**Project-10**

**IMPLEMENTING AN ORDERED LIST WITH A BINARY SEARCH TREE**

**Name: Pawan Dhungana**

**Project no: 10**

**Due Date: December 03, 2018**

**Design Document**

**Introduction**

An ordered list is a sequence of elements, all of the same type, that is ordered by the elements' values. A binary search tree (BST) is a binary tree in which the value at each node is greater than the values in its left subtree and less than the values in its right subtree. This project implements an ordered list using a BST and exercises that list in a menu-driven program.

**Data Structures**

The program uses a class called BST to implement an ordered list ADT. Under this class, different functions and data member are declared. Item is used as the data type of the items in the list. Two constructors are used: BST() is an inline constructor and BST(const List& t) is a copy constructor. A destructor is used that deletes the dynamic memory in the tree that represents a list.

**Functions**

Eight member functions are used in the program: make\_empty(), insert(), remove(), menu(), empty(), length(),present()and a friend function. These functions are declared under private section of the class. Each of these functions have their private recursive function that takes in the root value as a parameter so are not kept in the public section of the class. Make\_empty() re-initializes the list to be empty, insert() inserts values into the list, remove() removes values from the list, length() returns the length(the number of values) of the list, present() determine if a particular value is present in the list, empty() checks if the list is empty, menu() displays the menu and the friend function writes the contents of the list.

Besides, BST() is an inline constructor and ~BST() is the destructor that deallocates the memory.

**Main Program**

The main program provides the list of choice to the user. The user can insert value, remove value, find length, determine if an element is present in the list, empty the list and look at the menu from the main program.

**User Document**

An ordered list is a sequence of elements, all of the same type, that is ordered by the elements' values. A binary search tree (BST) is a binary tree in which the value at each node is greater than the values in its left subtree and less than the values in its right subtree. This project implements an ordered list using a BST and exercises that list in a menu-driven program.

The program's name is Project10.cpp, to compile and run it, simply enter:

g++ Project10.cpp

a.out

A run of the program might look like this:

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 v into the list.

r v -- Remove the value v 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

The list : 14 15 27 33 42

--> r 33

--> w

The list: 14 15 27 42

--> p 22

The value 22 is NOT present in the list.

--> p 42

The value 42 is present in the list.

--> i 88

--> i 51

--> l

The list contains 6 values.

--> q

**Code Listing:**

#include<iostream>

#include<cstdlib>

using namespace std;

void menu();

class BST

{

public:

//constructors

BST() { root = NULL; }

//BST( const BST& t );

//destructor

~BST()

{

destroy(root);

}

//typedef

typedef int Item ;

//member functions

bool empty() const { return root == NULL; }

void make\_empty();

void insert ( const Item entry );

void remove ( Item target );

int length();

bool present ( Item entry );

void write();

friend std::ostream& operator <<( std::ostream& out\_s, const BST &t );

private:

//Data members

struct Node

{

Item data;

Node \*left;

Node \*right;

};

Node \*root;

//recursive functions

void destroy(Node\* &t);

void r\_insert(Node\* &t, Item entry);

void r\_remove ( Node\* &t, Item target );

void remove\_node ( Node\* &t );

int length(Node\* &t);

void help\_print ( std::ostream& out\_s, Node \*t ) const;

};

int main()

{

char select;

int entry;

int x;

BST b;

//displays the menu.

menu();

do

{

cout <<endl<< "--> ";

cin >> select;

switch ( select )

{

case 'e':

b.make\_empty();

break;

case 'i': cin >> entry ;

b.insert(entry);

break;

case 'w':

cout<<"The list: "<<b;

break;

case 'r':

cin>>entry;

b.remove(entry);

break;

case 'p':cin>>entry;

if(b.present(entry)==1)

cout<<"The value "<<entry<<" is present in the list.";

else

cout<<"The value "<<entry<<" is NOT present in the list.";

break;

case 'l':

x=b.length();

cout<<"The list contains "<<x<<" values.";

break;

case 'h': menu();

break;

default:

;

}

} while ( select != 'q' );

return 0;

}

void menu()

{

cout <<endl;

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

cout<<"an ordered list of integers, which is initially empty."<<endl;

cout<<"In the following commands, v is any 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;

