Andrew Tran

Lab Experience 10

You are to do Lab 6.1 and project 6.2. Use cirqtester.cpp to test your circular queue.

**Queue.h – HEADER**

/\*--- Queue.h -------------------------------------------------------------

This header file contains the declaration of class Queue.

Basic operations:

Constructor: Constructs an empty queue

empty: Checks if a queue is empty

enqueue: Modifies a queue by adding a value at the back

front: Accesses the front queue value; leaves queue unchanged

dequeue: Modifies a queue by removing the value at the front

display: Displays the queue elements from front to back

Class Invariant:

1. The queue elements (if any) are stored in consecutive positions

in myArray, beginning at position myFront.

2. 0 <= myFront, myBack < QUEUE\_CAPACITY

3. Queue's size < QUEUE\_CAPACITY

Written by: Larry R. Nyhoff

Written for: Lab Manual for ADTs, Data Structures, and Problem

Solving with C++, 2E

Lab #6.1

--------------------------------------------------------------------------\*/

#ifndef QUEUE

#define QUEUE

#include<string>

#include<iostream>

#include<fstream>

using namespace std;

const int QUEUE\_CAPACITY = 128;

typedef int QueueElement;

class Queue

{

public:

/\*\*\*\*\* Function Members \*\*\*\*\*/

/\*\*\*\*\* Constructor \*\*\*\*\*/

Queue();

/\*-----------------------------------------------------------------------

Construct a Queue object.

Precondition: None.

Postcondition: An empty Queue object has been constructed; myFront

and myBack are initialized to -1 and myArray is an array with

QUEUE\_CAPACITY elements of type QueueElement.

----------------------------------------------------------------------\*/

/\*\*\*\*\* Copy \*\*\*\*\*/

Queue(const Queue & original);

/\*\*\*\*\* Destructor \*\*\*\*\*/

~Queue();

/\*------------------------------------------------------------------------

Class destructor

Precondition: None

Postcondition: The linked list in the stack has been deallocated.

------------------------------------------------------------------------\*/

/\*\*\*\*\* Assignment \*\*\*\*\*/

const Queue & operator= (const Queue & rightHandSide);

/\*------------------------------------------------------------------------

Assignment Operator

Precondition: rightHandSide is the stack to be assigned and is

received as a const reference parameter.

Postcondition: The current stack becomes a copy of rightHandSide

and a const reference to it is returned.

------------------------------------------------------------------------\*/

bool empty() const;

/\*-----------------------------------------------------------------------

Check if queue is empty.

Precondition: None.

Postcondition: True is returned if the queue is empty and false is

returned otherwise.

----------------------------------------------------------------------\*/

void enqueue(const QueueElement & value);

/\*-----------------------------------------------------------------------

Add a value to a queue.

Precondition: value is to be added to this queue.

Postcondition: value is added to back of queue provided there is space;

otherwise, a queue-full message is displayed and execution is

terminated.

-----------------------------------------------------------------------\*/

void display(ostream & out) const;

/\*-----------------------------------------------------------------------

Output the values stored in the queue.

Precondition: ostream out is open.

Postcondition: Queue's contents, from front to back, have been output

to out.

-----------------------------------------------------------------------\*/

QueueElement front() const;

/\*-----------------------------------------------------------------------

Retrieve value at front of queue (if any).

Precondition: Queue is nonempty.

Postcondition: Value at front of queue is returned, unless queue is

empty; in that case, an error message is displayed and a "garbage

value" is returned.

----------------------------------------------------------------------\*/

void dequeue();

/\*-----------------------------------------------------------------------

Remove value at front of queue (if any).

Precondition: Queue is nonempty.

Postcondition: Value at front of queue has been removed, unless queue

is empty; in that case, an error message is displayed and

execution is terminated.

----------------------------------------------------------------------\*/

private:

/\*\*\*\*\* Data Members \*\*\*\*\*/

int myFront,

myBack;

int myCapacity;

QueueElement \*myArray = new QueueElement[myCapacity];

}; // end of class declaration

//overloaded output operator

ostream &operator<<(ostream &out, const Queue &q);

#endif

**Queue.cpp – IMPLEMENTATION**

/\*--- Queue.cpp -----------------------------------------------------------

This implementation contains definitions of the function members for

the class Queue.

