CS124 Lab5 - lab.h

- One third of this header file is literally just include files for the stuff we need from the FLTK library.

```
#include <iostream>
#include <fstream>
#include <string>
#include <list>
#include <Fl/Fl.H>
#include <Fl/Fl_Double_Window.H>
#include <Fl/Fl_Button.H>
#include <Fl/Fl_Input.H>
#include <Fl/Fl_Secret_Input.H>
#include <Fl/fl_ask.H>
#include <FL/Fl_Text_Editor.H>
#include <FL/Fl_Text_Buffer.H>
#include <Fl/Fl_Text_Display.H>
#include <FL/fl_message.H>
#include <FL/fl_message.H>
#include "Subscriber.h"
#include "BinaryTree.h"
using namespace std;
```

CS124 Lab5 - lab.h

```
const int width = 400;
const int height = 400;

const int maxUsers = 10;\

struct Message{
    string from;
    string subject;
    string msg;
};
```

- I just arbitrarily set the window size to be 400x400
- The message struct contains three strings that an email would usually have.
- This is for incoming messages.

CS124 Lab5 - lab.h

- Lower third of the header file consists of our function prototypes and external variables
- We need to have external declarations to declare in other files like our callbacks

```
extern Fl_Double_Window ew;
void loginCB(Fl_Widget*);
void writeMsgCB(Fl_Widget*);
void CB(Fl_Widget*);
void createCB(Fl_Widget*);
void addSubCB(Fl_Widget*);
void MsgCB(Fl_Widget*);
void cancelCB(Fl_Widget*);
void cancelMsgCB(Fl_Widget*);
void createSub(string name, string password, Subscriber &sub, BinaryTree &tree);
extern Fl_Input* loginInput;
extern Fl_Secret_Input* passInput;
extern Fl_Double_Window* sw;
extern Fl_Double_Window* mailWin;
extern Subscriber users[maxUsers];
extern Fl_Text_Editor* msg;
extern Fl_Text_Buffer* msgBuf;
extern Fl_Double_Window* composeWin;
extern Fl_Double_Window* newUsrWin;
extern Fl_Input* nameIn;
extern Fl_Input* passwordIn;
```

CS124 Lab5 - BinaryTree.h

```
#ifndef BINARYTREE_H
#define BINARYTREE_H
#include "Subscriber.h" // includes #include <string>

class BinaryTree
{
private:
    struct TreeNode
    {
        Subscriber value;
        TreeNode *left;
        TreeNode *right;
    };

    TreeNode *root;

    void insert(TreeNode *&, TreeNode *&);
    void displayInOrder(TreeNode *) const;

    void deleteNode(Subscriber &, TreeNode *&);
    void makeDeletion(TreeNode *&);
```

- Class based implementation of the binary search tree
- The struct for TreeNode calls the Subscriber struct within it for modular reasons

CS124 Lab5 - BinaryTree.h

```
public:
    BinaryTree()
    { root = nullptr; }

    ~BinaryTree()
    {}

    void insertNode(Subscriber &);
    bool searchNode(string);
    void remove(Subscriber &);

    void displayInOrder() const
    { displayInOrder(root); }
};
#endif
```

- Public member functions include the insertNode, searchNode, remove, and displayInOrder functions
- All the functions in here are only applicable to tree objects

CS124 Lab5 - Subscriber.h

```
#ifndef Subscriber_h
#define Subscriber_h
#include <string>
using namespace std;

struct Subscriber
{
    string name;
    string password;
};

#endif /* Subscriber_h */
```

- Self-explanatory struct for subscribers; contains string for name and password.

CS124 Lab5 - searchNode.cpp

```
#include "BinaryTree.h"

bool BinaryTree::searchNode(string name)
{
    TreeNode *nodePtr = root;

    while (nodePtr)
    {
        if ((nodePtr->value).name == name)
            return true;
        else if (name < (nodePtr->value).name)
            nodePtr = nodePtr->left;
        else
            nodePtr = nodePtr->right;
    }
    return false;
}
```

- Iterative implementation of the search function
- If the value of the node is equal to the person we're looking for then we have found the node
- If the value is less than the parameter we entered; we go to the left child of the node
- Else if the value is greater than the parameter we entered we go to the right child
- We keep doing this in the while look until we return true.

