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

Driver program to test the Queue class.

Chris Dang CSCI 1107

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

#include <iostream>

using namespace std;

#include "LQueue.h"

void print(Queue q)

{ q.display(cout); }

int main()

{

Queue q1;

cout << "Queue created. Empty? " << boolalpha << q1.empty() << endl;

cout << "How many elements to add to the queue? \n9 elements\n";

int numItems = 9;

for (int i = 1; i <= numItems; i++)

q1.enqueue(111\*i);

cout << "Contents of queue q1:\n";

q1.display(cout);

cout << endl;

cout << "\nFront of q1 is " << q1.front() << endl ;

/\* Queue q2;

q2 = q1;

cout << "Contents of queue q2 after q2 = q1:\n";

q2.display(cout) ;

cout << endl;

cout << "Queue q2 empty? " << q2.empty() << endl;

cout << "Front value in q2: " << q2.front() << endl;

\*/

while (!q1.empty())

{

cout << "Remove front -- Queue contents: ";

q1.dequeue();

cout << endl ;

q1.display(cout);

cout << endl;

}

cout << "Queue q1 empty? " << q1.empty() << endl;

cout << "Front value in q1?" << endl << q1.front() << endl;

cout << "Trying to remove front of q1: " << endl;

q1.dequeue();

}

/\*--- LQueue.cpp ----------------------------------------------------------

This file implements LQueue member functions.

Chris Dang CSCI 1107

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

#include <new>

using namespace std;

#include "LQueue.h"

//--- Definition of Queue constructor

Queue::Queue()

: myBack(0)

{}

//--- Definition of Queue copy constructor

Queue::Queue(const Queue & original)

{

myBack = 0;

if (!original.empty())

{

Queue::NodePointer first ;

// Copy first node

first = myBack = new Queue::Node(original.front());

Queue::NodePointer origPtr ;

// Set pointer to run through original's linked list

if (original.myBack->next != 0) // if there is more than 1 item in original

origPtr = original.myBack->next;

else // if there is only 1 item in original;

origPtr = original.myBack;

while (origPtr != 0)

{

myBack->next = new Queue::Node(origPtr->data);

myBack = myBack->next;

origPtr = origPtr->next;

}

}

}

//--- Definition of Queue destructor

Queue::~Queue()

{

// Set pointer to run through the queue

Queue::NodePointer prev, ptr ;

if (myBack == myBack)

prev = myBack ;

else

prev = myBack->next ;

while (prev != 0)

{

if (ptr != myBack)

ptr = prev->next;

delete prev;

if (ptr != NULL)

prev = ptr;

else

prev = 0 ;

}

}

//--- Definition of assignment operator

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

{

if (this != &rightHandSide) // check that not q = q

{

this->~Queue(); // destroy current linked list

if (rightHandSide.empty()) // empty queue

myBack = 0;

else

{ // copy rightHandSide's list

Queue::NodePointer first ;

// Copy first node

first = myBack = new Queue::Node(rightHandSide.front());

// Set pointer to run through rightHandSide's linked list

Queue::NodePointer rhsPtr = rightHandSide.myBack->next;

while (rhsPtr != rightHandSide.myBack)

{

myBack->next = new Queue::Node(rhsPtr->data);

myBack = myBack->next;

rhsPtr = rhsPtr->next;

}

myBack->next = first ; // sets back to point to first in list

}

}

return \*this;

}

//--- Definition of empty()

bool Queue::empty() const

{

return (myBack == 0);

}

//--- Definition of enqueue()

void Queue::enqueue(const QueueElement & value)

{

Queue::NodePointer newptr = new Queue::Node(value);

if (empty()) { // if list is empty

myBack = newptr;

myBack->next = myBack ;

}

else {

NodePointer first ;

if (myBack->next == myBack)

first = myBack ;

else

first = myBack->next ;

myBack->next = newptr;

myBack = newptr;

myBack->next = first ;

}

//cout << myBack ;

}

//--- Definition of display()

void Queue::display(ostream & out) const {

if (!empty()){

Queue::NodePointer ptr = myBack -> next;

do { // outputs all data except last element

out << ptr->data << " \n";

ptr = ptr -> next ;

} while (ptr != myBack);

if (myBack->next != myBack) //if only 1 item in list, list it once

out << myBack->data ; //outputs last element

}

}

//--- Definition of front()

QueueElement Queue::front() const

{

if (!empty()) {

if (myBack->next != myBack) // if there is more than 1 item in original

return (myBack->next->data);

else // if there is only 1 item in original;

return (myBack->data);

}

else

{

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

" -- returning garbage \*\*\*\n";

QueueElement \* temp = new(QueueElement);

QueueElement garbage = \*temp; // "Garbage" value

delete temp;

return garbage;

}

}

//--- Definition of dequeue()

void Queue::dequeue()

{

NodePointer first ;

if (!empty()) {

Queue::NodePointer ptr = myBack->next; // points to node that is to be deleted

first = ptr->next ; // get a new front of list

// cout << "\nDeleting ptr: " << ptr->data << endl ;

delete ptr;

if (!empty())

myBack->next = first ;

if (myBack->next == myBack) { // if there is more than 1 item in list,

myBack = 0 ;}

}

else

cerr << "\*\*\* Queue is empty -- can't remove a value \*\*\*\n";

}

/\*-- LQueue.h -------------------------------------------------------------

This header file defines a Queue data type.

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 top queue value; leaves queue unchanged

dequeue: Modifies queue by removing the value at the front

display: Displays all the queue elements

Note: Execution terminates if memory isn't available for a queue element.

Chris Dang CSCI 1107

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

#include <iostream>

#ifndef LQUEUE

#define LQUEUE

typedef int QueueElement;

class Queue

{

public:

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

/\*\*\*\*\* Constructors \*\*\*\*\*/

Queue();

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

Construct a Queue object.

Precondition: None.

Postcondition: An empty Queue object has been constructed.

(myFront and myBack are initialized to null pointers).

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

Queue(const Queue & original);

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

Copy Constructor

Precondition: original is the queue to be copied and is received

as a const reference parameter.

Postcondition: A copy of original has been constructed.

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

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

~Queue();

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

Class destructor

Precondition: None.

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

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

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

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

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

Assignment Operator

Precondition: rightHandSide is the queue to be assigned and is

received as a const reference parameter.

Postcondition: The current queue becomes a copy of rightHandSide

and a reference to it is returned.

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

bool empty() const;

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

Check if queue is empty.

Precondition: None.

Postcondition: Returns true if queue is empty and false 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 at back of queue.

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

void display(ostream & out) const;

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

Display 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 the 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 allowed to proceed.

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

private:

/\*\*\* Node class \*\*\*/

class Node

{

public:

QueueElement data;

Node \* next;

//--- Node constructor

Node(QueueElement value, Node \* link = 0)

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

Precondition: value and link are received

Postcondition: A Node has been constructed with value in its

data part and its next part set to link (default 0).

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

{ data = value; next = link; }

};

typedef Node \* NodePointer;

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

NodePointer myBack; // pointer to back of queue

}; // end of class declaration

#endif

