NATIONAL UNIVERSITY
OF COMPUTER SCIENCE AND TECHNOLOGY

Assignment #3

Student name: *Eqan Ahmad*
19F0256

Course: *Data Structures(CS 218)* – Professor: *Dr Hashim Yasin*
Publishing Date: *November 29, 2020*

Question 1

Terminal Color Scheme

```
#include <ostream>
namespace Color {
  const std::string red("\033[1;31m");
  const std::string green("\033[1;32m");
  const std::string blue("\033[1;34m");
  const std::string yellow("\033[1;33m");
  const std::string cyan("\033[1;36m");
  const std::string magenta("\033[1;35m");
  const std::string reset("\033[0m");
}
```

C++ Custom Shell

```
#include <stdlib.h>
#include <iostream>
#include "colormod.h"
class CShell
{
public:
    int input()
    {
        std::string cmdin;
        std::cin >> cmdin;
        if (cmdin == "clear")
        {
            system("clear");
            return 10;
        }
        else if (cmdin == "history")
        {
            system("history");
            return 10;
        }
        else if (cmdin == "times")
        {
            system("times");
            return 10;
        }
        else if (cmdin == "time")
        {
            system("time");
            return 10;
        }
        else if (cmdin == "exit")
        {
            exit(0);
            return 10;
        }
        else if (cmdin == "end")
            return 0;
        else if (cmdin >= "0" && cmdin <= "9")
            return stoi(cmdin);
        return 0;
    }
};
```

Header File

```cpp
#include <queue>
#include <string>
#include "C++Shell.h" // ? Custom Made Shell Implementation specifically made for this Program
using namespace std;  // ? Adding this at the header files as we are following just the standard C++ implementation
                      // NOTE The binary tree is being aligned on the basis of the CGPA of a person

struct node
{
    float cgpa;
    int rollNumber;
    string name;
    node *left = NULL;
    node *right = NULL;
};
node *rootNode = NULL;
node *createNode(const float &cgpa, const int &rollNumber, const std::string &name)
{
    node *tempNode;
    tempNode = new node;
    tempNode->cgpa = cgpa;
    tempNode->rollNumber = rollNumber;
    tempNode->name = name;
    return tempNode;
}
class BST
{
public:
    void mainMenu();
    static int countNode;

private:
    CShell obj;
    node *inputNode(node *traverseNode, const float &cgpa, const int &rollNumber, const string &name);
    string dataTypes(const int &index);
    void readFile();
    void writeFile();
    void dataExtraction(string &file);
    node *largestNode(node *index);
    node *smallestNode(node *index);
    node *deleteNode(node *traverseNode, const int &rollNumber);
    int height(node *traverseNode);
    inline int numberOfEdges(node *tempNode);
    int maxDepth(node *rootNode);
    void levelorderTraversal(node *rootNode);
    void inorderTraversal(node *traverseNode, ofstream &outFile) const;
    void preorderTraversal(node *traverseNode, ofstream &outFile) const;
    void postorderTraversal(node *traverseNode, ofstream &outFile) const;
    bool searchParticularProfile(node *traverseNode, const int &rollNumber);
    void printNodesAtLevel(node *rootNode, const int &currentLevel,
   const int &destLevel);
    bool isComplete(node *rootNode, const int &index);
    int numberOfNodes(node *traverseNode);
    inline void dummyFunction(const int &option);
    inline void dummyFunction();
};
int BST::countNode = 0;
```