}

void BST::insert(Item entry)

{

r\_insert(root,entry);

}

void BST::r\_insert(Node\* &t, Item entry)

{

if(t==NULL)

{

t=new Node;

t->data=entry;

t->left=NULL;

t->right=NULL;

}

else if(entry<t->data)

r\_insert(t->left,entry);

else //entry > t->data

r\_insert(t->right,entry);

}

void BST::remove ( Item target )

{

r\_remove ( root, target );

}

void BST::r\_remove ( Node\* &t, Item target )

{

if ( t->data == target )

remove\_node ( t );

else if ( target < t->data )

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

else // target > t->data

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

}

void BST::remove\_node ( Node\* &t )

{

Node\* ptr;

Node\* back;

if ( t->left == NULL && t->right == NULL ) // removing a leaf

{

delete t;

t = NULL;

}

else if ( t->left == NULL ) // removing a node with right child only

{

ptr = t;

t = t->right;

delete ptr;

}

else if ( t->right == NULL ) // removing a node with left child only

{

ptr = t;

t = t->left;

delete ptr;

}

else //removing a value whose node has two children

{

back = t;

ptr = t->right;

while ( ptr->left != NULL )

{

back = ptr;

ptr = ptr->left;

}

t->data = ptr->data;

if ( back == t )

remove\_node ( back->right );

else

remove\_node ( back->left );

}

}

bool BST::present(Item target)//to search if the value is in the list

{

Node \*p;

p=root;

while(true)

{

if(p==NULL)

return false;

if(target<p->data)

p=p->left;

else if(target==p->data)

return true;

else

p=p->right;

}

}

void BST::make\_empty() //to make the list empty

{

destroy(root);

root=NULL;

}

void BST::destroy(Node\* &p)

{

if(p!=NULL)

{

destroy(p->left);

destroy(p->right);

delete p;

}

}

int BST::length()//te return the length of the list

{

return length(root);

}

int BST::length(Node\* &p)

{

if(p==NULL)

return 0;

else

return length(p->left)+1+length(p->right);

}

std::ostream& operator << ( std::ostream& out\_s, const BST &t )

{

t.help\_print ( out\_s, t.root );

return out\_s;

}

void BST::help\_print ( std::ostream& out\_s, Node \*t ) const

{

if ( t != NULL )

{

help\_print ( out\_s, t->left );

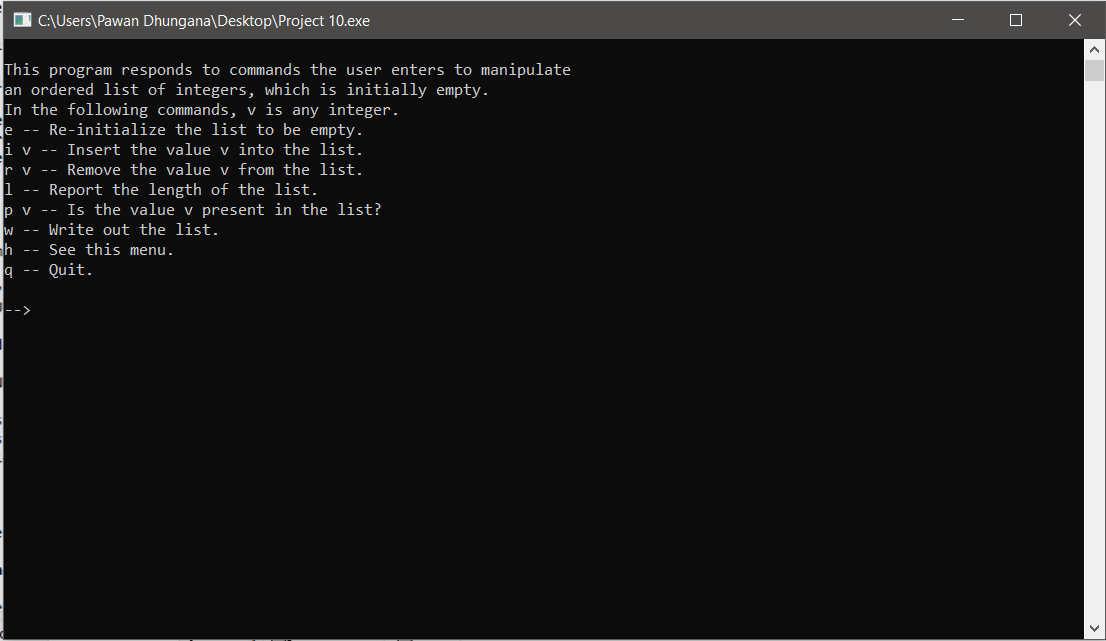
out\_s << t->data << ' ';

