#pragma once

#include <iostream>

#include <string>

#include "LQueue.h"

using namespace std;

//ostream & operator<<(ostream &out, const Token &t);

class Token {

public:

Token() {

word = "";

line = 0;

}

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

Purpose: Default constructor for Token

Pre: none

Post: word is empty and line is 0

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

Token(string w, int l) {

word = w;

line = l;

}

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

Purpose: Explicit constructor for Token

Pre: word and line are assigned by parameters

Post: word and line are defined

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

string getWord(){ return word; }

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

Purpose: Returns the word

Pre: word is assigned

Post: word returned

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

void setWord(string w) { word = w; }

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

Purpose: set the word

Pre: w is a string

Post: word is set to w

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

void setLine(int l) { line = l; }

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

Purpose: set the number of lines for token

Pre: l is a positive int

Post: line = l

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

void display(ostream &out)const {

out << word;

out << lineQ;

}

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

Purpose: displays the word followed by the queue

Pre: output is valid

Post: set output

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

void queueLine(int l) {lineQ.enqueue(l);}

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

Purpose: enqueues the int into the list

Pre: line is a positive int

Post: int is enqueued

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

int getLine() { return line; }

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

Purpose: gets the int inside the Token

Pre: line is assigned

Post: line is returned

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

private:

string word;

int line; //line in text document in which the word appears

Queue<int> lineQ;

};

ostream & operator<<(ostream & out, const Token &t) {

t.display(out);

return out;

}

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

Purpose: calls display

Pre: out and token are valid

Post: returns the ostream

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

#pragma once

/\* BST.h contains the declaration of class template BST.

Basic operations:

Constructor: Constructs an empty BST

empty: Checks if a BST is empty

search: Search a BST for an item

insert: Inserts a value into a BST

remove: Removes a value from a BST

inorder: Inorder traversal of a BST -- output the data values

graph: Output a grapical representation of a BST

Private utility helper operations:

search2: Used by delete

inorderAux: Used by inorder

graphAux: Used by graph

Other operations described in the exercises:

destructor

copy constructor

assignment operator

preorder, postorder, and level-by-level traversals

level finder

Note: Execution terminates if memory isn't available for a new BST node.

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

#include <iostream>

#include "Token.h"

#include <string>

using namespace std;

class BST

{

public:

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

BST();

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

Construct a BST object.

Precondition: None.

Postcondition: An empty BST has been constructed.

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

bool empty() const;

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

Check if BST is empty.

Precondition: None.

Postcondition: Returns true if BST is empty and false otherwise.

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

bool search(string & item) const;

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

Search the BST for item.

Precondition: None.

Postcondition: Returns true if item found, and false otherwise.

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

void insert(Token & item);

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

Insert item into BST.

Precondition: None.

Postcondition: BST has been modified with item inserted at proper

position to maintain BST property.

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

void remove(Token item);

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

Remove item from BST.

Precondition: None.

Postcondition: BST has been modified with item removed (if present);

BST property is maintained.

Note: remove uses auxiliary function search2() to locate the node

containing item and its parent.

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

void inorder(ostream & out) const;

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

Inorder traversal of BST.

Precondition: ostream out is open.

Postcondition: BST has been inorder traversed and values in nodes

have been output to out.

Note: inorder uses private auxiliary function inorderAux().

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

void graph(ostream & out) const;

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

Graphic output of BST.

Precondition: ostream out is open.

Postcondition: Graphical representation of BST has been output to out.

Note: graph() uses private auxiliary function graphAux().

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

private:

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

class BinNode

{

public:

Token data;

BinNode \* left;

BinNode \* right;

// BinNode constructors

// Default -- data part is default DataType value; both links are null.

BinNode()

: left(0), right(0)

{}

// Explicit Value -- data part contains item; both links are null.

BinNode(Token item)

: data(item), left(0), right(0)

{}

};// end of class BinNode declaration

typedef BinNode \* BinNodePointer;

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

void search2(Token & item, bool & found,

BinNodePointer & locptr, BinNodePointer & parent) const;

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

Locate a node containing item and its parent.

Precondition: None.

Postcondition: locptr points to node containing item or is null if

not found, and parent points to its parent.#include <iostream>

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

void inorderAux(ostream & out,

BinNodePointer subtreePtr) const;

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

Inorder traversal auxiliary function.

Precondition: ostream out is open; subtreePtr points to a subtree

of this BST.

Postcondition: Subtree with root pointed to by subtreePtr has been

output to out.

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

void graphAux(ostream & out, int indent,

BinNodePointer subtreeRoot) const;

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

Graph auxiliary function.

Precondition: ostream out is open; subtreePtr points to a subtree

of this BST.