CPP File

```cpp
#include "DSCourse3-Q1.h"
#include <fstream>

node *BST::inputNode(node *traverseNode, const float &cgpa, const int &rollNumber, const string &name)
{
    if (traverseNode == NULL)
    {
        countNode++;
        cout << Color::cyan << "[+] Data has been successfully added!\n";
        traverseNode = createNode(cgpa, rollNumber, name);
        return traverseNode;
    }
    else if (cgpa <= traverseNode->cgpa)
        traverseNode->left = inputNode(traverseNode->left, cgpa, rollNumber, name);
    else
        traverseNode->right = inputNode(traverseNode->right, cgpa, rollNumber, name);
    return traverseNode;
}
node *BST::largestNode(node *index)
{
    while (index->right != NULL)
        index = index->right;
    return index;
}
node *BST::smallestNode(node *index)
{
    while (index->left != NULL)
        index = index->left;
    return index;
}
node *BST::deleteNode(node *traverseNode, const int &rollNumber)
{
    if (traverseNode == NULL)
    {
        cout << "Node not found!\n";
        return traverseNode;
    }
    else if (rollNumber < traverseNode->rollNumber)
        traverseNode->left = deleteNode(traverseNode->left, rollNumber);
    else if (rollNumber > traverseNode->rollNumber)
        traverseNode->right = deleteNode(traverseNode->right, rollNumber);
    else if (rollNumber == traverseNode->rollNumber)
    {
        if (traverseNode->left == NULL && traverseNode->right == NULL) // ? Already is a Leaf Node
            free(traverseNode);
        else if (traverseNode->left == NULL) // ? Right Child is present
        {
            node *temp = traverseNode;
            traverseNode = traverseNode->right;
            free(temp);
        }
        else if (traverseNode->right == NULL) // ? Left child is present
        {
            node *temp = traverseNode;
            traverseNode = traverseNode->left;
            free(temp);
        }
        else // ? Both left and Right childs are present
        {
            node *temp = smallestNode(traverseNode->right);
            traverseNode->rollNumber = temp->rollNumber;
                // ? Not deleting the node just repla cing its cgpa, as
    its the parent node with unpredicted childs
            traverseNode->right = deleteNode(traverseNode->right, temp->
    rollNumber); // ? Now sending the original cgpa for deletion
        }
        countNode--;
        cout << Color::cyan << "[+] Node has been successfully
    eliminated!\n";
    }

    return traverseNode;
}
int BST::height(node *traverseNode)
{
    if (traverseNode == NULL)
        return 0;
    int leftHeight = height(traverseNode->left);
    int rightHeight = height(traverseNode->right);
    int currentHeight = max(leftHeight, rightHeight) + 1;
    return currentHeight;
}
inline int BST::numberOfEdges(node *tempNode)
{
    return tempNode == NULL ? 0 : numberOfNodes(tempNode) - 1;
}
int BST::maxDepth(node *rootNode)
{
    if (rootNode == NULL)
        return 0;
    int leftDepth = maxDepth(rootNode->left);
    int rightDepth = maxDepth(rootNode->right);
    return (leftDepth > rightDepth) ? ++leftDepth : ++rightDepth;
}
void BST::levelorderTraversal(node *rootNode)
{
    ofstream outFile("data.txt");
    if (rootNode == NULL)
        return;
    queue<node *> traverseNode; // ? To temporarily store nodes and
    print later
    traverseNode.push(rootNode);
    while (!traverseNode.empty())
    {
        node *currentNode = traverseNode.front();
        cout << Color::yellow << "Name: " << currentNode->name << ' ' <<
     "CGPA: " << currentNode->cgpa << ' ' << "Roll Number: " <<
    currentNode->rollNumber << '\n';
```

```
        outFile << "Name: " << currentNode->name << '\n'
                << "CGPA: " << currentNode->cgpa << '\n'
                << "RollNumber: " << currentNode->rollNumber << '\n';
        if (currentNode->left != NULL)
            traverseNode.push(currentNode->left);
        if (currentNode->right != NULL)
            traverseNode.push(currentNode->right);
        traverseNode.pop();
    }
    outFile.close();
}
void BST::inorderTraversal(node *traverseNode, ofstream &outFile) const
{
    if (traverseNode == NULL)
        return;
    inorderTraversal(traverseNode->left, outFile);
    cout << Color::yellow << "Name: " << traverseNode->name << ' ' << "
   CGPA: " << traverseNode->cgpa << ' ' << "Roll Number: " <<
   traverseNode->rollNumber << '\n';
    outFile << "Name: " << traverseNode->name << '\n'
            << "CGPA: " << traverseNode->cgpa << '\n'
            << "RollNumber: " << traverseNode->rollNumber << '\n';
    inorderTraversal(traverseNode->right, outFile);
}
void BST::preorderTraversal(node *traverseNode, ofstream &outFile) const
{
    if (traverseNode == NULL)
        return;
    outFile << "Name: " << traverseNode->name << '\n'
            << "CGPA: " << traverseNode->cgpa << '\n'
            << "RollNumber: " << traverseNode->rollNumber << '\n';
    cout << Color::yellow << "Name: " << traverseNode->name << ' ' << "
   CGPA: " << traverseNode->cgpa << ' ' << "Roll Number: " <<
   traverseNode->rollNumber << '\n';
    preorderTraversal(traverseNode->left, outFile);
    preorderTraversal(traverseNode->right, outFile);
}
void BST::postorderTraversal(node *traverseNode, ofstream &outFile)
   const
{
    if (traverseNode == NULL)
        return;
    postorderTraversal(traverseNode->left, outFile);
    postorderTraversal(traverseNode->right, outFile);
    outFile << "Name: " << traverseNode->name << '\n'
            << "CGPA: " << traverseNode->cgpa << '\n'
            << "RollNumber: " << traverseNode->rollNumber << '\n';
    cout << Color::yellow << "Name: " << traverseNode->name << ' ' << "
   CGPA: " << traverseNode->cgpa << ' ' << "Roll Number: " <<
   traverseNode->rollNumber << '\n';
}
bool BST::searchParticularProfile(node *traverseNode, const int &
   rollNumber)
{
    if (traverseNode == NULL)
        return false;

    if (traverseNode->rollNumber == rollNumber)
    {
        cout << Color::yellow << "Name: " << traverseNode->name << ' ' << "CGPA: " << traverseNode->cgpa << '\n';
        return true;
    }
    bool leftResult = searchParticularProfile(traverseNode->left, rollNumber);
    if (leftResult)
        return true;
    bool rightResult = searchParticularProfile(traverseNode->right, rollNumber);
    return rightResult;
}
void BST::printNodesAtLevel(node *rootNode, const int &currentLevel, const int &destLevel)
{
    if (rootNode == NULL)
        return;
    if (currentLevel == destLevel)
    {
        cout << Color::yellow << "Name: " << rootNode->name << ' ' << "CGPA: " << rootNode->cgpa << ' ' << "Roll Number: " << rootNode->rollNumber << '\n';
        return;
    }
    printNodesAtLevel(rootNode->left, currentLevel + 1, destLevel);
    printNodesAtLevel(rootNode->right, currentLevel + 1, destLevel);
}
bool BST::isComplete(node *rootNode, const int &index)
{
    int numberOfNode = numberOfNodes(rootNode);
    if (rootNode == NULL)
        return (true);
    if (index >= numberOfNode)
        return (false);
    return (isComplete(rootNode->left, 2 * index + 1) && isComplete(rootNode->right, 2 * index + 2));
}
int BST::numberOfNodes(node *traverseNode)
{
    if (traverseNode == NULL)
        return 0;
    return 1 + numberOfNodes(traverseNode->left) + numberOfNodes(traverseNode->right);
}
void BST::dummyFunction(const int &option)
{
    string dummy;
    system("clear");
    cout << Color::cyan << "Press Any Key[Except Enter] To Continue\n";
    cin >> dummy;
}
void BST::dummyFunction()
{
    string dummy;
    cout << Color::cyan << "Press Any Key[Except Enter] To Continue\n";
    cin >> dummy;
    system("clear");
```

