question1.cpp

```cpp
# include <iostream>
# include <cstdlib>
using namespace std;
struct nod {
    int info;
    struct nod *l;
    struct nod *r;
}*r;
class BST {
    public://functions declaration
        void search(nod *, int);
        void find(int, nod **, nod **);
        void insert(nod *, nod *);
        void del(int);
        void casea(nod *,nod *);
        void caseb(nod *,nod *);
        void casec(nod *,nod *);
        void show(nod *, int);
        BST() {
            r = NULL;
        }
};
void BST::find(int i, nod **par, nod **loc) {           //function called to find an int i from BST, given 2 node pointer pointers par (parent of i) and loc (location of i) to be assigned
    nod *ptr, *ptrsave;
    if (r == NULL) {
        *loc = NULL;            //sets parent and location to NULL if BST's root (r) is NULL and returns.
        *par = NULL;
        return;
    }
    if (i == r->info) {         //if root contains i, then set loc as root, and it has no parent hence par is NULL
        *loc = r;
        *par = NULL;
        return;
    }
    if (i < r->info)
        ptr = r->l;             //searches i by traversing tree according to comparison, choose first subtree left or right
    else
        ptr = r->r;
    ptrsave = r;
    while (ptr != NULL) {       //loops until end of BST reached
        if (i == ptr->info) {   //if i found at position (ptr) then parent is stored in par and loc is stored as position (ptr)
            *loc = ptr;
            *par = ptrsave;
            return;
        }
        ptrsave = ptr;          //else position(ptr) is assigned to other variable for next iteration
        if (i < ptr->info)      //choose left/right child by comparison
            ptr = ptr->l;
        else
```

```
            ptr = ptr->r;
    }
    *loc = NULL;                //if end of BST reached without finding i the loc set to NULL and par set tp last node visited
    *par = ptrsave;
}
//FIND finds the location of an Integer i and stores its location in a node* value called loc, and also gives its parent's location in node* called par.



void BST::search(nod*root,int data) {       //search function called to find integer data from a specified root in parameter
    int depth = 0;              //depth variable to count how 'deep' a value is from root
    nod *temp = new nod;        //temp root value for traversing
    temp = root;                //initialize temp from root for traversal
    while(temp != NULL) {
        depth++;                //every layer checked is shown as increment in depth
        if(temp->info == data) {    //if data at current layer is wwhat we need to find then output the depth or layer from root it was found
            cout<<"\nData found at depth: "<<depth<<endl;
            return;
        } else if(temp->info > data)    //otherwise continue traversal based on comparisons
            temp = temp->l;
        else
            temp = temp->r;
    }
    cout<<"\n Data not found"<<endl;    //if bst traversed without finding then output data not found/
    return;
}
//search basically find the layers in which data to be found exists, and outputs it upon encountering it.

//Find gives the nlocation of the value to be found and its parent's location
//Search gives the layer of a value to be searched from a starting root node.

void BST::insert(nod *tree, nod *newnode) {
    if (r == NULL) {
        r = new nod;
        r->info = newnode->info;
        r->l= NULL;
        r->r= NULL;
        cout<<"Root Node is Added"<<endl;
        return;
    }
    if (tree->info == newnode->info) {
        cout<<"Element already in the tree"<<endl;
        return;
    }
    if (tree->info > newnode->info) {
        if (tree->l != NULL)
            insert(tree->l, newnode);
        else {
            tree->l= newnode;
            (tree->l)->l = NULL;
            (tree->l)->r= NULL;
```

```
            cout<<"Node Added To Left"<<endl;
            return;
        }
    } else {
        if (tree->r != NULL)
            insert(tree->r, newnode);
            else {
                tree->r = newnode;
                (tree->r)->l= NULL;
                (tree->r)->r = NULL;
                cout<<"Node Added To Right"<<endl;
                return;
            }
    }
}
void BST::del(int i) {
    nod *par, *loc;
    if (r == NULL) {
        cout<<"Tree empty"<<endl;
        return;
    }
    find(i, &par, &loc);
    if (loc == NULL) {
        cout<<"Item not present in tree"<<endl;
        return;
    }
    if (loc->l == NULL && loc->r == NULL) {
        casea(par, loc);
        cout<<"item deleted"<<endl;
    }
    if (loc->l!= NULL && loc->r == NULL) {
        caseb(par, loc);
        cout<<"item deleted"<<endl;
    }
    if (loc->l== NULL && loc->r != NULL) {
        caseb(par, loc);
        cout<<"item deleted"<<endl;
    }
    if (loc->l != NULL && loc->r != NULL) {
        casec(par, loc);
        cout<<"item deleted"<<endl;
    }
    free(loc);
}
void BST::casea(nod *par, nod *loc) {
    if (par == NULL) {
        r= NULL;
    } else {
        if (loc == par->l)
