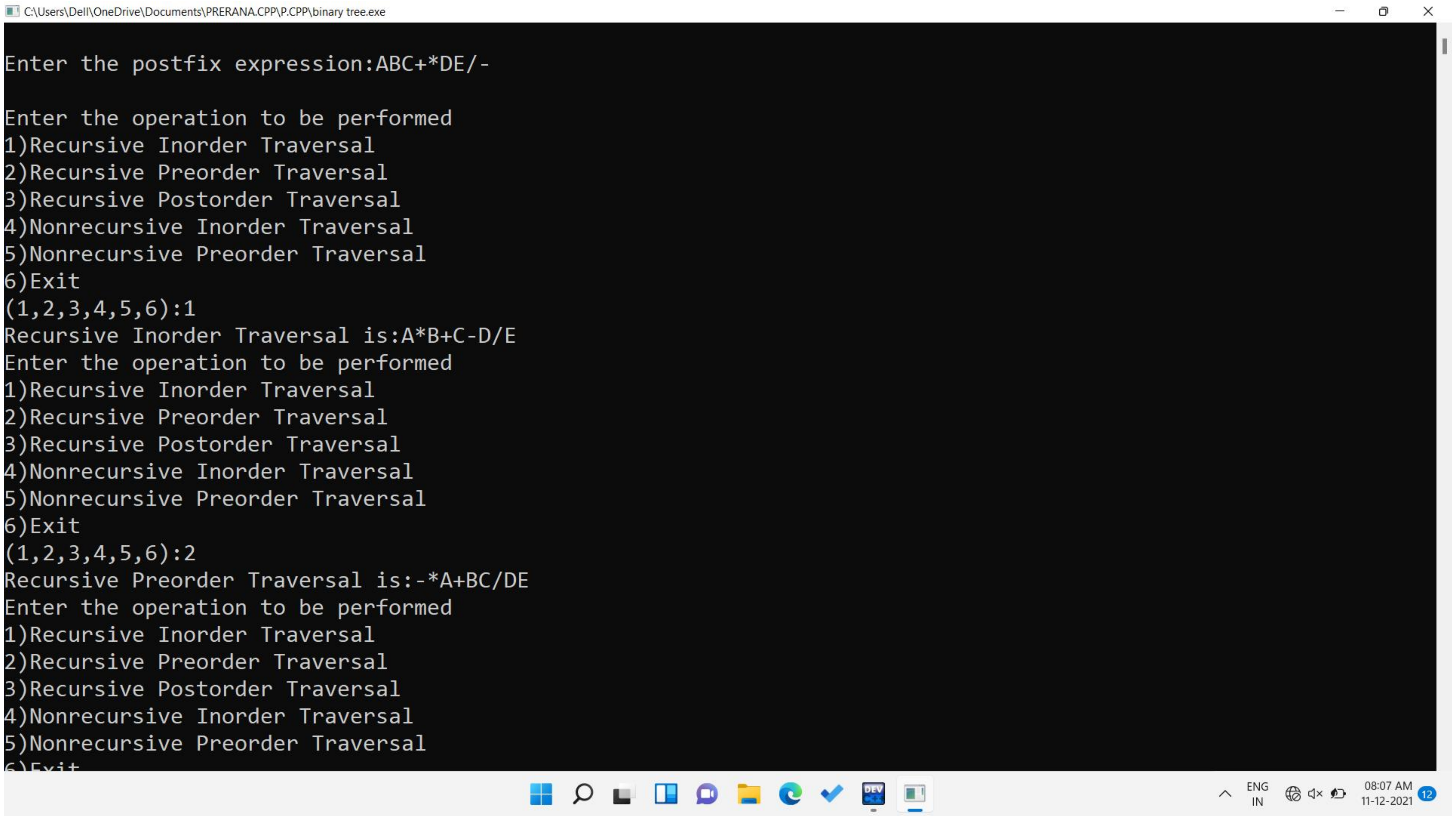
```
//                                      Assignment no-4
// Name- Prerana Rajesh Gajare      Class-SEIT      RollNo-SI41
/*PROBLEM STATEMENT:-
        Construct an Expression Tree from postfix expression. Perform recursive and non-recursive In-order,
        pre-order and post-order traversals.
