Author: Michael LaPan
Class: ADV c++
Assignment: Program8

Table of contents:

orderLinkedList.h

```
//*******************************************
// The ListNode class creates a type used to                    *
// store a node of the linked list.                              *
// PRECONDITIONS:                                                *
//   Choice for ItemType implements 'cout'                       *
//   as well as "==" and "<" operators                            *
//*******************************************
#ifndef OrderOrderLinkedList_H
#define OrderOrderLinkedList_H

template <class ItemType>
class ListNode
{
public:
        ItemType info;              // Node value
        ListNode<ItemType> *next;    // Pointer to the next node

        // Constructor
        ListNode (ItemType nodeValue)
        {
                info = nodeValue;
                next = NULL;
        }
};

//*******************************************
// OrderLinkedList class                                        *
//*******************************************

template <class ItemType>
class OrderLinkedList
{
private:
        ListNode<ItemType> *head;       // List head pointer
        ListNode<ItemType> *currentPos;  // Pointer to "current" list item
        int length;                  // Length

public:
        OrderLinkedList();                                  // Constructor
        ~OrderLinkedList();                                  // Destructor
        OrderLinkedList( const OrderLinkedList<ItemType>& anotherList ); // Copy constructor
        void operator= ( const OrderLinkedList<ItemType>& );          // Assignment op

        void insertNode(ItemType);
        void deleteNode(ItemType);
        bool searchList(ItemType& item);

        int getLength();
        void displayList();

        void resetList();
        ItemType getNextItem();       // Iterator
        bool atEnd();
};

//*************************************************
```

```
// Constructor                                                  *
// Initial list head pointer and length                         *
//***************************************************
template <class ItemType>
OrderLinkedList<ItemType>::OrderLinkedList()
{
        head = NULL;
        length = 0;
}

//***************************************************
// displayList shows the value stored in each node              *
// of the linked list pointed to by head.                        *
// Precondition: "cout" operator enabled for                     *
// ItemType data type.                                          *
//***************************************************

template <class ItemType>
void OrderLinkedList<ItemType>::displayList()
{
        ListNode<ItemType> *nodePtr;

        nodePtr = head;
        while (nodePtr != NULL)
        {
                cout << nodePtr->info << endl;
                nodePtr = nodePtr->next;
        }
}
//***************************************************
// The insertNode function inserts a node with                  *
// newValue copied to its value member.                         *
//***************************************************

template <class ItemType>
void OrderLinkedList<ItemType>::insertNode(ItemType newValue)
{
        //Michael Edit!!
        //had to set previous new to null.
        ListNode<ItemType> *newNode, *nodePtr, *previousNode = NULL;

        // Allocate a new node & store newValue
        newNode = new ListNode<ItemType>(newValue);

        // If there are no nodes in the list
        // make newNode the first node
        if (head == NULL)
        {
                head = newNode;
                newNode->next = NULL;
        }
        else   // Otherwise, insert newNode
        {
                // Initialize nodePtr to head of list and previousNode to NULL.
                nodePtr = head;
                previousNode = NULL;

                // Skip all nodes whose value member is less
```

```
                // than newValue.
                while (nodePtr != NULL && nodePtr->info < newValue)
                {
                        previousNode = nodePtr;
                        nodePtr = nodePtr->next;
                }

                // If the new node is to be the 1st in the list,
                // insert it before all other nodes.
                if (previousNode == NULL)
                {
                        head = newNode;
                        newNode->next = nodePtr;
                }
                else    // Otherwise, insert it after the prev. node.
                {
                        previousNode->next = newNode;
                        newNode->next = nodePtr;
                }
        }
        length++;
}

//**************************************************
// The deleteNode function searches for a node                                  *
// with searchValue as its value. The node, if found,                   *
// is deleted from the list and from memory.                            *
//**************************************************

template <class ItemType>
void OrderLinkedList<ItemType>::deleteNode(ItemType searchValue)
{
        ListNode<ItemType> *nodePtr, *previousNode = NULL;

        // If the list is empty, do nothing.
        if (!head)
                return;

        // Determine if the first node is the one.
        if (head->info == searchValue)
        {
                nodePtr = head->next;
                delete head;
                head = nodePtr;
        }
        else
        {
                // Initialize nodePtr to head of list
                nodePtr = head;

                // Skip all nodes whose value member is
                // not equal to searchValue.