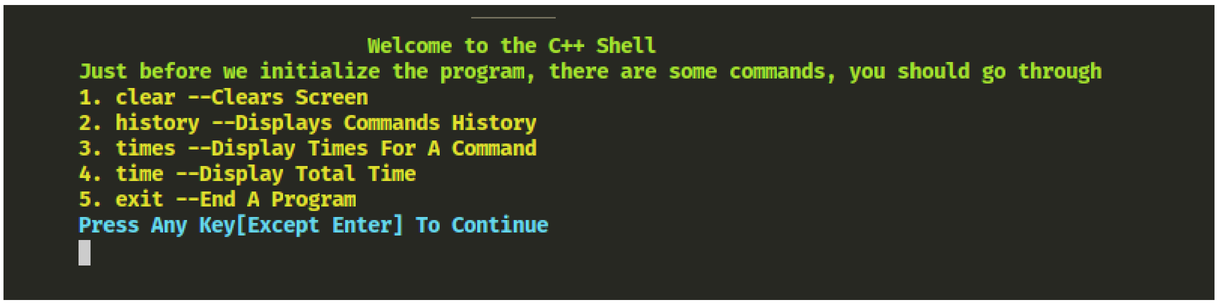
```
}
string BST::dataTypes(const int &index)
{
    switch (index)
    {
    case 0:
        return "Name: ";
        break;
    case 1:
        return "CGPA: ";
        break;
    case 2:
        return "RollNumber: ";
        break;
    }
    return "";
}

void BST::dataExtraction(string &file)
{
    string tempString, resultantString, name;
    int rollNumber, j = 0;
    float cgpa;
    for (int i = 0; i < file.length(); i++)
    {
        if (tempString == dataTypes(j))
        {
            resultantString += file[i];
            if (file[i] == '\n')
            {
                if (j == 0)
                    name = resultantString;
                else if (j == 1)
                    cgpa = stof(resultantString);
                else if (j == 2)
                {
                    rollNumber = stoi(resultantString);
                    rootNode = inputNode(rootNode, cgpa, rollNumber,
   name);
                    j = -1;
                }
                j++;
                tempString = "", resultantString = "";
            }
        }
        else if (file[i] == '\n')
            tempString = "";
        else
            tempString += file[i];
    }
}
void BST::readFile()
{
    string tempData, file;
    ifstream inputFile("data.txt");
    int i = 0;
    if (inputFile.is_open())
    {
        while (getline(inputFile, tempData))
            file += tempData + "\n";
        inputFile.close();
    }
    else
        cout << "Unable to open file";
    cout << Color::yellow << file << '\n';
    cout << Color::cyan << "[+] Information Extracted Successfully!\n";
    dataExtraction(file);
}
void BST::writeFile()
{
    if (rootNode == NULL)
        readFile();
    if (rootNode == NULL)
        return;
    cout << Color::magenta << "0. PreOrder\n1. PostOrder\n2. LevelOrder\
n3. InOrder\n";
    cout << Color::cyan << "CShell > ";
    int option = obj.input();
    node *traverseNode = rootNode;
    switch (option)
    {
    case 0:
    {
        ofstream outFile("data.txt");
        preorderTraversal(traverseNode, outFile);
        outFile.close();
        break;
    }
    case 1:
    {
        ofstream outFile("data.txt");
        postorderTraversal(traverseNode, outFile);
        outFile.close();
        break;
    }
    case 2:
        levelorderTraversal(traverseNode);
        break;
    case 3:
    {
        ofstream outFile("data.txt");
        inorderTraversal(traverseNode, outFile);
        outFile.close();
        break;
    }
    default:
        cout << Color::red << "[-] The Option is out of bounds, Please
try again\n";
        break;
    }
}

void BST::mainMenu()
{
    int rollNumber, option;
    float cgpa;
    string name;
    node *traverseNode;
    system("clear");
    cout << Color::green << "\t\t\tWelcome to the C++ Shell \nJust before we initialize the program, there are some commands, you should go through\n";
    cout << Color::yellow << "1. clear --Clears Screen\n2. history --Displays Commands History\n3. times --Display Times For A Command\n4. time --Display Total Time\n5. exit --End A Program\n";
    dummyFunction();
loop:
    cout << Color::magenta << "0. Extract File\n1. Insertion\n2. Traverse & Display\n3. Smallest & Largest GPA Profile\n4. Search Profile\n5. Display All Nodes At A particular Level\n6. Delete A Particular Node\n7. BST Balance Check\n8. Extra Functions\n";
    cout << Color::cyan << "CShell > ";
    option = obj.input();
    if (option == 10)
    {
        dummyFunction(option);
        goto loop;
    }
    switch (option)
    {
    case 0:
        system("clear");
        readFile();
        break;
    case 1:
    {
        system("clear");
        cout << Color::yellow << "Enter Name[Press '.' & Enter To Stop]: ";
        cin.ignore();
        getline(cin, name, '.');
        cout << "Enter RollNumber: ";
        cin >> rollNumber;
        cout << "Enter CGPA: ";
        cin >> cgpa;
        rootNode = inputNode(rootNode, cgpa, rollNumber, name);
        break;
    }
    case 2:
        system("clear");
        writeFile();
        break;
    case 3:
    {
        system("clear");
        cout << Color::magenta << "0. Smallest Node\n1. Largest Node\n";
        cout << Color::cyan << "CShell > ";
        int option = obj.input();
        traverseNode = rootNode;
        switch (option)
        {
        case 0:
            traverseNode = smallestNode(traverseNode);
            cout << Color::yellow << "Name: " << traverseNode->name << '
```

```
 ' << "CGPA: " << traverseNode->cgpa << ' ' << "Roll Number: " <<
traverseNode->rollNumber << '\n';
                break;
            case 1:
                traverseNode = largestNode(traverseNode);
                cout << Color::yellow << "Name: " << traverseNode->name << '
 ' << "CGPA: " << traverseNode->cgpa << ' ' << "Roll Number: " <<
traverseNode->rollNumber << '\n';
                break;
            default:
                cout << Color::red << "[-] The Option is out of bounds,
Please try again\n";
                break;
            }
        }
        break;
        case 4:
            system("clear");
            traverseNode = rootNode;
            cout << Color::yellow << "Enter RollNumber: ";
            cin >> rollNumber;
            if (!searchParticularProfile(traverseNode, rollNumber))
                cout << Color::red << "[-] Profile Not Found\n";
            break;
        case 5:
            system("clear");
            int currentLevel, destLevel;
            traverseNode = rootNode;
            cout << Color::yellow << "Enter Current Level";
            cin >> currentLevel;
            cout << "Enter Destination Level: ";
            cin >> destLevel;
            printNodesAtLevel(traverseNode, currentLevel, destLevel);
            break;
        case 6:
            system("clear");
            cout << Color::yellow << "Enter RollNumber: ";
            cin >> rollNumber;
            traverseNode = rootNode;
            deleteNode(traverseNode, rollNumber);
            break;
        case 7:
            system("clear");
            traverseNode = rootNode;
            cout << Color::yellow << "Enter Level: ";
            cin >> option;
            (isComplete(traverseNode, option)) ? (cout << Color::green << "
The binary tree is complete!\n") : (cout << Color::red << "The binary
 tree is not complete!\n");
            break;
        case 8:
            system("clear");
            cout << Color::magenta << "0. Height Of Tree\n1. Number Of Edges
 Of Tree\n2. Depth Of Tree\n";
            cout << Color::cyan << "CShell > ";
            option = obj.input();
            traverseNode = rootNode;
            switch (option)
            {
            case 0:
                cout << Color::cyan << "Height Of Tree: " << height(traverseNode) << '\n';
                break;
            case 1:
                cout << Color::cyan << "Number Of Edges Of Tree: " << numberOfEdges(traverseNode) << '\n';
                break;
            case 2:
                cout << Color::cyan << "Max Depth Of Tree: " << maxDepth(traverseNode) << '\n';
                break;
            default:
                cout << Color::red << "[-] The Option is out of bounds, Please try again\n";
                break;
            }
            break;
        default:
            cout << Color::red << "[-] The Option is out of bounds, Please try again\n";
            break;
        }
        Color::reset;
        dummyFunction();
        goto loop;
    }
```