*/
//Source Code:-
#include <iostream>
using namespace std;

//class tree
class tree
{
    char data;//Character data
    tree *left,*right;//Left and Right pointer
    public:
        //declaring funtions
        tree* create(char);
        tree* expression(char[]);
        void inorder(tree *);
        void preorder(tree *);
        void postorder(tree *);
        void nrecinorder(tree *);
        void nrecpreorder(tree *);
};

//Class stack
class stack
{

    public:
        tree *t;//declaring the pointer t of tree datatype
        stack *top;//Top pointer
        stack *link;//Link pointer
    //Default constructor
    stack()
    {
        top=NULL;//Initailising top as NULL
    }
    //declaring functions
    void push(tree *);
    void pop();
};

//TO PUSH AN ITEM IN STACK BY CREATING A NEW NODE
```

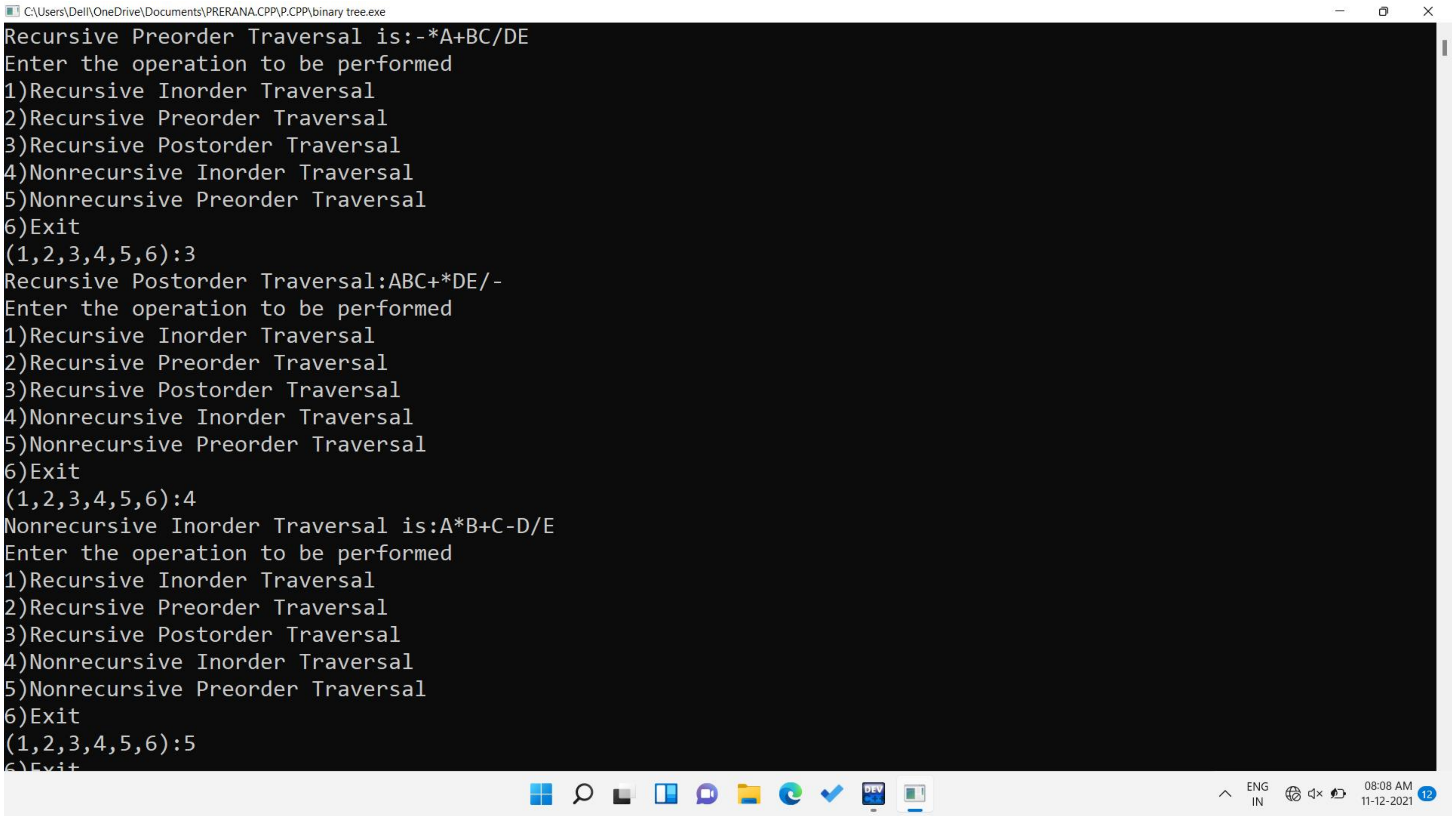
```
void stack::push(tree *item)
{

    stack *newnode=new stack();//Create a new node using new keyword
    newnode->t=item;//Store item value in t pointer of newnode
    newnode->link=NULL;//Store null value in link of newnode

    if(top==NULL)//If top is NULL store value of newnode in top
    {
        top = newnode;
    }
    else//Store top value in link of newnode and value of newnode in top
    {
        newnode->link=top;
        top = newnode;
    }
}

//TO POP/DELETE AN ELEMENT FROM TREE
void stack::pop()
{

    if (top==NULL)//If top is null then tree is empty
    {
        cout<<"Tree is empty";
    }
    else
    {
    stack *ptr1 = top;//Store top in prt1 and
    top=top->link;//increment the top by 1

    delete ptr1;
    }

}

tree* tree ::create(char operand)
{

    tree *newnode=new tree();//Create a newnode for the data operand using new keyword
    newnode->data=operand;//Store data in newnode of data
    newnode->left=NULL;//initialise left and right pointers to NULL and return newnode.
    newnode->right=NULL;
    return newnode;
}

//recursive traversal

//To perform Inorder Traversal of tree(LVR)
void tree::inorder(tree* ptr)    {
    if(ptr==NULL)
        return;

    inorder(ptr->left);
    cout<<ptr->data;
    inorder(ptr->right);
}

//To perform Preorder Traversal of tree(VLR)