                while (nodePtr != NULL && nodePtr->info != searchValue)
                {
                        previousNode = nodePtr;
                        nodePtr = nodePtr->next;
                }
```

```
                // If nodePtr is not at the end of the list,
                // link the previous node to the node after
                // nodePtr, then delete nodePtr.
                if (nodePtr)
                {
                        previousNode->next = nodePtr->next;
                        delete nodePtr;
                }
        }
        length--;
}

//************************************************
// Linear search                                 *
// Post: If found, item's key matches an element's *
// key in the list and a copy of that element has *
// been stored in item; otherwise, item is       *
// unchanged.  Return value is boolean to indicate *
// status of search.                             *
//************************************************

template <class ItemType>
bool OrderLinkedList<ItemType>::searchList(ItemType& item)
{
        bool moreToSearch;
        ListNode<ItemType>* nodePtr;

        nodePtr = head;             // Start search from head of list
        bool found = false;         // Assume value not found
        moreToSearch = (nodePtr != NULL);

        while (moreToSearch && !found)
        {
                if (nodePtr->info < item)
                {
                        nodePtr = nodePtr->next;
                        moreToSearch = (nodePtr != NULL);
                }
                else if (item == nodePtr->info)
                {
                        found = true;
                        item = nodePtr->info;
                }
                else
                        moreToSearch = false;
        }
        return found;
}

//************************************************
// Iterator reset function                       *
// Resets pointer of current item in list to the  *
// head of the list.                              *
//************************************************

template <class ItemType>
void OrderLinkedList<ItemType>::resetList()
        // Post: Current position has been initialized.
```

```
{
	currentPos = head;
}

//*************************************************
// Function: Gets the next element in list as
//	referenced by currPtr
// Pre:  Current position is defined.
//	Element at current position is not last in list.
// Post: Current position is updated to next position.
//	item is a copy of element at current position.
//*************************************************
template <class ItemType>
ItemType OrderLinkedList<ItemType>::getNextItem()
{
	ItemType item;

	if (currentPos == NULL)
		currentPos = head;  // wrap if getnext is called at past-end
	//else
	item = currentPos->info;
	currentPos = currentPos->next;

	return item;
}

//*************************************************
// Observer function to return current list length	*
//*************************************************
template <class ItemType>
int OrderLinkedList<ItemType>::getLength()
{
	return length;
}

//*************************************************
// Observer function to determine if current		*
// is the end of the list				*
//*************************************************
template <class ItemType>
bool OrderLinkedList<ItemType>::atEnd()
{
	if (currentPos == NULL)
		return true;
	else
		return false;
}

//*************************************************
// Copy Constructor					*
//*************************************************
template<class ItemType>
OrderLinkedList<ItemType>::OrderLinkedList( const OrderLinkedList<ItemType>& anotherList )
{
	ListNode<ItemType>* ptr1;
	ListNode<ItemType>* ptr2;

	if (anotherList.head == NULL)
```

```
                head = NULL;
        else
        {
                head = new ListNode<ItemType>(anotherList.head->info);
                ptr1 = anotherList.head->next;
                ptr2 = head;
                while (ptr1 != NULL)
                {
                        ptr2->next = new ListNode<ItemType>(ptr1->info);
                        ptr2 = ptr2->next;
                        ptr1 = ptr1->next;
                }
                ptr2->next = NULL;
        }
        length = anotherList.length;
}

//************************************************
// Overloaded Assignment Operator                                *
//************************************************
template<class ItemType>
void OrderLinkedList<ItemType>::operator=( const OrderLinkedList<ItemType>& anotherList )
{
        ListNode<ItemType>* ptr1;
        ListNode<ItemType>* ptr2;

        if (anotherList.head == NULL)
                head = NULL;
        else
        {
                head = new ListNode<ItemType>(anotherList.head->info);
                ptr1 = anotherList.head->next;
                ptr2 = head;
                while (ptr1 != NULL)
                {
                        ptr2->next = new ListNode<ItemType>(ptr1->info);
                        ptr2 = ptr2->next;
                        ptr1 = ptr1->next;
                }
                ptr2->next = NULL;
        }
        length = anotherList.length;
}

//************************************************
// Destructor                                                    *
// This function deletes every node in the list.                 *
//************************************************

template <class ItemType>
OrderLinkedList<ItemType>::~OrderLinkedList()
{
        ListNode<ItemType> *nodePtr, *nextNode;

        nodePtr = head;
        while (nodePtr != NULL)
        {
                nextNode = nodePtr->next;
```

```
                delete nodePtr;
                nodePtr = nextNode;
        }
}

#endif
```

