Contents

1 binaryTree.h

These functions were primarily taken from the book.

```
#ifndef BINARY_TREE_H
#define BINARY_TREE_H
#include <string>

using namespace std;

class Node
{
        public:

                string data;
                string animal;
                Node* left;
                Node* right;
                friend class BinaryTree;

};
```

```
A binary tree in which each node has two children.
class BinaryTree
{
        public:

 Constructs an empty tree.
                BinaryTree();
*******************************************************
 Constructs a tree with one node and no children.
                BinaryTree(string root_data, string root_animal);
*******************************************************
 Constructs a binary tree.
                BinaryTree(string root_data, string root_animal, BinaryTree
                    left, BinaryTree right);
 root data is the data for the root, left the left subtree
right the right subtree

*******************************************************
 Returns the height of this tree.
                int height() const;

*******************************************************
 Checks whether this tree is empty.
                bool empty() const;

*******************************************************
```

```
Gets the data at the root of this tree.
                string data() const;      return the root data
                string animal() const;
 here I added the string "animal", which I will use in
my gif search in lab.cpp

*****************************************************
 Gets the left subtree of this tree.
                BinaryTree left() const;

*****************************************************
 Gets the right subtree of this tree.
                BinaryTree right() const;

                Node* root;
*****************************************************
        private:
 Returns the height of the subtree whose root is the given node.
                int height(const Node* n) const;
 param n a node or nullptr return the height of the subtree,
or 0 if n is nullptr
};

#endif
```

2 binaryTree.cpp

The following functions are from the book.

```
#include <algorithm>
#include "binaryTree.h"
using namespace std;

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

BinaryTree::BinaryTree(string root_data, string root_animal)
{
        root = new Node;
        root->data = root_data;
        root->animal = root_animal;
        root->left = nullptr;
        root->right = nullptr;
}

BinaryTree::BinaryTree(string root_data, string root_animal, BinaryTree left
    , BinaryTree right)
{
        root = new Node;
        root->data = root_data;
        root->animal = root_animal;
        root->left = left.root;
        root->right = right.root;
}
```

```
int BinaryTree::height(const Node* n) const
{
        if (n == nullptr) { return 0; }
        else { return 1 + max(height(n->left), height(n->right)); }
}

int BinaryTree::height() const
{
        return height(root);
}

bool BinaryTree::empty() const
{
        return root == nullptr;
}

string BinaryTree::data() const
{
        return root->data;
}

string BinaryTree::animal() const
{
        return root->animal;
}
```

```
BinaryTree BinaryTree::left() const
{
        BinaryTree result;
        result.root = root->left;
        return result;
}

BinaryTree BinaryTree::right() const
{
        BinaryTree result;
        result.root = root->right;
        return result;
}
```

3 buildTree.cpp

```
#include "binaryTree.h"
#include <cstdlib>
#include <iostream>
#include <string>
 To this tree, which was provided by the book, I added a last "layer"
of questions so the program could confirm if the animal presented was
correct or not- if it wasn't, the program would ask questions to add a
new animal to the list.
BinaryTree buildTree()
{
        BinaryTree question_tree(
          BinaryTree("Is it a mammal?", "initialization",
                BinaryTree("Does it have stripes?", "none",
                  BinaryTree("Is it a carnivore?", "none",
                        BinaryTree("It is a tiger.", "none",
                          BinaryTree("Thank you for playing!","tiger"),
                          BinaryTree("What kind of animal is it: ","new")),
                        BinaryTree("It is a zebra.", "none",
                          BinaryTree("Thank you for playing!","zebra"),
                          BinaryTree("What kind of animal is it:","new"))),
                  BinaryTree("Is it a pig.","none",
                        BinaryTree("Thank you for playing!","pig"),
                        BinaryTree("What kind of animal is it:","new"))),
```

```
                BinaryTree("Does it fly?", "none",
                  BinaryTree("It is an eagle.", "none",
                        BinaryTree("Thank you for playing!", "eagle"),
                        BinaryTree("What kind of animal is it:","new")),
                  BinaryTree("Does it swim?", "none",
                        BinaryTree("It is a penguin.", "none",
                          BinaryTree("Thank you for playing!", "penguin"),
                          BinaryTree("What kind of animal is it:","new")),
                        BinaryTree("It is an ostrich.", "ostrich",
                          BinaryTree("Thank you for playing!", "ostrich"),
                          BinaryTree("What kind of animal is it:","new")))))
                             );
        return question_tree;
}
```