void tree::preorder(tree* ptr) {
    if(ptr==NULL)
       return;

    cout<<ptr->data;
    preorder(ptr->left);
    preorder(ptr->right);
}

//To perform Postorder Traversal of tree(LRV)
void tree::postorder(tree* ptr) {
    if(ptr==NULL)
       return;

    postorder(ptr->left);
    postorder(ptr->right);
    cout<<ptr->data;
}

//non recursive traversal
void tree :: nrecinorder(tree *root)
{
    stack s;//Creating an object of class stack
    tree *curr=root;//Store root value in curr
    while(1)
    {
        while(curr)
        {
            s.push(curr);//Push curr in stack

            curr=curr->left;//move towards left
        }
        if(s.top!=NULL)
        {
            curr=s.top->t;//Store top element of stack in curr
            s.pop();//pop the element from stack
            cout<<curr->data;
            curr=curr->right;
        }
        else
            break;

    }
}

void tree :: nrecpreorder(tree *root)
{
    stack s;//Creating an object of class stack
    tree *curr=root;//Store root value in curr
    while(1)
    {
        while(curr)
        {
            cout<<curr->data;
            if(curr->right)
            {
                s.push(curr->right);//Push curr of right in stack
            }
            curr=curr->left;//Move towards left
        }
        if(s.top!=NULL)
        {
            curr=s.top->t;//Store top element of stack in curr
            s.pop();//pop the element from stack
        }
        else
            break;
    }
}
tree* tree ::expression(char postfix[])
{
    int i=0;
    stack s;//Creating an object of class stack
    tree *ptr;

```

```cpp
    while(postfix[i]!='\0')//Excecute the loop till the equation ends
    {
        ptr=create(postfix[i]);//Create node for each value of postfix array and store it in ptr
        if(isalnum(postfix[i]))//If data is alphabet or numeric directly push into the stack
        {
            s.push(ptr);
        }
        else
        {
            ptr->right=s.top->t;//Store top element of stack in ptr right
            s.pop();//Pop the data
            ptr->left=s.top->t;//Store top element of stack in ptr left
            s.pop();//Pop the data