Main File

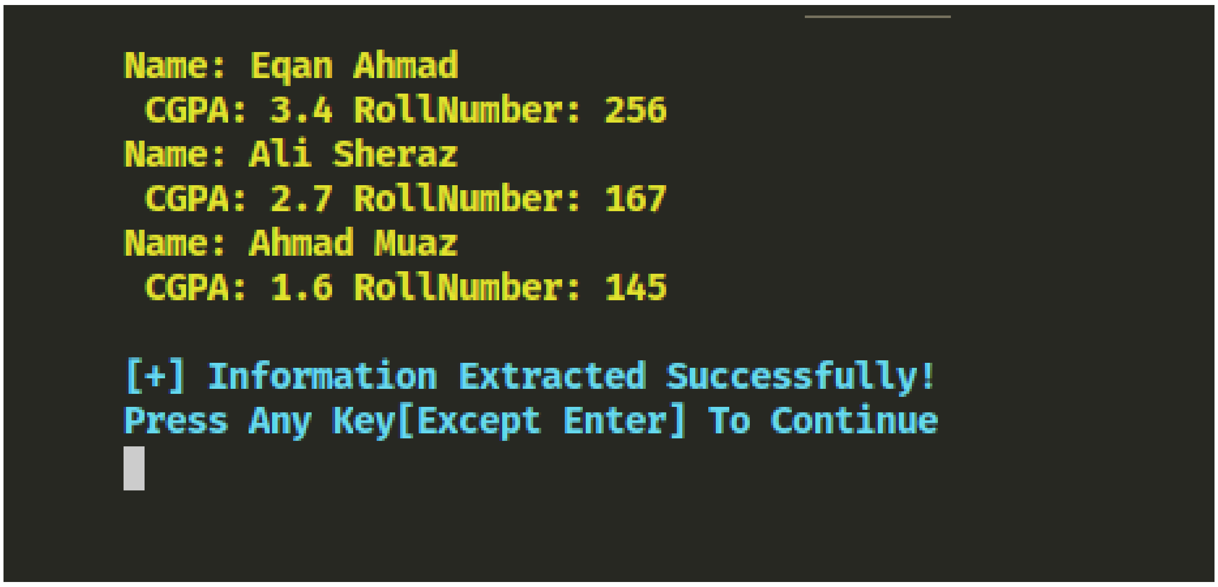
```
#include "DSCourse3-Q1v2.cpp"
int main()
{
    BST obj;
    obj.mainMenu();
}
```

Note This program is specifically made for Unix/Linux command line, As a linux User i wanted to integrate shell like commands into my program and my drive was enough to combine some fresh colorschemes with Bash Like commands into this program.