4 getImage.cpp

```
#include <iostream>
#include <fstream>
#include <curl/curl.h>
#include "mashape.h"

 handleData is a callback function to get the picture
size_t handleData(char* p, size_t s, size_t n, std::ofstream* u)
{
    u->write(p,s*n);
    return s * n;
    If this returns zero, it means there's nothing else to receive
}

void getImage(std::string u)
{
    CURL *curl;
    CURLcode res;
    std::ofstream ofs("animal.gif", std::ios::binary);
    curl_global_init(CURL_GLOBAL_DEFAULT);

    curl = curl_easy_init();

    if(curl)
    {
        curl_easy_setopt(curl, CURLOPT_URL, u.c_str());

        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, handleData);

        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &ofs);
```

```
            res = the return code
            res = curl_easy_perform(curl);

            print out an error statement if it fails
            if(res != CURLE_OK)
            {
                fprintf(stderr, "curl_easy_perform() failed: %s\n",
                        curl_easy_strerror(res));
            }
            curl_easy_cleanup(curl);
        }
        curl_global_cleanup();
}
```

5 getImageURL.cpp

```
The following is very similar to the getImage file,
but it returns the URL instead of just performing a void function
#include <stdio.h>
#include <curl/curl.h>
#include "mashape.h"

size_t handleData(char* p, size_t s, size_t n, std::string* u)
{
    *u += p;
    return s * n;
}

std::string getImageURL(std::string k)
{
    CURL *curl;
    CURLcode res;
    std::string s = "gif: ";

    std::string r = url + api + "&q=" + k + "&limit=1";

    curl_global_init(CURL_GLOBAL_DEFAULT);

    curl = curl_easy_init();
```

```
    if(curl)
    {
        curl_easy_setopt(curl,
                         CURLOPT_URL, r.c_str());

        curl_easy_setopt(curl,
                         CURLOPT_WRITEFUNCTION, handleData);

        curl_easy_setopt(curl,
                         CURLOPT_WRITEDATA, &s);

        res = curl_easy_perform(curl);

        if(res != CURLE_OK)
        {
            fprintf(stderr, "curl_easy_perform() failed: %s\n",
                    curl_easy_strerror(res));
        }
        curl_easy_cleanup(curl);
    }
    curl_global_cleanup();
    return s;
}
```

6 lab.cpp

```
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <config.h>
#include <curl/curl.h>
#include <FL/Fl_Cairo_Window.H>
#include <FL/Fl_Value_Input.H>
#include <FL/Fl_GIF_Image.H>
#include <FL/Fl_Box.H>
#include <FL/fl_ask.H>
#include "binaryTree.h"

const int WIDTH = 300; const int HEIGHT = 300;

std::string getImageURL(std::string);
void getImage(std::string k);

std::string parseJson(std::string j);
BinaryTree buildTree();
void preOrder(std::ofstream& o, BinaryTree bt);

std::string new_animal;
std::string new_question;

void readFile(std::ifstream& i, BinaryTree bt);
```

```
int main()
{
        BinaryTree question_tree = buildTree();
        std::ofstream ofs("animalsChoice");

        preOrder(ofs, question_tree);

        bool done = false;

 This is where my "animal" string variable comes into use- with an
animal of "new", the program knows to ask the user for the name of a
new animal and question.
        while (!done)
        {
                BinaryTree left = question_tree.left();
                BinaryTree right = question_tree.right();

                if (left.empty() && right.empty())
                {
                        if (question_tree.animal() == "new") {
                                new_animal = fl_input(
                                                "Enter new animal: ");
                                new_question = fl_input(
                                                "Enter a question where 'yes
                                                    ' applies to this animal
                                                     and 'no' applies to the
                                                     last animal mentioned:
                                                    ");
                        }
                        else {
                                std::cout << question_tree.data() << std::
                                    endl;
```

```
                        std::cout << "The animal was a(n)" <<
                            question_tree.animal() << "." << std::
                            endl;
                }
                done = true;
        }

        else
        {
                std::string question = question_tree.data();
                const char *c = question.c_str();

                switch( fl_choice(c, "Yes", "No", 0) )
                {
                        case 0:
                                {
                                        std::cout << "Yes" << std::
                                            endl;
                                        question_tree = left;
                                        break;
                                }       //yes
                        case 1:
                                {
                                        std::cout << "No" << std::
                                            endl;
                                        question_tree = right;
                                        break;
                                }       //No
                }
        }
}
```

```
    std::string s = getImageURL(question_tree.animal());
        s = parseJson(s);
    std::cout << s << std::endl;

    Fl_Cairo_Window cw(WIDTH, HEIGHT);
    Fl_Box b(10, 10, WIDTH, HEIGHT);

        getImage(s);

     go get the file returned by getImageURL store in a file

    Fl_GIF_Image i("animal.gif");
    b.image(&i);

    cw.show();

    return Fl::run();
}
```

7 mashape.h

```
#include <string>
This code was provided in class in order to allow us to find the
necessary URLs from giphy.
const std::string url = "https://api.giphy.com";
const std::string key =
"X-RapidAPI-Key: DrFjbSmOJnmshTt0NowBkUY1WpcXp1bDqRvjsn4MrMqadGpwlM";
const std::string js = "Accept: application/json";
const std::string api = "/v1/gifs/search?api_key=
    evJNRw4gT9hrC9P2hfQPe22czt629zPa";
```