            s.push(ptr);//push the pointer
        }
        i++;
    }
    ptr=s.top->t;//Store top element of stack in ptr
    s.pop();//Pop the data
    return ptr;
}


int main()
{
    tree t1;//Object of class tree
    stack s1;//Object of class stack
    tree *p;//To store the Postfix Expression
    char postfix[20];//Declaring a postfix array of size  20
    int l;
    //Accepting the expression as input
    cout<<"\nEnter the postfix expression:";
    cin>>postfix;
    p=t1.expression(postfix);
    do
    {
        cout<<"\nEnter the operation to be performed\n1)Recursive Inorder Traversal\n2)Recursive Preorder Traversal\n3)Recursive Postorder Traversal\n4)Nonrecursive Inorder Traversal\n5)Nonrecursive Preorder Traversal\n6)Exit\n(1,2,3,4,5,6):";
        cin>>l;
        switch(l)
        {

            case 1:
                cout<<"Recursive Inorder Traversal is:";
                t1.inorder(p);//Calling inorder function
                break;
            case 2:
                cout<<"Recursive Preorder Traversal is:";
                t1.preorder(p);//Calling preorder function
                break;
            case 3:
                cout<<"Recursive Postorder Traversal:";
                t1.postorder(p);//Calling postorder function
                break;
            case 4:
                cout<<"Nonrecursive Inorder Traversal is:";
                t1.nrecinorder(p);//Calling nrecinorder function
                break;
            case 5:

                cout<<"Nonrecursive Preorder Traversal is:";
                t1.nrecpreorder(p);//Calling nrecpreorder function
                break;
            case 6:
                cout<<"The End";
                break;
            default:
                cout<<"Wrong Choice";
        }
    }while(l!=6);
}


```



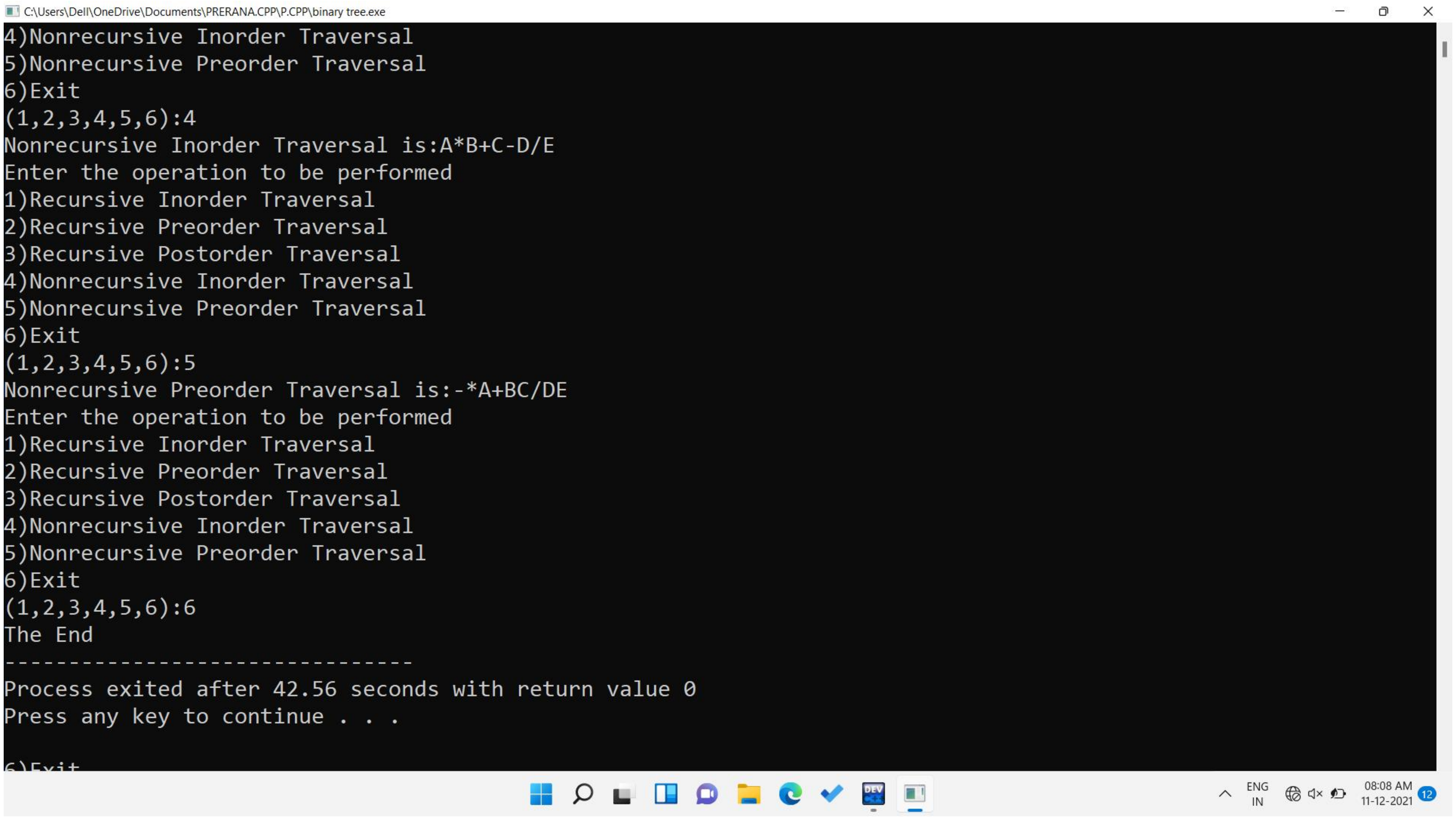
C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary tree.exe

```
Enter the postfix expression:ABC+*DE/-

Enter the operation to be performed
1)Recursive Inorder Traversal
2)Recursive Preorder Traversal
3)Recursive Postorder Traversal
4)Nonrecursive Inorder Traversal
5)Nonrecursive Preorder Traversal
6)Exit
(1,2,3,4,5,6):1
