

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

bool ioQueue::incrementAll() {

// Receives – Nothing

// Task - Increment the time in I/O queue of all the processes

// Returns - Incremented everything or queue is empty

//return false if the queue is empty

if (isEmpty()) return false;

//if queue is not empty, increment all the processes in the queue

else {

int counter = 0, temp = front;

while (counter != size) {

if (temp == io\_max) temp = 0;

theQ[temp]->time\_in\_ioQ++;

counter++; temp++;

}

}

return true;

}

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