Written by: Larry R. Nyhoff

Written for: Lab Manual for ADTs, Data Structures, and Problem

Solving with C++, 2E

Lab #6.1

--------------------------------------------------------------------------\*/

#include<string>

#include<iostream>

#include<fstream>

using namespace std;

#include "Queue.h"

//default constructor

Queue::Queue()

{

myCapacity = 10;

myArray = new(nothrow)QueueElement[myCapacity];

if (myArray != 0)

{

myFront = 0;

myBack = 0;

}

else

{

cout << "Insufficient Space" << endl;

exit(1107);

}

}

//copy constructor

//Precons: original Queue

//Postcons: lnewly copied myArray

Queue::Queue(const Queue & original)

: myCapacity(original.myCapacity), myFront(original.myFront),myBack(original.myBack)

{

myArray = new(nothrow) QueueElement[myCapacity];

if (myArray != 0)

{

for (int i = myFront; i != myBack; i = (i + 1) % myCapacity)

myArray[i] = original.myArray[i];

}

else

{

cout << "Insufficient Memory" << endl;

exit(1107);

}

}

//destructor

Queue::~Queue()

{

delete[] myArray;

}

const Queue & Queue::operator= (const Queue & rightHandSide)

{

if (this != &rightHandSide)

{

delete[] myArray;

myCapacity = rightHandSide.myCapacity;

myArray = new QueueElement[myCapacity];

if (myArray == 0)

{

cout << "Insufficient Space" << endl;

exit(1107);

}

}

myFront = rightHandSide.myFront;

myBack = rightHandSide.myBack;

for (int i = myFront; i != myBack; i = (i + 1) % myCapacity)

{

myArray[i] = rightHandSide.myArray[i];

}

return \*this;

}

//Empty function, returns if the queue is empty

//Precons: none

//Postcons: bool

bool Queue::empty() const

{

return (myFront == myBack);

}

//Enqueue function, adds a value to the queue

//Precons: a value to enqueue

//Postcons: none

void Queue::enqueue(const QueueElement & value)

{

int newBack = (myBack + 1) % QUEUE\_CAPACITY;

if (newBack != myFront) // queue isn't full

{

myArray[myBack] = value;

myBack = newBack;

}

else

{

cerr << "\*\*\* Queue full -- can't add new value \*\*\*\n"

"Must increase value of QUEUE\_CAPACITY in Queue.h\n";

exit(1);

}

}

//Display function, displays current queue

//Precons: outstream

//Postcons: none

void Queue::display(ostream & out) const

{

for (int i = myFront; i != myBack; i = (i + 1)%QUEUE\_CAPACITY)

out << myArray[i] << " ";

cout << endl;

}

//overloaded output operator

//Precons: outstream, Queue to output

//Postcons: outstream

ostream &operator<<(ostream &out, const Queue &q)

{

q.display(out);

return out;

}

//front function, returns value at the front of the queue

//Precons: none

//Postcons: front value

QueueElement Queue::front() const

{

if ( !empty() )

return (myArray[myFront]);

else

{

cerr << "\*\*\* Queue is empty -- returning garbage value \*\*\*\n";

QueueElement garbage = 0;

return garbage;

}

}

//dequeue function, moves myFront ptr forward

//Precons: none

//Postcons: none

void Queue::dequeue()

{

if ( !empty() )

myFront = (myFront + 1) % QUEUE\_CAPACITY;

else

{

cerr << "\*\*\* Queue is empty -- "

"can't remove a value \*\*\*\n";

exit(1);

}

}

**Circqtester.cpp – DRIVER**

#include<string>

#include<iostream>

#include<fstream>

#include "Queue.h"

using namespace std;

int main(){

Queue q1;

q1.enqueue(10);

q1.enqueue(20);

q1.enqueue(30);

q1.display(cout);

cout << q1 << endl;

{

Queue q;

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

cout << "in { } for queue" << q << endl;

cout << "Testing destructor " << endl;

}

Queue q2(q1);

cout << "Testing copy constructor " << endl;

cout << q2 << endl;

Queue q3;

q3 = q2;

cout << "Testing assignment operator." << endl;

cout << q3 << endl;

cout <<"Testing front method: " << q3.front() << endl;

while(!q3.empty()){

q3.dequeue();

cout << q3 << endl;

}// end while

return 0;

}// end main

