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

Programmer: Tim Wrenn Class: CSCI 1107

Basic Operations:

constructor: constructs an empty Queue

empty : checks if the Queue is empty

enqueue : Inserts an element into the Queue

front : Returns the top queue element

dequeue : deletes the top element from the Queue

display : Displays the entire Queue

Note: Program does not terminate if error conditions are met.

Chris Dang CSCI 1107

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

#include <iostream>

#include <iomanip>

#include <string>

#include <cctype>

using namespace std;

#ifndef CIRCULARQ\_H

#define CIRCULARQ\_H

template <typename qElement>

class CircularQ{

private:

class Node{

public:

qElement data;

Node \* next;

Node(qElement d){

data = d;

next = NULL;

}// end Node Constructor

};

public:

typedef Node \* NodePointer;

CircularQ(){ last = NULL; } // Create an empty Q

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

Default Constructor creates an empty Queue

Pre: None

Post: An empty Queue is created and the last pointer is initialized to NULL

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

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

Display the entire Queue

Pre: Ostream out is open.

Post: Outputs the contents of the Queue in FIFO order

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

void display(ostream &out) const{ // Helper method

NodePointer first; // Need a pointer

if (last == NULL)

cout << "Queue is empty \n\n" ;

else{

first = last->next; // no, intialize to first item in the list

do{

out << first->data << " "; // output the item

first = first->next; // move along

}while(first != last->next); // have we traversed the entire list?

}// end else

}//display method

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

Copy Constructor: creates a copy of the original Queue

Pre: Original is the Queue to be copied and is received as a const reference

parameter

Post: A copy of the original has been constructed

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

CircularQ(const CircularQ &origQ){ // Copy constructor

if(origQ.last != NULL){

NodePointer origLast = origQ.last,

origFirst = origLast->next,

newNode;

last = new Node(origFirst->data);

last->next = last;

origFirst = origFirst ->next;

while(origFirst != origLast->next){

newNode = new Node(origFirst->data);

newNode->next = last -> next;

last->next = newNode;

last = newNode;

origFirst = origFirst ->next;

}// end while

}// end if

}//end copy constructor

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

Destructor: deletes the Queue

Pre: None

Post: The queue is destroyed and empty queue is created

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

~CircularQ(){ // Destructor

NodePointer first;

if(last != NULL){ // empty list?

first = last->next; // no, intialize to first item in the list

while(first != last){

last->next = first -> next; // avoid memory leak

delete first; // get rid of first node

first = last->next; // adjust to the first node.

}// end while

delete first; // make sure the last node is gone

last = NULL; // set to an empty list

}// end if

}// end destructor

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

Overloaded Assignment Operator: Mimics the assignment operator

Pre: Original is the Queue to be copied and is received as a const reference

parameter

Post: The current Queue is replaced by a copy of the Queue of rhs and

a reference to it is returned.

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

CircularQ &operator=(const CircularQ &rhs){

if(this != &rhs){

this->~CircularQ();

if(rhs.last != NULL){

NodePointer rhsLast = rhs.last,

rhsFirst = rhsLast->next,

newNode;

last = new Node(rhsFirst->data);

last->next = last;

rhsFirst = rhsFirst ->next;

while(rhsFirst != rhsLast->next){

newNode = new Node(rhsFirst->data);

newNode->next = last -> next;

last->next = newNode;

last = newNode;

rhsFirst = rhsFirst ->next;

}// end while

}// end if

}

return \*this;

}// end assignment operator

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

enqueue: Inserts an item into the front of the Queue.

Pre: Queue must exist and the parameter item must be initialized.

Post: The item is inserted into the Queue.

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

template <typename qElement>

void enqueue(qElement item){

NodePointer newNode = new Node(item); // Need a node for the Q

if(last == NULL){ // empty Q

last = newNode; // start constructing the Q

last->next = last; // pointer to itself since one item

}

else{

newNode->next = last->next; // Chain it

last->next = newNode;

last = newNode;

}

}// end enqueue

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

front: Returns the first item in the queue.

Pre: Queue must exist

Post: Either an error message will be displayed if the Queue is empty or

the front of the Queue is returned.

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

qElement front(){

//Element garbage = -9999999999999999;

if(empty()){

cout << "Queue is empty returning garbage" << endl;

return \*new qElement;

}

NodePointer first = last->next;

return (first->data);

}// return front of Q

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

dequeue: Deletes the item located in the front of the Queue.

Pre: Checks to see if the Queue is empty

Post: Displays an error message if the Queue is empty otherwise the element

at the front of the Queue is deleted.

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

void dequeue(){

if(empty())

cout << "Error ----- Cannot delete an item from an empty Queue"

<< endl << endl;

else{

NodePointer first = last->next;

if(first == last)

last = NULL; // Queue will be empty

else

last->next = first->next;

delete first; // eliminate the node

}

}// end dequeue

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

empty: Checks to see if the Queue is empty

Pre: A queue must be created

Post: Returns either true if the Queue is empty, false otherwise.

