CSS 342 Programming Assignment #3 Due: Wednesday, April 25, 2007 (from syllabus) Spring 2007

Note that the midterm is Monday, April 30. Recursion will not be covered on the midterm.

**PART 1.** Develop an OrderedList template class. Test it on two different types of objects available on the sample code page:

1. Employee -- to be sorted alphabetically by last name, then first
2. NodeData -- to be sorted by numerical value, then char

Put class definition and implementation into a single .h file.

Existing samples:

**Array class:**

Simple int array class: [array.h](http://courses.washington.edu/css342/zander/Code/array.h) ... [array.cpp](http://courses.washington.edu/css342/zander/Code/array.cpp) ... [arraydriver.cpp](http://courses.washington.edu/css342/zander/Code/arraydriver.cpp) ... with [output](http://courses.washington.edu/css342/zander/Code/arrayoutput)

Same int array class with pre/post condition comments: [array.h](http://courses.washington.edu/css342/zander/Code/array2.h) ... [array.cpp](http://courses.washington.edu/css342/zander/Code/array2.cpp)

Array class as a template [arraytemplate.h](http://courses.washington.edu/css342/zander/Code/arraytemplate.h) includes class def and implementation ... [arraytemplatedriver.cpp](http://courses.washington.edu/css342/zander/Code/arraytemplatedriver.cpp) ... with [output](http://courses.washington.edu/css342/zander/Code/arraytemplateoutput)

**List class:**

List class (singly linked list): [list.h](http://courses.washington.edu/css342/zander/Code/list.h) ... [list.cpp](http://courses.washington.edu/css342/zander/Code/list.cpp) ... [listdriver.cpp](http://courses.washington.edu/css342/zander/Code/listdriver.cpp)

Nodedata class: [nodedata.h](http://courses.washington.edu/css342/zander/Code/nodedata.h) ... [nodedata.cpp](http://courses.washington.edu/css342/zander/Code/nodedata.cpp)

Employee class: [employee.h](http://courses.washington.edu/css342/zander/Code/employee.h) ... [employee.cpp](http://courses.washington.edu/css342/zander/Code/employee.cpp)

The List class includes the following functions. For the examples, assume the following are declared and suppose there are n items of data in the list .

ifstream infile1("data31.txt"), infile2("data32.txt");

bool success;

Employee\* oneEmployee;

List company1, company2, company3, company4;

1. All necessary constructors and destructor. (Should all take O(n) time.)

2. void buildList : given datafile, build an ordered list (note that buildList puts the responsibility on the Object class to know how to read from a file.) (insertion sort, O(n2) time) E.g., company1.buildList(infile1);

3. bool insert : insert one Object\* into its correct sorted position, return whether insertion was successful

4. bool remove: remove first parameter from the list and return (as parameter) the Object\* from the list if it is found, return value is whether removal was successful. E.g.,

Employee target("duck", "donald"); // assume constructor exists

success = company1.remove(target, oneEmployee);

if (success) {

cout << "removed: " << \*oneEmployee << endl;

delete oneEmployee; // could be inserted into another list

}

5. bool retrieve : given an Object, returns the Object\* (parameter) to it. If there are duplicates in the list, the first one encountered is retrieved. E.g.,

Employee target("duck", "donald");

success = company1.retrieve(target, oneEmployee);

if (success) { cout << "Found in list: " << \*oneEmployee << endl; }

6. void merge -- takes 2 sorted lists and merge into one long sorted list (No new memory is allocated. At termination of function, the two parameter lists are empty unless one is also the current object. Duplicates are allowed in a merged list.) (O(n) time, so insert must not be used or time would be O(n2)) E.g.,

// after this merge, company1 and company2 are empty,

// company3 contains all employees of company1 and company2

company3.merge(company1, company2);

// after merge, company3 is empty,

// company4 contains all employees of company3 and company4

company4.merge(company4, company3);

7. void intersect -- takes 2 sorted lists and finds the items in common in both lists (New memory is allocated; at termination of the function, the two parameter lists are unchanged unless one is also the current object.) E.g.,

// after intersect, company1 and company2 are unchanged,

// company3 holds intersection of company1 and 2

company3.intersect(company1, company2);

// after intersect, company3 is unchanged,

// company4 holds the intersection of company3 and 4

company4.intersect(company3, company4);

8. overload << : to display a List (does not use endl in it anywhere)

9. bool isEmpty : determine whether a list is empty

10. void makeEmpty : empty out the list, deallocate all memory for all Nodes and each Object

11. operator= : to assign one list to another (Takes O(n) time.)

