

LAB 4

Implementaion of A BST

1.isEmpty()

The function returns true if the search Tree is empty.

With ARRAY:

The function checks if the element at the first position is -1 (considering -1 as a null value). If so, the function returns true.

File Help main.cpp - CE2020_Lab4_50_51 - Visual Studio Code

ArrayBST.h M ArrayBst.cpp M main.cpp U X AbstractBST.h M

main.cpp > main()

```
1 #include <iostream>
2 #include "AbstractBST.h"
3 #include "ArrayBST.h"
4 using namespace std;
5
6 int main(){
7     cout << "\n\n\n";
8     cout << "----- Implementation of Binary Search Tree -----"
9     << endl;
10    AbstractBST *binaryTree = new ArrayBst();
11    if(binaryTree->isEmpty()){
12        cout << "The binary tree is empty." << endl;
```

PROBLEMS DEBUG CONSOLE TERMINAL

----- Implementation of Binary Search Tree -----

The binary tree is empty.

The binary Search Tree is:

The Element is :0

The Element is :40

The Element is :35

The Element is :50

The Element is :20

The Element is :37

The Element is :48

The Element is :60

The Element is : -1

The Element is :21

The Element is : -1

Code: Express Off Git Graph Reconnect to Discord Ln 7, Col 19 Spaces: 4 UTF-8 CRLF C++ Go Live Win3

2.Add(key,value)

The function adds an element on the correct position in the binary search tree.

At first the function checks if the tree is empty .A while loop runs until the key is -1: If so the new node is added as the root of the tree. If the tree is not empty, the key of the node ToBeAdded is compared with the key of the current root.

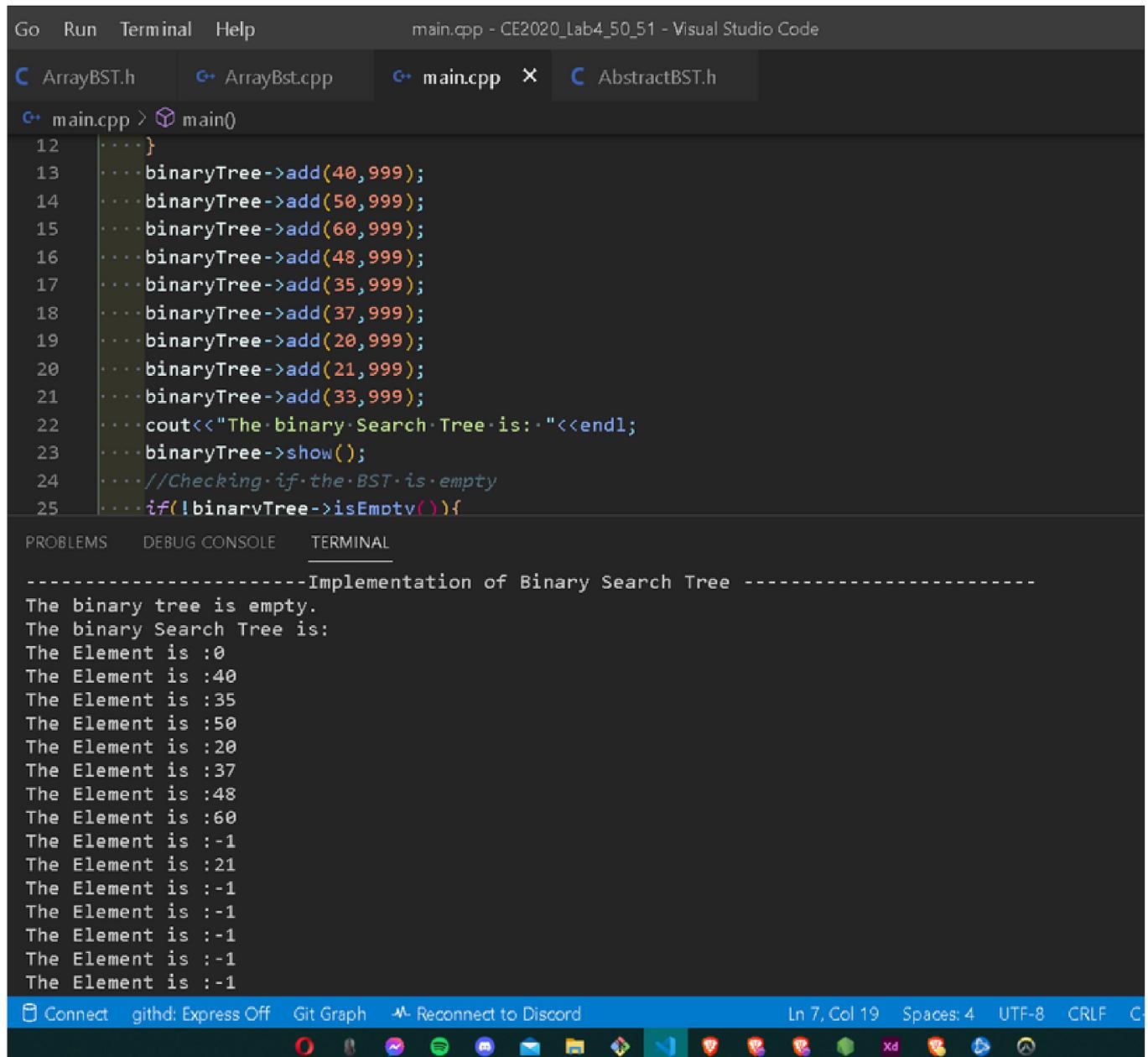
if root.key > nodeToBeAdded.key :