CS124 Lab5 - remove.cpp

- Calls the deleteNode function to remove the root node

```
#include "BinaryTree.h"

void BinaryTree::remove(Subscriber &sub)
{
    deleteNode(sub, root);
}
```

CS124 Lab5 - makeDeletion.cpp

- TODO!!!

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

void BinaryTree::makeDeletion(TreeNode *&nodePtr)
{
    TreeNode *tempNodePtr;

    if (nodePtr == nullptr)
        cout << "Cannot delete empty node.\n";
    else if (nodePtr->right == nullptr)
    {
        tempNodePtr = nodePtr;
        nodePtr = nodePtr->left;
        delete tempNodePtr;
    }
    else if (nodePtr->left == nullptr)
    {
        tempNodePtr = nodePtr;
        nodePtr = nodePtr->right;
        delete tempNodePtr;
    }
```

CS124 Lab5 - makeDeletion.cpp

```
    else
    {
        tempNodePtr = nodePtr->right;
        while (tempNodePtr->left)
            tempNodePtr = tempNodePtr->left;
        tempNodePtr->left = nodePtr->left;
        tempNodePtr = nodePtr;
        nodePtr = nodePtr->right;
        delete tempNodePtr;
    }
}
```

- TODO!!

CS124 Lab5 - insert.cpp

```
#include "BinaryTree.h"

void BinaryTree::insert(TreeNode *&nodePtr, TreeNode *&newNode)
{
    if (nodePtr == nullptr)
        nodePtr = newNode;
    else if ((newNode->value).name < (nodePtr->value).name)
        insert(nodePtr->left, newNode);
    else
        insert(nodePtr->right, newNode);
}
```

- If there are no nodes in the tree then we create the first node i.e. the root.
- If the current node we are pointing to has a greater value than the one we are trying to insert then we insert to the left of it.
- If the current node we are pointing to has a lesser value than the one we are tryign to insert then we insert to the right of it.

CS124 Lab5 - insertNode.cpp

```
#include "BinaryTree.h"
#include "lab.h"

void BinaryTree::insertNode(Subscriber &sub)
{
    TreeNode *newNode;

    newNode = new TreeNode;
    newNode->value = sub;
    newNode->left = newNode->right = nullptr;

    insert(root, newNode);
    fl_message("You’ve successfully created a new subscriber!");
}
```

- Allocates a new node to insert
- Assigns the data in subscriber to the value of the tree node
- Sets the left and right child to null and then calls the insert function
- If the insert is successful, we let the user know by producing a message popup

CS124 Lab5 - displayInOrder.cpp

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

void BinaryTree::displayInOrder(TreeNode *nodePtr) const
{
    if (nodePtr)
    {
        displayInOrder(nodePtr->left);
        cout << (nodePtr->value).name << endl;
        displayInOrder(nodePtr->right);
    }
}
```

- Recursive implementation of the display in order function
- We are displaying with priority towards the front of the alphabet(A)

CS124 Lab5 - deleteNode.cpp

```
#include "BinaryTree.h"

void BinaryTree::deleteNode(Subscriber &sub, TreeNode *&nodePtr)
{
    if (sub.name < (nodePtr->value).name)
        deleteNode(sub, nodePtr->left);
    else if (sub.name > (nodePtr->value).name)
        deleteNode(sub, nodePtr->right);
    else
        makeDeletion(nodePtr);
}
```

- Takes two parameters: subscriber address and node pointer
- Does comparisons of nodes until correct node is found then calls makeDeletion function

CS124 Lab5 - main.cpp

```
//g++ *.cpp -std=c++11 -I ~/fltk-1.3.3 ‘fltk-config --cxxflags --ldflags --use-cairo‘ -o main
#include "lab.h"
#include "BinaryTree.h"
#include "Subscriber.h"
Fl_Double_Window ew(width,height);
Fl_Double_Window* sw;
Fl_Double_Window* mailWin;
Fl_Button* loginButton;
Fl_Input* loginInput;
Fl_Secret_Input* passInput;
Fl_Text_Editor* msg;
Fl_Text_Buffer* msgBuf;
Fl_Button* writeMsg;
Fl_Button* addSub;
Fl_Button* deleteSub;
Subscriber users[maxUsers];
int main(){
    users[0].name="admin";      //Hard coding sysop login for now
    users[0].password="p";
    ew.label("eMail");
```