12. bool operator== and operator!= : compare two lists

Notes

Implement the list using Nodes defined similar to the following. The Node can be a struct or a class, can be internal or external to the List class. (It's okay if you want to use the name NodeData instead of Object so you can copy and use the List class code from the website as is.)

struct Node {

Object\* data;

Node\* next;

};

NodeData data files are formatted so that each line of data includes an int and a char. For example,

100 x

You can assume correctly formatted data (each line will always have one int and one char). So that file reading works in a similar manner under both unix and windows, make sure the last line of data has an end-of-line character (under windows, cursor is on the line below the last line of good data). Checking for eof as in the List class’ buildList routine should work correctly, so do it that way.

Employee data files are formatted so that each line of data is the employee's name (last name then first name) followed by IDnumber and salary. For example,

mouse mickey 8584 55000

You can assume correctly formatted data (each line will always have two strings and two ints).

See the web for turn-in information.

**PART 2.** (Written exercises)

#1. Using the definition of big-O, prove that f(x) = 2x5 + x3 + 4x – 10 is O(x5) .

#2. Consider your implementation of selection sort in assignment 1. Give an analysis of the running time (find a tight big-O) for your selection sort. Include a copy of just the code for the sort with this answer.

To compile using the sample lab3 main, call it lab3.cpp, on mead, enter:   
        g++ lab3.cpp nodedata.cpp employee.cpp   
You can find sample Employee code with the List code (on the sample code page).   
**Hard copy turn-in:**   list.h (which includes the function bodies) and part 2 problems.   
**Electronic copy turn-in:**   list.h (which includes the function bodies) (all lowercase).   
        Do NOT put ANY endl in operator<< of your List class.   
        The operator<< for List should not display anything but the object, no extra blanks, no tabs, no endls ...   
        I will provide my own main, classes, and data files to test your List class.   
        Any classes used to test your List class will have setNodeData, ==, !=, <, >, << .

#include "list.h"

#include "nodedata.h"

#include "employee.h"

//---------------------- testCopyConstructor --------------------------------

// Not a member function, pass by value forces a copy to be made.

// After makeEmpty, the pass by reference list will be empty, while

// the pass by value list will be the original list as in main.

// Also tests destructor behind the scenes.

//---------------------------------------------------------------------------

template <typename Object>

void testCopyConstructor(List<Object>& byRefList, List<Object> byValList) {

cout << "Test Copy Constructor and Destructor" << endl;

cout << "------------------------------------" << endl;

byRefList.makeEmpty();

cout << "after emptying, byRefList is: " << endl << byRefList << endl;

cout << "byValList is: " << endl << byValList << endl;

}

int main() {

// list of NodeData, to be sorted by numerical value, then char

List<NodeData> mylist, mylist2, mylist3;

NodeData\* p = new NodeData(20, 'n'); mylist.insert(p);

p = new NodeData(10, 'f'); mylist.insert(p);

p = new NodeData(16, 'u'); mylist.insert(p);

p = new NodeData(25, '!'); mylist.insert(p);

p = new NodeData(16, 'u'); mylist2.insert(p);

p = new NodeData(15, 't'); mylist2.insert(p);

p = new NodeData(19, 'f'); mylist2.insert(p);

p = new NodeData(14, 's'); mylist2.insert(p);

p = new NodeData(25, '!'); mylist2.insert(p);

p = new NodeData(18, 'f'); mylist2.insert(p);

cout << "mylist:" << endl << mylist << endl;

cout << "mylist2:" << endl << mylist2 << endl;

mylist3.intersect(mylist, mylist2);

cout << "mylist3:" << endl << mylist3 << endl;

// many lists of employees,

// to be sorted alphabetically by last name, then first

bool success;

Employee\* oneEmployee;

List<Employee> company1, company2, company3, company4, company5, company6,

company7, company8, company9, company10, company11;

ifstream infile1("data31.txt"), infile2("data32.txt"),

infile3("data33.txt"), infile4("data34.txt");

// for electronic version:

//ifstream infile1("../data31.txt"), infile2("../data32.txt"),

// infile3("../data33.txt"), infile4("../data34.txt");

company1.buildList(infile1);

company2.buildList(infile2);

company3.buildList(infile3);

company4.buildList(infile4);

company6 = company4;

company7 = company1;

company8 = company2;

company10 = company11 = company3;

cout << "test -- merge 1 and 2 into 5" << endl;

cout << "----------------------------" << endl;

cout << "Company1:" << endl << company1 << endl;

cout << "Company2:" << endl << company2 << endl;

company5.merge(company1, company2);

cout << "company1 is " << (company1.isEmpty() ? "empty" : "not empty")