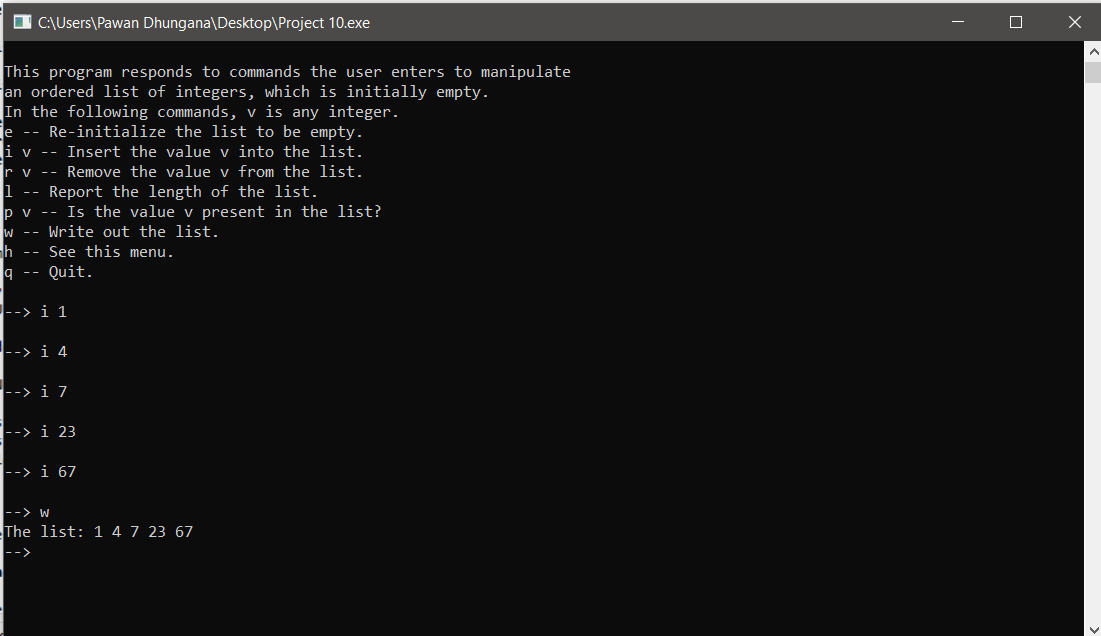
help\_print ( out\_s, t->right );

