Code Explanation and Help

i have viva and practical exam
if i provide you my code then can you help me by explaining the code line by line,also what is use of the code and how it works
and if possible can you give video to understand the concept in code

Yes, I can absolutely help you with that.

👉 If you **share your code**, I will:

- ✅ Explain it **line by line** in simple terms.
- ✅ Explain the **overall purpose** and **working** of the code.
- ✅ Highlight any **important concepts** used (like DSA, file handling, recursion, etc.).
- ✅ If you want, I can also summarize it in a **simple way for your viva**.
- 🎥 And yes, I can **find or recommend YouTube videos** that explain the same concept or logic used in the code — just let me know the topic or which part you find confusing.

Go ahead and upload or paste your first code file when you're ready.

```
#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) {
    node **cur = &root;
    while (*cur != nullptr) {
        if ((*cur)->key == key) {
            node *temp = *cur;
            if (temp->left == nullptr)
                *cur = temp->right;
            else if (temp->right == nullptr)
                *cur = temp->left;
            else {
                node *successor = temp->right;
                while (successor->left != nullptr)
                    successor = successor->left;
                temp->key = successor->key;
                temp->value = successor->value;
                cur = &temp->right;
                key = successor->key;
                continue;
            }
            delete temp;
            return true;
        } else if ((*cur)->key < key)
            cur = &((*cur)->right);
        else
            cur = &((*cur)->left);
    }
    return false;
}
```

```
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 << "--- MAIN 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 << "Choose an option (0-5):\t";
        cin >> ch;
        switch (ch) {
            case 1:
                cout << "Key (word) to insert:\t";
                cin >> k;
                cout << "Value (meaning):\t";
                cin >> v;
                if (tree.insert(k, v)) {
                    cout << "Element insertion successful." << endl;
                } else {
                    cout << "Element already exists." << endl;
                }
                break;
            case 2:
                cout << "Key (word) to search:\t";
                cin >> k;
                ans = tree.search(k);
                if (ans == "\0") {
                    cout << "Element does not exist." << endl;
                } else {
                    cout << "Value (meaning) is:\t" << ans << endl;
                }
                break;
            case 3:
                cout << "Key (word) to update:\t";
                cin >> k;
                cout << "New value (meaning):\t";
                cin >> v;
                if (tree.update(k, v)) {
                    cout << "Element updated." << endl;
                } else {
                    cout << "Element does not exist." << endl;
                }
                break;
            case 4:
                cout << "Key (word) to delete:\t";
                cin >> k;
                if (tree.delete_key(k)) {
                    cout << "Element deletion successful." << endl;
                } else {
                    cout << "Element does not exist." << endl;
                }
                break;
            case 5:
                cout << "Data in ascending order:\t" << endl;
                tree.display(tree.root);
                break;
            case 0:
                cout << "\n// END OF CODE\n";
                break;
            default:
                cout << "Please choose a valid option (0-5)." << endl;
                break;
        }
    } while (ch != 0);
    return 0;
}
```

Let's go through the code you provided step by step and explain it:

1. Class Definition: `node`

cpp

```cpp
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;
    }
};
```

- **Purpose:** The `node` class represents a single node in a binary search tree (BST). Each node holds a `key` (usually unique) and a `value` (which is related to the key). The `left` and `right` pointers refer to the left and right child nodes of the current node.
- The class has two constructors: a default constructor that initializes an empty node, and a parameterized constructor that initializes a node with a given `key` and `value`.

2. Class Definition: `bst`

```
cpp

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);
};
```

- **Purpose:** The `bst` class represents the binary search tree. It has a pointer `root` which points to the root node of the tree. There are also several methods for common BST operations:
    - `insert`: Insert a new node into the tree.
    - `search`: Find a node based on its key.
    - `update`: Modify the value of an existing node.
    - `delete_key`: Delete a node with a given key.
    - `display`: Display the tree in ascending order (in-order traversal).