<< endl;

cout << "company2 is " << (company2.isEmpty() ? "empty" : "not empty")

<< endl;

cout << "Company5:" << endl << company5 << endl;

cout << "test == " << endl;

cout << "--------" << endl;

Employee emp1("moose", "bullwinkle"); // last name, first name

Employee emp2("squirrel", "rocky");

// knows emp1 and emp2 are in the lists, so deletes without checking

success = company10.remove(emp1, oneEmployee); delete oneEmployee;

success = company11.remove(emp2, oneEmployee); delete oneEmployee;

cout << "Company10:" << endl << company10 << endl;

cout << "Company11:" << endl << company11 << endl;

cout << "company10 " << (company10 == company11 ? "equals":"does not equal")

<< " company11" << endl;

success = company11.remove(emp1, oneEmployee); delete oneEmployee;

cout << "Company11:" << endl << company11 << endl;

cout << "company10 " << (company10 == company11 ? "equals":"does not equal")

<< " company11" << endl;

cout << "test == and !=: merge 7 and 8 into 9 (same as 5)" << endl;

cout << "------------------------------------------------" << endl;

cout << "Company7:" << endl << company7 << endl;

cout << "Company8:" << endl << company8 << endl;

company9.merge(company8, company7);

cout << "Company9:" << endl << company9 << endl;

cout << "company5 " << (company5 == company9 ? "equals" : "does not equal")

<< " company9" << endl;

cout << "company5 " << (company5 != company9 ? "does not equal" : "equals")

<< " company9" << endl;

// note after merge, 3 is empty, 4 is merged 3 and 4

cout << "test -- merge 3 and 4 into 4" << endl;

cout << "----------------------------" << endl;

cout << "Company3:" << endl << company3 << endl;

cout << "Company4:" << endl << company4 << endl;

company4.merge(company3, company4);

cout << "company3 is " << (company3.isEmpty() ? "empty" : "not empty")

<< endl;

cout << "Company4:" << endl << company4 << endl;

cout << "test -- merge empty 3 and 4 into 4, no change to 4" << endl;

cout << "--------------------------------------------------" << endl;

cout << "Company3:" << endl << company3 << endl;

cout << "Company4:" << endl << company4 << endl;

company4.merge(company3, company4);

cout << "company3 is " << (company3.isEmpty() ? "empty" : "not empty")

<< endl;

cout << "Company4 again (same):" << endl << company4 << endl;

cout << "test -- merge 4 and empty 3 into 4, no change to 4" << endl;

cout << "--------------------------------------------------" << endl;

company4.merge(company4, company3);

cout << "company3 is " << (company3.isEmpty() ? "empty" : "not empty")

<< endl;

cout << "Company4 again (same):" << endl << company4 << endl;

cout << "test -- merge 6 and empty 3 into 4" << endl;

cout << "----------------------------------" << endl;

company4.merge(company6, company3);

cout << "company3 is " << (company3.isEmpty() ? "empty" : "not empty")

<< endl;

cout << "company6 is " << (company6.isEmpty() ? "empty" : "not empty")

<< endl;

cout << "Company4:" << endl << company4 << endl;

cout << "test remove and retrieve on 4" << endl;

cout << "-----------------------------" << endl;

Employee emp3("aaaaa", "aa");

Employee emp4("zzzzz", "zz");

success = company4.retrieve(emp3, oneEmployee);

if (success) cout << "Found in list: " << \*oneEmployee << endl;

else cout << "Did not find aaaaa in list" << endl;

success = company4.retrieve(emp4, oneEmployee);

if (success) cout << "Found in list: " << \*oneEmployee << endl;

else cout << "Did not find zzzzz in list" << endl;

Employee emp5("ghost", "casper");

Employee emp6("rella", "barbara");

Employee emp7("bear", "bertha");

success = company4.remove(emp5, oneEmployee);

if (success) {

cout << "Removed: " << \*oneEmployee << endl;

delete oneEmployee; // could insert elsewhere

}

else

cout << "Did not find ghost in list" << endl;

success = company4.remove(emp6, oneEmployee);

if (success) {

cout << "Removed: " << \*oneEmployee << endl;

delete oneEmployee;

}

else

cout << "Did not find rella in list" << endl;

success = company4.remove(emp7, oneEmployee);

if (success) {

cout << "Removed: " << \*oneEmployee << endl;

delete oneEmployee;

}

else

cout << "Did not find bear in list" << endl;

cout << "Company4 after removes:" << endl << company4 << endl;

testCopyConstructor(company10, company10);

cout << "back in main, Company10:" << endl << company10 << endl;

return 0;

}