Problem.h

```
#include <string>
#include <ostream>
#include <iostream>


using namespace std;

/*
*problem class
*this class will hold information for one help
*desk problem
*/
class problem
{
public:
	problem();//constructor
private:
	int problemCode;
	int criticality;
	string Date;
	string desc;
	string lName;
	string fName;
	//--------------------------------
	//i have a bug or in my code
	//but every other item is blank
	//couldnt find the bug, but its the reason
	//why these are 54 not 27
	//--------------------------------
	int probCodeArray[54];//holds the problem code
	string probCodeDiscArray[54];//holds the discription for the problem code
public:
	void setProbCode(int toSet);//set
	int getProbCode();//get
	void setCriticality(int toSet);//set
	int getCriticality();//get
	void setDate(string toSet);//set
	string getDate();//get
	void setdesc(string toSet);//set
	string getConcat();//get
	string getLName();//get
	string getFName();//get
	void setFName(string toSet);//set
	void setLName(string toSet);//set
	bool operator< (const problem& toTest);//overloading of <
	bool operator== (const problem& isEql);//overloading of ==
	bool operator!= (const problem& isEql);//overloading of !=
	void popArray();//populateing problem code array
	void addDisc();//add decription for each problem
```

```
	/*
	*extending (friend of the class) of <<
	*this one through me for a loop, but with some googling i got it!
	*i pass in the stream? and then the object to print
	*then useing the stream i pass in i output how the object should
	*be printted with cout.
	*/
	friend ostream &operator<<(ostream &output, problem &D)
	{
		output.clear();
		output << "    " << D.criticality << "     " << D.Date << "  " << D.fName << " " << D.lName << "  " << D.desc;
		return output;
	}
};
```

ProblemList.h

```
#include <string>
#include <iostream>
#include <ostream>
using namespace std;

#include "orderLinkedList.h"
#include "problem.h"

/*
*problemList class
*the problem list class holds all of the problems
*and acts as sort of the manager class
*/
class ProblemList
{

private:
	OrderLinkedList<problem> head;	// List head pointer
public:
	ProblemList(string toRead);//constructor
	ProblemList ProblemList::operator+= (ProblemList& toAdd);//overloaded +=
	ProblemList ProblemList::operator-= (ProblemList& toAdd);//overloaded -=
	int getLen();//returns the len of the linked list
	void writeTop(int numToWrite);//writed the top
	void writeBottom(int numToWrite);//writed the bottem
	OrderLinkedList<problem> getHead();//returns the head
};
```

Driver.cpp

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

#include "ProblemList.h"

/*
**********************
*MADE BY MICHAEL LApAN
**********************
*this program is a help desk mock up
*it proccesses helkp desk tickets,
*sorts and stores them. then shows top
*25 then bottem 25
*/
int main()
{
	ProblemList problems("problems.txt");

	ProblemList newProblems("newproblems.txt");

	ProblemList solvedProblems("resolvedproblems.txt");

		problems += newProblems;
		problems -= solvedProblems;
		//couldnt easily incorporate this into class
		cout << "Priority  Submitted     By      Description" << endl;
		problems.writeTop(25);
		system("pause");
		//couldnt easily incorporate this into class
		cout << "Priority  Submitted     By      Description" << endl;
		problems.writeBottom(25);
	system("pause");
}
```

Problem.cpp

```
#include "problem.h"
#include <fstream>
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

//constructor
problem::problem()
{
	problemCode = 0;//set to 0
	criticality = 0;// set to 0
	popArray();//read the file into array
}

//overloading of ==
//if obj passed in equals this return true
bool problem::operator== (const problem& isEql)
{
	//everything must equal this
	if (isEql.criticality == criticality && isEql.Date == Date
		&& isEql.fName == fName && isEql.lName == lName
		&& isEql.problemCode == problemCode)
	{
		return true;
	}
	return false;
}