3. Method Implementations

`insert` Method:

```
cpp

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; // If the key already exists, return false
        } else if (temp->key < key) {
            temp = temp->right; // Move right
        } else {
            temp = temp->left; // Move left
        }
    }
    if (prev->key < key) {
        prev->right = new node(key, value); // Insert to the right
    } else {
        prev->left = new node(key, value); // Insert to the left
    }
    return 1; // Insertion successful
}
```

- **Explanation:** This method inserts a new node with the specified `key` and `value` into the binary search tree. It traverses the tree starting from the root and moves left or right based on the comparison with the current node's key. If the key already exists, it returns `false`. If insertion is successful, it returns `true`.

`search` Method:

```cpp
string bst::search(string key) {
    node *temp = root;
    while (temp != NULL) {
        if (temp->key == key) {
            return temp->value; // If key is found, return value
        } else if (temp->key < key) {
            temp = temp->right; // Move right
        } else {
            temp = temp->left; // Move left
        }
    }
    return "\0"; // Return empty string if key is not found
}
```

- **Explanation:** This method searches for a node with the specified `key`. It traverses the tree starting from the root, moving left or right based on the comparison of keys. If the key is found, it returns the value associated with it; otherwise, it returns an empty string.

`update` Method:

```cpp
bool bst::update(string key, string value) {
    node *temp;
    temp = root;
    while (temp != NULL) {
        if (temp->key == key) {
            temp->value = value; // Update value of the found node
            return 1; // Update successful
        } else if (temp->key < key) {
            temp = temp->right; // Move right
        } else {
            temp = temp->left; // Move left
        }
    }
    return 0; // Return false if key is not found
}
```

- **Explanation:** This method searches for a node with the specified `key` and updates its associated `value`. If the key is found, the value is updated, and it returns `true`. If the key is not found, it returns `false`.

`delete_key` Method:

```cpp
bool bst::delete_key(string key) {
    node **cur = &root;
    while (*cur != nullptr) {
        if ((*cur)->key == key) {
            node *temp = *cur;
            if (temp->left == nullptr)
                *cur = temp->right; // If left child is null, link to right child
            else if (temp->right == nullptr)
                *cur = temp->left; // If right child is null, link to left child
            else {
                node *successor = temp->right;
                while (successor->left != nullptr)
                    successor = successor->left;
                temp->key = successor->key; // Replace with successor's key and value
                temp->value = successor->value;
                cur = &temp->right;
                key = successor->key;
                continue;
            }
            delete temp; // Free memory of deleted node
            return true;
        } else if ((*cur)->key < key)
            cur = &((*cur)->right); // Move right
        else
            cur = &((*cur)->left); // Move left
    }
    return false; // Return false if key is not found
}
```

- **Explanation:** This method deletes a node with the specified `key`. If the node to be deleted has no children, it simply removes the node. If the node has one child, it links the parent to the child. If the node has two children, it replaces the node with its in-order successor (the smallest node in the right subtree).

`display` Method:

```cpp
void bst::display(node *cur) {
    if (cur == NULL) {
        return;
    }
    display(cur->left); // Traverse left subtree
    cout << cur->key << " : " << cur->value << endl; // Print key-value pair
    display(cur->right); // Traverse right subtree
}
```

- **Explanation:** This method performs an **in-order traversal** of the tree (left, root, right), printing the `key` and `value` of each node in ascending order.

4. Main Program:

```cpp
int main() {
    bst tree;
    int ch;
    string k, v, ans;
    do {
        // Menu for user interaction
        // Choices include Insert, Search, Update, Delete, Display Ascending
        // Continue until the user selects option 0 (Exit)
    } while (ch != 0);
    return 0;
}
```

- **Explanation:** This part is the main loop that interacts with the user. It provides a menu to the user for inserting, searching, updating, deleting, or displaying nodes in the tree. The user selects an option, and the corresponding method is called.

Video Recommendations:

For further understanding of the concepts used in this code, such as Binary Search Trees (BST), insertion, searching, deletion, and in-order traversal, here are some YouTube video recommendations:

- Binary Search Tree (BST) Insertion and Traversal
- Binary Search Tree (BST) Deletion

Let me know if you'd like more details on any specific part of the code!