

Code Explanation 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 definitely help you with that!

✓ **Here's what I can do for you:**

1. **Explain your code line by line** – so you understand *exactly* what each part is doing.
2. **Explain the purpose of the code** – what problem it solves and why it's useful.
3. **Explain how the code works** – the logic and flow behind it.
4. **Suggest videos** – I'll find and share high-quality YouTube videos (from trusted sources) that explain the **same concept** used in your code.
(Note: I can't create videos myself, but I'll fetch the best ones for your topic.)

Please upload or paste your code here, and let me know:

- What language it's in (Python, C++, etc.)
- Which concept it's related to (e.g., AVL tree, hashing, graph, file handling, etc.)

Would you like to start with a specific file or code snippet right now?



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

struct node{
    char data;
    node *left;
    node *right;
};

class tree{
    char prefix[50];
public:
    node *top;
    void expression(char []);
    void display(node *);
    void deletion(node *node);
};
```

```

class stack1{
public:
    node *data[30];
    int top;
    stack1(){
        top=-1;
    }
    int empty(){
        if(top== -1){
            return 1;
        }
        return 0;
    }
    void push(node *p){
        data[++top]=p;
    }

    node *pop(){

        return(data[top--]);
    }
};

void tree::expression(char prefix[]){
    char c;
    stack1 s;
    node *t1,*t2;
    int len,i;
    len=strlen(prefix);
    for(i=len-1;i>=0;i--){
        top = new node;
        top->left=NULL;
        top->right=NULL;

        if(isalpha(prefix[i])){
            top->data=prefix[i];
            s.push(top);
        }else if(prefix[i]=='+' || prefix[i]=='-' || prefix[i]=='*' || prefix[i]=='/'){
            t2 = s.pop();
            t1=s.pop();
            top->data=prefix[i];
            top->left = t2;
            top->right=t1;
            s.push(top);
        }
    }
    top = s.pop();
}

void tree::display(node *top){

```

```

stack1 s1,s2;

node *T = top;
s1.push(T);
while(!s1.empty()){
    T = s1.pop();
    s2.push(T);

    if(T->left!=NULL){
        s1.push(T->left);
    }

    if(T->right!=NULL){
        s1.push(T->right);
    }
}

while(!s2.empty()){

    top = s2.pop();
    cout<<top->data;

}
cout<<endl;
}

void tree::deletion(node *node){
    if(node==NULL){
        return;
    }
    deletion(node->left);
    deletion(node->right);
    cout<<"Deleting node: "<<node->data<<endl;
    free(node);
}

int main(){
    tree t;
    char exp1[20];

    int ch;
    do{
        cout<<"1 -> Enter prefix expression"<<endl;
        cout<<"2 -> Display postfix Expression"<<endl;
        cout<<"3 -> Deletion"<<endl;
        cout<<"4 -> Exit"<<endl;
        cout<<"Choose an option (1-4):\t";
        cin>>ch;

        switch(ch){

```

```

case 1:
    cout<<"Enter the expression (eg.: +--a*bc/def):\t";
    cin>>exp1;
    t.expression(exp1);
    break;
case 2:
    t.display(t.top);
    break;
case 3:
    t.deletion(t.top);
    break;
case 4:
    cout<<"\n// END OF CODE\n"<<endl;
    break;
default:
    cout<<"Choose a valid option (1-4).";
    break;
}
}while(ch!=4);
}

```

Let's go over this code line by line and explain its purpose and functionality in detail. This program is designed to work with **prefix expressions**, **postfix conversion**, and **tree node deletion**.

Code Explanation:

1. Header Files:

```

cpp

#include<iostream>
#include<string.h>
using namespace std;

```

- ``iostream``: Used for input/output operations like ``cin`` and ``cout``.
- ``string.h``: Provides string handling functions like ``strlen()`` to calculate the length of a string.

2. Node Structure:

```

cpp

struct node{
    char data;
    node *left;
    node *right;
};

```

- This defines a **binary tree node**. Each node holds:
 - A ``char`` data (``data``), which is a single character (used in the prefix expression).

- Two pointers `left` and `right` to connect to the left and right children of the node.

3. Tree Class Definition:

```
cpp

class tree{
    char prefix[50];
    public:
        node *top;
        void expression(char []);
        void display(node *);
        void deletion(node *);
};
```

- **`tree` class:** The core class of this program that handles:
 - Storing the top node of the tree (`top`).
 - `expression(char [])`: Converts a prefix expression into a binary tree.
 - `display(node *)`: Displays the expression in postfix notation.
 - `deletion(node *)`: Deletes all nodes of the tree recursively.

4. Stack Class Definition:

```
cpp

class stack1{
    public:
        node *data[30];
        int top;
        stack1(){
            top=-1;
        }
        int empty(){
            if(top==-1){
                return 1;
            }
            return 0;
        }
        void push(node *p){
            data[++top]=p;
        }

        node *pop(){
            return(data[top--]);
        }
};
```

- **`stack1` class:** Implements a stack data structure to help convert the prefix expression into a binary tree.
 - `data[30]`: Holds a maximum of 30 nodes (pointers to `node` objects).
 - `top`: Points to the top of the stack.
 - `empty()`: Returns 1 if the stack is empty, otherwise 0.
 - `push(node *p)`: Pushes a node onto the stack.
 - `pop()`: Pops a node from the stack and returns it.

5. Expression Function (Converts Prefix to Binary Tree):

cpp

```
void tree::expression(char prefix[]){
    char c;
    stack<node> s;
    node *t1,*t2;
    int len,i;
    len=strlen(prefix);
    for(i=len-1;i>=0;i--){
        top = new node;
        top->left=NULL;
        top->right=NULL;

        if(isalpha(prefix[i])){
            top->data=prefix[i];
            s.push(top);
        }else if(prefix[i]=='+'||prefix[i]=='-'||prefix[i]=='*'||prefix[i]=='/'){
            t2 = s.pop();
            t1=s.pop();
            top->data=prefix[i];
            top->left = t2;
            top->right=t1;
            s.push(top);
        }
    }
    top = s.pop();
}
```

- **Purpose:** This function converts a **prefix expression** (e.g., `+-a*bc/def`) into a **binary tree**.
 - It processes the prefix expression from right to left.
 - When an **operand (alpha character)** is encountered, it creates a node with that character and pushes it onto the stack.
 - When an **operator (+, -, *, /)** is encountered:
 - Pops two nodes from the stack (`t2` and `t1`).
 - Creates a new node with the operator as the root.
 - The first popped node (`t2`) becomes the left child, and the second popped node (`t1`) becomes the right child.
 - Pushes the new node back onto the stack.
 - Finally, the top of the stack holds the root node of the tree.

6. Display Function (Postfix Traversal):

cpp

```
void tree::display(node *top){
    stack<node> s1,s2;

    node *T = top;
    s1.push(T);
    while(!s1.empty()){
        T = s1.pop();
        s2.push(T);

        if(T->left!=NULL){
            s1.push(T->left);
        }
    }
```

```

        if(T->right!=NULL){
            s1.push(T->right);
        }
    }

    while(!s2.empty()){
        top = s2.pop();
        cout<<top->data;
    }
    cout<<endl;
}

```

- **Purpose:** This function performs **post-order traversal** on the binary tree and prints the result.
 - It uses two stacks (`s1` and `s2`) to simulate the post-order traversal.
 - The idea is to visit the left child, right child, and then the root.
 - The final result is printed in **postfix notation** (e.g., for the expression `+-a*bc/def`, it will print the postfix equivalent `abc*def/-+`).

7. Deletion Function (Deletes the Tree Nodes):

```

cpp

void tree::deletion(node *node){
    if(node==NULL){
        return;
    }
    deletion(node->left);
    deletion(node->right);
    cout<<"Deleting node: "<<node->data<<endl;
    free(node);
}

```

- **Purpose:** This function recursively deletes all the nodes in the tree.
 - It starts from the root and recursively deletes the left and right children.
 - After visiting both children, it frees the memory for the current node.
 - It prints a message indicating the deletion of each node.

8. Main Function:

```

cpp

int main(){
    tree t;
    char exp1[20];

    int ch;
    do{
        cout<<"1 -> Enter prefix expression"<<endl;
        cout<<"2 -> Display postfix Expression"<<endl;
        cout<<"3 -> Deletion"<<endl;
        cout<<"4 -> Exit"<<endl;
        cout<<"Choose an option (1-4):\t";
        cin>>ch;

        switch(ch){

            case 1:
                cout<<"Enter the expression (eg.: +-a*bc/def):\t";

```

```

        cin>>exp1;
        t.expression(exp1);
        break;
    case 2:
        t.display(t.top);
        break;
    case 3:
        t.deletion(t.top);
        break;
    case 4:
        cout<<"\n// END OF CODE\n"<<endl;
        break;
    default:
        cout<<"Choose a valid option (1-4).";
        break;
    }
}while(ch!=4);
}

```

- **Purpose:** This is the main driver of the program.
 - It allows the user to:
 - Enter a prefix expression and convert it to a binary tree.
 - Display the postfix equivalent of the expression.
 - Delete the tree nodes.
 - Exit the program.
 - The user selects options using a menu interface.

Summary of Program Flow:

- **Prefix Expression:** The user inputs a prefix expression.
- **Conversion:** The expression is converted into a binary tree.
- **Postfix Output:** The tree is traversed in post-order to print the postfix notation.
- **Tree Deletion:** Finally, the tree is deleted, freeing all allocated memory.

Video Recommendations:

Here are some high-quality YouTube video links to help you understand the concepts:

1. **Prefix, Infix, and Postfix Expressions:** [Video on Prefix, Infix, and Postfix](#)
2. **Binary Trees:** [Video on Binary Trees](#)
3. **Tree Traversals (Pre-order, In-order, Post-order):** [Video on Tree Traversals](#)

Let me know if you'd like more clarification or additional code explanations!