

1 Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Lab09, put each programming exercise into its own subdirectory of this directory, zip the entire Lab09 directory up into the file Lab09.zip, and then submit this zip file to Lab #9.

Make sure that you:

- Follow the coding and documentation standards for the course as published in the MyClasses page for the class.
- Check the contents of the zip file before uploading it. Make sure all the files are included.
- Make sure that the file was submitted correctly to MyCLasses.

All non-templated class structures are to have their own guarded specification file (.h) and implementation file (.cpp) that has the same name as the class. All templated class structures are to be guarded and written entirely in their (.h) file. No inline coding in the class specification. In addition you must create a make file that compiles and links the project on a Linux computer with a Debian or Debian branch flavor.

This exercise is to add in a function to the binary tree class and the linked list class that will apply a function to the entire collection.

1. Create a new directory and put in the following main program.

```
#include <algorithm>
#include <cctype>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>

#include "BinaryTree.h"
#include "LinkedList.h"

using namespace std;

template <class T> void squarer(T &i) { i = sqrt(i); }
template <class T> void sqr(T &i) { i = i * i; }
template <class T> void pt(T &item) { cout << item << " "; }
void toupperchar(char &c) { c = toupper(c); }
void uppercase(string &str) { for_each(str.begin(), str.end(), toupperchar); }

void div() { cout << "\n-----\n\n"; }

int main() {
    srand(time(0));

    // Create the trees.
    BinaryTree<int> bt;
```

```
bt.insertNode(10);
bt.insertNode(4);
bt.insertNode(12);
bt.insertNode(7);
bt.insertNode(15);

bt.PrintTree();
cout << endl;

bt.apply(pt);
cout << endl << endl;

bt.apply(sqr);
bt.PrintTree();
cout << endl;

div();

BinaryTree<double> btd;

btd.insertNode(10);
btd.insertNode(4);
btd.insertNode(12);
btd.insertNode(7);
btd.insertNode(15);

btd.PrintTree();
cout << endl;

btd.apply(squarert);
btd.PrintTree();
cout << endl;

btd.apply(pt);
cout << endl;

div();

BinaryTree<string> btstr;

btstr.insertNode("Steve");
btstr.insertNode("John");
btstr.insertNode("Sue");
btstr.insertNode("Kim");
btstr.insertNode("Tom");

btstr.PrintTree();
cout << endl;

btstr.apply(uppercase);
btstr.PrintTree();
cout << endl;

btstr.apply(pt);
cout << endl;

div();

LinkedList<int> LLi;
LLi.appendNode(10);
LLi.appendNode(5);
LLi.appendNode(17);
LLi.appendNode(121);
LLi.appendNode(15);

LLi.displayList();
LLi.apply(pt);
cout << endl;
```

```

    LLi.apply(sqr);
    LLi.apply(pt);
    cout << endl;

    div();

    LinkedList<string> LLs;
    LLs.appendNode("Steve");
    LLs.appendNode("John");
    LLs.appendNode("Sue");
    LLs.appendNode("Kim");
    LLs.appendNode("Tom");

    LLs.apply(pt);
    cout << endl;
    LLs.apply(uppercase);
    LLs.apply(pt);
    cout << endl;

    return 0;
}

```

2. Copy over the header files for the binary tree and for the linked list classes, we want the templated versions for these, of course.
3. Add a function called apply to both the linked list and binary tree classes. At least in the case of the binary tree, you will probably want two functions, one recursive and the other not recursive that starts up the recursive one. These functions are to obviously be templated. The non-recursive version should be public and take in as a parameter a function pointer to a templated function that is void and takes a single parameter of the template type by reference (to update the value stored in the node). The recursive version is to be either private or protected and take in two parameters, one will be a pointer to the current node being visited and the other a pointer to the function to apply to the node value.

For example, in the code above, the function `void pt(T &item)` for printing would work here. The `apply` functions will traverse the entire collection class and apply the parameter function to each node in the collection. So then the call `bt.apply(pt)` will apply the `print` function to each node in the binary tree, using in-order traversal. Hence printing the in-order traversal to the console.

When the above main program is run the output will be the following.

```

      15
    12
  10
    7
  4
4 7 10 12 15

      225
    144
  100
    49
  16

```

```
-----  
      15  
     12  
    10  
     7  
     4  
  
     3.87298  
     3.4641  
  3.16228  
     2.64575  
     2  
  
2 2.64575 3.16228 3.4641 3.87298
```

```
-----  
      Tom  
     Sue  
Steve  
      Kim  
    John  
  
     TOM  
     SUE  
STEVE  
      KIM  
    JOHN  
  
JOHN KIM STEVE SUE TOM
```

```
-----  
    10  
    5  
   17  
  121  
   15  
10 5 17 121 15  
100 25 289 14641 225
```

```
-----  
Steve John Sue Kim Tom  
STEVE JOHN SUE KIM TOM
```