

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

```
1 #include <ostream>
2 namespace Color {
3     const std::string red("\033[1;31m");
4     const std::string green("\033[1;32m");
5     const std::string blue("\033[1;34m");
6     const std::string yellow("\033[1;33m");
7     const std::string cyan("\033[1;36m");
8     const std::string magenta("\033[1;35m");
9     const std::string reset("\033[0m");
10 }
```

C++ Custom Shell

```
1 #include <stdlib.h>
2 #include <iostream>
3 #include "colormod.h"
4 class CShell
5 {
6 public:
7     int input()
8     {
9         std::string cmdin;
10        std::cin >> cmdin;
11        if (cmdin == "clear")
12        {
13            system("clear");
14            return 10;
15        }
16        else if (cmdin == "history")
17        {
18            system("history");
19            return 10;
20        }
21        else if (cmdin == "times")
22        {
23            system("times");
24            return 10;
25        }
26    }
27 }
```

```
26     else if (cmdin == "time")
27     {
28         system("time");
29         return 10;
30     }
31     else if (cmdin == "exit")
32     {
33         exit(0);
34         return 10;
35     }
36     else if (cmdin == "end")
37         return 0;
38     else if (cmdin >= "0" && cmdin <= "9")
39         return stoi(cmdin);
40     return 0;
41 }
42 };
```

Header File

```

1 #include <queue>
2 #include <string>
3 #include "C++Shell.h" // ? Custom Made Shell Implementation specifically
   made for this Program
4 using namespace std; // ? Adding this at the header files as we are
   following just the standard C++ implementation
5 // NOTE The binary tree is being aligned on the
   basis of the CGPA of a person
6
7 struct node
8 {
9     float cgpa;
10    int rollNumber;
11    string name;
12    node *left = NULL;
13    node *right = NULL;
14 };
15 node *rootNode = NULL;
16 node *createNode(const float &cgpa, const int &rollNumber, const std::
   string &name)
17 {
18     node *tempNode;
19     tempNode = new node;
20     tempNode->cgpa = cgpa;
21     tempNode->rollNumber = rollNumber;
22     tempNode->name = name;
23     return tempNode;
24 }
25 class BST
26 {
27 public:
28     void mainMenu();
29     static int countNode;
30
31 private:
32     CShell obj;
33     node *inputNode(node *traverseNode, const float &cgpa, const int &
   rollNumber, const string &name);
34     string dataTypes(const int &index);
35     void readFile();
36     void writeFile();
37     void dataExtraction(string &file);
38     node *largestNode(node *index);
39     node *smallestNode(node *index);
40     node *deleteNode(node *traverseNode, const int &rollNumber);
41     int height(node *traverseNode);
42     inline int numberOfEdges(node *tempNode);
43     int maxDepth(node *rootNode);
44     void levelorderTraversal(node *rootNode);
45     void inorderTraversal(node *traverseNode, ofstream &outFile) const;
46     void preorderTraversal(node *traverseNode, ofstream &outFile) const;
47     void postorderTraversal(node *traverseNode, ofstream &outFile) const
   ;
48     bool searchParticularProfile(node *traverseNode, const int &
   rollNumber);
49     void printNodesAtLevel(node *rootNode, const int &currentLevel,

```

```
50     const int &destLevel);  
51     bool isComplete(node *rootNode, const int &index);  
52     int numberOfNodes(node *traverseNode);  
53     inline void dummyFunction(const int &option);  
54     inline void dummyFunction();  
55 };  
56 int BST::countNode = 0;
```