//adddisc
//adds the discription for each problem code
void problem::addDisc()
{
	//loop through array looking for correct problem code
	for (size_t i = 0; i < 54; i++)
	{
		if (probCodeArray[i] == problemCode)
		{
			desc = probCodeDiscArray[i + 1];
		}
	}
}

//reads everything from the file into the array
void problem::popArray()
{
	int n = 0;
	int t = 0;
	string toParse;
	ifstream wordFile2("problemlist.txt");

	//if file not found
	if (wordFile2.fail())
	{
		cout << "Problem opening document file";
		exit(-1);
	}
```

```
	// Build list of words in document
	getline(wordFile2, toParse);	// Get first word
	while (!wordFile2.eof())
	{
		stringstream ss(toParse);
		string s;

		//split at -
		while (getline(ss, s, '-')) {
			if (t == 0)
			{
				probCodeArray[n] = atoi(s.c_str());
				t++;
			}
			else if (t == 1)
			{
				probCodeDiscArray[n] = s;
				t = 0;
			}
			n++;
		}
		getline(wordFile2, toParse);	// Get second word
	}
	wordFile2.close();
}
//overloading of !=
//if obj passed in does not equals any part of
//this. return true
bool problem::operator!= (const problem& isEql)
{
	if (isEql.criticality != criticality || isEql.Date != Date
		|| isEql.fName != fName || isEql.lName != lName
		|| isEql.problemCode != problemCode)
	{
		return true;
	}
	return false;
}

//overloading of the < operator
//if criticality of obj getting passed in is
//>then this criticality then return true
bool problem::operator< (const problem& toTest)
{
	if (toTest.criticality == criticality)
	{
		if (toTest.Date > Date)
		{
			return true;
		}
		else
		{
			return false;
		}
	}
	else
	{
```

```
                        if (toTest.criticality >= criticality)
                        {
                                return true;
                        }
                        else
                        {
                                return false;
                        }
                }
}

//set problem code
void problem::setProbCode(int toSet)
{
        problemCode = toSet;
        addDisc();
}

//return problemcode
int problem::getProbCode()
{
        return problemCode;
}

//set criticality
void problem::setCriticality(int toSet)
{
        criticality = toSet;
}