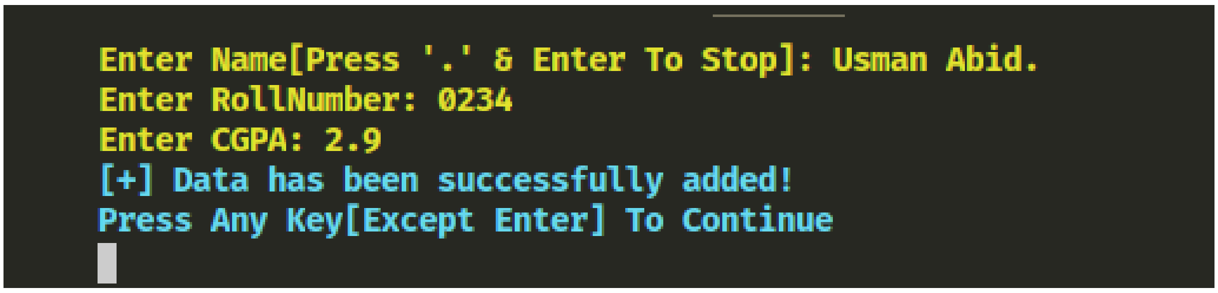
```
                        Welcome to the C++ Shell
Just before we initialize the program, there are some commands, you should go through
1. clear --Clears Screen
2. history --Displays Commands History
3. times --Display Times For A Command
4. time --Display Total Time
5. exit --End A Program
Press Any Key[Except Enter] To Continue
```

Figure 1: ScreenShot



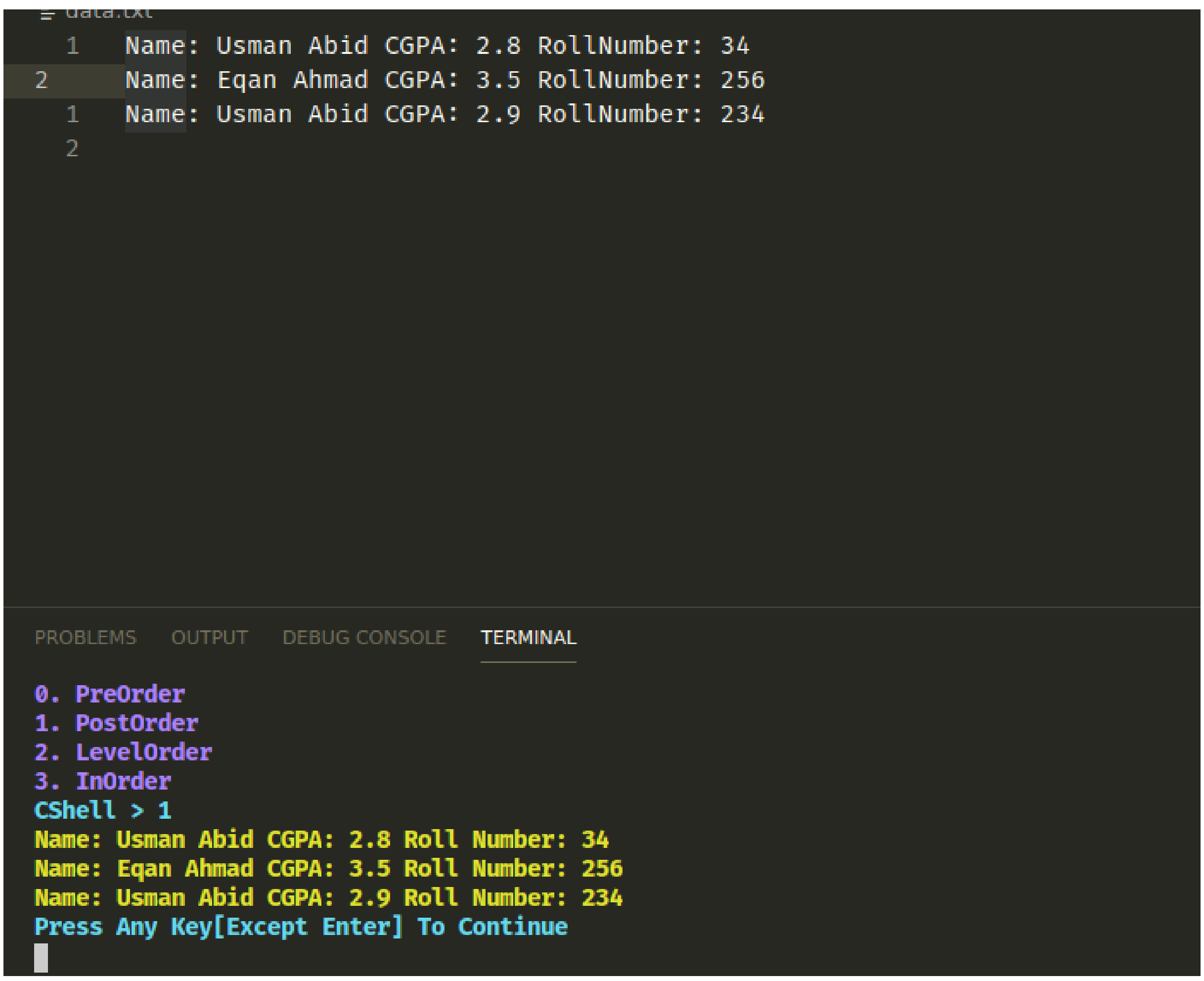
```
Name: Eqan Ahmad
 CGPA: 3.4 RollNumber: 256
Name: Ali Sheraz
 CGPA: 2.7 RollNumber: 167
Name: Ahmad Muaz
 CGPA: 1.6 RollNumber: 145

[+] Information Extracted Successfully!
Press Any Key[Except Enter] To Continue
```