CPP File

```

1 #include "DSCourse3-Q1.h"
2 #include <fstream>
3
4 node *BST::inputNode(node *traverseNode, const float &cgpa, const int &
   rollNumber, const string &name)
5 {
6     if (traverseNode == NULL)
7     {
8         countNode++;
9         cout << Color::cyan << "[+] Data has been successfully added!\n"
10        ;
11         traverseNode = createNode(cgpa, rollNumber, name);
12         return traverseNode;
13     }
14     else if (cgpa <= traverseNode->cgpa)
15         traverseNode->left = inputNode(traverseNode->left, cgpa,
16         rollNumber, name);
17     else
18         traverseNode->right = inputNode(traverseNode->right, cgpa,
19         rollNumber, name);
20     return traverseNode;
21 }
22 node *BST::largestNode(node *index)
23 {
24     while (index->right != NULL)
25         index = index->right;
26     return index;
27 }
28 node *BST::smallestNode(node *index)
29 {
30     while (index->left != NULL)
31         index = index->left;
32     return index;
33 }
34 node *BST::deleteNode(node *traverseNode, const int &rollNumber)
35 {
36     if (traverseNode == NULL)
37     {
38         cout << "Node not found!\n";
39         return traverseNode;
40     }
41     else if (rollNumber < traverseNode->rollNumber)
42         traverseNode->left = deleteNode(traverseNode->left, rollNumber);
43     else if (rollNumber > traverseNode->rollNumber)
44         traverseNode->right = deleteNode(traverseNode->right, rollNumber
45 );
46     else if (rollNumber == traverseNode->rollNumber)
47     {
48         if (traverseNode->left == NULL && traverseNode->right == NULL)
49             // ? Already is a Leaf Node
50             free(traverseNode);
51         else if (traverseNode->left == NULL) // ? Right Child is present
52         {
53             node *temp = traverseNode;
54             traverseNode = traverseNode->right;
55             free(temp);
56         }
57         else if (traverseNode->right == NULL) // ? Left Child is present
58         {
59             node *temp = traverseNode;
60             traverseNode = traverseNode->left;
61             free(temp);
62         }
63     }
64     return traverseNode;
65 }

```

```

51     }
52     else if (traverseNode->right == NULL) // ? Left child is present
53     {
54         node *temp = traverseNode;
55         traverseNode = traverseNode->left;
56         free(temp);
57     }
58     else // ? Both left and Right childs are present
59     {
60         node *temp = smallestNode(traverseNode->right);
61         traverseNode->rollNumber = temp->rollNumber;
62         // ? Not deleting the node just replacing its cgpa, as
        its the parent node with unpredicted childs
63         traverseNode->right = deleteNode(traverseNode->right, temp->
        rollNumber); // ? Now sending the original cgpa for deletion
64     }
65     countNode--;
66     cout << Color::cyan << "[+] Node has been successfully
        eliminated!\n";
67 }
68 return traverseNode;
69 }
70 int BST::height(node *traverseNode)
71 {
72     if (traverseNode == NULL)
73         return 0;
74     int leftHeight = height(traverseNode->left);
75     int rightHeight = height(traverseNode->right);
76     int currentHeight = max(leftHeight, rightHeight) + 1;
77     return currentHeight;
78 }
79 inline int BST::numberOfEdges(node *tempNode)
80 {
81     return tempNode == NULL ? 0 : numberOfNodes(tempNode) - 1;
82 }
83 int BST::maxDepth(node *rootNode)
84 {
85     if (rootNode == NULL)
86         return 0;
87     int leftDepth = maxDepth(rootNode->left);
88     int rightDepth = maxDepth(rootNode->right);
89     return (leftDepth > rightDepth) ? ++leftDepth : ++rightDepth;
90 }
91 void BST::levelorderTraversal(node *rootNode)
92 {
93     ofstream outFile("data.txt");
94     if (rootNode == NULL)
95         return;
96     queue<node *> traverseNode; // ? To temporarily store nodes and
        print later
97     traverseNode.push(rootNode);
98     while (!traverseNode.empty())
99     {
100         node *currentNode = traverseNode.front();
101         cout << Color::yellow << "Name: " << currentNode->name << ' ' <<
            "CGPA: " << currentNode->cgpa << ' ' << "Roll Number: " <<
            currentNode->rollNumber << '\n';

