

# lab6

Bhupinder Singh

May 4, 2019

## Contents

1	<code>binaryTree.h</code>	1
2	<code>binaryTree.cpp</code>	5
3	<code>buildTree.cpp</code>	8
4	<code>getImage.cpp</code>	9
5	<code>getImageurl.cpp</code>	12
6	<code>getResponse.cpp</code>	15
7	<code>lab.cpp</code>	16
8	<code>mashape.h</code>	19
9	<code>parseJson.cpp</code>	20



# 1 binaryTree.h

GOAL --> In this lab we are studying the API, implementation and application of a binary decision tree. The author gives us all the code needed to build the tree and play a guessing game with the computer trying to guess our chosen animal from its decision tree. We are enhancing that game by: Add new member function to BinaryTree to modify the Node of the object that calls it (data, left, and right).

```

#ifndef BINARY_TREE_H
#define BINARY_TREE_H
#include <string>

using namespace std;

class Node
{
    private:

        string data;
        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);    @param rootdata the data for the root
BinaryTree(string root_data);    @param rootdata the data for the root

*****

Constructs a binary tree.
BinaryTree(string root_data, BinaryTree left, BinaryTree right);
    @param rootdata the data for the root
    @param left the left subtree
    @param right the right subtree

*****

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

*****

* Checks whether this tree is empty.
bool empty() const; * @return true if this tree is empty

*****

* Gets the data at the root of this tree.
string data() const;    * @return the root data

*****

* Gets the left subtree of this tree.
BinaryTree left() const;    * @return the left child of the root

*****

```

```

        * Gets the right subtree of this tree.
        BinaryTree right() const;    * @return the right child of the root

void setLeft(std::string l);
void setRight(std::string r);
void setData(std::string d);

*****

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

        *****

        Node* root;
};

#endif

```

## 2 binaryTree.cpp

```
#include <algorithm>
#include "binaryTree.h"

using namespace std;

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

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

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

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

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

void BinaryTree::setLeft(std::string l)

```



```
root->left = new BinaryTree(l);  
void BinaryTree::setRight(string r)  
root->right = new BinaryTree(r);  
void BinaryTree::setData(std::string d)  
root->data = d;
```

### 3 buildTree.cpp

```
#include "binaryTree.h"
```

Is there a more efficient way to return this tree?  
So we don't have to return it by value? By reference  
not good since it is a local variable.

```
BinaryTree buildTree()  
{  
    BinaryTree question_tree(  
        BinaryTree("Is it a mammal?",  
            BinaryTree("Does it have stripes?",  
                BinaryTree("Is it a carnivore?",  
                    BinaryTree("It is a tiger."),  
                    BinaryTree("It is a zebra.")),  
                BinaryTree("It is a pig.")),  
            BinaryTree("Does it fly?",  
                BinaryTree("It is an eagle."),  
                BinaryTree("Does it swim?",  
                    BinaryTree("It is a penguin."),  
                    BinaryTree("It is an ostrich.")))))  
    );  
    return question_tree;  
}
```

## 4 getImage.cpp

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

    callback function to receive the image
size_t handleData2(char* p, size_t s, size_t n, std::ofstream* u)
{
    size_t ns = s * n;

    u->write(p, ns);

    return s * n; //indicates if there is more data or not
    //zero means no more packets to receive
}

// "https://media3.giphy.com/media/wPud2z0g029Xy/200.gif"

// expect s is a valid URL of a gif and we want to save it in a file
void getImage(std::string k)
{
    CURL *curl;

    CURLcode respcode = CURLE_OK;

    std::ofstream ofs("animal.gif", std::ios::binary);

    // std::string r = url + api;
```

```

curl_global_init(CURL_GLOBAL_DEFAULT);

curl = curl_easy_init();

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

    //std::cout << r << std::endl;

    curl_easy_setopt(curl,
                      CURLOPT_WRITEFUNCTION, handleData2);

    curl_easy_setopt(curl,
                      CURLOPT_WRITEDATA, &ofs); //change to write to a
                                                file

    Perform the request, res will get the return code
    respcode = curl_easy_perform(curl);