Figure 2: ScreenShot



```
Enter Name[Press '.' & Enter To Stop]: Usman Abid.
Enter RollNumber: 0234
Enter CGPA: 2.9
[+] Data has been successfully added!
Press Any Key[Except Enter] To Continue
```

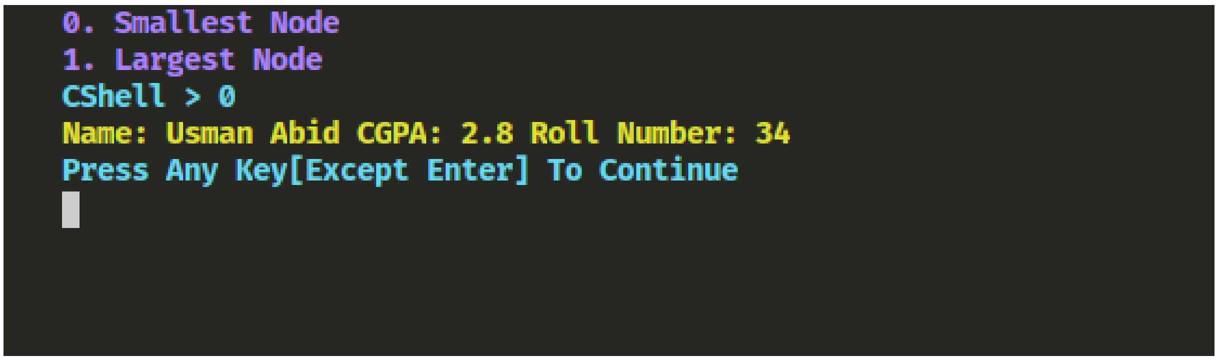
Figure 3: ScreenShot



```
Name: Usman Abid CGPA: 2.8 RollNumber: 34
Name: Eqan Ahmad CGPA: 3.5 RollNumber: 256
Name: Usman Abid CGPA: 2.9 RollNumber: 234
```

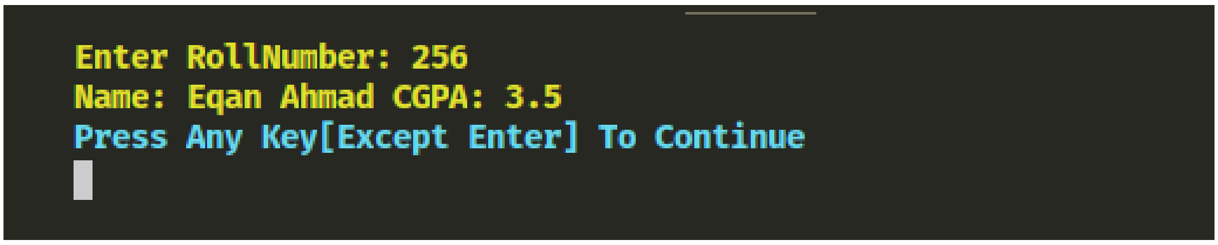
```
0. PreOrder
1. PostOrder
2. LevelOrder
3. InOrder
CShell > 1
Name: Usman Abid CGPA: 2.8 Roll Number: 34
Name: Eqan Ahmad CGPA: 3.5 Roll Number: 256
Name: Usman Abid CGPA: 2.9 Roll Number: 234
Press Any Key[Except Enter] To Continue
```

Figure 4: ScreenShot



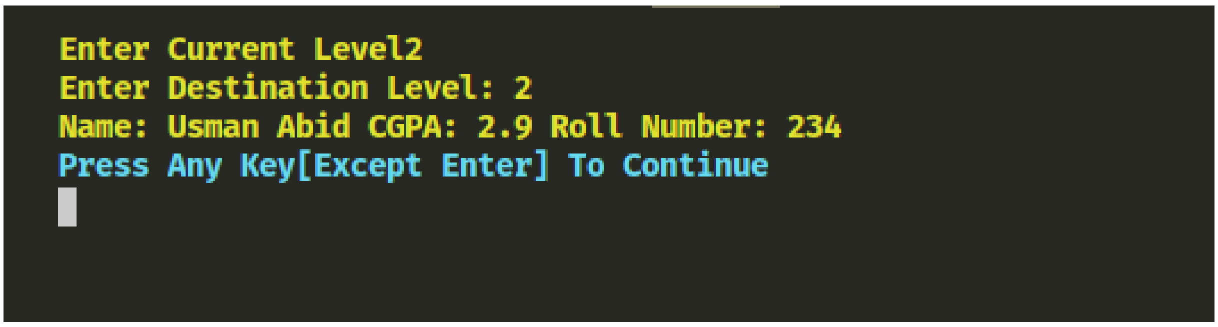
```
0. Smallest Node
1. Largest Node
CShell > 0
Name: Usman Abid CGPA: 2.8 Roll Number: 34
Press Any Key[Except Enter] To Continue
```