New root is set to be the left child:

else:

new root is set to be the right child:

This process is iterated until an empty node is found.



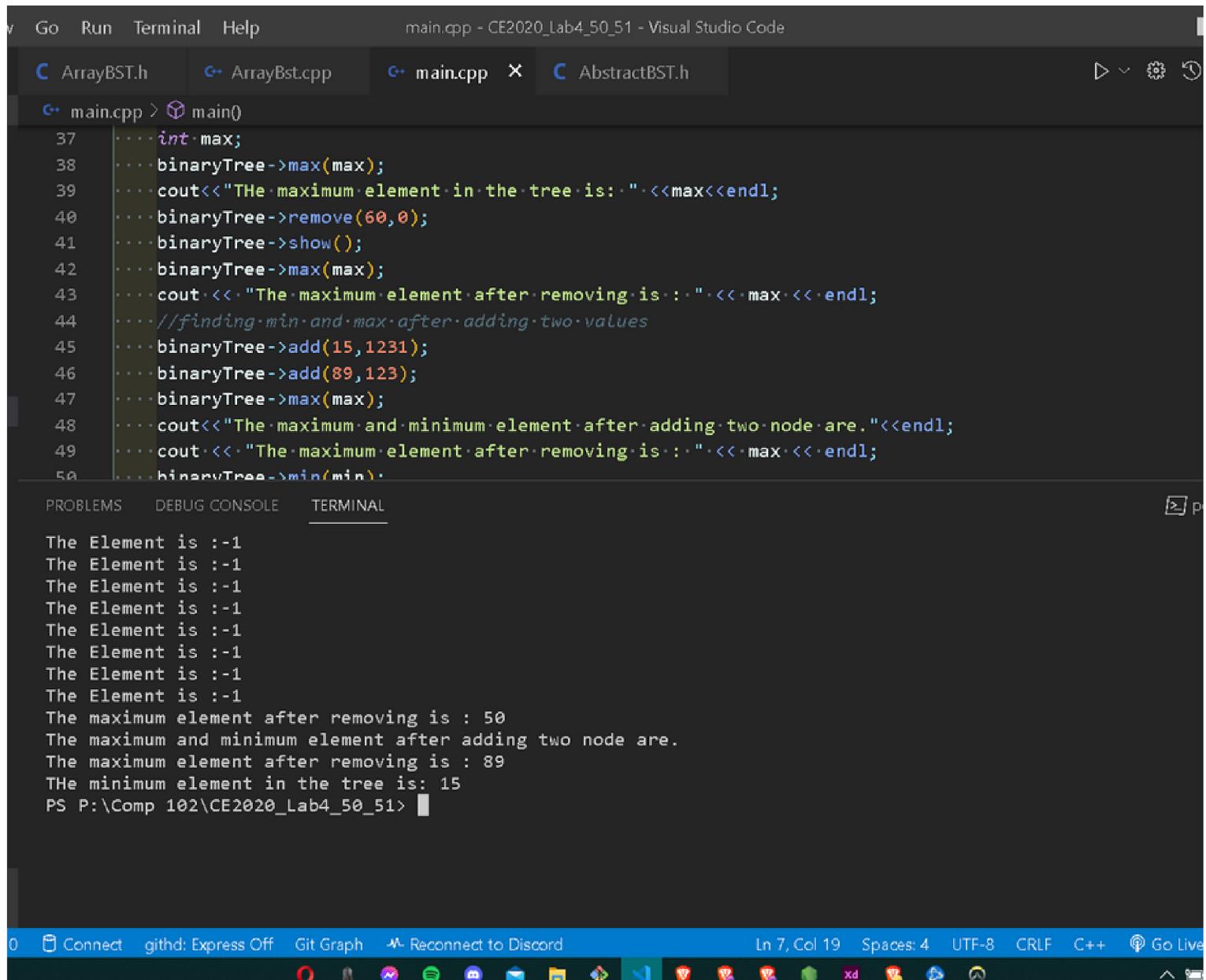
The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files: ArrayBST.h, ArrayBst.cpp, main.cpp (active), and AbstractBST.h.
- Code Editor:** Displays the `main.cpp` file with code for adding elements to a binary search tree and printing its structure.
- Output:** Shows the terminal output of the program execution.
- Bottom Status Bar:** Includes icons for GitHub, Express, Git Graph, Reconnect to Discord, and various system status indicators.

```
-----Implementation of Binary Search Tree -----
The binary tree is empty.
The binary Search Tree is:
The Element is :0
The Element is :40
The Element is :35
The Element is :50
The Element is :20
The Element is :37
The Element is :48
The Element is :60
The Element is : -1
The Element is :21
The Element is : -1
```