```

```

102     outFile << "Name: " << currentNode->name << '\n'
103           << "CGPA: " << currentNode->cgpa << '\n'
104           << "RollNumber: " << currentNode->rollNumber << '\n';
105     if (currentNode->left != NULL)
106         traverseNode.push(currentNode->left);
107     if (currentNode->right != NULL)
108         traverseNode.push(currentNode->right);
109     traverseNode.pop();
110 }
111 outFile.close();
112 }
113 void BST::inorderTraversal(node *traverseNode, ofstream &outFile) const
114 {
115     if (traverseNode == NULL)
116         return;
117     inorderTraversal(traverseNode->left, outFile);
118     cout << Color::yellow << "Name: " << traverseNode->name << ' ' << "
CGPA: " << traverseNode->cgpa << ' ' << "Roll Number: " <<
traverseNode->rollNumber << '\n';
119     outFile << "Name: " << traverseNode->name << '\n'
120           << "CGPA: " << traverseNode->cgpa << '\n'
121           << "RollNumber: " << traverseNode->rollNumber << '\n';
122     inorderTraversal(traverseNode->right, outFile);
123 }
124 void BST::preorderTraversal(node *traverseNode, ofstream &outFile) const
125 {
126     if (traverseNode == NULL)
127         return;
128     outFile << "Name: " << traverseNode->name << '\n'
129           << "CGPA: " << traverseNode->cgpa << '\n'
130           << "RollNumber: " << traverseNode->rollNumber << '\n';
131     cout << Color::yellow << "Name: " << traverseNode->name << ' ' << "
CGPA: " << traverseNode->cgpa << ' ' << "Roll Number: " <<
traverseNode->rollNumber << '\n';
132     preorderTraversal(traverseNode->left, outFile);
133     preorderTraversal(traverseNode->right, outFile);
134 }
135 void BST::postorderTraversal(node *traverseNode, ofstream &outFile)
const
136 {
137     if (traverseNode == NULL)
138         return;
139     postorderTraversal(traverseNode->left, outFile);
140     postorderTraversal(traverseNode->right, outFile);
141     outFile << "Name: " << traverseNode->name << '\n'
142           << "CGPA: " << traverseNode->cgpa << '\n'
143           << "RollNumber: " << traverseNode->rollNumber << '\n';
144     cout << Color::yellow << "Name: " << traverseNode->name << ' ' << "
CGPA: " << traverseNode->cgpa << ' ' << "Roll Number: " <<
traverseNode->rollNumber << '\n';
145 }
146 bool BST::searchParticularProfile(node *traverseNode, const int &
rollNumber)
147 {
148     if (traverseNode == NULL)
149         return false;
150
151     if (traverseNode->rollNumber == rollNumber)

```

```

152     {
153         cout << Color::yellow << "Name: " << traverseNode->name << ' '
154         << "CGPA: " << traverseNode->cgpa << '\n';
155         return true;
156     }
157     bool leftResult = searchParticularProfile(traverseNode->left,
158     rollNumber);
159     if (leftResult)
160         return true;
161     bool rightResult = searchParticularProfile(traverseNode->right,
162     rollNumber);
163     return rightResult;
164 }
165 void BST::printNodesAtLevel(node *rootNode, const int &currentLevel,
166 const int &destLevel)
167 {
168     if (rootNode == NULL)
169         return;
170     if (currentLevel == destLevel)
171     {
172         cout << Color::yellow << "Name: " << rootNode->name << ' ' << "
173         CGPA: " << rootNode->cgpa << ' ' << "Roll Number: " << rootNode->
174         rollNumber << '\n';
175         return;
176     }
177     printNodesAtLevel(rootNode->left, currentLevel + 1, destLevel);
178     printNodesAtLevel(rootNode->right, currentLevel + 1, destLevel);
179 }
180 bool BST::isComplete(node *rootNode, const int &index)
181 {
182     int numberOfNode = numberOfNodes(rootNode);
183     if (rootNode == NULL)
184         return (true);
185     if (index >= numberOfNode)
186         return (false);
187     return (isComplete(rootNode->left, 2 * index + 1) && isComplete(
188     rootNode->right, 2 * index + 2));
189 }
190 int BST::numberOfNodes(node *traverseNode)
191 {
192     if (traverseNode == NULL)
193         return 0;
194     return 1 + numberOfNodes(traverseNode->left) + numberOfNodes(
195     traverseNode->right);
196 }
197 void BST::dummyFunction(const int &option)
198 {
199     string dummy;
200     system("clear");
201     cout << Color::cyan << "Press Any Key[Except Enter] To Continue\n";
202     cin >> dummy;
203 }
204 void BST::dummyFunction()
205 {
206     string dummy;
207     cout << Color::cyan << "Press Any Key[Except Enter] To Continue\n";
208     cin >> dummy;
209     system("clear");