8 parseJson.cpp

```
#include <string>
This parseJson is based on how the urls were originally printing in
the terminal before they were parsed.  Each one had a beginning in common
and could pretty reliably be divided into separate urls based on those
beginning statements.  By finding the start and end of the urls
and the characters we wanted to remove, the program is able to parse
the url we want.
std::string parseJson(std::string j)
{
    std::string header = "\"original\":{\"url\":\"" ;
    int first = j.find(header); int last = j.find(".gif", first + 1);
    int length = last - first + 4 - header.length();

    j = j.substr(first + header.length(),length);
    std::string url = "";

    for (int i = 0; i < j.length(); ++i) {
                if (j[i] != '\\'){
                  url += j[i];
          }
    }
    return url;
}
```

9 preOrder.cpp

```
#include <fstream>
#include <iostream>
#include <string>
#include "binaryTree.h"
This code was provided in class to output the binary tree to our output file.
using namespace std;

void preOrder(std::ofstream& o, BinaryTree bt)
{
        if (bt.empty())
        {
                return;
        }

        o << bt.data() << std::endl;

        preOrder(o, bt.left());

        preOrder(o, bt.right());

}
```

10 readFile.cpp

```
 These were the functions I was going to use to add the new
animals/questions as well as read from the animalsChoice file to create
another tree.  It does not work and is not linked to the program.

#include <fstream>
#include <iostream>
#include <string>
#include "binaryTree.h"

using namespace std;

void readFile(std::ifstream& i, BinaryTree bt)
{
        fstream file;
        file.open("animalChoice.txt");  //opens file to read
}
```

```
void addNode(Node* root, string choices, int length, string data, string
    animal)
{
    Node* top = root;
    int i = 0;
    while (i < length){

for these if statements, I was intending to check for something in the
lines that indicated whether the last choice was yes or no, or left or
right.
        if(choices[i] == ''){
            root = root->left;
        }
        if(choices[i] == ''){
            root = root->right;
        }
        i++;
    }
    Node* newnode = new Node;
    newnode->data = data; newnode->animal = animal;
    root = newnode;              //add recursive function?
}
```

11 unittest.cpp

```
Unit testing each BinaryTree member function using Catch2.
#include <iostream>
#include "catch.hpp"
#include "binaryTree.h"
BinaryTree buildTree();

#ifdef UNIT_TEST
TEST_CASE( "Tests" ){
        SECTION("First Constructor")
        {
                BinaryTree newTree;
                CHECK(newTree.root == nullptr);
        }
This just constructs an empty tree, so we only need to check for nullptr

        SECTION("Second Constructor")
        {
                BinaryTree newTree("Is this your question?", "animal");
                CHECK(newTree.data() == "Is this your question?");
                CHECK(newTree.animal() == "animal");
        }
 This function initializes with data values- I tested for both data
(which the function started with) and animal (which I added on my own)
```

```
        SECTION("Third Constructor")
        {
                BinaryTree left; BinaryTree right;
                BinaryTree thirdTree("Question?", "animal", left, right);
                CHECK(thirdTree.data()== "Question?");
                CHECK(thirdTree.animal()== "animal");
                CHECK(left.empty()); CHECK(right.empty());
        }
```

This function checks that the data values are correct and that there are empty branches both "left" and "right"

```
        SECTION("Height")
        {
                BinaryTree testTree = buildTree();
                CHECK(testTree.height() == 5);
        }
```

As I extended the tree with a new batch of leaves, my initial tree is a height of 5 rather than the 4 the tree the book had.

```
        SECTION("Empty")
        {
                BinaryTree emptyTree;
                CHECK(emptyTree.empty() == true);
        }
```

Checks that the tree is empty.

```
        SECTION("Data")
        {
                BinaryTree dataTree = buildTree();
                CHECK(dataTree.data() == "Is it a mammal?");
                CHECK(dataTree.animal() == "initialization");
        }
 This checks the data from buildTree itself.  The first question is of
course "Is it a mammal?" while I set the first 'animal' string to
"initialization" to distinguish it.

        SECTION("Left")
        {
                BinaryTree testTree = buildTree();
                BinaryTree leftTree = testTree.left();
                CHECK(leftTree.root == testTree.root->left);
        }
This in essence reverses the steps of the "left" function for binary
tree (which sets the root of the tree to "root->left")

                SECTION("Right")
        {
                BinaryTree testTree = buildTree();
                BinaryTree rightTree = testTree.right();
                CHECK(rightTree.root == testTree.root->right);
        }
This is the same as the left function, but for the right side.
}
#endif
```