//get criticality
int problem::getCriticality()
{
        return criticality;
}

//set date
void problem::setDate(string toSet)
{
        Date = toSet;
}

//return date
string problem::getDate()
{
        return Date;
}

//set desc
void problem::setdesc(string toSet)
{
        desc = toSet;
}

//return desc
string problem::getConcat()
{
```

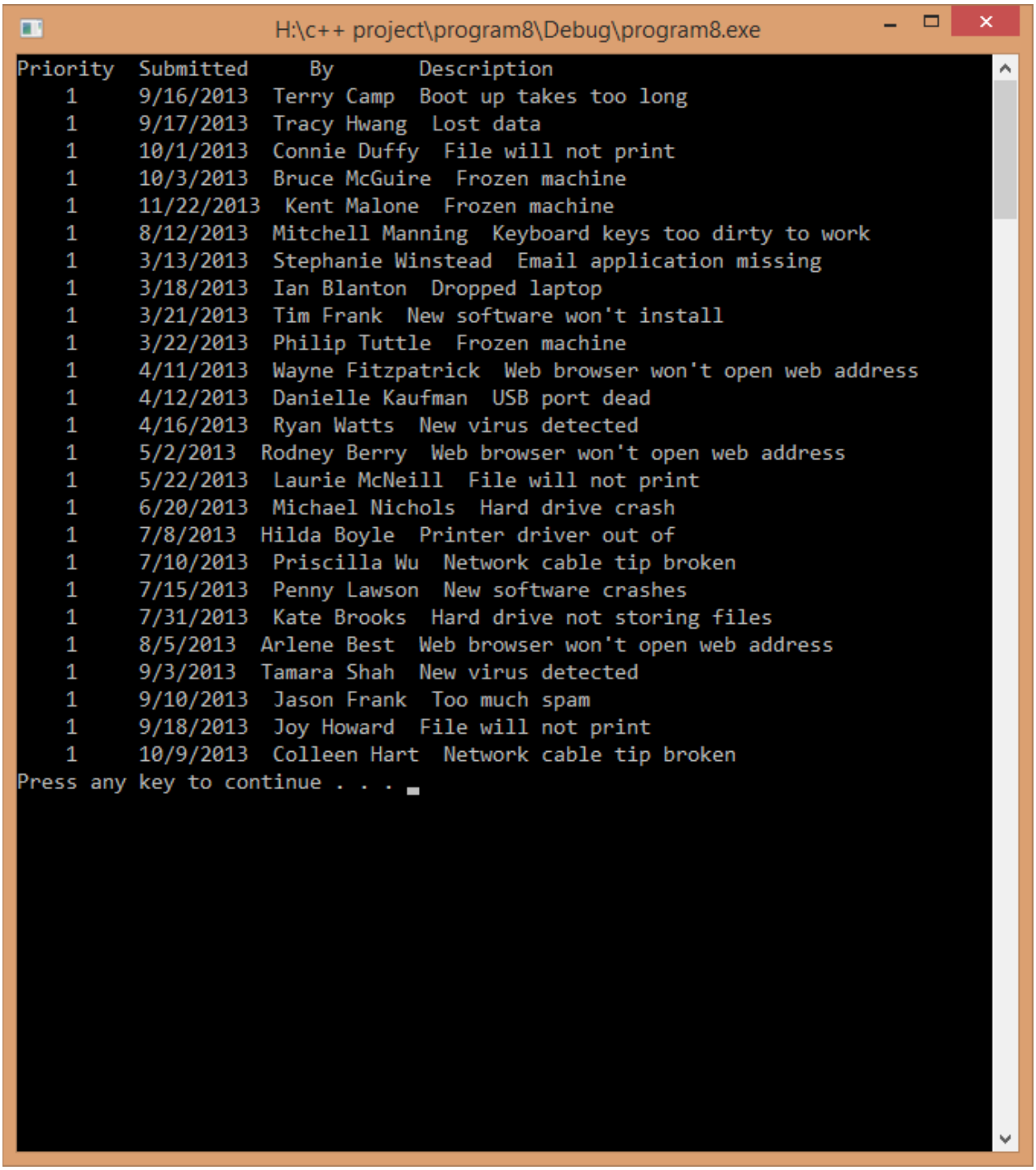
```
	return desc;
}

//return last name
string problem::getLName()
{
	return lName;
}

//return first name
string problem::getFName()
{
	return fName;