3.max()

A property of a binary search tree states that the element greater than the node is kept as the right child of the parent node. So the leftmost node is the maximum element in the binary search tree.
A for loop runs until the left most element is figured out.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files: ArrayBST.h, ArrayBst.cpp, main.cpp (active), and AbstractBST.h.
- Code Editor:** The main.cpp file contains C++ code for a binary search tree. It includes operations for adding nodes, removing nodes, and finding the maximum element. The code is as follows:

```
37 |     int max;
38 |     binaryTree->max(max);
39 |     cout<<"The maximum element in the tree is: "<<max<<endl;
40 |     binaryTree->remove(60,0);
41 |     binaryTree->show();
42 |     binaryTree->max(max);
43 |     cout<<"The maximum element after removing is: "<<max<<endl;
44 |     //finding min and max after adding two values
45 |     binaryTree->add(15,1231);
46 |     binaryTree->add(89,123);
47 |     binaryTree->max(max);
48 |     cout<<"The maximum and minimum element after adding two node are."<<endl;
49 |     cout<<"The maximum element after removing is: "<<max<<endl;
50 |     binaryTree->min(min);
```

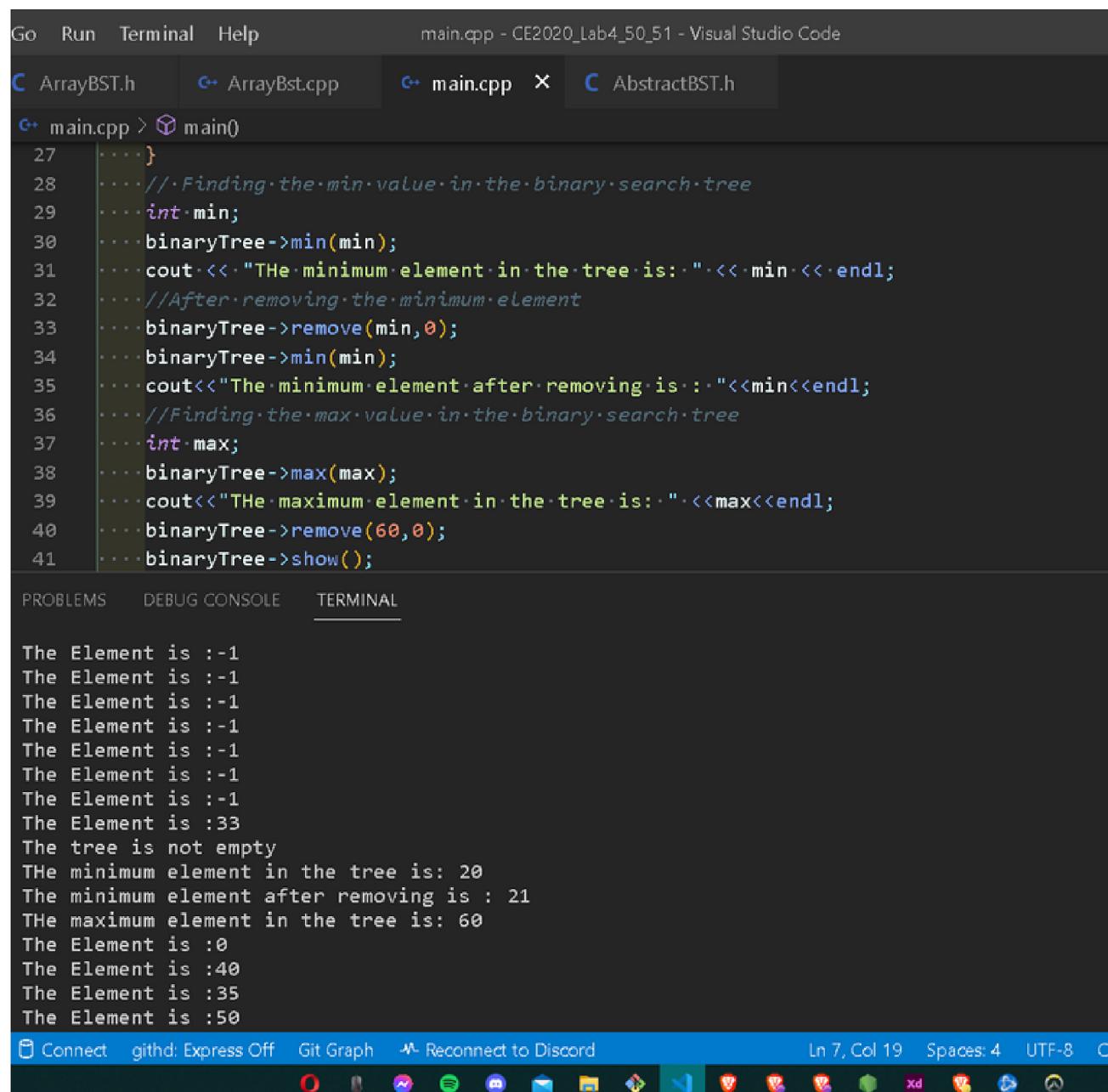
- Terminal:** Displays the output of the program. The output shows the maximum element being printed multiple times as -1, followed by the maximum element after removing 60 (50), and finally the maximum and minimum elements after adding two nodes (15 and 89).

```
The Element is :-1
The maximum element after removing is : 50
The maximum and minimum element after adding two node are.
The maximum element after removing is : 89
The minimum element in the tree is: 15
PS P:\Comp 102\CE2020_Lab4_50_51> |
```

- Bottom Bar:** Shows various icons for connecting to GitHub, Git Graph, Discord, and other tools.

4.min()

A property of a binary search tree states that the element smaller than the node is kept as the left child of the parent node. So the rightmost node is the minimum element in the binary search tree. A for loop runs until the rightmost element is figured out.



