University of Washington Bothell

CSS 342: Data Structures, Algorithms, and Discrete Mathematics

Program 3: Linked Lists

Purpose

This programming assignment will require the usage of dynamic memory, pointers and linked lists. It will also require the usage of File IO and templates. Also, required is a good understanding of overloading assignment and copy constructors. Learnings from previous assignments, especially operator overloading will also be required.

Problem: The List.

You will build a class for a fully ordered list. The list class, called List342, will be templatized so that different types of objects can be stored in it.

```
List342<string> some_strings;
List342<int> list_of_integers;
List342<Child> children;
List342<MyRandomObj> ordered_list_random_objs;
```

The following member functions should be created for the List342 class. **Please make sure the signatures are exact or points will be deducted as the test programs will not compile.**

There is code in this document which you should test your program with to make sure it works. These test cases are not exhaustive but if these test cases do not build the grader will not be able to test your code.

Please make sure that the proper constructors/destructors are created for the class.

bool BuildList(string file_name) : Given a string which represents a file, open the file, read the objects from file, and build an ordered list. Note that BuildList puts the responsibility on the Object class to know how to read from a file. That is, do not have object specific logic in the implementation. You can insert each item into the list as you read it from the file. If a list already has an item do not add it to the list; only add the new items to the list.

Assume that the text in the file is well-formed for the object being read into.

bool Insert(T* obj) : Insert an object in the list in the correct place. Return true or false depending on whether the insert was successful. Duplicates of an object are not allowed and should return false. Note that a pointer to the object is passed in but the Insert should create a copy of the object to keep in the list.

bool Remove(T target, T& result): Remove the target item from the list. Return the item using a second parameter that is passed in by reference. If the item was found and removed return true, else return false.

bool Peek(T target, T& result) const: The same as Remove except the item is not removed from the list. Again, the second item is returned by reference.

int Size() const: return the number of items in the list.

void DeleteList(): Remove all items from the list. Deallocate all memory appropriately. This includes not only the nodes in the list but the items being pointed to by the nodes.

bool Merge(List342& list1): Takes a sorted list and merges into the calling sorted list (no new memory should be allocated). At termination of function, the list passed in (list1) should be empty (unless it is the calling list). No duplicates are allowed.

The following operators should be overloaded for the List342<>. Please make sure signatures are correct on overloads.

+, += : This should add two lists together. The lists are assumed to be sorted and the returned list must also be sorted. Use an efficient sort algorithm and avoid unnecessary data allocations. Duplicates are not allowed and expected.

<< : Display the list and only the List object, no extra blanks, no tabs, no endl. Display the items on the list in increasing order. See examples below.

== and **!=** : check for equality or inequality.

= : Assignment. Make a deep copy of all new memory.

Details on testing:

We will utilize multiple different types to test the list. For instance, we will create a list of ints and a list of strings. As List342<> is templatized we will also test with a set of objects we create. For instance, I will test your list with a Bird class—I will make sure I have the appropriate operators overloaded in that class so that it will work with your list.

In order for you to test your List342<> I have some code below which utilizes a Child class.

The child class contains a first name (string), last name (string), and age (int). Stream input, >>, is of the form "firstname lastname age" in the file. Assume the test in the file is well formed w/o errors.

Here is an example file:

George Washington 12
Lyndon Johnson 12
Joseph Smith 4
Thomas Paine 17
Thomas Paine 17

The Child class compares names alphabetically by last name first; first name second; and then age. A child is considered a duplicate if all three fields (firstName, lastName, age) are equivalent.

I provide code below which utilizes the Child class and the List342<>.

Other Rules

1) Implement the List342<> class using Nodes defined as below. It can be internal or external to the List342<> class whatever you find easiest.

   Note that the struct contains a pointer to the next Node and a ***pointer*** to the data. This is required for this program.

```
struct Node {
      T *data;
      Node *next;
};
```

2) Due to vicissitudes of different IDEs and templating we will be factoring the code into files a little sub-optimally. **Put all code into the List342.h file—this includes the implementation (.cpp) as well as the .h.**

   TURN IN (.zip file):
   - list342.h (Note: the implementation is at the bottom of the file)
   - Output of your program running the example code below

Example code followed by output:

```
#include <iostream>
#include "list342.h"
#include "child.h"
using namespace std;

int main()
{
	Child c1("Angie", "Ham", 7), c2("Pradnya", "Dhala", 8),
		c3("Bill", "Vollmann", 13), c4("Cesar", "Ruiz", 6);
	Child c5("Piqi", "Tangi", 7), c6("Russell", "Wilson", 13),
		c7("Hank", "Aaron", 3), c8("Madison", "Fife", 7);
	Child c9("Miles", "Davis", 65), c10("John", "Zorn", 4), c11;
	List342<Child> class1, class2, soccer, chess;
	int a = 1, b = -1, c = 13;

	class1.Insert(&c1);
	class1.Insert(&c2);
	class1.Insert(&c3);
	class1.Insert(&c4);
	class1.Insert(&c5);
	class1.Insert(&c6);
	class1.Insert(&c5);
	cout << "class1: " << class1 << endl;

	if (! class1.Insert(&c1))
	{
		cout << "ERROR::: Duplicate" << endl;
	}

	class2.Insert(&c4);
	class2.Insert(&c5);
	class2.Insert(&c6);
	class2.Insert(&c7);
	class2.Insert(&c10);
	cout << "Class2: " << class2 << endl;

	class1.Merge(class2);
	class2.Merge(class1);
	class1.Merge(class2);
	class1.Merge(class1);
	cout << "class1 and 2 Merged: " << class1 << endl;

	class1.Remove(c4, c11);
	class1.Remove(c5, c11);
	class1.Remove(c11, c11);
	if (class1.Remove(c1, c11))
	{
		cout << "Removed from class1, student " << c11 << endl;
```

```
	}
	cout << "class1: " << class1 << endl;

	soccer.Insert(&c6);
	soccer.Insert(&c4);
	soccer.Insert(&c9);
	cout << "soccer: " << soccer << endl;
	soccer += class1;
	cout << "soccer += class1 : " << soccer << endl;

	List342<Child> football = soccer;
	if (football == soccer)
	{
		cout << "football: " << football << endl;
	}
	if (football.Peek(c6, c11))
	{
		cout << c11 << " is on the football team" << endl;
	}
	soccer.DeleteList();

	List342<int> numbers;
	numbers.Insert(&a);
	numbers.Insert(&b);
	numbers.Insert(&c);
	cout << "These are the numbers: " << numbers << endl;
	numbers.DeleteList();
}
```

OUTPUT:

```
class1: PradnyaDhala8AngieHam7CesarRuiz6PiqiTangi7BillVollmann13RussellWilson13
ERROR::: Duplicate
Class2: HankAaron3CesarRuiz6PiqiTangi7RussellWilson13JohnZorn4
class1 and 2 Merged:
HankAaron3PradnyaDhala8AngieHam7CesarRuiz6PiqiTangi7BillVollmann13RussellWilson13JohnZorn4
Removed from class1, student AngieHam7
class1: HankAaron3PradnyaDhala8BillVollmann13RussellWilson13JohnZorn4
soccer: MilesDavis65CesarRuiz6RussellWilson13
soccer += class1 :
HankAaron3MilesDavis65PradnyaDhala8CesarRuiz6BillVollmann13RussellWilson13JohnZorn4
football: HankAaron3MilesDavis65PradnyaDhala8CesarRuiz6BillVollmann13RussellWilson13JohnZorn4
RussellWilson13 is on the football team
These are the numbers: -1113
```