Postcondition: Graphical representation of subtree with root pointed

to by subtreePtr has been output to out, indented indent spaces.

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

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

BinNodePointer myRoot;

}; // end of class template declaration

//--- Definition of constructor

BST::BST(){

myRoot = NULL;

}

//--- Definition of empty()

bool BST::empty() const

{

return myRoot == 0;

}

//--- Definition of search()

bool BST::search(string & item) const{

BinNodePointer locptr = myRoot;

bool found = false;

while (!found && locptr != NULL)

{

if (item < locptr->data.getWord()) { // descend left

locptr = locptr->left;

}

else if (locptr->data.getWord() > item) {// descend right

locptr = locptr->right;

}

else // item found

found = true;

}

return found;

}

//--- Definition of insert()

void BST::insert(Token & item)

{

BinNodePointer

locptr = myRoot, // search pointer

parent = 0; // pointer to parent of current node

bool found = false; // indicates if item already in BST

while (!found && locptr != 0)

{

parent = locptr;

if (item.getWord() < locptr->data.getWord()) // descend left

locptr = locptr->left;

else if (locptr->data.getWord() < item.getWord()) // descend right

locptr = locptr->right;

else // item found

found = true;

}

if (!found)

{ // construct node containing item

locptr = new BinNode(item);

if (parent == 0) // empty tree

myRoot = locptr;

else if (item.getWord() < parent->data.getWord()) // insert to left of parent

parent->left = locptr;

else // insert to right of parent

parent->right = locptr;

}

else {//found

locptr->data.queueLine(item.getLine());

}

}

//--- Definition of remove()

void BST::remove(Token item)

{

bool found; // signals if item is found

BinNodePointer

x, // points to node to be deleted

parent; // " " parent of x and xSucc

search2(item, found, x, parent);

if (!found)

{

cout << "Item not in the BST\n";

return;

}

//else

if (x->left != 0 && x->right != 0)

{ // node has 2 children

// Find x's inorder successor and its parent

BinNodePointer xSucc = x->right;

parent = x;

while (xSucc->left != 0) // descend left

{

parent = xSucc;

xSucc = xSucc->left;

}

// Move contents of xSucc to x and change x

// to point to successor, which will be removed.

x->data = xSucc->data;

x = xSucc;

} // end if node has 2 children

// Now proceed with case where node has 0 or 2 child

BinNodePointer

subtree = x->left; // pointer to a subtree of x

if (subtree == 0)

subtree = x->right;

if (parent == 0) // root being removed

myRoot = subtree;

else if (parent->left == x) // left child of parent

parent->left = subtree;

else // right child of parent

parent->right = subtree;

delete x;

}

//--- Definition of inorder()

void BST::inorder(ostream & out) const

{

inorderAux(out, myRoot);

}

//--- Definition of graph()

void BST::graph(ostream & out) const

{

graphAux(out, 0, myRoot);

}

//--- Definition of search2()

void BST::search2(Token & item, bool & found,

BinNodePointer & locptr,

BinNodePointer & parent) const

{

locptr = myRoot;

parent = 0;

found = false;

while (!found && locptr != 0)

{

if (item.getWord() < locptr->data.getWord()) // descend left

{

parent = locptr;

locptr = locptr->left;

}

else if (locptr->data.getWord() < item.getWord()) // descend right

{

parent = locptr;

locptr = locptr->right;

}

else // item found

found = true;

}

}

//--- Definition of inorderAux()

void BST::inorderAux(ostream & out,

BinNodePointer subtreeRoot) const

{

if (subtreeRoot != 0)

{

inorderAux(out, subtreeRoot->left); // L operation

out << subtreeRoot->data << " "; // V operation

inorderAux(out, subtreeRoot->right); // R operation

}

}

//--- Definition of graphAux()

#include <iomanip>

void BST::graphAux(ostream & out, int indent,

BinNodePointer subtreeRoot) const

{

if (subtreeRoot != 0)

{

graphAux(out, indent + 8, subtreeRoot->right);

out << setw(indent) << " " << subtreeRoot->data << endl;

graphAux(out, indent + 8, subtreeRoot->left);

}

}

#include "LQueue.h"

#include "BST.h"

#include <iostream>

#include <string>

#include <fstream>

#include "Token.h"

using namespace std;

const int SIZE = 26;//number of characters in the alphabet

void readFile(ifstream & inputFile, BST tree[]);

void displayConcord(BST tree[]);

int main() {

BST concord[SIZE];

ifstream infile;

infile.open("binaryTreeRead.txt");

if (!infile) {//check file opens

cerr << "File did not open correctly, exiting program" << endl;

exit(1107);