Figure 5: ScreenShot



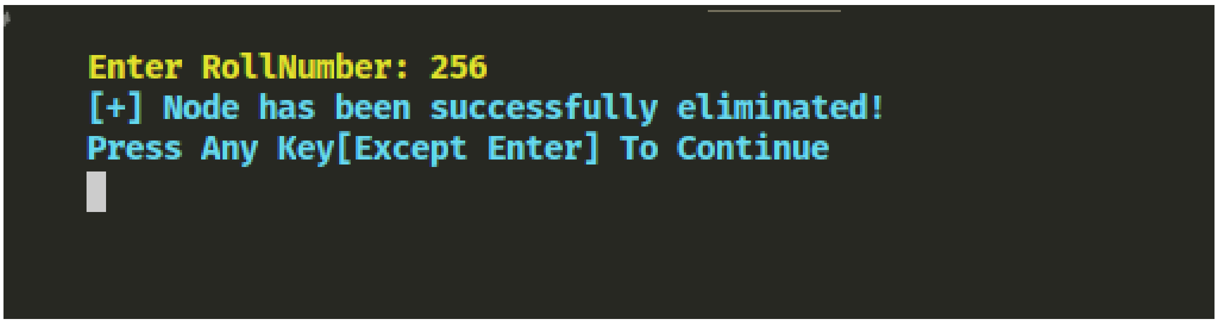
```
Enter RollNumber: 256
Name: Eqan Ahmad CGPA: 3.5
Press Any Key[Except Enter] To Continue
```

Figure 6: ScreenShot



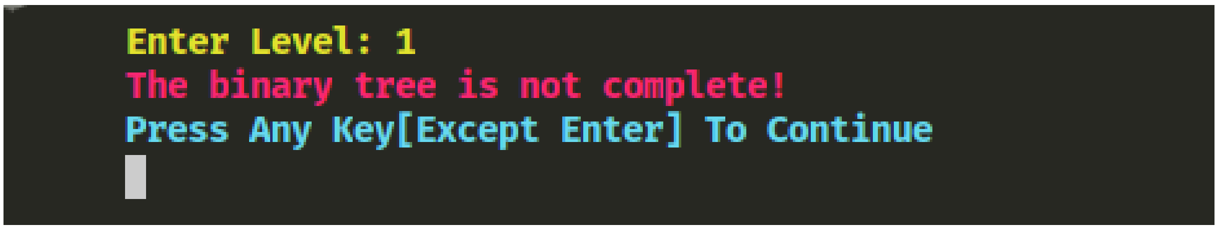
```
Enter Current Level2
Enter Destination Level: 2
Name: Usman Abid CGPA: 2.9 Roll Number: 234
Press Any Key[Except Enter] To Continue
```

Figure 7: ScreenShot



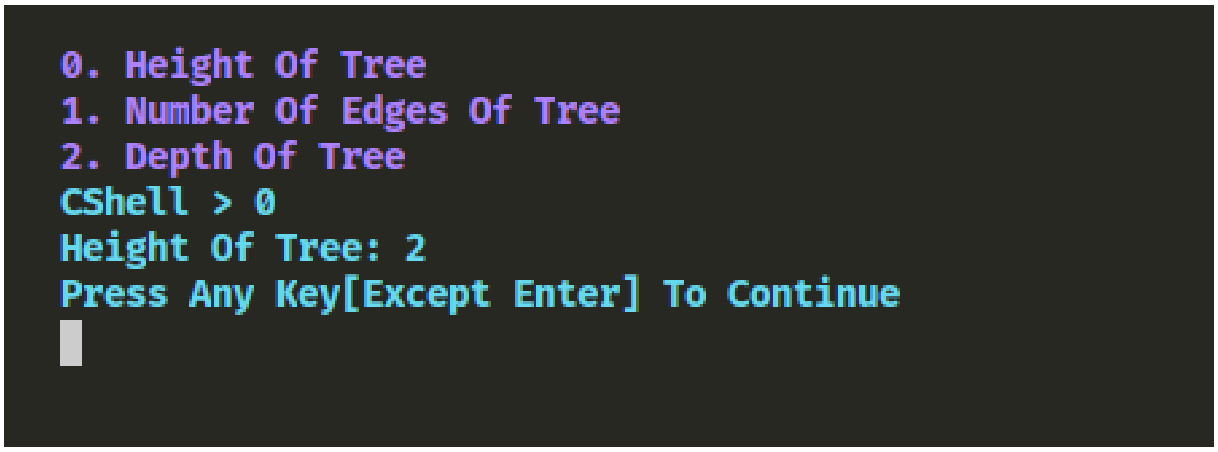
```
Enter RollNumber: 256
[+] Node has been successfully eliminated!
Press Any Key[Except Enter] To Continue
```