            par->l = NULL;
        else
            par->r = NULL;
    }
}
void BST::caseb(nod *par, nod *loc) {
    nod *child;
```

```
157     if (loc->l!= NULL)
158         child = loc->l;
159     else
160         child = loc->r;
161     if (par == NULL)
162         r = child;
163     else {
164         if (loc == par->l)
165             par->l = child;
166         else
167             par->r = child;
168     }
169 }
170 void BST::casec(nod *par, nod *loc) {
171     nod *ptr, *ptrsave, *suc, *parsuc;
172     ptrsave = loc;
173     ptr = loc->r;
174     while (ptr->l!= NULL) {
175         ptrsave = ptr;
176         ptr = ptr->l;
177     }
178     suc = ptr;
179     parsuc = ptrsave;
180     if (suc->l == NULL && suc->r == NULL)
181         casea(parsuc, suc);
182     else caseb(parsuc, suc);
183     if (par == NULL)
184         r = suc;
185     else {
186         if (loc == par->l)
187             par->l = suc;
188         else
189             par->r= suc;
190     }
191     suc->l = loc->l;
192     suc->r= loc->r;
193 }
194 //print the tree
195 void BST::show(nod *ptr, int level) {
196     int i;
197     if (ptr != NULL) {
198         show(ptr->r, level+1);
199         cout<<endl;
200         if (ptr == r)
201             cout<<"Root->: ";
202         else {
203             for (i = 0; i < level; i++)
204                 cout<<" ";
205         }
206         cout<<ptr->info;
207         show(ptr->l, level+1);
208     }
209 }
210
211 int main() {
212     int c, n,item;
```

```
    BST bst;
    nod *t;
    while (1) {
        cout<<"1.Insert Element "<<endl;
        cout<<"2.Delete Element "<<endl;
        cout<<"3.Search Element"<<endl;
        cout<<"4.Display the tree"<<endl;
        cout<<"5.Quit"<<endl;
        cout<<"Enter your choice : ";
        cin>>c;
        switch(c) {
            case 1:
                t = new nod;
                cout<<"Enter the number to be inserted: ";
                cin>>t->info;
                bst.insert(r, t);
                break;

            case 2:
                if (r == NULL) {
                    cout<<"Tree is empty, nothing to delete"<<endl;
                    continue;
                }
                cout<<"Enter the number to be deleted: ";
                cin>>n;
                bst.del(n);
                break;
            case 3:
                cout<<"Search:"<<endl;
                cin>>item;
                bst.search(r,item);
                break;
            case 4:
                cout<<"Display BST:"<<endl;
                bst.show(r,1);
                cout<<endl;
                break;
            case 5:
                exit(1);
            default:
                cout<<"Wrong choice!"<<endl;
        }
    }
}
```

question2.cpp

```cpp
# include <iostream>
# include <cstdlib>
using namespace std;
struct nod {
    int info;
    struct nod *l;
    struct nod *r;
}*r;
class BST {
    public://functions declaration
//      void search(nod *, int);
        void findNsearch(int, nod **, nod **);
        void insert(nod *, nod *);
        void del(int);
        void casea(nod *,nod *);
        void caseb(nod *,nod *);
        void casec(nod *,nod *);
        void show(nod *, int);
        BST() {
            r = NULL;
        }
};
void BST::findNsearch(int i, nod **par, nod **loc) {            //function called to find an int i from BST, given 2 node pointer pointers par (parent of i) and loc (location of i) to be assigned
                                                                //Search functionality merged into findNsearch, search now counts from root of entire tree instead of specified subtree

    int depth = 0;      //depth counter from search
    nod *ptr, *ptrsave;

    if (r == NULL) {
        *loc = NULL;            //sets parent and location to NULL if BST's root (r) is NULL and returns.
        *par = NULL;
        cout<<"\n Data not found because BST is empty"<<endl;
        return;
    }
    depth++;    //first layer
    if (i == r->info) {         //if root contains i, then set loc as root, and it has no parent hence par is NULL
        *loc = r;
        *par = NULL;
        cout<<"\nData found at depth: "<<depth<<endl;
        return;
    }
    if (i < r->info)
        ptr = r->l;             //searches i by traversing tree according to comparison, choose first subtree left or right
    else
        ptr = r->r;
    ptrsave = r;
    while (ptr != NULL) {       //loops until end of BST reached
        depth++;                //increment of depth per each layer traversed
        if (i == ptr->info) {   //if i found at position (ptr) then parent is stored in par and loc is stored as position (ptr)
```

```
50              *loc = ptr;
51              *par = ptrsave;
52              nod*t = *par;
53              cout<<"Location of "<<i<<" : "<<*loc<<endl;                  //siaplaying loc and par
to show find functionality works.
54              cout<<"Parent of "<<i<<" : "<<t->info<<" @ "<<*par<<endl;
55              cout<<"\nData found at depth: "<<depth<<endl;                //showing depth to show
search functionality works.