    Check for errors
    if(respcode != CURLE_OK)
    {
        fprintf(stderr, "curl_easy_perform() failed: %s\n",
                curl_easy_strerror(respcode));
    }

    always cleanup
    curl_easy_cleanup(curl);

```

```
    }  
    curl_global_cleanup();  
    //return;  
}
```

## 5 getImageurl.cpp

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

sudo apt install libcurl3-dev

callback function to receive the image
size_t handleData(char* p, size_t s, size_t n, std::string* u)
{
    *u += p;

    return s * n; //indicates if there is more data or not
    //zero means no more packets to receive
}

// "https://media3.giphy.com/media/wPud2z0g029Xy/200.gif"

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

    //~ struct curl_slist * slist1 = NULL;
    //~ slist1 = curl_slist_append(slist1, key.c_str());
    //~ slist1 = curl_slist_append(slist1, js.c_str());

    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_HTTPHEADER, slist1);

    std::cout << r << std::endl;

    curl_easy_setopt(curl,
                     CURLOPT_WRITEFUNCTION, handleData);

    curl_easy_setopt(curl,
                     CURLOPT_WRITEDATA, &s);

    Perform the request, res will get the return code
    res = curl_easy_perform(curl);

    Check for errors
    if(res != CURLE_OK)
    {
        fprintf(stderr, "curl_easy_perform() failed: %s\n",
                curl_easy_strerror(res));
    }

    always cleanup

```

```
        curl_easy_cleanup(curl);  
    }  
  
    curl_global_cleanup();  
  
    return s;  
}
```



## 6   getResponse.cpp

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

std::string getResponse(BinaryTree t)
{
    std::string r;
    do
    {
        cout << t.data() << " (y/n): ";
        cin >> r;
    } while (r != "y" && r != "n");
    return r;
}
```

## 7 lab.cpp

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

Compilation command:

```
g++ -g -I/home/debian -o "lab" "lab.cpp" -lcurl getImageURL.cpp -I/home/debian/fttk-1.3.4-2
'fttk-config --cxxflags --ldflags --use-images --use-cairo'
g++ -g -I/home/debian -o "lab" "lab.cpp" -lcurl getImage.cpp -I/home/debian/fttk-1.3.4-2
'fttk-config --cxxflags --ldflags --use-images --use-cairo'
```

```
//Fl_Cairo_Window cw;
const int WIDTH = 300;
const int HEIGHT = 300;

std::string getImageURL(std::string);
void getImage(std::string);
std::string parseJson(std::string s);

int main()
{
    std::string s = getImageURL("cat"); //return the location URL of a cat
    image
    s = parseJson(s);
```

```

//~ std::cout << s << std::endl;

Fl_Cairo_Window cw(WIDTH, HEIGHT);

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

    go get the file returned by getImageURL store in a file
getImage("https://media3.giphy.com/media/wPud2z0g029Xy/200.gif");

Fl_GIF_Image i("animal.gif");

b.image(&i);

cw.show();

//~ while (true)
//~ {
//~ switch( fl_choice("Empty Trash?", "Yes", "No", 0) )
//~ {
//~ case 0:
//~ {std::cout << "Yes" << std::endl; break;} //yes
//~ case 1:
//~ {std::cout << "No" << std::endl; break;} //No (
//~ default)
//~ }
//~ }

//std::cout << fl_input("Enter new animal: ");

return Fl::run();

```

}

## 8 mashape.h

```
#include <string>

const std::string url = "https://api.giphy.com";
const std::string key =
    "X-RapidAPI-Key: DrFjbSm0JnmshTt0NowBkUY1WpcXp1bDqRvj4sn4MrMqadGpwlM";
const std::string js = "Accept: application/json";
const std::string api = "/v1/gifs/search?api_key=
    evJNRw4gT9hrC9P2hfQPe22czt629zPa";
```

## 9 parseJson.cpp