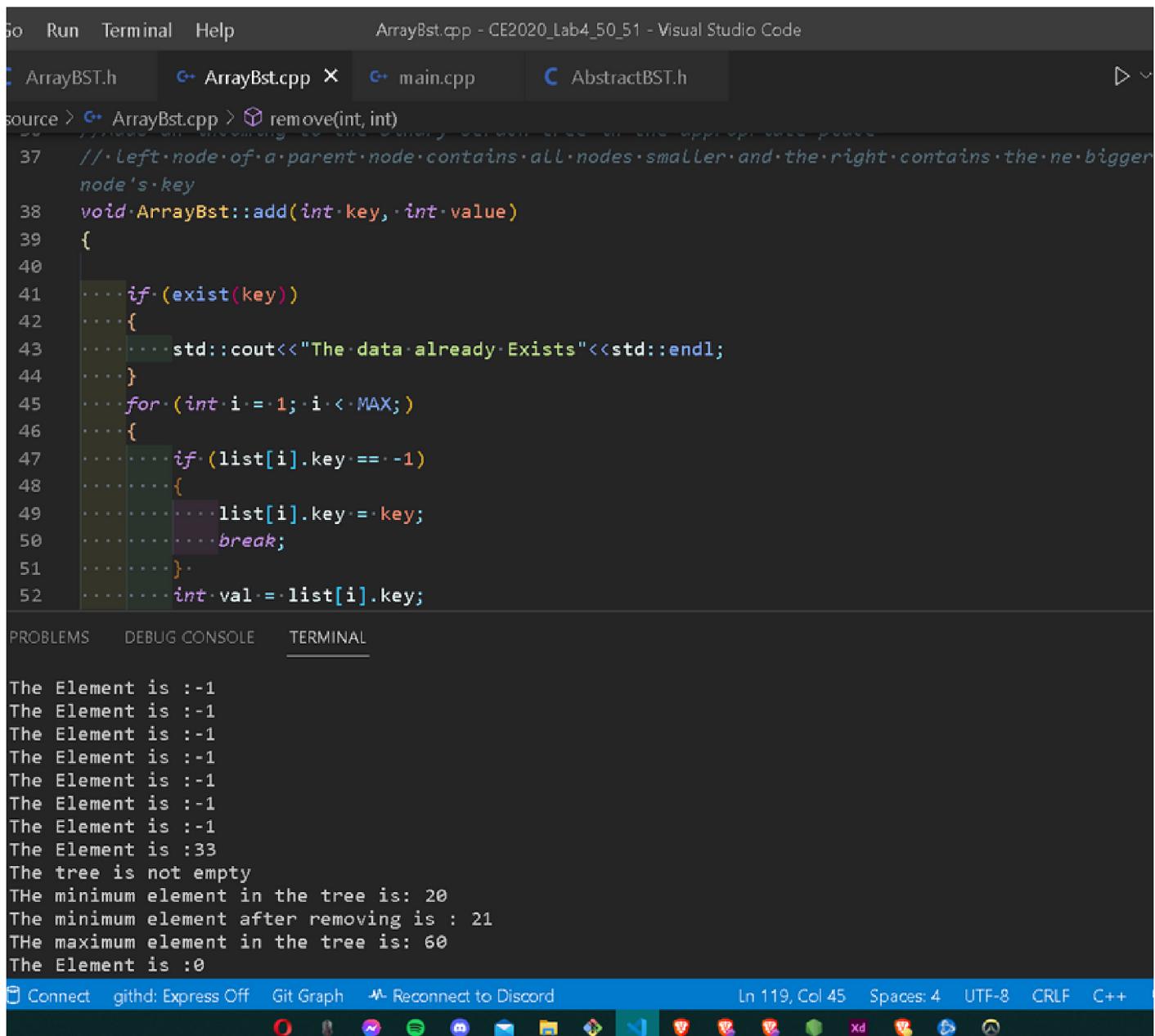
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files: ArrayBST.h, ArrayBst.cpp, main.cpp (selected), and AbstractBST.h.
- Code Editor:** Displays the main.cpp file content. The code implements a binary search tree and demonstrates finding the minimum and maximum values.
- Terminal:** Shows the output of the program execution. The output is:

```
The Element is :-1
The Element is :33
The tree is not empty
THe minimum element in the tree is: 20
The minimum element after removing is : 21
THe maximum element in the tree is: 60
The Element is :0
The Element is :40
The Element is :35
The Element is :50
```
- Bottom Status Bar:** Shows connection status (Connect, githd: Express Off, Git Graph, Reconnect to Discord), file information (Ln 7, Col 19, Spaces: 4, UTF-8), and a toolbar with various icons.

5.exist()

The exist function works similar to that of add. Only exceptions is that the key of node is also checked if it is equal to that of the root. If so true is returned , else iterates until a null node is found and false is returned in that case.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files: ArrayBST.h, ArrayBst.cpp (active), main.cpp, AbstractBST.h.
- Code Editor:** Displays the `ArrayBst.cpp` file. The code implements an array-based binary search tree. It includes a check for existing keys and a loop to find the index for a new key. The code is annotated with comments explaining its logic.
- Terminal:** Shows the output of the program's execution. The output includes:
 - Multiple lines of "The Element is :-1" (likely from a loop or a series of invalid inputs).
 - "The tree is not empty"
 - "THe minimum element in the tree is: 20"
 - "The minimum element after removing is : 21"
 - "THe maximum element in the tree is: 60"
 - "The Element is :0"
- Bottom Bar:** Includes icons for Connect, GitHub, Git Graph, Reconnect to Discord, and various extension icons. Status information: Line 119, Column 45, Spaces: 4, UTF-8, CRLF, C++.

6. show()

The function iterates through each element of the array and prints it in the console. The null node is also printed in this case.