56              return;
57          }
58          ptrsave = ptr;          //else position(ptr) is assigned to other variable for next
iteration
59          if (i < ptr->info)      //choose left/right child by comparison
60              ptr = ptr->l;
61          else
62              ptr = ptr->r;
63      }
64      *loc = NULL;                //if end of BST reached without finding i the loc set to NULL and
par set tp last node visited
65      *par = ptrsave;
66      cout<<"\n Data not found, end of BST reached"<<endl;
67  }
68
69  void BST::insert(nod *tree, nod *newnode) {
70      if (r == NULL) {
71          r = new nod;
72          r->info = newnode->info;
73          r->l= NULL;
74          r->r= NULL;
75          cout<<"Root Node is Added"<<endl;
76          return;
77      }
78      if (tree->info == newnode->info) {
79          cout<<"Element already in the tree"<<endl;
80          return;
81      }
82      if (tree->info > newnode->info) {
83          if (tree->l != NULL)
84              insert(tree->l, newnode);
85          else {
86              tree->l= newnode;
87              (tree->l)->l = NULL;
88              (tree->l)->r= NULL;
89              cout<<"Node Added To Left"<<endl;
90              return;
91          }
92      } else {
93          if (tree->r != NULL)
94              insert(tree->r, newnode);
95              else {
96                  tree->r = newnode;
97                  (tree->r)->l= NULL;
98                  (tree->r)->r = NULL;
99                  cout<<"Node Added To Right"<<endl;
100                 return;
101             }
102     }
```

```
103 }
104 void BST::del(int i) {
105     nod *par, *loc;
106     if (r == NULL) {
107         cout<<"Tree empty"<<endl;
108         return;
109     }
110     findNsearch(i, &par, &loc);
111     if (loc == NULL) {
112         cout<<"Item not present in tree"<<endl;
113         return;
114     }
115     if (loc->l == NULL && loc->r == NULL) {
116         casea(par, loc);
117         cout<<"item deleted"<<endl;
118     }
119     if (loc->l!= NULL && loc->r == NULL) {
120         caseb(par, loc);
121         cout<<"item deleted"<<endl;
122     }
123     if (loc->l== NULL && loc->r != NULL) {
124         caseb(par, loc);
125         cout<<"item deleted"<<endl;
126     }
127     if (loc->l != NULL && loc->r != NULL) {
128         casec(par, loc);
129         cout<<"item deleted"<<endl;
130     }
131     free(loc);
132 }
133 void BST::casea(nod *par, nod *loc) {
134     if (par == NULL) {
135         r= NULL;
136     } else {
137         if (loc == par->l)
138             par->l = NULL;
139         else
140             par->r = NULL;
141     }
142 }
143 void BST::caseb(nod *par, nod *loc) {
144     nod *child;
145     if (loc->l!= NULL)
146         child = loc->l;
147     else
148         child = loc->r;
149     if (par == NULL)
150         r = child;
151     else {
152         if (loc == par->l)
153             par->l = child;
154         else
155             par->r = child;
156     }
157 }
158 void BST::casec(nod *par, nod *loc) {
```

```
159     nod *ptr, *ptrsave, *suc, *parsuc;
160     ptrsave = loc;
161     ptr = loc->r;
162     while (ptr->l!= NULL) {
163         ptrsave = ptr;
164         ptr = ptr->l;
165     }
166     suc = ptr;
167     parsuc = ptrsave;
168     if (suc->l == NULL && suc->r == NULL)
169         casea(parsuc, suc);
170     else caseb(parsuc, suc);
171     if (par == NULL)
172         r = suc;
173     else {
174         if (loc == par->l)
175             par->l = suc;
176         else
177             par->r= suc;
178     }
179     suc->l = loc->l;
180     suc->r= loc->r;
181 }
182 //print the tree
183 void BST::show(nod *ptr, int level) {
184     int i;
185     if (ptr != NULL) {
186         show(ptr->r, level+1);
187         cout<<endl;
188         if (ptr != r) {
189             for (i = 0; i < level; i++)
190                 cout<<" ";
191         }
192         cout<<ptr->info;
193         if (ptr==r)
194             cout<<"\t<-Root";
195         show(ptr->l, level+1);
196     }
197 }
198
199 int main() {
200     int c, n,item;
201     BST bst;
202     nod *t;
203     nod *a;
204     nod *b;
205     while (1) {
206         cout<<"1.Insert Element "<<endl;
207         cout<<"2.Delete Element "<<endl;
208         cout<<"3.Search Element"<<endl;
209         cout<<"4.Display the tree"<<endl;
210         cout<<"5.Quit"<<endl;
211         cout<<"Enter your choice : ";
212         cin>>c;
213         switch(c) {
214             case 1:
```

```
215                 t = new nod;
216                 cout<<"Enter the number to be inserted: ";
217                 cin>>t->info;
218                 bst.insert(r, t);
219                 break;
220 
221             case 2:
222                 if (r == NULL) {
223                     cout<<"Tree is empty, nothing to delete"<<endl;
224                     continue;
225                 }
226                 cout<<"Enter the number to be deleted: ";
227                 cin>>n;
228                 bst.del(n);
229                 break;
230             case 3:
231                 cout<<"Find and Search:"<<endl;
232                 cin>>item;
233                 bst.findNsearch(item,&a,&b);
234                 break;
235             case 4:
236                 cout<<"Display BST:"<<endl;
237                 bst.show(r,1);
238                 cout<<endl;
239                 break;
240             case 5:
241                 exit(1);
242             default:
243                 cout<<"Wrong choice!"<<endl;
244         }
245     }
246 }
```