- The build command is written on the top of the file for reference
- Fltk requires the use of many global declarations of each widget you want to create
- We hard coded the first account for now due to time constraint and convenience

CS124 Lab5 - main.cpp

```
int x = 5*width/8;
int y = 1*height/8;
int w = 1*width/4;
int h = 1*height/8;
loginButton = new Fl_Button(x,y,w,h,"Login");
loginButton->callback(loginCB);

x = 1*width/4;
h = 1*height/16;
loginInput = new Fl_Input(x,y,w,h, "Name: ");

y += 30;
passInput = new Fl_Secret_Input(x,y,w,h, "Password: ");
```

- The login buton uses Fl Button which is a nice and simple button that will call a callback function when clicked
- The name and password fields use Fl Input and Fl Secret Input respectively for text input
- The Secret Input function makes it so that your text shows up as censored asterisks when typing in your password
- We are able to check for correct name and password in the loginCB callback. We will go into that later.

CS124 Lab5 - main.cpp

```
    //This currently is the *very* rudimentary screen for mail display
    mailWin =  new Fl_Double_Window(width, height);

    writeMsg = new Fl_Button(width * 0.03, height * 0.03, 110, 30, "Compose Mail");
    writeMsg->callback(writeMsgCB);

    addSub = new Fl_Button(width * 0.55, height * 0.03, 70, 30, "Add Sub");
    addSub->callback(addSubCB);

    deleteSub = new Fl_Button(width * 0.74, height * 0.03, 90, 30, "Delete Sub");
    //deleteSub->callback(deleteSubCB);

    ew.show();
    Fl::run();
}
```

- I made the mail window in the main file because...programmer discretion...
- This is the window that is supposed to show recieved/incoming mail
- There are three buttons currently that can take you to the compose mail window, add subscriber window, and remove subscriber window

CS124 Lab5 - loginCB.cpp

```
#include "lab.h"
void loginCB(Fl_Widget*){
    string v = loginInput->value();
    string vp = passInput->value();
    if(v == users[0].name && vp == users[0].password){
        string msg = "Hello ";
        msg += v;
        int choice = fl_choice(msg.c_str(), "Exit", "Go to mail", 0);
        switch(choice){
            case 0: exit(EXIT_SUCCESS);
            case 1: mailWin->show();
                    ew.hide();;
        }
    }
    else{
        fl_message("Sorry, You entered the wrong credentials. Try again.");
    }
}
```

- The login callback stores the value of the input fields for name and password and stores them in their own strings
- We then make a check if the entered values match with the user's account credentials
- If the information entered is wrong then we show a message that alerts the user that the information entered was wrong and allows the user to try again
- If the info entered is correct then the user is greeted by the system via a message popup and asked if they want to exit or go to their inbox
- If they opt to go to their inbox then we will hide the initial login screen and show the mail window
- Pressing the exit button will exit the program

CS124 Lab5 - writeMsgCB.cpp

```
#include "lab.h"
Fl_Double_Window* composeWin;
Fl_Input* to;
Fl_Input* subject;
Fl_Text_Editor* message;
Fl_Text_Buffer* textbuf;
Fl_Button* send;
Fl_Button* cancelMsg;

void writeMsgCB(Fl_Widget*){
    int msgX = width * 0.2;
    int msgY = height * 0.05;
    int msgW = width * 0.77;

    composeWin = new Fl_Double_Window(width, height);

    to = new Fl_Input(msgX, msgY, msgW, 30, "To: ");
    subject = new Fl_Input(msgX, msgY + 40, msgW, 30, "Subject: ");
```

- The writeMsgCB is the callback for when you click on the "Compose" button and will create a new window which allows for the user to type into the To, Subject, and Message fields
- To and Subject uses Fl Input to retrieve text and we can use the same method of $->value()$ to get the input and save to a file with ofstream and ifstream