}

readFile(infile, concord);

displayConcord(concord);

return 0;

}

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

Purpose: Read characters from file and puts into string once a space is

reached, then input string and line number into tree

Pre: file opens correctly

Post: File is read and all words have been put into the tree

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

void readFile(ifstream & inputFile, BST tree[]) {

string inputString;

Token inToken;

int lineNum = 0;//increments the number of newline characters

char stringChar;

while (inputFile.get(stringChar)) {

if (stringChar == '\n')

lineNum++;//increment line number each newline

else if (stringChar == ' ') {//space indicates end of word

inToken = Token(inputString, lineNum); //sets token to string and line value

inToken.queueLine(lineNum);

tree[toupper(inputString[0] - 'A')].insert(inToken);

inputString = ""; //reset string

}

else if (isalpha(stringChar))

inputString += toupper(stringChar);

}

}

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

Purpose: Displays each bst

Pre: BST is a filled tree

Post: display of bsts

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

void displayConcord(BST tree[]) {

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

cout << static\_cast<char>('A'+i) << ": ";//displays character value of tree

tree[i].inorder(cout);

cout << endl;

}

}

#pragma once

/\*--- LQueue.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

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

#include <iostream>

#include <iomanip>

using namespace std;

const int QUEUE\_CAPACITY = 10;

template <typename QueueElement>

class Queue {

public:

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

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

Queue();

const Queue(const Queue & original);

~Queue();

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

bool empty() const;

void enqueue(const QueueElement & value);

QueueElement front() const;

void display(ostream & out) const;

void dequeue();

private:

/\*\*\*\*\* Node Class \*\*\*\*\*/

class Node {

public:

QueueElement data;

Node \*next;

//---Node constructor

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

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

Precondition: value and link are recieved

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

template <typename QueueElement>

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

template <typename QueueElement>

Queue<QueueElement>::Queue() {

myBack = NULL;

}

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

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.

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

template <typename QueueElement>

Queue<QueueElement>::Queue(const Queue & original) {

myBack = NULL;

if (!original.empty()) {

//Copy first node

NodePointer cur = original.myBack->next;

do {

this->enqueue(cur->data);

cur = cur->next;

} while (cur != original.myBack->next);

}

}

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

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.

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

template <typename QueueElement>

Queue<QueueElement>::~Queue() {

while (!empty())//goes through whole list and dequeues

dequeue();

}

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

Class destructor

Precondition: None.

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

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

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

template <typename QueueElement>

const Queue<QueueElement> & Queue<QueueElement>::operator=(const Queue<QueueElement> & rightHandSide) {

if (this != &rightHandSide) {

this->~Queue();

if (rightHandSide.empty())

myBack = NULL;

else {

NodePointer rhsPtr = rightHandSide.myBack->next;

do {

this->enqueue(rhsPtr->data);

rhsPtr = rhsPtr->next;

} while (rhsPtr != rightHandSide.myBack->next);

}

}

return \*this;

}

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

Assignment Operator

Precondition: rightHandSide is the queue to be assigned and is recieved

as a const reference parameter.

Postcondition: The current queue becomes a copy of rightHandSide

and a reference to it is returned

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

template <typename QueueElement>

bool Queue<QueueElement>::empty() const

{

return (myBack == NULL);

}

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

Check if queue is empty.

Precondition: None.

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

returned otherwise.

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

template <typename QueueElement>

void Queue<QueueElement>::enqueue(const QueueElement & value)

{

NodePointer newptr = new Node(value);

if (empty()) {

myBack = newptr;

myBack->next = myBack;

}

else {

newptr->next = myBack->next;

myBack->next = newptr;

myBack = newptr;

}

}

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

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.

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

template <typename QueueElement>

void Queue<QueueElement>::display(ostream & out) const

{

NodePointer ptr = myBack->next;

do

{

out << showpoint << setprecision(1) << fixed << " -> " << ptr->data;

ptr = ptr->next;

} while (ptr != myBack->next);

}

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

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.

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

template <typename QueueElement>

QueueElement Queue<QueueElement>::front() const

{

if (!empty())

return myBack->next->data;

else

{

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

QueueElement garbage = 0;

return garbage;

}

}

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

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.

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

template <typename QueueElement>

void Queue<QueueElement>::dequeue()

{

if (!empty()) {

NodePointer ptr = myBack->next;//set pointer to front

if (ptr == myBack) {

myBack = NULL;

delete ptr;

}

else {

myBack->next = myBack->next->next;

delete ptr;

}

}

else

cerr << "List is empty" << endl;

}

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

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.

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

template <typename QueueElement>

ostream & operator<<(ostream &out, const Queue<QueueElement> &q) {

q.display(out);

return out;

}