```



```
202 }
203 string BST::dataTypes(const int &index)
204 {
205     switch (index)
206     {
207     case 0:
208         return "Name: ";
209         break;
210     case 1:
211         return "CGPA: ";
212         break;
213     case 2:
214         return "RollNumber: ";
215         break;
216     }
217     return "";
218 }
219
220 void BST::dataExtraction(string &file)
221 {
222     string tempString, resultantString, name;
223     int rollNumber, j = 0;
224     float cgpa;
225     for (int i = 0; i < file.length(); i++)
226     {
227         if (tempString == dataTypes(j))
228         {
229             resultantString += file[i];
230             if (file[i] == '\n')
231             {
232                 if (j == 0)
233                     name = resultantString;
234                 else if (j == 1)
235                     cgpa = stof(resultantString);
236                 else if (j == 2)
237                 {
238                     rollNumber = stoi(resultantString);
239                     rootNode = inputNode(rootNode, cgpa, rollNumber,
240 name);
241                     j = -1;
242                 }
243                 j++;
244                 tempString = "", resultantString = "";
245             }
246             else if (file[i] == '\n')
247                 tempString = "";
248             else
249                 tempString += file[i];
250         }
251     }
252 void BST::readFile()
253 {
254     string tempData, file;
255     ifstream inputFile("data.txt");
256     int i = 0;
257     if (inputFile.is_open())
258     {
```

```

259     while (getline(inputFile, tempData))
260         file += tempData + "\n";
261     inputFile.close();
262 }
263 else
264     cout << "Unable to open file";
265 cout << Color::yellow << file << '\n';
266 cout << Color::cyan << "[+] Information Extracted Successfully!\n";
267 dataExtraction(file);
268 }
269 void BST::writeFile()
270 {
271     if (rootNode == NULL)
272         readFile();
273     if (rootNode == NULL)
274         return;
275     cout << Color::magenta << "0. PreOrder\n1. PostOrder\n2. LevelOrder\n3. InOrder\n";
276     cout << Color::cyan << "CShell > ";
277     int option = obj.input();
278     node *traverseNode = rootNode;
279     switch (option)
280     {
281     case 0:
282     {
283         ofstream outFile("data.txt");
284         preorderTraversal(traverseNode, outFile);
285         outFile.close();
286         break;
287     }
288     case 1:
289     {
290         ofstream outFile("data.txt");
291         postorderTraversal(traverseNode, outFile);
292         outFile.close();
293         break;
294     }
295     case 2:
296         levelorderTraversal(traverseNode);
297         break;
298     case 3:
299     {
300         ofstream outFile("data.txt");
301         inorderTraversal(traverseNode, outFile);
302         outFile.close();
303         break;
304     }
305     default:
306         cout << Color::red << "[-] The Option is out of bounds, Please try again\n";
307         break;
308     }
309 }
310
311 void BST::mainMenu()
312 {
313     int rollNumber, option;
314     float cgpa;