}

//set first name
void problem::setFName(string toSet)
{
	fName = toSet;
}

//set last name
void problem::setLName(string toSet)
{
	lName = toSet;
}
```

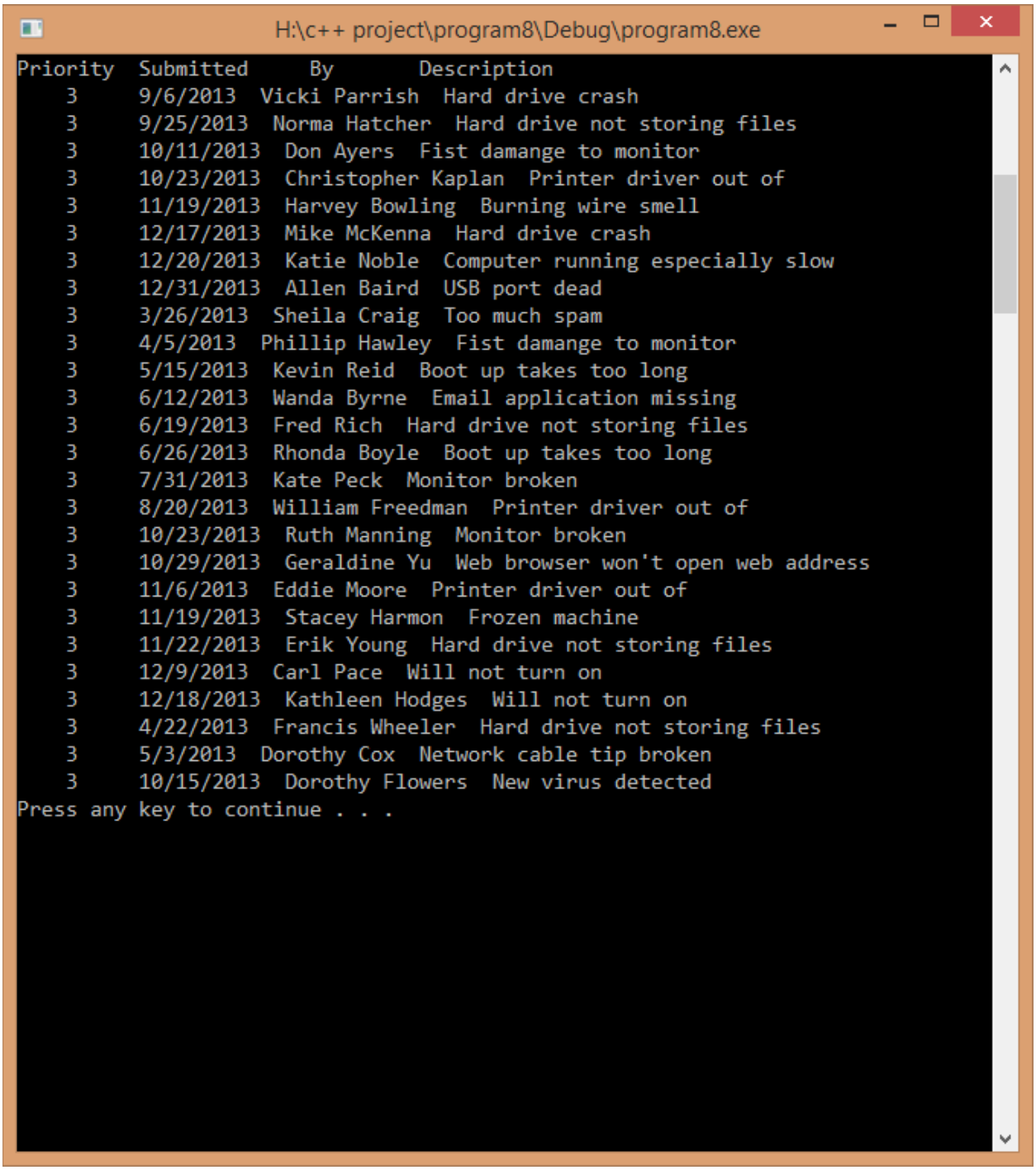
ProblemList.cpp

```
#include <fstream>
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

#include "ProblemList.h"
#include "orderLinkedList.h"

/*
*problem list constructor
*reads the file and puts it into a llinked list of problems
*and populates the array
*/
ProblemList::ProblemList(string toRead)
{
	string aWord;
	problem prob;
	int i = 0;
	ifstream wordFile(toRead);
	//if file not found
	if (wordFile.fail())
	{
		cout << "Problem opening document file";
		exit(-1);
	}

	// Build list of words in document
	wordFile >> aWord;		// Get first word
	while (!wordFile.eof())
	{
		stringstream ss(aWord);
		string s;

		//split word appart at the comma
		while (getline(ss, s, ',')) {

			//i then count the commas and assign everything.
			if (i == 0)
			{
				prob.setProbCode(atoi(s.c_str()));
			}
			else if (i == 1)
			{
				prob.setCriticality(atoi(s.c_str()));
			}
			else if (i == 2)
			{
				prob.setDate(s);
			}
			else if (i == 3)
			{
				prob.setFName(s);
			}
			else if (i == 4)
			{
				prob.setLName(s);
```

```
                                        i = -1;
                                }
                                i++;
                        }
                        head.insertNode(prob);

                        wordFile >> aWord;          // Get next word

                }

                wordFile.close();//close the file
}

//returns the length of the linked list
int ProblemList::getLen()
{
        return head.getLength();
}

//returns the head of the linked list
OrderLinkedList<problem> ProblemList::getHead()
{
        return head;
}

/*
*write top
*writes the top of the list. using a user defined number
*/
void ProblemList::writeTop(int numToWrite)
{
        for (size_t i = 0; i < numToWrite; i++)
        {
                //advance till users num
                cout << head.getNextItem() << endl;
        }
}

//writes the bottem of the list
//using user defined number
void ProblemList::writeBottom(int numToWrite)
{
        int len = head.getLength();

        //the number i use here is 155.
        //for some reason genlength returns the wrong number
        //it returns 205 not the real length of 155
        for (size_t i = 0; i <= 155; i++)
        {
                if (i >= 155 - numToWrite)
                {
                        cout << head.getNextItem() << endl;
                }
                head.getNextItem();
        }
}