Go Run Terminal Help main.cpp - CE2020_Lab4_50_51 - Visual Studio Code

C ArrayBST.h C ArrayBst.cpp C main.cpp X C AbstractBST.h

main.cpp > main()

```
17     ....binaryTree->add(35,999);
18     ....binaryTree->add(37,999);
19     ....binaryTree->add(20,999);
20     ....binaryTree->add(21,999);
21     ....binaryTree->add(33,999);
22     ....cout<<"The binary Search Tree is:"<<endl;
23     ....binaryTree->show();
24     ....//Checking if the BST is empty
25     ....if(!binaryTree->isEmpty()){
26     ....    cout<<"The tree is not empty"<<endl;
27     ....}
28     ....//Finding the min value in the binary search tree
```

PROBLEMS DEBUG CONSOLE TERMINAL

```
The Element is :40
The Element is :35
The Element is :50
The Element is :20
The Element is :37
The Element is :48
The Element is :60
The Element is : -1
The Element is :21
The Element is : -1
The Element is :33
```

Connect githd: Express Off Git Graph Reconnect to Discord Ln 7, Col 19 Spaces: 4 UTF-8 CRLF C

7.remove()

This function works similar to the exist function in the aspect of finding the node to be deleted.

After the node is found , node can have :

i) One child:

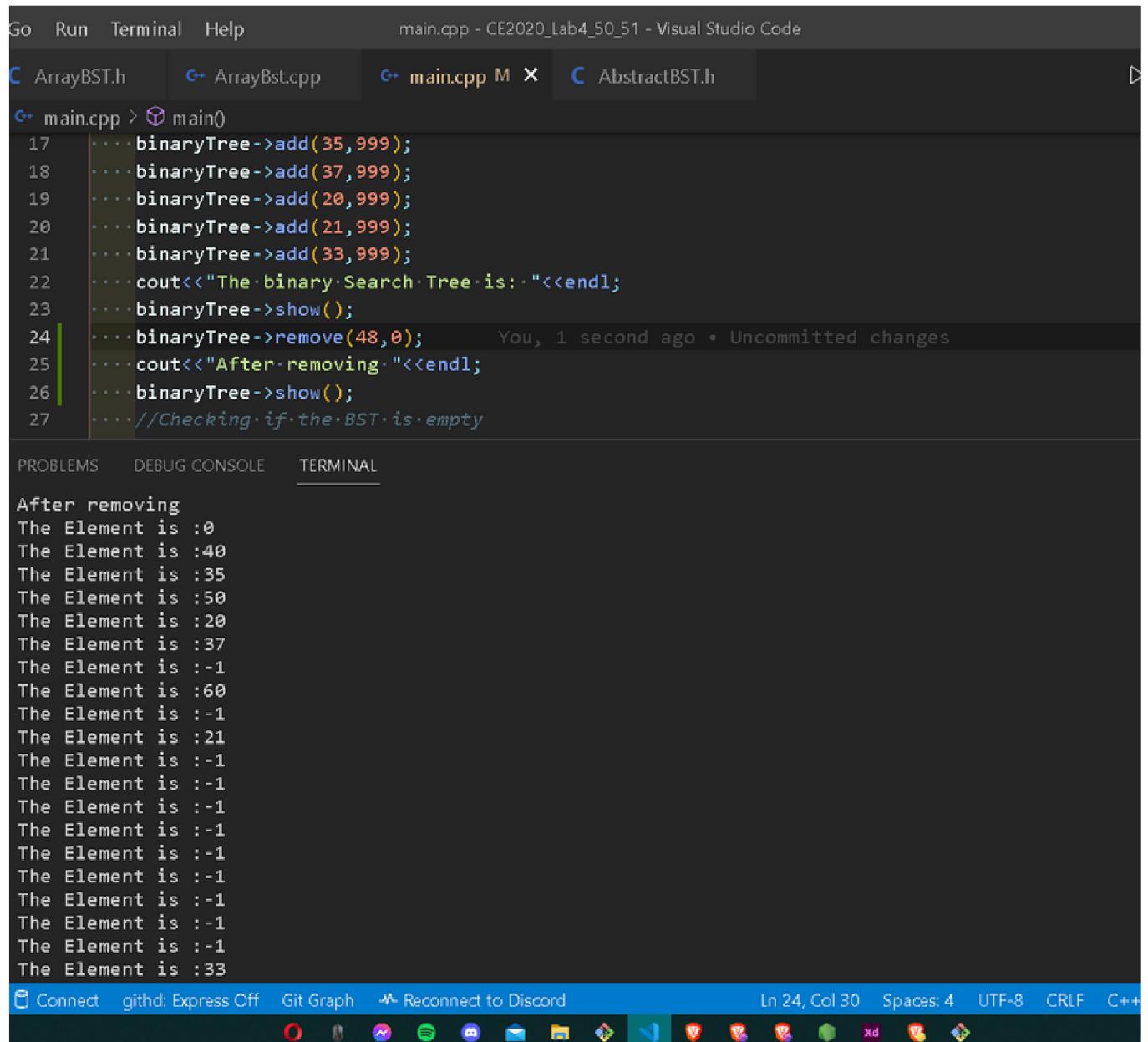
In this case , the node is removed and the child is kept as the new node in the parent's place. . And a function call is made to delete child.

