**Sandesh Ghimire**

**CSCI 301 section 3**

**Computer Science 2**

**Project 9**

**12th November**

**Introduction**

Binary tree is a tree where may degree is two, and in which we distinguish left and right children. Binary search tree is a tree where data are arranged in such an order that a greater value always goes to the right part of the tree. In this project, we are implementing BST by using linked list. The feature of BST helps us to find data very easily.

**Data Structures:**

The data structures used in this project is Binary tree. Binary tree has at most two children. In this project we used binary tree in such a way that the greater value children are always in right part of the tree.

**Functions:**

The program contains a class named List in the header file " *List.h* “. The class list contains many public functions that are used to manipulate the linked list according to the user's decision the various function to manipulate the linked list are as follows:

*Tree() {root = NULL;}*: This is the constructor for the Binary tree.

*~Tree():* This is the destructor for the binary tree.

*void insert (Item empty):* This function inserts the value given by user into the binary tree. Firstly, it checks the proper place by using loop and then create a new Node. Lastly, the new node is pointed by the proper place (either right or left).

*void remove (Item target):* This function removes a value from the Tree. We did it recursively by breaking down the whole part into three function.

*void help\_remove(Node \* &t,Item target):* This function performs the recursion for removing the data in the list.

*void remove\_node(Node \* &t):* This function finds the node to remove and then deletes it.

*length ():* This function returns the length of the tree.

*bool present(const Item& target):* This function checks if an item exists on the binary tree or not.

*void Display:* This function prints the existing values of the tree.

*friend std::ostream& operator << (std::ostream& out\_s,const list& l):* This is a friend function to overload the “<<” operator.

**Pointers and Structures:**

The pointers are used in order to initialize the linked list and access the contents of the linked list. Node is used to make two pointers that points two children of out binary tree. One of them points right and another left. It is also used to hold the value of children.

**The Main Program:**

The main program utilizes the class and print outs the menu as well as selects the cases and performs actions accordingly.

At first, the main program calls the *display()* function to display the menu. Then the program asks the user to enter the desired operation. The program then selects the case that the user inputs and performs the actions accordingly. The programs continue to run until the user enters the letter " q ".

**Code**

**Main.cpp**

#include <iostream>

#include "list.h"

#include "list.cpp"

#include<iomanip>

#include<cstdlib>

#include <cmath>

/\*Binary tree is a tree where may degree is two, and in which we distinguish left and right children.

Binary search tree is a tree where data are arranged in such an order that a greater value always goes to the right part of the tree.

In this project, we are implementing BST by using linked list. The feature of BST helps us to find data very easily.

\*/

using namespace std;

void display(){ //function to display menu

cout<<"This program responds to commands the user enters to "<<endl<<

"manipulate an ordered list of integers, which is "<<endl<<

"initially empty. In the following commands, v is any"<<endl<<

"integer."<<endl;

cout<<"e -- Re-initialize the list to be empty."<<endl;

cout<<"i v -- Insert the value v into the list."<<endl;

cout<<"r v -- Remove the value v from the list."<<endl;

cout<<"l -- Report the length of the list."<<endl;

cout<<"p v -- Is the value v present in the list?"<<endl;

cout<<"w -- Write out the list."<<endl;

cout<<"h -- See this menu."<<endl;

cout<<"q -- Quit."<<endl;

}

int main(){ //The main function

Tree l;

display();

char choice;

int num;

bool quit = false;

while(choice != 'q') //using loop to stay on the program

{

cout<<" --> ";

cin>>choice;

switch (choice)

{

case 'i':

cin >> num;

l.Insert(num);

break;

case 'r': //if choice is r call remove

{

cin>>num;

l.Remove(num);

break;

}

case 'l': //if choice is l call length

{

int len=l.Length();

cout<<"The list contains "<<len <<" elements"<<endl;

break;

}

case 'p': //if choice is p call present

{

cin>>num;

if (l.Present(num)){

cout<<"The value "<<num<<" is present in the list"<<endl;

}

else{

cout<<"The value "<<num<<" is not present in the list"<<endl;

}

break;

}

case 'w': //if choice is w call display

{

cout<<"The list: (";

l.Display();

cout<<")"<<endl;

break;

}

case 'h': //if choice is h call display menu

{

display();

break;

}

case 'q':{ //if choice is q call quit the program

quit=true;

break;

}

default:{ //if entered extra then menu

cout<<"Wrong choice"<<endl;

break;

}

if (quit) //if not quit

{

break;

}

cin>>choice;

}

}

}

**List.cpp**

#include <iostream>

#include "list.h"

using namespace std;

Tree::~Tree() //destructor

{

destroy(root);//function to delete the tree

}

int Tree::Length()//find the length of tree

{

return r\_length(root);//calling the function

}

int Tree::r\_length(Node \*r){//find the length

if (r==NULL){return 0;}//starting the pre-condition

else{return r\_length(r->left)+1+r\_length(r->right);}//if not satisfied

}

bool Tree::Present(Item target)//function to check if the value is present in tree or not

{

Node \*p=root;//declaring a pointer

while(true)//starting the pre condition

{

if (p==NULL)//condition

return false;

else if (target<p->data)//condition

p=p->left;

else if(target=p->data)//condition

return true;

else//execute this if not satisfied any of above condition

p=p->right;

}

}

void Tree::Insert(Item entry)//to insert data

{

Node \*p, \*back, \*temp;//decalaring the pointers

p=root;//giving root value

back=NULL;//giving NULL value

//searching the place to put the new value

while (p!=NULL)//starting the loop

{

back=p;

if(entry<p->data)

p=p->left;

else

p=p->right;