Figure 8: ScreenShot



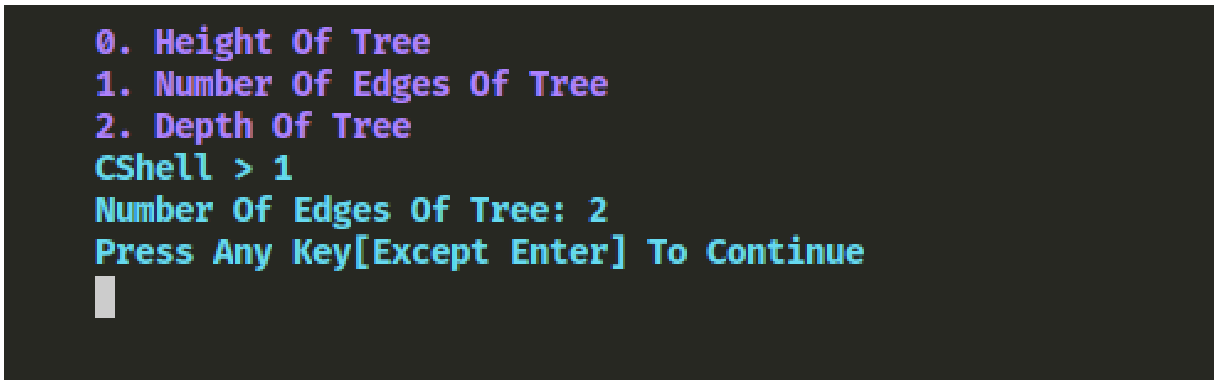
```
Enter Level: 1
The binary tree is not complete!
Press Any Key[Except Enter] To Continue
```

Figure 9: ScreenShot



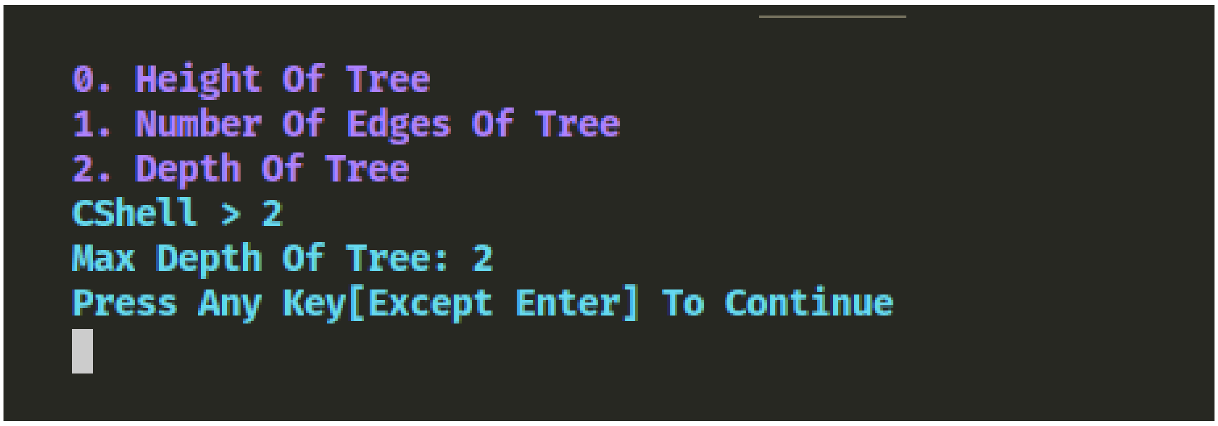
```
0. Height Of Tree
1. Number Of Edges Of Tree
2. Depth Of Tree
CShell > 0
Height Of Tree: 2
Press Any Key[Except Enter] To Continue
```

Figure 10: ScreenShot



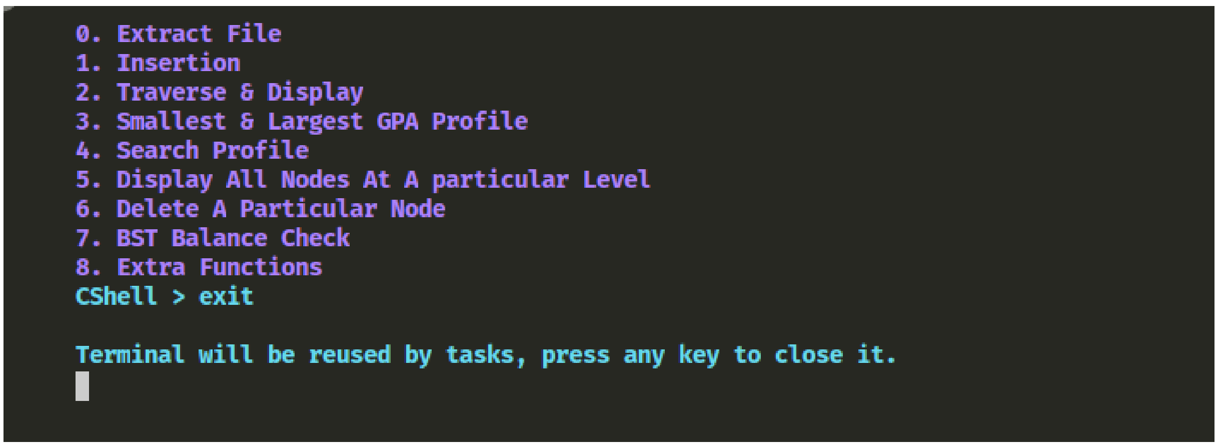
```
0. Height Of Tree
1. Number Of Edges Of Tree
2. Depth Of Tree
CShell > 1
Number Of Edges Of Tree: 2
Press Any Key[Except Enter] To Continue
```

Figure 11: ScreenShot



```
0. Height Of Tree
1. Number Of Edges Of Tree
2. Depth Of Tree
CShell > 2
Max Depth Of Tree: 2
Press Any Key[Except Enter] To Continue
```

Figure 12: ScreenShot



```
0. Extract File
1. Insertion
2. Traverse & Display
3. Smallest & Largest GPA Profile
4. Search Profile
5. Display All Nodes At A particular Level
6. Delete A Particular Node
7. BST Balance Check
8. Extra Functions
CShell > exit

Terminal will be reused by tasks, press any key to close it.
```

Figure 13: ScreenShot