```

```

315     string name;
316     node *traverseNode;
317     system("clear");
318     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";
319     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";
320     dummyFunction();
321 loop:
322     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";
323     cout << Color::cyan << "CShell > ";
324     option = obj.input();
325     if (option == 10)
326     {
327         dummyFunction(option);
328         goto loop;
329     }
330     switch (option)
331     {
332     case 0:
333         system("clear");
334         readFile();
335         break;
336     case 1:
337     {
338         system("clear");
339         cout << Color::yellow << "Enter Name[Press '.' & Enter To Stop]:
";
340         cin.ignore();
341         getline(cin, name, '.');
342         cout << "Enter RollNumber: ";
343         cin >> rollNumber;
344         cout << "Enter CGPA: ";
345         cin >> cgpa;
346         rootNode = inputNode(rootNode, cgpa, rollNumber, name);
347         break;
348     }
349     case 2:
350         system("clear");
351         writeFile();
352         break;
353     case 3:
354     {
355         system("clear");
356         cout << Color::magenta << "0. Smallest Node\n1. Largest Node\n";
357         cout << Color::cyan << "CShell > ";
358         int option = obj.input();
359         traverseNode = rootNode;
360         switch (option)
361         {
362         case 0:
363             traverseNode = smallestNode(traverseNode);
364             cout << Color::yellow << "Name: " << traverseNode->name << '

```

```

    ' << "CGPA: " << traverseNode->cgpa << ' ' << "Roll Number: " <<
    traverseNode->rollNumber << '\n';
365     break;
366     case 1:
367         traverseNode = largestNode(traverseNode);
368         cout << Color::yellow << "Name: " << traverseNode->name << '
    ' << "CGPA: " << traverseNode->cgpa << ' ' << "Roll Number: " <<
    traverseNode->rollNumber << '\n';
369         break;
370     default:
371         cout << Color::red << "[-] The Option is out of bounds,
    Please try again\n";
372         break;
373     }
374 }
375 break;
376 case 4:
377     system("clear");
378     traverseNode = rootNode;
379     cout << Color::yellow << "Enter RollNumber: ";
380     cin >> rollNumber;
381     if (!searchParticularProfile(traverseNode, rollNumber))
382         cout << Color::red << "[-] Profile Not Found\n";
383     break;
384 case 5:
385     system("clear");
386     int currentLevel, destLevel;
387     traverseNode = rootNode;
388     cout << Color::yellow << "Enter Current Level";
389     cin >> currentLevel;
390     cout << "Enter Destination Level: ";
391     cin >> destLevel;
392     printNodesAtLevel(traverseNode, currentLevel, destLevel);
393     break;
394 case 6:
395     system("clear");
396     cout << Color::yellow << "Enter RollNumber: ";
397     cin >> rollNumber;
398     traverseNode = rootNode;
399     deleteNode(traverseNode, rollNumber);
400     break;
401 case 7:
402     system("clear");
403     traverseNode = rootNode;
404     cout << Color::yellow << "Enter Level: ";
405     cin >> option;
406     (isComplete(traverseNode, option)) ? (cout << Color::green << "
    The binary tree is complete!\n") : (cout << Color::red << "The binary
    tree is not complete!\n");
407     break;
408 case 8:
409     system("clear");
410     cout << Color::magenta << "0. Height Of Tree\n1. Number Of Edges
    Of Tree\n2. Depth Of Tree\n";
411     cout << Color::cyan << "CShell > ";
412     option = obj.input();
413     traverseNode = rootNode;
414     switch (option)

```

```
415     {
416         case 0:
417             cout << Color::cyan << "Height Of Tree: " << height(
traverseNode) << '\n';
418             break;
419         case 1:
420             cout << Color::cyan << "Number Of Edges Of Tree: " <<
numberOfEdges(traverseNode) << '\n';
421             break;
422         case 2:
423             cout << Color::cyan << "Max Depth Of Tree: " << maxDepth(
traverseNode) << '\n';
424             break;
425         default:
426             cout << Color::red << "[-] The Option is out of bounds,
Please try again\n";
427             break;
428     }
429     break;
430     default:
431         cout << Color::red << "[-] The Option is out of bounds, Please
try again\n";
432         break;
433     }
434     Color::reset;
435     dummyFunction();
436     goto loop;
437 }
```

Main File

```
1 #include "DSCourse3-Q1v2.cpp"
2 int main()
3 {
4     BST obj;
5     obj.mainMenu();
6 }
```

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.

