DATA STRUCTURES AND ALGORITHMS LABORATORY

Group B
Assignment No. 2

NAME :- OJUS PRAVIN JAISWAL
ROLL NO. :- SACO19108
DIVISION :- A

Group B - Assignment 2

Title: Construct an expression tree from the given prefix expression eg. +--a*bc/def and traverse it using post order traversal and then delete the entire tree

Objectives:

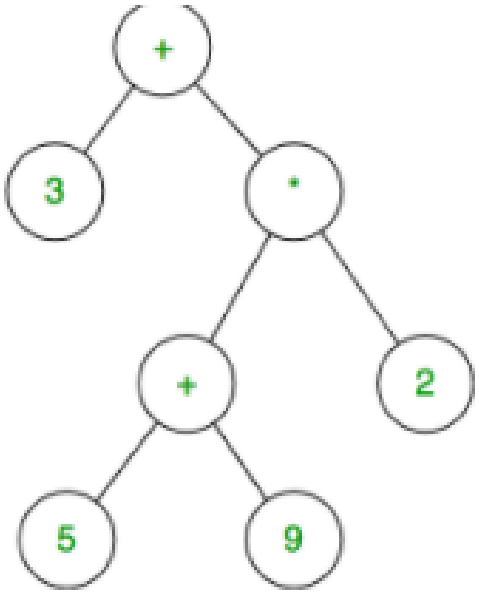
- To explain the concept of Tree & Binary Tree.
- To analyze the working of various Tree operations.

Outcome:

Students will be able to use various set of operations on Binary search.

Theory:

The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand so for example expression tree for
3 + ((5+9)*2) would be:



Given a character array a[] representing a prefix expression. The task is to build an Expression Tree for the expression and then print the infix and postfix expression of the built tree.

DATA STRUCTURE LABORATORY

Examples:

Input: a[] = "*+ab-cd"

Output: The Infix expression is:

a + b * c – d

The Postfix expression is:

a b + c d – *

Input: a[] = "+ab"

Output: The Infix expression is:

a + b

The Postfix expression is:

a b +

Algorithm:

Begin

class ExpressionTree which has following functions:

function push() to push nodes into the tree:

If stack is null

then push the node as first element

Else

push the node and make it top

function pop() to pop out nodes from the tree:

If stack is null

then print underflow

Else

Pop out the node and update top

function insert() to insert characters:

If it is digit

then push it.

Else if it is operator

Then pop it.

```
Else
  Print "invalid Expression"

function postOrder() for postorder traversal:
If tree is not empty
  postOrder(ptr->l)
  postOrder(ptr->r)
  Print root as ptr->d

function inOrder() for inorder traversal:
If tree is not empty
  inOrder(ptr->l)
  Print root as ptr->d
  inOrder(ptr->r)

function preOrder() for preorder traversal:
If tree is not empty
  Print root as ptr->d
  preOrder(ptr->l)
  preOrder(ptr->r)
End
```