```
#include <string>

std::string parseJson(std::string j)
{
    std::string original = "\"original\":{\"url\":\"";
    int start = j.find(original);
    int end = j.find(".gif", start + 1);
    int len = end - start + 4 - original.length();
    j = j.substr(start + original.length(), len);
    std::string url = "";
    for (int i = 0; i < j.length(); ++i) {
        if (j[i] != '\\')
            url += j[i];
    }
    return url;
}
```

## 10 Treedemo.cxx

This program demonstrates a decision tree for an animal guessing game.

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

using namespace std;

BinaryTree buildTree();
std::string getResponse(BinaryTree);

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

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

    preOrder(o, bt.left());

    preOrder(o, bt.right());
}
```

```

int main()
{
    Build a binary tree called question tree
    BinaryTree question_tree = buildTree(); //calls
    BinaryTree tree = question_tree;

    std::ofstream ofs("animals");

    preOrder(ofs, question_tree);

    //preOrder(std::cout, question_tree);

    bool done = false;

```

In this loop we ask the user some questions.  
 the user enters either yes or no.  
 based on their answer, we decide which way to go next.

```

while (!done)
{
    BinaryTree left = question_tree.left();
    BinaryTree right = question_tree.right();

    if (left.empty() && right.empty())
    {
        char yn;
        cout << question_tree.data() << "(Y/N)" << std::endl;
        cin >> yn;
    }
}

```

this part of the code is the added part.  
 it asks the user a final time if they have the desired result.  
 then



```

if(yn == 'y')
    done = true;
else {
    std::string animal, question;
    cout << "Whats the correct animal?" << std::endl;
    cin >> animal;
    cout << "What would the question leading to
            this answer be?" << std::endl;
    getline(cin, question);

    BinaryTree a(animal); //used constructor
                           that makes no child nodes.
    BinaryTree d(question_tree.data()); //same
                                         thing with current animal
    BinaryTree n(question, a, d); //making
                                   question tree with 2 child nodes and the
                                   question u want to replace node with
    question_tree = n; // replacing
    done = true; //exit loop
}

}

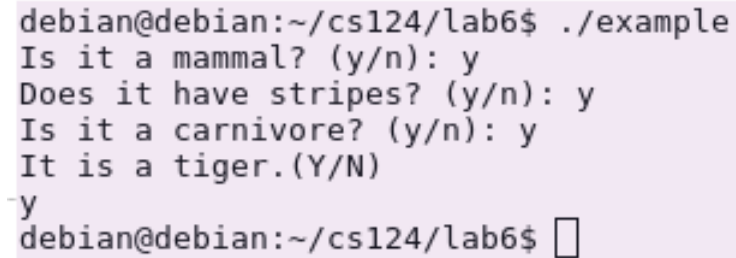
else
{
    string response = getResponse(question_tree);

    if (response == "y"){question_tree = left;}

    else {question_tree = right;}
}
}

```

```
    ofstream newtree("result");  
    preOrder(newtree, question_tree); //equal to the new tree that was built  
}
```



```
debian@debian:~/cs124/lab6$ ./example  
Is it a mammal? (y/n): y  
Does it have stripes? (y/n): y  
Is it a carnivore? (y/n): y  
It is a tiger.(Y/N)  
y  
debian@debian:~/cs124/lab6$
```

This first image shows the basic binary tree given to us.

```
debian@debian:~/cs124/lab6$ ./example
Is it a mammal? (y/n): y
Does it have stripes? (y/n): y
Is it a carnivore? (y/n): y
It is a tiger.(Y/N)
n
Whats the correct animal?
Leopard
What would the question be?
Is it yellow or orange?
debian@debian:~/cs124/lab6$ ☐
```

This second image goes one step beyond and asks the user whether the answer they received was right or wrong. if the answer was right, the program ends. if the answer was wrong, the program asks the user to choose the right answer and also the question that would lead up to that answer.

Due to my surgery, I did not have time to finish all of the lab. i had to first catch up with people and then work on the rest of the lab. If i had more time, the next thing i would do is implement a file system as mentioned in the lab. I would make the program read the text from a file. Every single time the user wants to enter another answer, he would add it to the file with the question pertaining to it. Sorry i could not complete all of the lab :( What happened was really unlucky timing.