Name: Khushal Patil Class: SE-II (R-Batch)

Roll No.: 64

Code:

```
#include <iostream>
#include <string>
using namespace std;

class node {
   public:
    string key, value;
    node *left, *right;
    node() {
        key = "";
        value = "";
        left = NULL;
        right = NULL;
    }
    node(string key, string value) {
        this->key = key;
        this->value = value;
        left = NULL;
        right = NULL;
    }
};

class bst {
   public:
    node *root;
    bst(){
         root = NULL;
    }
    bst(string key, string value) {
        root = new node(key, value);
    }

    bool insert(string, string);
    string search(string);
    bool update(string, string);
    bool delete_key(string);
    void display(node *cur);

};
```

```
bool bst::insert(string key, string value) {
    if (root == NULL) {
        root = new node(key, value);
        return 1;
    }
    node *temp, *prev;
    prev = root;
    temp = root;
    while (temp != NULL) {
        prev = temp;
        if (temp->key == key) {
            return 0;
        } else if (temp->key < key) {
            temp = temp->right;
        } else {
            temp = temp->left;
        }
    }
    if (prev->key < key) {
        prev->right = new node(key, value);
    } else {
        prev->left = new node(key, value);
    }
    return 1;
}

string bst::search(string key) {
    node *temp = root;
    while (temp != NULL) {
        if (temp->key == key) {
            return temp->value;
        } else if (temp->key < key) {
            temp = temp->right;
        } else {
            temp = temp->left;
        }
    }
    return "\0";
}

bool bst::update(string key, string value) {
    node *temp;
    temp = root;
    while (temp != NULL) {
```

```
        if (temp->key == key) {
            temp->value = value;
            return 1;
        } else if (temp->key < key) {
            temp = temp->right;
        } else {
            temp = temp->left;
        }
    }
    return 0;
}

bool bst::delete_key(string key) {
    if (root == NULL) {
        return 0;
    }

    node *temp, *prev;
    prev = root;
    temp = root;

    if (temp->key == key) {

        if (temp->left == NULL && temp->right == NULL) {
            root = NULL;
            delete temp;
        } else if (temp->left != NULL && temp->right == NULL) {
            root = temp->left;
            delete temp;
        } else if (temp->left == NULL && temp->right != NULL) {
            root = temp->right;
            delete temp;
        } else {
            node *l_temp = temp->left;
            node *l_prev = temp;
            if (l_temp->right == NULL) {
                l_prev->left = l_temp->left;
            } else {
                while (l_temp->right != NULL) {
                    l_prev = l_temp;
                    l_temp = l_temp->right;
                }
                l_prev->right = l_temp->left;
            }
```

```
        l_temp->right = temp->right;
        l_temp->left = temp->left;
        root = l_temp;
        delete temp;
    }
    return 1;
} else if (temp->key < key) {
    temp = temp->right;
} else {
    temp = temp->left;
}

while (temp != NULL) {
    if (temp->key == key) {
        if (temp->left == NULL && temp->right == NULL) {
            if (temp->key < prev->key) {
                prev->left = NULL;
            } else {
                prev->right = NULL;
            }
            delete temp;
        } else if (temp->left != NULL && temp->right == NULL) {
            if (temp->key < prev->key) {
                prev->left = temp->left;
                delete temp;
            } else {
                prev->right = temp->left;
                delete temp;
            }
        } else if (temp->left == NULL && temp->right != NULL) {
            if (temp->key < prev->key) {
                prev->left = temp->right;
                delete temp;
            } else {
                prev->right = temp->right;
                delete temp;
            }
        } else {
            node *l_temp = temp->left;
            node *l_prev = temp;
            if (l_temp->right == NULL) {
                l_prev->left = l_temp->left;
            } else {
```

```
                    while (l_temp->right != NULL) {
                        l_prev = l_temp;
                        l_temp = l_temp->right;
                    }
                    l_prev->right = l_temp->left;
                }

                if (temp->key < prev->key) {
                    prev->left = l_temp;
                } else {
                    prev->right = l_temp;
                }
                l_temp->left = temp->left;
                l_temp->right = temp->right;
                delete temp;
            }
            return 1;
        } else if (temp->key < key) {
            prev = temp;
            temp = temp->right;
        } else {
            prev = temp;
            temp = temp->left;
        }
    }
    return 0;
}

void bst::display(node *cur) {
    if (cur == NULL) {
        return;
    }
    display(cur->left);
    cout << cur->key << " : " << cur->value << endl;
    display(cur->right);
}

int main() {
    bst tree;
    int ch;
    string k, v, ans;
    do {
        cout << "MENU" << endl;
        cout << "1. Insert" << endl;
```

```
cout << "2. Search" << endl;
cout << "3. Update" << endl;
cout << "4. Delete" << endl;
cout << "5. Display Ascending" << endl;
cout << "0. Exit" << endl;
cout << "~ Enter your Choice:";
cin >> ch;
switch (ch) {
    case 1:
        cout << "Enter key to insert:";
        cin >> k;
        cout << "Enter value:";
        cin >> v;
        if (tree.insert(k, v)) {
            cout << "Element Inserted Successfully" << endl;
        } else {
            cout << "Element Already Present" << endl;
        }
        break;
    case 2:
        cout << "Enter key to search:";
        cin >> k;
        ans = tree.search(k);
        if (ans == "\0") {
            cout << "Element Not Found" << endl;
        } else {
            cout << "Value is " << ans << endl;
        }
        break;
    case 3:
        cout << "Enter key to Update:";
        cin >> k;
        cout << "Enter new value:";
        cin >> v;
        if (tree.update(k, v)) {
            cout << "Element Updated Successfully" << endl;
        } else {
            cout << "Element Not Present" << endl;
        }
        break;
    case 4:
        cout << "Enter key to Delete:";
        cin >> k;
        if (tree.delete_key(k)) {
```

```
                    cout << "Element Deleted Successfully" << endl;
                } else {
                    cout << "Element Not Present" << endl;
                }
                break;
            case 5:
                cout << "Data in Ascending order is " << endl;
                tree.display(tree.root);
                break;
            case 0:
                cout << "Thank You!" << endl;
                break;
            default:
                cout << "Please Enter a valid choice" << endl;
                break;
        }
    } while (ch != 0);
    return 0;
}
```