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

bool empty(){return last == NULL;}

private:

NodePointer last;

};

// Non-member functions

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

operator<<: Overloaded operator which allows you to display the contents

of the entire Queue

Pre: ostream must be opened and the const parameter is passed by reference

Post: Calls the method display that displays the entire Queue.

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

template <typename qElement>

ostream &operator<<(ostream& out, const CircularQ<qElement> &q){

q.display(out);

return out;

}// end operator<<

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

join: Appends q2 to the end of q1

Pre: Must pass in Circular q1 and q2

Post: Returns an appended circular queue

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

template <typename qElement>

CircularQ<qElement> join(CircularQ<qElement> q1, CircularQ<qElement> q2) {

while(!q2.empty()) {

q1.enqueue(q2.front()) ;

q2.dequeue() ;

} // end while

return q1 ;

}// end join

#endif

#include <iostream>

#include "circq.h"

using namespace std;

class Student {

private:

int studentId ;

string name ;

public:

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

Default Constructor creates a student object

Pre: None

Post: A student profile is created with placeholder student ID and name

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

Student(){studentId = 000; name = "Jamie Doe";}

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

read

Pre: Pass in istream object

Post: Records number and name for student

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

void read (istream &in) {

in >> studentId ; // record student ID

in >> name ; // record student name

} // end read

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

display

pre: must pass ostream operator

post: displays student ID and name of student

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

void display(ostream &out) const{

out << "Student ID: " << studentId << " Name: " << name ;

} // end display

} ; // end student class

// non-member function for student

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

operator<<: Overloaded operator which allows you to display the contents

of the entire Queue

Pre: ostream must be opened and the const parameter is passed by reference

Post: Calls the method display that displays the entire Queue.

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

ostream &operator<<(ostream& out, const Student &student) {

student.display(out) ;

return out;

} // end overloaded << op

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

operator>>: Overloaded operator which allows you to insert records

Pre: istream must be opened and the student record passed by value

Post: inserts student into record

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

istream &operator>>(istream& in, Student &student) {

student.read(in) ;

return in ;

} // end overloaded << op

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

operator<<: Overloaded operator which allows you to display the contents

of the entire student Queue

Pre: ostream must be opened and the const parameter is passed by reference.

Must be used with student queue.

Post: Calls the method display that displays the entire Queue.

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

template <typename qElement>

void displayStudent(ostream &out, CircularQ<qElement> student) {

while (!student.empty()) {

out << \*(student.front()) << endl ;

student.dequeue() ;

}// end while

}// end displayStudent

int main(){

CircularQ <double> qdub1 ; // holds 0.0, 1.1, 2.2 ... 9.9

CircularQ <double> qdub2 ; // holds user inputted doubles

CircularQ <double> qdub ; // tests assignment operator

CircularQ <string> qstring ; // holds strings

CircularQ <Student> qstu ; // holds students names

CircularQ <Student\*> stuPtrQ ; // holds anonimous variables

CircularQ <CircularQ<double>> qq ; // holds qdub1, qdub2, qdub

for (int i = 0; i <= 9 ; i++) // loads queue with 0, 1.1, 2.2 ... 9.9

qdub1.enqueue(i \* 1.1) ;

cout << "Please enter 5 double values: " ; // loads queue with 5 user values

for (int i = 0; i < 5 ; i++) {

double temp ;

cin >> temp ;

qdub2.enqueue(temp) ;

} // end for

cout << "qdub1 contains: " << qdub1 << endl ;

cout << "qdub2 contains: " << qdub2 << endl << endl ;

qdub = join(qdub1, qdub2) ; // testing assignment operator

cout << "qdub contains: " << qdub << endl ;

cout << "\nCreating student account.\n " ; // testing student class

Student s ;

cout << "Enter ID for student and name for student: " ; // testing student class

cin >> s ;

cout << s << endl ;

// tests Queue holding student records

cout << "Enter number of students: " ; // get number of students from user

int num ;

cin >> num ;

for (int i = 0; i < num; i++) { // get student records from user

Student temp ;

cout << "Enter ID for student and name for student: " ;

cin >> temp ;

qstu.enqueue(temp) ;

} // end for

cout << "Students are: " << qstu ; // display records

cout << endl << endl ;

// tests Queue holding student records that are anonymous

cout << "Enter number of students: " ; // get number of students from user

int num2 ;

cin >> num2 ;

for (int i = 0; i < num2; i++) { // get student records from user

Student temp ;

cout << "Enter ID for student and name for student: " ;

cin >> temp ;

Student \*anon ;

anon = new Student ;

\*anon = temp ;

stuPtrQ.enqueue(anon) ;

} // end for

// displaying student info with memory addresses

cout << "Memory locations of students are: " << stuPtrQ << endl ;

cout << "Students are:\n" ;

displayStudent(cout, stuPtrQ) ;

cout << endl << endl ;

// loading qq with queues

qq.enqueue(qdub1) ;

qq.enqueue(qdub2) ;

qq.enqueue(qdub) ;

cout << "qq contains: " << qq ; /// output queue

return 0;

}// end main