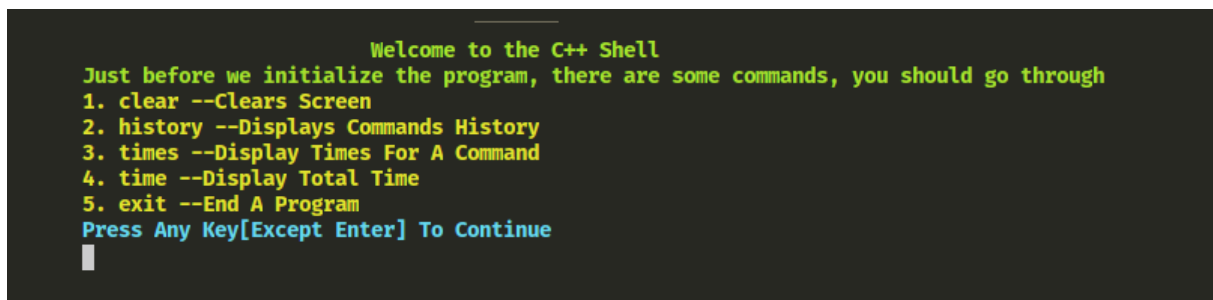


Figure 1: ScreenShot

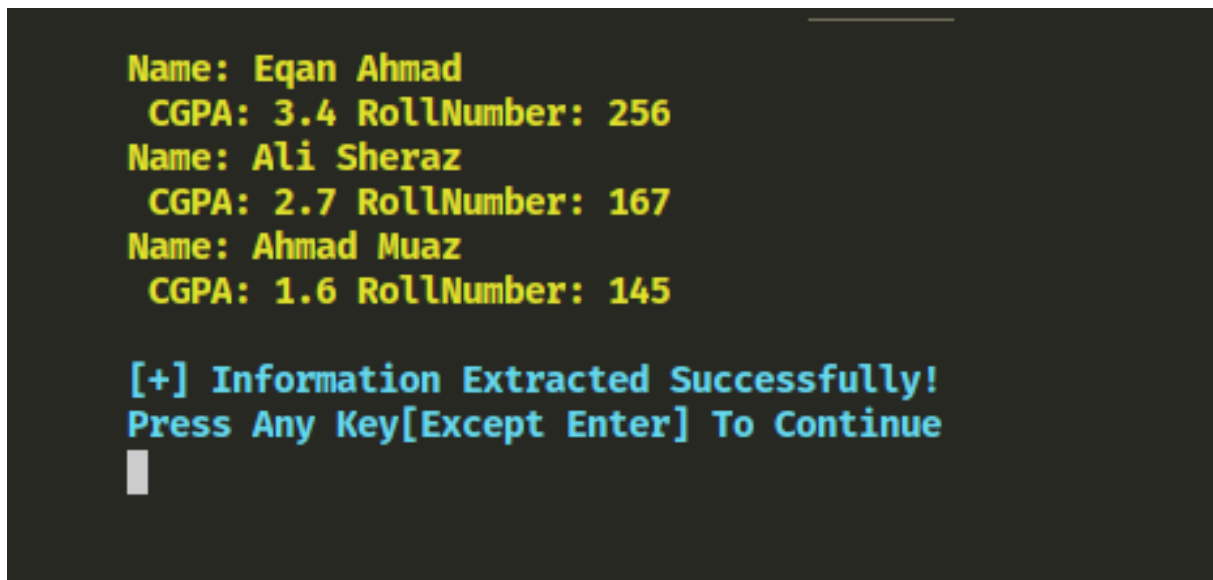


Figure 2: ScreenShot

```
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

```
= data.txt  
1 Name: Usman Abid CGPA: 2.8 RollNumber: 34  
2 Name: Eqan Ahmad CGPA: 3.5 RollNumber: 256  
1 Name: Usman Abid CGPA: 2.9 RollNumber: 234  
2  
  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL  
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