ii) Two children.

In this case the largest element from the left sub tree replaces the node to be deleted. And a function call is made to delete the node with the largest element.

iii)No child:

The node is just deleted.



```
Go Run Terminal Help main.cpp - CE2020_Lab4_50_51 - Visual Studio Code
C ArrayBST.h C ArrayBst.cpp C main.cpp M X C AbstractBST.h
C main.cpp > main()
17     ...binaryTree->add(35,999);
18     ...binaryTree->add(37,999);
19     ...binaryTree->add(20,999);
20     ...binaryTree->add(21,999);
21     ...binaryTree->add(33,999);
22     ...cout<<"The binary Search Tree is: "<<endl;
23     ...binaryTree->show();
24     ...binaryTree->remove(48,0);      You, 1 second ago • Uncommitted changes
25     ...cout<<"After removing: "<<endl;
26     ...binaryTree->show();
27     ...//Checking if the BST is empty

PROBLEMS DEBUG CONSOLE TERMINAL

After removing
The Element is :0
The Element is :40
The Element is :35
The Element is :50
The Element is :20
The Element is :37
The Element is :1
The Element is :60
The Element is :1
The Element is :21
The Element is :1
The Element is :33

Ln 24, Col 30  Spaces: 4  UTF-8  CRLF  C++
```

Lab Report: Lab 4

Linked List Implementation of BST

1. Bool isEmpty():

This function checks whether the tree is empty or not. If the tree is empty it returns 0 otherwise it returns 1.

```
//function to determine if the tree is empty or not//
bool BST_linkedlist ::isEmpty (){
    if (root == NULL)
    {
        return true;
    }
    else {
        return false;
    }
}
```

2. Bool searchBST(*int targetkey*)

This function checks whether the key inserted is present in the **BST** or not. If the key is present, it returns 1(true) with a message “the Key is in the tree”, whereas returns 0(false) with a message “the key is not in the tree” if not

present.

```
bool BST_linkedlist ::searchBST(int targetkey){//function to check whether the key is present in the tree or not
    if(isEmpty()){
        cout<<"The tree is empty"<<endl;
        return 0;
    }
    node*temp = root;
    while(temp != NULL){
        if (temp -> key == targetkey) {
            cout<<"The key is in the tree"<<endl;
            return true;
        }
        else if(temp->key>targetkey){
            temp=temp->left_child;
        }
        else if(temp->key<targetkey){
            temp=temp->right_child;
        }
    }
    cout<<"The key is not in the tree"<<endl;
    return false;
}
```

3. Void addBST(*node *targetnode, node *newnode*):

This function adds node into the tree. The targetnode is root by default,

where as the newnode is the node to be added.

```
void BST_linkedlist :: addBST(node *targetnode, node *newnode){//function to add node into the tree

    if(this->root==NULL){
        root = newnode;

        cout<<"Root added "<<endl;
        //showData(root);
    }

    else if(targetnode->key > newnode->key){
        if(targetnode->left_child == nullptr) // if left child's position is empty
        {
            targetnode->left_child = newnode;
        }
        else
        {
            addBST(targetnode->left_child, newnode);
        }
    }
    else {
        if(targetnode->right_child == nullptr) // if right child's position is empty
        {
            targetnode->right_child = newnode;
        }
        else
        {
            addBST(targetnode->right_child, newnode);
        }
    }
}
```