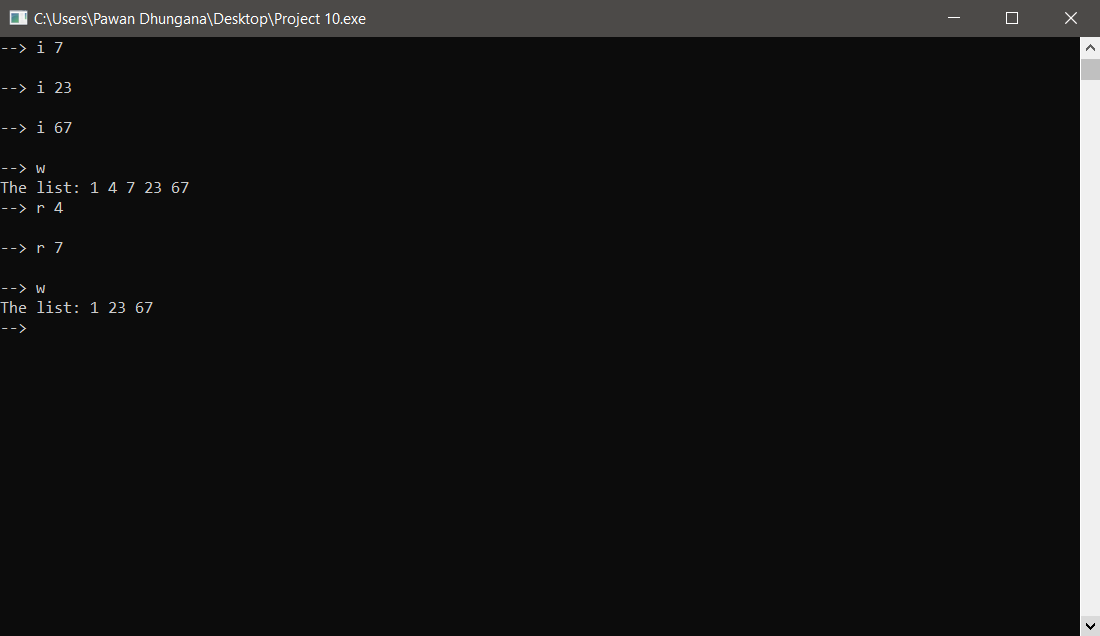
}

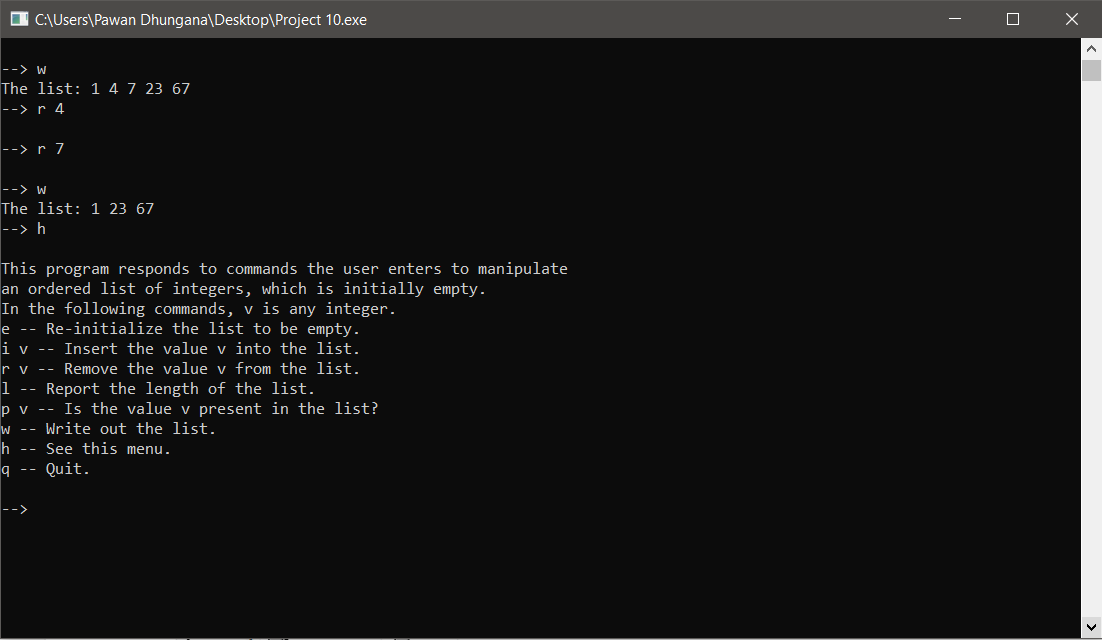
}

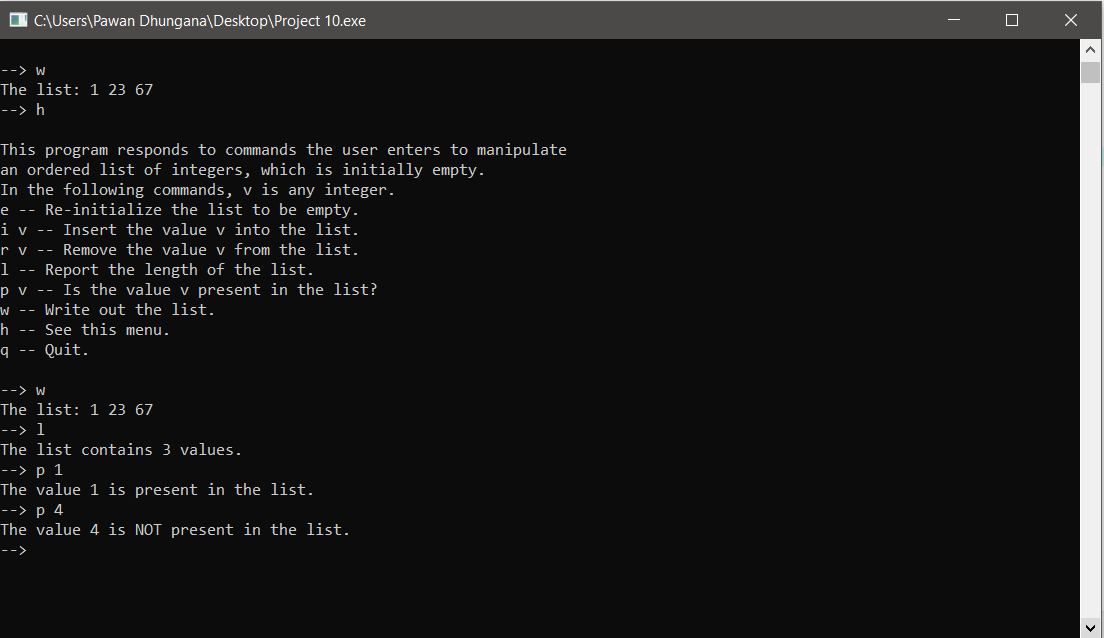
**Test Document**

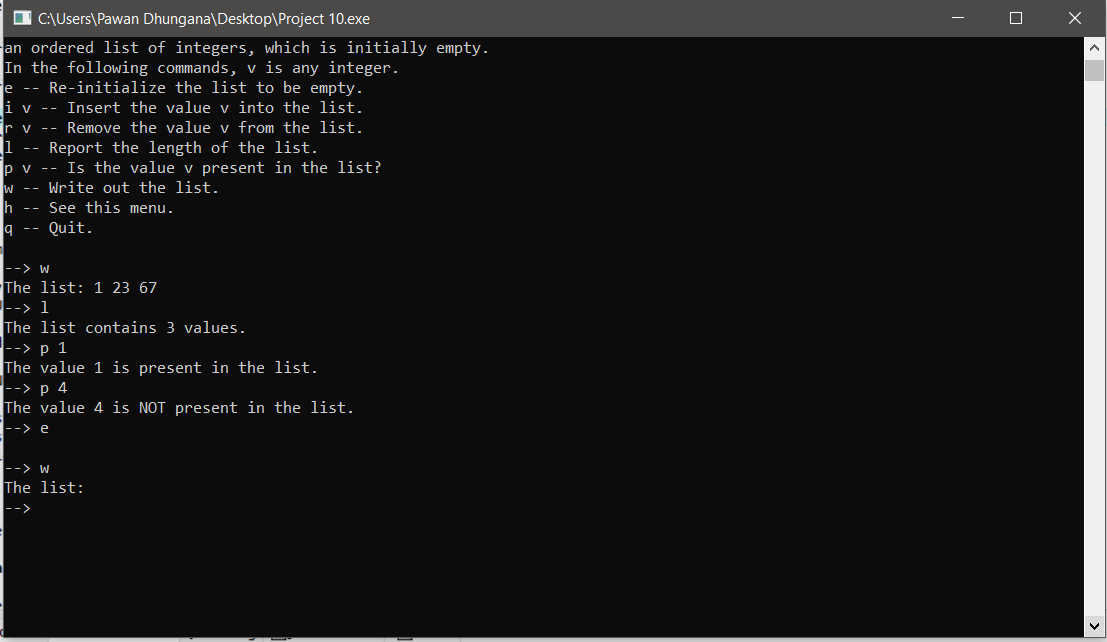


****

****

****

****

**Summary**

In this project, we implemented a program that implements an ordered list ADT using a pointer-based binary search tree. The program writes to the terminal instructions and a menu of commands, and it prompts for the user’s input. The program is a menu driven program and user has the options to insert, remove, write length, print elements, check presence of elements and empty the list.