Output:

MENU

1. Insert

2. Search

3. Update

4. Delete

5. Display Ascending

0. Exit

~ Enter your Choice:1

Enter key to insert:c

Enter value:cat

Element Inserted Successfully

Enter key to insert:d

Enter value:dog

Element Inserted Successfully

Enter key to insert:b

Enter value:bat

Element Inserted Successfully

Enter key to insert:a

Enter value:apple

Element Inserted Successfully

MENU

1. Insert

2. Search

3. Update

4. Delete

5. Display Ascending

0. Exit

~ Enter your Choice:5

Data in Ascending order is

a : apple

b : bat

c : cat

d : dog

MENU

1. Insert

2. Search

3. Update

4. Delete

5. Display Ascending

0. Exit

~ Enter your Choice:2

Enter key to search:d

Value is dog

MENU

1. Insert

2. Search

3. Update

4. Delete

5. Display Ascending

0. Exit

~ Enter your Choice:3

Enter key to Update:c

Enter new value:catlog

Element Updated Successfully

MENU

1. Insert

2. Search

3. Update

4. Delete

5. Display Ascending

0. Exit

~ Enter your Choice:5

Data in Ascending order is

a : apple

b : bat

c : catlog

d : dog

MENU

1. Insert

2. Search

3. Update

4. Delete

5. Display Ascending

0. Exit

~ Enter your Choice:4

Enter key to Delete:c

Element Deleted Successfully

MENU

1. Insert

2. Search

3. Update

4. Delete

5. Display Ascending

0. Exit

~ Enter your Choice:5

Data in Ascending order is

a : apple

b : bat

d : dog