

```

1: //                                     Assignment no-4
2: // Name- Prerana Rajesh Gajare      Class-SEIT      RollNo-SI41
3: /*PROBLEM STATEMENT:-
4:      Construct an Expression Tree from postfix expression. Perform
      recursive and non-recursive In-order,
5:      pre-order and post-order traversals.
6: */
7: //Source Code:-
8: #include <iostream>
9: using namespace std;
10:
11: //class tree
12: class tree
13: {
14:     char data;//Character data
15:     tree *left,*right;//Left and Right pointer
16:     public:
17:         //declaring funtions
18:         tree* create(char);
19:         tree* expression(char[]);
20:         void inorder(tree *);
21:         void preorder(tree *);
22:         void postorder(tree *);
23:         void nrecinorder(tree *);
24:         void nrecpreorder(tree *);
25: };
26:
27: //Class stack
28: class stack
29: {
30:
31:     public:
32:         tree *t;//declaring the pointer t of tree datatype
33:         stack *top;//Top pointer
34:         stack *link;//Link pointer
35:         //Default constructor
36:         stack()
37:         {
38:             top=NULL;//Initailising top as NULL
39:         }
40:         //declaring functions
41:         void push(tree *);
42:         void pop();
43: };
44:
45: //TO PUSH AN ITEM IN STACK BY CREATING A NEW NODE

```

```

46: void stack::push(tree *item)
47: {
48:
49:     stack *newnode=new stack();//Create a new node using new keyword
50:     newnode->t=item;//Store item value in t pointer of newnode
51:     newnode->link=NULL;//Store null value in link of newnode
52:
53:     if(top==NULL)//If top is NULL store value of newnode in top
54:     {
55:         top = newnode;
56:     }
57:     else//Store top value in link of newnode and value of newnode in
        top
58:     {
59:         newnode->link=top;
60:         top = newnode;
61:     }
62: }
63:
64: //TO POP/DELETE AN ELEMENT FROM TREE
65: void stack::pop()
66: {
67:
68:     if (top==NULL)//If top is null then tree is empty
69:     {
70:         cout<<"Tree is empty";
71:     }
72:     else
73:     {
74:         stack *ptr1 = top;//Store top in ptr1 and
75:         top=top->link;//increment the top by 1
76:
77:         delete ptr1;
78:     }
79:
80: }
81:
82: tree* tree ::create(char operand)
83: {
84:
85:     tree *newnode=new tree();//Create a newnode for the data operand
        using new keyword
86:     newnode->data=operand;//Store data in newnode of data
87:     newnode->left=NULL;//initialise left and right pointers to NULL
        and return newnode.
88:     newnode->right=NULL;

```

```

89:     return newnode;
90: }
91:
92: //recursive traversal
93:
94: //To perform Inorder Traversal of tree(LVR)
95: void tree::inorder(tree* ptr)    {
96:     if(ptr==NULL)
97:         return;
98:
99:     inorder(ptr->left);
100:    cout<<ptr->data;
101:    inorder(ptr->right);
102: }
103:
104: //To perform Preorder Traversal of tree(VLR)
105: void tree::preorder(tree* ptr) {
106:     if(ptr==NULL)
107:         return;
108:
109:     cout<<ptr->data;
110:     preorder(ptr->left);
111:     preorder(ptr->right);
112: }
113:
114: //To perform Postorder Traversal of tree(LRV)
115: void tree::postorder(tree* ptr) {
116:     if(ptr==NULL)
117:         return;
118:
119:     postorder(ptr->left);
120:     postorder(ptr->right);
121:     cout<<ptr->data;
122: }
123:
124: //non recursive traversal
125: void tree :: nrecinorder(tree *root)
126: {
127:     stack s; //Creating an object of class stack
128:     tree *curr=root; //Store root value in curr
129:     while(1)
130:     {
131:         while(curr)
132:         {
133:             s.push(curr); //Push curr in stack

```

```

134:
135:         curr=curr->left; //move towards left
136:     }
137:     if(s.top!=NULL)
138:     {
139:         curr=s.top->t; //Store top element of stack in curr
140:         s.pop(); //pop the element from stack
141:         cout<<curr->data;
142:         curr=curr->right;
143:     }
144:     else
145:         break;
146:
147: }
148: }
149:
150: void tree :: nrecpreorder(tree *root)
151: {
152:     stack s; //Creating an object of class stack
153:     tree *curr=root; //Store root value in curr
154:     while(1)
155:     {
156:         while(curr)
157:         {
158:             cout<<curr->data;
159:             if(curr->right)
160:             {
161:                 s.push(curr->right); //Push curr of right in stack
162:             }
163:             curr=curr->left; //Move towards left
164:         }
165:         if(s.top!=NULL)
166:         {
167:             curr=s.top->t; //Store top element of stack in curr
168:             s.pop(); //pop the element from stack
169:         }
170:         else
171:             break;
172:     }
173: }
174: tree* tree :: expression(char postfix[])
175: {
176:     int i=0;
177:     stack s; //Creating an object of class stack
178:     tree *ptr;
179:

```

```

180:     while(postfix[i]!='\0')//Excecute the loop till the equation ends
181:     {
182:         ptr=create(postfix[i]);//Create node for each value of
        postfix array and store it in ptr
183:         if(isalnum(postfix[i]))//If data is alphabet or numeric
        directly push into the stack
184:         {
185:             s.push(ptr);
186:         }
187:         else
188:         {
189:             ptr->right=s.top->t;//Store top element of stack in ptr
        right
190:             s.pop();//Pop the data
191:             ptr->left=s.top->t;//Store top element of stack in ptr
        left
192:             s.pop();//Pop the data
193:             s.push(ptr);//push the pointer
194:         }
195:         i++;
196:     }
197:     ptr=s.top->t;//Store top element of stack in ptr
198:     s.pop();//Pop the data
199:     return ptr;
200: }
201:
202:
203: int main()
204: {
205:     tree t1;//Object of class tree
206:     stack s1;//Object of class stack
207:     tree *p;//To store the Postfix Expression
208:     char postfix[20];//Declaring a postfix array of size 20
209:     int l;
210:     //Accepting the expression as input
211:     cout<<"\nEnter the postfix expression:";
212:     cin>>postfix;
213:     p=t1.expression(postfix);
214:     do
215:     {
216:         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):";
217:         cin>>l;
218:         switch(l)

```

```

219:         {
220:
221:             case 1:
222:                 cout<<"Recursive Inorder Traversal is:";
223:                 t1.inorder(p);//Calling inorder function
224:                 break;
225:             case 2:
226:                 cout<<"Recursive Preorder Traversal is:";
227:                 t1.preorder(p);//Calling preorder function
228:                 break;
229:             case 3:
230:                 cout<<"Recursive Postorder Traversal:";
231:                 t1.postorder(p);//Calling postorder function
232:                 break;
233:             case 4:
234:                 cout<<"Nonrecursive Inorder Traversal is:";
235:                 t1.nrecinorder(p);//Calling nrecinorder function
236:                 break;
237:             case 5:
238:
239:                 cout<<"Nonrecursive Preorder Traversal is:";
240:                 t1.nrecpreorder(p);//Calling nrecpreorder function
241:                 break;
242:             case 6:
243:                 cout<<"The End";
244:                 break;
245:             default:
246:                 cout<<"Wrong Choice";
247:         }
248:     }while(l!=6);
249: }
250:
251:

```

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
- 6)Exit

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


```
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
```