//overloading of the += operator
```

```
ProblemList ProblemList::operator+= (ProblemList& toAdd)
{
	//assign the list to add to a temp.
	OrderLinkedList<problem> temp = toAdd.getHead();	// List head pointer

	//reset it, to ensure starting from the front
	temp.resetList();
	for (size_t i = 0; i < toAdd.getLen(); i++)
	{
		//insert the new item into the list
		head.insertNode(temp.getNextItem());
	}
	return toAdd;
}

//overloading of the -= operator
ProblemList ProblemList::operator-= (ProblemList& toAdd)
{
	//assign the list to add to a temp.
	OrderLinkedList<problem> temp = toAdd.getHead();	// List head pointer

	//reset it, to ensure starting from the front
	temp.resetList();
	for (size_t i = 0; i < toAdd.getLen(); i++)
	{
		//delete item in oldlist
		head.deleteNode(temp.getNextItem());
	}
	return toAdd;
}
```


H:\c++ project\program8\Debug\program8.exe
Priority Submitted By Description
1 9/16/2013 Terry Camp Boot up takes too long
1 9/17/2013 Tracy Hwang Lost data
1 10/1/2013 Connie Duffy File will not print
1 10/3/2013 Bruce McGuire Frozen machine
1 11/22/2013 Kent Malone Frozen machine
1 8/12/2013 Mitchell Manning Keyboard keys too dirty to work
1 3/13/2013 Stephanie Winstead Email application missing
1 3/18/2013 Ian Blanton Dropped laptop
1 3/21/2013 Tim Frank New software won't install
1 3/22/2013 Philip Tuttle Frozen machine
1 4/11/2013 Wayne Fitzpatrick Web browser won't open web address
1 4/12/2013 Danielle Kaufman USB port dead
1 4/16/2013 Ryan Watts New virus detected
1 5/2/2013 Rodney Berry Web browser won't open web address
1 5/22/2013 Laurie McNeill File will not print
1 6/20/2013 Michael Nichols Hard drive crash
1 7/8/2013 Hilda Boyle Printer driver out of
1 7/10/2013 Priscilla Wu Network cable tip broken
1 7/15/2013 Penny Lawson New software crashes
1 7/31/2013 Kate Brooks Hard drive not storing files
1 8/5/2013 Arlene Best Web browser won't open web address
1 9/3/2013 Tamara Shah New virus detected
1 9/10/2013 Jason Frank Too much spam
1 9/18/2013 Joy Howard File will not print
1 10/9/2013 Colleen Hart Network cable tip broken
Press any key to continue . . .


H:\c++ project\program8\Debug\program8.exe
Priority Submitted By Description
3 9/6/2013 Vicki Parrish Hard drive crash
3 9/25/2013 Norma Hatcher Hard drive not storing files
3 10/11/2013 Don Ayers Fist damange to monitor
3 10/23/2013 Christopher Kaplan Printer driver out of
3 11/19/2013 Harvey Bowling Burning wire smell
3 12/17/2013 Mike McKenna Hard drive crash
3 12/20/2013 Katie Noble Computer running especially slow
3 12/31/2013 Allen Baird USB port dead
3 3/26/2013 Sheila Craig Too much spam
3 4/5/2013 Phillip Hawley Fist damange to monitor
3 5/15/2013 Kevin Reid Boot up takes too long
3 6/12/2013 Wanda Byrne Email application missing
3 6/19/2013 Fred Rich Hard drive not storing files
3 6/26/2013 Rhonda Boyle Boot up takes too long
3 7/31/2013 Kate Peck Monitor broken
3 8/20/2013 William Freedman Printer driver out of
3 10/23/2013 Ruth Manning Monitor broken
3 10/29/2013 Geraldine Yu Web browser won't open web address
3 11/6/2013 Eddie Moore Printer driver out of
3 11/19/2013 Stacey Harmon Frozen machine
3 11/22/2013 Erik Young Hard drive not storing files
3 12/9/2013 Carl Pace Will not turn on
3 12/18/2013 Kathleen Hodges Will not turn on
3 4/22/2013 Francis Wheeler Hard drive not storing files
3 5/3/2013 Dorothy Cox Network cable tip broken
3 10/15/2013 Dorothy Flowers New virus detected
Press any key to continue . . .