Recursive Inorder Traversal is:A*B+C-D/E
Enter the operation to be performed
1)Recursive Inorder Traversal
2)Recursive Preorder Traversal
3)Recursive Postorder Traversal
4)Nonrecursive Inorder Traversal
5)Nonrecursive Preorder Traversal
6)Exit
(1,2,3,4,5,6):2
Recursive Preorder Traversal is:-*A+BC/DE
Enter the operation to be performed
1)Recursive Inorder Traversal
2)Recursive Preorder Traversal
3)Recursive Postorder Traversal
4)Nonrecursive Inorder Traversal
5)Nonrecursive Preorder Traversal
```



C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary tree.exe

```
Recursive Preorder Traversal is:-*A+BC/DE
Enter the operation to be performed
1)Recursive Inorder Traversal
2)Recursive Preorder Traversal
3)Recursive Postorder Traversal
4)Nonrecursive Inorder Traversal
5)Nonrecursive Preorder Traversal
6)Exit
(1,2,3,4,5,6):3
Recursive Postorder Traversal:ABC+*DE/-
Enter the operation to be performed
1)Recursive Inorder Traversal
2)Recursive Preorder Traversal
3)Recursive Postorder Traversal
4)Nonrecursive Inorder Traversal
5)Nonrecursive Preorder Traversal
6)Exit
(1,2,3,4,5,6):4
Nonrecursive Inorder Traversal is:A*B+C-D/E
Enter the operation to be performed
1)Recursive Inorder Traversal
2)Recursive Preorder Traversal
3)Recursive Postorder Traversal
4)Nonrecursive Inorder Traversal
5)Nonrecursive Preorder Traversal
6)Exit
(1,2,3,4,5,6):5
6)Exit
```


C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary tree.exe

```
4)Nonrecursive Inorder Traversal
5)Nonrecursive Preorder Traversal
6)Exit
(1,2,3,4,5,6):4
Nonrecursive Inorder Traversal is:A*B+C-D/E
Enter the operation to be performed
1)Recursive Inorder Traversal
2)Recursive Preorder Traversal
3)Recursive Postorder Traversal
4)Nonrecursive Inorder Traversal
5)Nonrecursive Preorder Traversal
6)Exit
(1,2,3,4,5,6):5
Nonrecursive Preorder Traversal is:-*A+BC/DE
Enter the operation to be performed
1)Recursive Inorder Traversal
2)Recursive Preorder Traversal
3)Recursive Postorder Traversal
4)Nonrecursive Inorder Traversal
5)Nonrecursive Preorder Traversal
6)Exit
(1,2,3,4,5,6):6
The End
--------------------------------
Process exited after 42.56 seconds with return value 0
Press any key to continue . . .

6)Exit
```