4. Void adddata(*int key, int value*):

This function passes the key and the data of the node that is to be inserted into the tree.

```
void BST_linkedlist ::adddata(int key, int value){ //function to add data into the tree
    node *addednode = new node();
    addednode ->key = key;
    addednode ->data = value;

    addBST(root, addednode);
}
```

5. Void removeBST(*int keyToDelete*):

This function is supposed to remove the node with the given key.

```
88 void BST_linkedlist ::removeBST(int keyToDelete)// function to remove the key from the tree
89 {
90     node *temp = root;
91     if(temp->key == keyToDelete){
92         node *temp2 = temp->left_child;
93         node *parent2 = temp;
94         while(temp2->right_child != nullptr){
95             parent2 = temp;
96             temp2 = temp2->right_child;
97         }
98         parent2->right_child = nullptr;
99         temp2->left_child = temp->left_child;
100        temp2->right_child = temp->right_child;
101        delete[] temp;
102        root = temp2;
103        return;
104    }
105    node *parent = root;
106    while(true){
107        if(temp->key == keyToDelete
108        )
109        {
110            break;
111        }else if(temp->key < keyToDelete
112        ){
113            parent = temp;
114            temp = temp->right_child;
115        }else if(temp->key > keyToDelete
116        ){
117            parent = temp;
118            temp = temp->left_child;
119        }
120        if(temp == nullptr){
121            return;
122        }
123    }
124 }
```

```

122     }
123 }
124 ~> if(temp->left_child == nullptr && temp->right_child == nullptr)
125 {
126
127     delete[] temp;
128     return;
129 ~> }else if(temp->left_child == nullptr && temp->right_child != nullptr){
130 ~>     if(parent->left_child == temp){
131         parent->left_child = temp->right_child;
132 ~>     }else{
133         parent->right_child = temp->right_child;
134     }
135     delete[] temp;
136     return;
137 ~> }else if(temp->left_child != nullptr && temp->right_child == nullptr){
138 ~>     if(parent->left_child == temp){
139         parent->left_child = temp->left_child;
140 ~>     }else{
141         parent->right_child = temp->left_child;
142     }
143     delete[] temp;
144     return;
145 ~> }else{
146     node *temp2 = temp->left_child;
147     node *parent2 = temp;
148 ~>     while(temp2->right_child != nullptr){
149         parent2 = temp;
150         temp2 = temp2->right_child;
151     }
152
153     parent2->right_child = nullptr;
154     if(parent->left_child == temp){
155         parent->left_child = temp2;
156     }else{
157         parent->right_child = temp2;
158     }
159     temp2->left_child = temp->left_child;
160     temp2->right_child = temp->right_child;
161     delete[] temp;
162     return;
163 }
164
165 }

```

6. int min (node *tempo):

This returns the minimum valued key in the tree.

7. int max (node *tempo):

This returns the maximum valued key in the tree.

```
int BST_linkedlist :: min(node *tempo){ //function to return the minimum key in the tree
    node *temporary = tempo;
    while (temporary->left_child != NULL){
        temporary = temporary->left_child;
    }
    return temporary->key;
}

int BST_linkedlist :: max(node *tempo){ //function to calculate the maximum key in the tree and return it
    node *temporary = tempo;
    while (temporary->right_child != NULL){
        temporary = temporary->right_child;
    }
    return temporary->key;
}
```

8. void showdata(node *n):

This shows the data of the desired node.

```
void BST_linkedlist ::showData(node *n){ //shows the data at the desired node
    if (n== NULL){
        cout<<"\n\nkey==NULL" << "value==NULL" << endl;
    }
    cout <<"Key = " << n->key << endl;
    cout <<"Data = " << n->data << endl;
}
```

OUTPUT:

```
PS D:\Labworks\LAB \$\CE2020_Lab4_50_51> g++ main.cpp .\source\linkedlistBST.cpp .\source\ArrayBst.cpp -I include  
PS D:\Labworks\LAB \$\CE2020_Lab4_50_51> ./a.exe
```

Which implementation do you want to see?

Press 1 for Array
Press 2 for Linkedlist

-----Implementation of BST using linkedlist-----

The list is empty
Root added

Checking if the tree is empty
The list is not empty
The smallest key in the tree is 1
The largest key in the tree is 52
Key = 12
Data = 222

Node removed
Key = 12
Data = 222
The key is in the tree
The key is not in the tree
PS D:\Labworks\LAB \\$\CE2020_Lab4_50_51> █