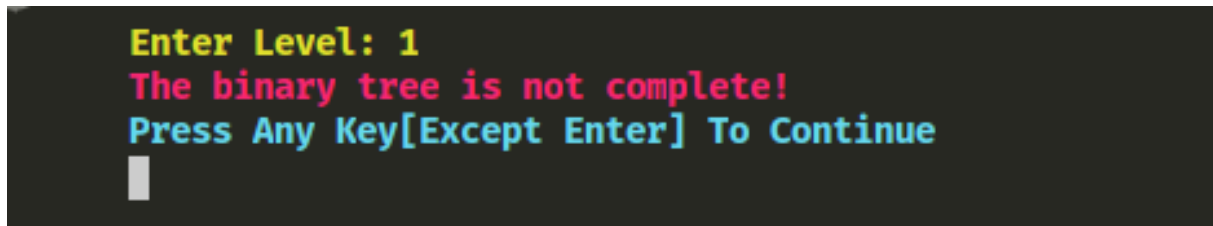


Figure 9: ScreenShot

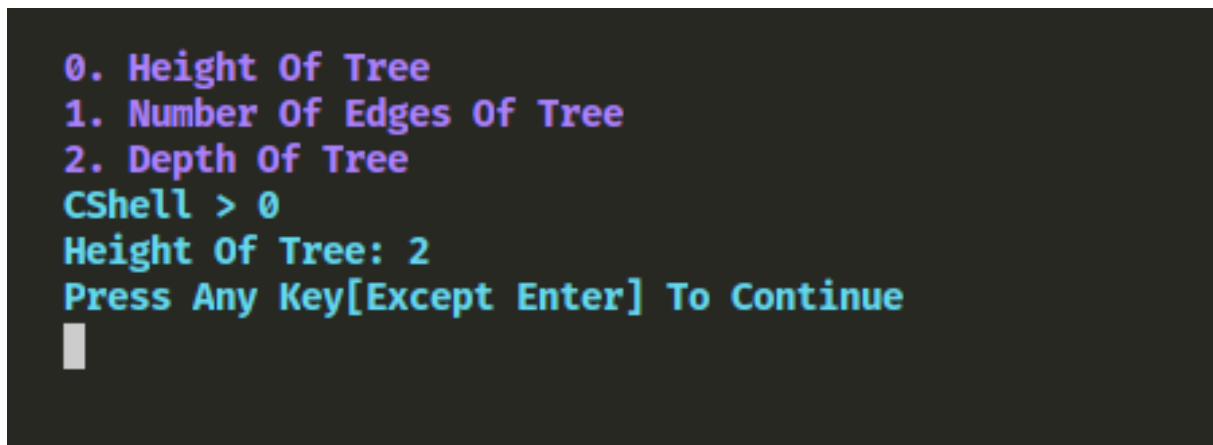


Figure 10: ScreenShot

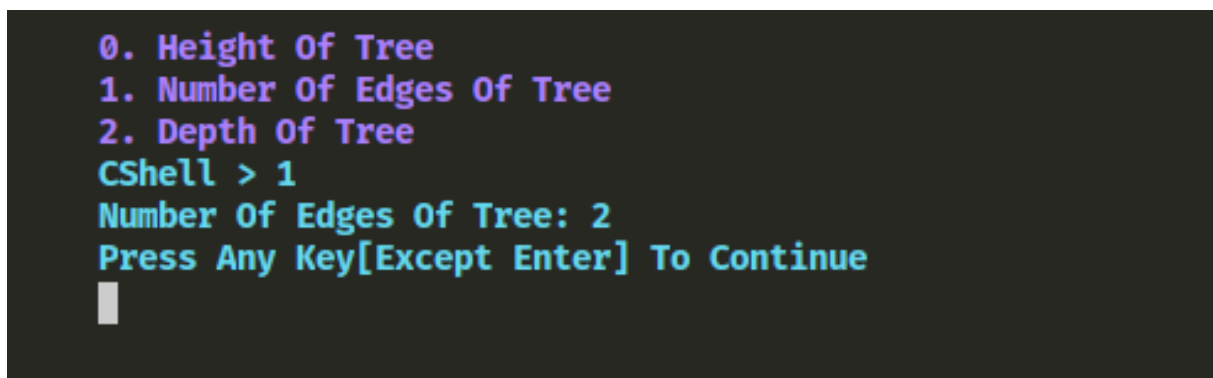


