

Bansilal Ramnath Agarwal Charitable Trust's Vishwakarma Institute of Information Technology Department of

Artificial Intelligence and Data Science

Student Name: Siddhesh Dilip Khairnar

Class: S.Y. Division: B Roll No: 272028

Semester: IV Academic Year: 2022 - 23

Subject Name & Code: Advanced Data Structures: ADUA22202

Title of Assignment: Write a program to construct an expression tree from postfix/prefix expression and perform In-order, pre-order and post-order traversals.

Date of Performance: 1/02/2023 Date of Submission: 10/02/2023

		PAGE NO.:	
	Assignment wo 2	DATE: / /	
	Name: Siddhesh Oilip Khairnar		
	Quision: B Rouno: 272028		
	PRN NO 22110398		
	Aim: Conseruct expression tree from Postfix/Region expression and		
	Payorm recursive and non recursive In order, bee-order and		
	Post order transcrals		
i	TOTAL ORDER THEORY OF THE THE THEORY OF THE THE THEORY OF THE THEORY OF THE THEORY OF THE THEORY OF THE THE THE THE THEORY OF THE THEORY OF THE THEORY OF THE		
	Pachagound Tulamatian :-		
	Background Information: -		
	Expression tree: An expression tree is a binary tree that is used to		
	represent expression. The leaf nades of the expression-tru are operant		
1	and the other nodes are operator. The structure of the expression tree is		
	deturnined by the order of the operator and operand in the original		
	deturning agric sauce of the special		
	expression.		
	Come to annuaison the a. The appelling expression is	converted to an	
*	Postix to expression true: The postin expression is converted to an		
	expression tree by traversing the expression from left to right for each		
	operand, a new modes is created in the tree and for each operation, two was		
	are popped from the stack and a new node is created with the operator		
	as it data and the hur pepped wales as it left or	0	

* Prefix to expression tree: The prefix expression is convicted to an expression true by traversing the expression from left to eight for each operand, a new mode is weated in the tree and for each operation, thus

nodes are popped from the stack and a new node is reasted with the

operator or it data and the true papped wodes as its left and eight children

	PAGENO.:			
	DATE: / /			
	* Expression Tree Algorithm (from Postfix expression):			
	1. Initialize an empty stack			
	2. Traverse me postin expression from lest to right			
3. If the current characta is an operand, anate a new mode with the				
	operand as it data and push it onto the stack.			
4	4. If the current character is an operator, poptuonodes from the			
	stack, areate a new mode with the operator as it data, and make			
	The two popped nodes is its left and right children.			
5	5 Repeal steps 3 and 4 until the end of the expression is reached			
6.	6. The final wales remaining on the stack is the root of the expression			
	tree.			
*	Expression Tree Algorithm (from brejix Expression):			
	Initialize an empy empty stack			
	Traverse The prefix expression from right toleft.			
3.	If me werent character is an operand, weate a new node with the oper-			
- 2	and as it data and push it onto the stack.			
4	If me witers character is an operator, poptuo node from the stack,			
1	create a new node with the operator as it data, and make the two			
	pepped nodes its left and right children.			
S.	Repeat Step 3 and 4 write the end of the expression is reached.			
6.	The final wood remaining on the stack is the root of the expression the			
	0			
*	In- order Traversal (Recursive)			
1.	. If current mode is not null:			
	Toawase the eyesen were by calling the is order traversal function			
	recursively on the left child of the warnt node.			
0	Visit The current word and print it data.			
	Traverse the eight sub teer by calling The in order traversal function			

on the eight chied of the werent was. * In Order Francisal (Non- recursive): 1. Initialize a stack and push the toot nade onto the stack 2 Repeat the following step until the stack is Emply. A peopthe top node from the stack and visit it left child. B. If the left child is not rule, push it onto the stack c. If he left child is well, visit the mode and it eight child d. If the right child is not mul, push it into stack * Bre-order Traininal (Recursive): 1. The current node is not mu! a visit the current mode and print it data b. Travelse the left subtree eyealling tupe order travasal function recursively on the left chied of the other current mode. c. Traverse hu right subtree by calling the pu-order traverse function recursively on the right child of the current wade. * post order Traversal (recursive) 1. If the current mode is not well: a Traverse the subtree by calling the post order traversal function recusively on the left child of the waters node. v. Traverse the right subtree by calling the post order traversal function recursively

CODE AND OUTPUT

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;
class Node
public:
    Node *left;
    Node *right;
    char value;
    Node(char value)
        this->value = value;
        this->left = NULL;
        this->right = NULL;
    }
};
bool is_operand(char c)
    if ((c \rightarrow a' & a' & a' & a' & a') | (c \rightarrow a' & a' & a' & a' & a')
        return true;
    return false;
Node *build_tree(std::string expression, bool is_postfix)
    std::stack<Node *> stack;
    for (int i = 0; i < expression.length(); i++)</pre>
    {
        char c = expression[i];
        if (is_operand(c))
             stack.push(new Node(c));
         }
        else
             Node *node = new Node(c);
             if (is_postfix)
                 node->right = stack.top();
                 stack.pop();
                 node->left = stack.top();
```

```
stack.pop();
            }
            else
            {
                 node->left = stack.top();
                 stack.pop();
                 node->right = stack.top();
                 stack.pop();
            stack.push(node);
        }
    return stack.top();
void in_order_traversal(Node *root)
    if (root != NULL)
        in_order_traversal(root->left);
        std::cout << root->value << " ";</pre>
        in_order_traversal(root->right);
    }
void pre_order_traversal(Node *root)
    if (root != NULL)
    {
        cout << root->value << " ";</pre>
        pre_order_traversal(root->left);
        pre_order_traversal(root->right);
    }
void post_order_traversal(Node *root)
    if (root != NULL)
        post_order_traversal(root->left);
        post_order_traversal(root->right);
        cout << root->value << " ";</pre>
    }
int main()
    string expression = "ab+ef*g*-";
    Node *root = build_tree(expression, false);
```

```
cout << "In-order: ";
  in_order_traversal(root);
  cout << endl;
  cout << "Pre-order: ";
  pre_order_traversal(root);
  cout << endl;
  cout << "Post-order: ";
  post_order_traversal(root);
  cout << endl;
  return 0;
}</pre>
```

```
PS C:\Program language\C++> cd "c:\Program language\C++\" ; if ($?) { g++ ads2.cpp -o ads2 } ; if ($?)
{ .\ads2 }
O In-order: g * f * e - b + a
Pre-order: - * g * f e + b a
Post-order: g f e * * b a + -
PS C:\Program language\C++>
```

Conclusion: Successfully implemented program for expression tree from postfix/prefix expression and perform In-order, pre-order and post-order traversals.