}

//making a new node

temp= new Node;

temp->data=entry;

temp->left=NULL;

temp->right=NULL;

//attaching the new node

if(back==NULL)//condition

root=temp;

else if(entry<back->data)

back->left=temp;

else//if not satisfied execute this

back->right=temp;

}

void Tree::Remove(Item target)//function to remove data from tree

{

help\_remove(root,target);//sub function of the function Remove()

}

void Tree::help\_remove(Node \*&t,Item target)

{

if(t->data==target)//starting condition

remove\_node(t);

else if(target<t->data)//another condition

help\_remove(t->left,target);

else//execute this if not satisfied

help\_remove(t->right,target);

}

void Tree::remove\_node(Node \* &t) //to remove the node

{

Node \*ptr, \*back;//declare pointers

if(t->left==NULL && t->right==NULL)//starting the base condition

{

delete t;//delete the node

t=NULL;

}

else if(t->left==NULL)//else delete left

{

ptr=t;

t=t->right;

delete ptr;

}

else if(t->right==NULL)//else delete right

{

ptr=t;

t=t->left;

delete ptr;

}

else//if not satisfied execute this part

{

back=t;

ptr=t->right;

while(ptr->left!=NULL)//starting the loop to remove

{

back=ptr;

ptr=ptr->left;

}

t->data=ptr->data;

if (back==t)//starting condition

remove\_node(back->right);

else//if not satisfied

remove\_node(back->left);

}

}

void Tree::Display(){//to display the contents

help\_display(root);//call the recursive fuction to display data

}

void Tree::help\_display(Node \*r)//function to display the data recursively

{

if (r != NULL)

{

help\_display(r->left);//call left

cout<<r->data<<" ";//print data

help\_display(r->right);//call right

}

}

ostream& operator<<(ostream& out\_s, const Tree& b){ //overloading the << operator

Tree::Node\*n;

Tree::Node\*m;

n= b.root;

m=b.root;

cout<<n->data<<"";//displaying if null

while (n->right != NULL && n->left !=NULL)

{

n=n->left;

m=m->right;

out\_s<<n->data<<"";//displaying if data are present

out\_s<<m->data<<"";

}

return out\_s;

}

void Tree::destroy(Node \*r)//function to destroy the whole tree used in destructor

{

if(r!=NULL){

destroy(r->left);//recursion

destroy(r->right);//recursion

delete r;//remove r node

}

}

**List.h**

#include <iostream>

#ifndef LIST\_H

#define LIST\_H

using namespace std;

class Tree{

public://public functions of the class

typedef int Item; //defining the Item using int

struct Node

{

Item data;

Node\* left;

Node\* right;

};

Node\*root;

Tree(){root=NULL;}//constructor

bool Empty(){return root==NULL;}//check if the tree is empty or not

~Tree(); //destructor

int Length();//find out the length of tree

int r\_length(Node \*r);

bool Present(Item target);//Is a particular value present in tree

void Insert(Item entry);//To insert data into tree

void Remove(Item target);//To remove data from tree

void help\_remove(Node \* &t,Item target);//recursive for remove

void remove\_node(Node \* &t);//to remove a node

void Display();//to display the content of tree

void destroy(Node \*r);//to destroy the tree

void help\_display(Node \*r);//to help display function recursively

friend std::ostream& operator << (std::ostream& out\_s, Tree& Tree1); //this is a friend function

};

#endif // LIST\_H

**User Document**

The program can be used by the user to initialize a Binary tree. The user will be able to reinitialize the tree to be empty, insert values inside the list, remove values from the list, check if a value is present in the list, display the length of the list. To compile the program user can open the executable file. As soon as the .exe file opens, the user will be asked to enter the desired action to be taken. The user will be able to manipulate through the list by using special codes which are available in the menu, are available in the menu.

**Testing**

-Test 1

This program responds to commands the user enters to

manipulate an ordered list of integers, which is

initially empty. In the following commands, v is any

integer.

e – Re-initialize the list to be empty.

i v -- Insert the value into the list.

r v -- Remove the value from the list

l – Report the length of the list.

p v -- Is the value v present in the list?

w -- write out the list.

h -- See this menu.

q – Quit.

-->i 27

-->i 42

-->i 15

-->i 33

-->i 14

--> w

List: (14 15 27 33 42 )

-->r 22

-->p 22

The value is 22 is not present in the list.

-->p 42

The value is 42 is present in the list.

-->i 88

-->i 51

--> q

-Test 2

This program responds to commands the user enters to

manipulate an ordered list of integers, which is

initially empty. In the following commands, v is any

integer.

e – Re-initialize the list to be empty.

i v -- Insert the value into the list.

r v -- Remove the value from the list

l – Report the length of the list.

p v -- Is the value v present in the list?

w -- write out the list.

h -- See this menu.

q – Quit.

-->i 12

-->i 33

-->i 1

-->i 22

-->i 10

--> w

List: (10 1 12 22 33 )

-->r 10

-->p 10

The value is 10 is not present in the list.

-->p 22

The value is 22 is present in the list.

-->i 98

-->i 55

-->l

The list contains 6 elements

--> w

List: ( 1 12 55 98 22 33)

--> q

**Summary**

In this project, we have implemented a program that initializes a binary tree and allows the user to manipulate it. The program reads the special code from the user. The user can know about the special codes from the menu which is displayed to the user as soon as the program starts. The user can then insert, remove and perform many other included actions to the tree along with printing the values in the tree. I used three files to keep my codes organized and easy to access. The problems and their solutions involving Binary tree that we learnt in class helped a to solve the given question.