Figure 11: ScreenShot

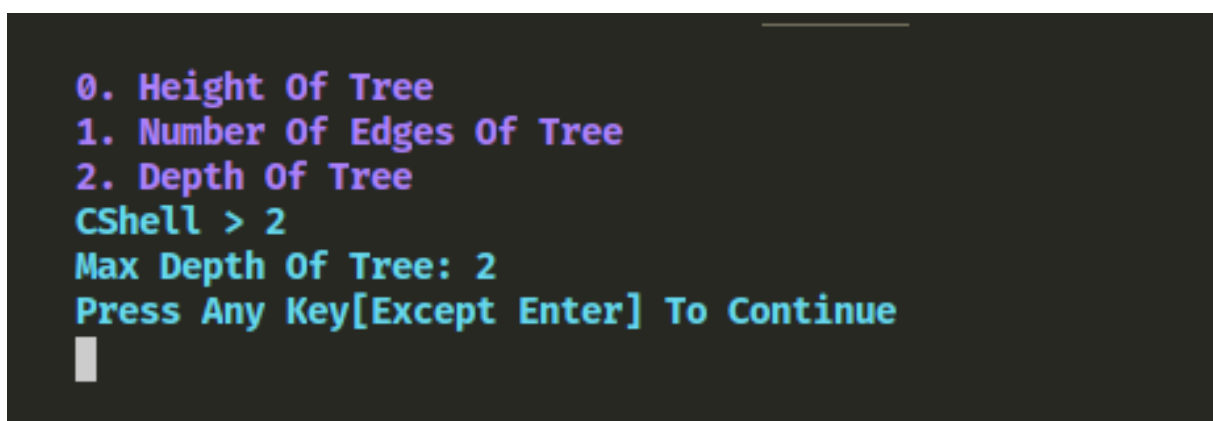
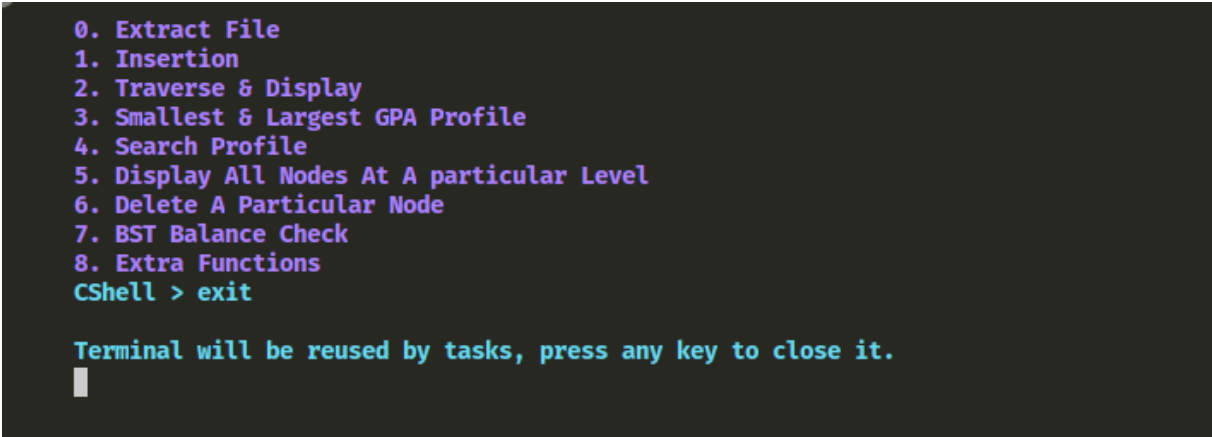


Figure 12: ScreenShot



```
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