CS124 Lab5 - writeMsgCB.cpp

```
    message = new Fl_Text_Editor(width * 0.03, msgY + 90, width * .94, 235, "");
    textbuf = new Fl_Text_Buffer();
    message->wrap_mode(1,35);
    message->buffer(textbuf);
    //USE char* text() to output our text to a string.
    //message->align(ALIGN_TOP);        We might need fltk 2.0 for this

    cancelMsg = new Fl_Button(width * 0.79, height * 0.9, 70, 30, "Cancel");
    cancelMsg->callback(cancelMsgCB);

    send = new Fl_Button(width * 0.64, height * 0.9, 50, 30, "Send");

    //TODO: Make callbacks for the send and cancel buttons with the
    //      binary search tree code.


    composeWin->show();
}
```

- The Message field uses Fl Text Editor because it supports multiple lines and has a function for autowrapping text
- Important note: when using Text Editor you must have a text buffer or else you will not be able to input text into the field
- The cancel button calls the cancelMsgCB which we will get to later
- At the end, we show the compose message window after all our GUI elements are defined and the actual window has been defined

CS124 Lab5 - addSubCB.cpp

- addSubCB is called when the 'Add Sub' button is clicked and draws up a small window
- This window contains the fields for name and password input using Fl Input

```
#include "lab.h"
Fl_Double_Window* newUsrWin;
Fl_Input* nameIn;
Fl_Input* passwordIn;
Fl_Button* create;
Fl_Button* cancel;

void addSubCB(Fl_Widget*){
    int inputWidth = width * 0.7;
    int inputX = width * 0.25;

    newUsrWin = new Fl_Double_Window(width, height/3);

    nameIn = new Fl_Input(inputX, height * 0.05, inputWidth, 30, "Name: ");
    passwordIn = new Fl_Input(inputX, height * 0.05 + 40, inputWidth, 30, "Password: ");

    create = new Fl_Button(width * 0.58, height/4, 70, 30, "Create");
    create->callback(createCB);
    cancel = new Fl_Button(width * 0.78, height/4, 70, 30, "Cancel");
    cancel->callback(cancelCB);

    newUsrWin->show();
}
```

CS124 Lab5 - cancelCB.cpp

```
#include "lab.h"

void cancelCB(Fl_Widget*){
    string warning = "Do you want to cancel the creation of this new subscriber?";
    int choice = fl_choice(warning.c_str(), "No", "Yes", 0);
    switch(choice){
        case 0: break;
        case 1: newUsrWin->hide();
    }
}
```

- When the cancel button on the 'Add Sub' window is clicked; we alert and ask the user if they are sure
- We give them a choice between confirming and declining their action by using fl choice
- fl choice returns an integer depending on what the user pressed and we can use that to determine what to do in the switch block
- If they choose to decline canceling the sub creation then we just break out and we return back to the 'Add Sub' window

CS124 Lab5 - createCB.cpp

```
#include "lab.h"
#include "Subscriber.h"

void createCB(Fl_Widget*){
    Subscriber sub;
    BinaryTree tree;

    string name = nameIn->value();
    string password = passwordIn->value();

    createSub(name, password, sub, tree);
}
```

- We make a Subscriber and BinaryTree object here named sub and tree respectively to pass into the createSub function
- However, I worry if I'm actually creating a new binary tree eachtime we want to create a new subscriber
- I store the values of the name and password into strings and pass it into createSub

CS124 Lab5 - createSub.cpp

```
#include "lab.h"
#include "BinaryTree.h"
void createSub(string name, string password, Subscriber &sub, BinaryTree &tree){
    sub.name = name;
    sub.password = password;

    tree.insertNode(sub);
}
```

- The createSub function takes 4 paramenters: name, password, subscriber obj, and tree
- This function essentially is just setting all the parameters to the correct places

CS124 Lab5 - cancelMsgCB.cpp

```
#include "lab.h"

void cancelMsgCB(Fl_Widget*){
    string warning = "Are you sure you want to cancel your message?";
    int choice = fl_choice(warning.c_str(), "No", "Yes", 0);
    switch(choice){
        case 0: break;
        case 1: composeWin->hide();
    }
}
```

- When the cancel button on the 'Compose' window is clicked; we alert and ask the user if they are sure
- This process is extremely similar to the cancelCB function for the subscribers screens. We could probably make this code reusable.

Build Script

Text written to file build.sh

```
pptexenv latex lab
dvipdf  lab
```

Bourne Shell

```
chmod +x  build.sh
```