Software Required: g++ / gcc compiler- / 64 bit Fedora, eclipse IDE

Input: prefix expression

DATA STRUCTURE LABORATORY

Program:

```
//================================================================
===========
// Name        : ExpressionTree.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : construct an expression tree for a prefix Expression in inorder, preorder and
postorder traversals
//================================================================
===========

#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cstring>
using namespace std;
//node declaration
class TREE_N
{
  public:
        char d;
        TREE_N *l, *r;
        TREE_N(char d)
        {
                this->d = d;
                this->l = NULL;
                this->r = NULL;
        }
};
```

```
// stack declaration
class StackNod
{
  public:
        TREE_N *treeN;
        StackNod *n;
        //constructor
        StackNod(TREE_N *treeN)
        {
                this->treeN = treeN;
                n = NULL;
        }
};

class ExpressionTree
{
  private:
        StackNod *top;

  public:
        ExpressionTree()
        {
                top = NULL;
        }
        void clear()
        {
                top = NULL;
        }

        void push(TREE_N *ptr)
        {
                if (top == NULL)
                        top = new StackNod(ptr);
```

```
        else
        {
                StackNod *nptr = new StackNod(ptr);
                nptr->n = top;
                top = nptr;
        }
}

TREE_N *pop()
{
        if (top == NULL)
        {
                cout << "Underflow" << endl;
                return 0;
        }
        else
        {
                TREE_N *ptr = top->treeN;
                top = top->n;
                return ptr;
        }
}

TREE_N *peek()
{
        return top->treeN;
}

void insert(char val)
{
        if (isalpha(val))
        {
                TREE_N *nptr = new TREE_N(val);
                push(nptr);
```

```
		}
		else if (isOperator(val))
		{
			TREE_N *nptr = new TREE_N(val);
			nptr->l = pop();
			nptr->r = pop();
			push(nptr);
		}
		else
		{
			cout << "Invalid Expression" << endl;
			return;
		}
	}

	/* bool isDigit(char ch) {
  return ch >= '0' && ch <= '9';
}*/

	bool isOperator(char ch)
	{
		return ch == '+' || ch == '-' || ch == '*' || ch == '/';
	}

	int toDigit(char ch)
	{
		return ch - '0';
	}

	void buildTree(string eqn)
	{
		for (int i = eqn.length() - 1; i >= 0; i--)
			insert(eqn[i]);
	}
```

```
void postfix()
{
        postOrder(peek());
}

void postOrder(TREE_N *ptr)
{
        if (ptr != NULL)
        {
                postOrder(ptr->l);
                postOrder(ptr->r);
                cout << ptr->d;
        }
}
void infix()
{
        inOrder(peek());
}

void inOrder(TREE_N *ptr)
{
        if (ptr != NULL)
        {
                inOrder(ptr->l);
                cout << ptr->d;
                inOrder(ptr->r);
        }
}
void prefix()
{
        preOrder(peek());
}
```

```
        void preOrder(TREE_N *ptr)
        {
                if (ptr != NULL)
                {
                        cout << ptr->d;
                        preOrder(ptr->l);
                        preOrder(ptr->r);
                }
        }
};

int main()
{
        string s, ch;
        int c;
        ExpressionTree et;
        cout << "\n----------EXPRESSION TREE----------\n";
another:
        cout << "\nEnter equation in Prefix form : ";
        cin >> s;
        et.buildTree(s);
        while (1)
        {
                cout << "\n-----Menu-----\n1) Prefix Form\n2) Infix Form\n3) Postfix Form\n4) Exit Loop\n";
                cout << "\nEnter your choice : ";
                cin >> c;
                switch (c)
                {
                case 1:
                        cout << "\nPrefix : ";
                        et.prefix();
                        cout << "\n";
                        break;
```

```
                case 2:
                        cout << "\nInfix : ";
                        et.infix();
                        cout << "\n";
                        break;
                case 3:
                        cout << "\nPostfix : ";
                        et.postfix();
                        cout << "\n";
                        break;
                case 4:
                        cout << "\nExitting Loop!!!";
                        cout << "\n";
                        goto loopexit;
                default:
                        cout << "\nWrong Choice Entered!!!";
                        cout << "\n";
                }
        }
loopexit:
        cout << "\nWant to convert another expression ? (Yes/No) : ";
        cin >> ch;
        if (ch == "No" || ch == "no" || ch == "NO" || ch == "N" || ch == "n")
        {
                cout << "\nExitting Program!!!";
                exit(0);
        }
        else if (ch == "Yes" || ch == "yes" || ch == "YES" || ch == "Y" || ch == "y")
        {
                goto another;
        }
        else
                goto loopexit;
}
```

Output:

```
----------EXPRESSION TREE----------

Enter equation in Prefix form : +--a*bc/def

-----Menu-----
1) Prefix Form
2) Infix Form
3) Postfix Form
4) Exit Loop

Enter your choice : 1

Prefix : +--a*bc/def

-----Menu-----
1) Prefix Form
2) Infix Form
3) Postfix Form
4) Exit Loop

Enter your choice : 2

Infix : a-b*c-d/e+f

-----Menu-----
1) Prefix Form
2) Infix Form
3) Postfix Form
4) Exit Loop

Enter your choice : 3

Postfix : abc*-de/-f+

-----Menu-----
1) Prefix Form
2) Infix Form
3) Postfix Form
4) Exit Loop

Enter your choice : 5

Wrong Choice Entered!!!

-----Menu-----
1) Prefix Form
2) Infix Form
3) Postfix Form
4) Exit Loop

Enter your choice : 4

Exitting Loop!!!
```

```
Want to convert another expression ? (Yes/No) : Yes

Enter equation in Prefix form : +ab

-----Menu-----
1) Prefix Form
2) Infix Form
3) Postfix Form
4) Exit Loop

Enter your choice : 1

Prefix : +ab

-----Menu-----
1) Prefix Form
2) Infix Form
3) Postfix Form
4) Exit Loop

Enter your choice : 2

Infix : a+b

-----Menu-----
1) Prefix Form
2) Infix Form
3) Postfix Form
4) Exit Loop

Enter your choice : 3

Postfix : ab+

-----Menu-----
1) Prefix Form
2) Infix Form
3) Postfix Form
4) Exit Loop

Enter your choice : 4

Exitting Loop!!!

Want to convert another expression ? (Yes/No) : No

Exitting Program!!!
[Program finished]
```

Conclusion: This program implements Binary expression tree data structure.