Navlakhi®

Prefix Postfix to ALL

Methodology and Program

By Abhishek Navlakhi

Semester 3: Data Structures

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<math.h>
#define MAX 50

struct node
{
            struct node* left_child;
            char x;
            struct node* right_child;
} * stack[50];

int top = -1;

struct node* CreateExpTreePostfix(char*);
struct node* CreateExpTreePrefix(char*);
void preorder(struct node* sr);
void inorder(struct node* sr);
void postorder(struct node* sr);
int Evaluate(struct node* sr);
void push(struct node*);
struct node* pop();
void Delete_Tree(struct node*);
```

```
void main( )
{
        struct node* root;
        char str[50];
        int z;
        char ch;
         clrscr();
        printf("Input expression is:\n1)Prefix\n2)Postfix ");
        ch=getche();
        if(ch=='1')
         {
              printf("\nEnter Prefix Expression:");
              gets(str);
              root = CreateExpTreePrefix(str);
         }
        else
          {
              printf("\nEnter Postfix Expression:");
              gets(str);
              root =  CreateExpTreePostfix(str);
          }
       printf("\nPrefix Exp.  :");
           preorder(root);
        printf("\nInfix Exp.   :");
           inorder(root);
        printf("\nPostfix Exp. :");
           postorder(root);

           z=Evaluate(root);
        printf("\nExpression Evaluated to: %d", z);
            Delete_Tree(root);

}
```

```
struct node* CreateExpTreePostfix(char* str)
{
        struct node* nleft, * nright, * nodeptr;
        while (*str)
        {
              nodeptr = (struct node *) malloc(sizeof(struct node));
               nodeptr->x = *str;
              if (*str == '+' ||  *str == '-' || *str == '/' || *str == '*' || *str == '^')
                {
                        nright = pop();
                        nleft = pop();
                         nodeptr->left_child = nleft;
                        nodeptr->right_child = nright;
                }
                else
               {
                     nodeptr->left_child = NULL;
                       nodeptr->right_child = NULL;
                }
                push(nodeptr);
                str++;
        }
         return pop();
}
```

```
struct node* CreateExpTreePrefix(char* str)
{
        struct node* nleft, * nright, * nodeptr;
        strrev(str);
        while (*str)
        {
                nodeptr = (struct node *) malloc(sizeof(struct node));
                nodeptr->x=*str;
               if (*str == '+' || *str == '-' || *str == '/' || *str == '*' || *str == '^')
                {
                        nleft = pop();
                        nright = pop();

                        nodeptr->left_child = nleft;
                        nodeptr->right_child = nright;
                 }
                else
                {
                        nodeptr->left_child = NULL;
                        nodeptr->right_child = NULL;
                }
                push(nodeptr);
                str++;
        }
        return pop();
}
```

```
void inorder(struct node* p)
{
        if (p != NULL)
        {
                inorder(p -> left_child) ;
                 printf("%c", p ->x) ;
                 inorder(p -> right_child) ;
        }
}

 void preorder(struct node* p)
{
        if (p != NULL)
        {
                 printf("%c", p -> x) ;
                 preorder(p -> left_child) ;
                 preorder(p -> right_child) ;
        }
}



void postorder(struct node* p)
{
          if (p != NULL)
        {
                postorder(p -> left_child) ;
                postorder(p -> right_child) ;
                printf("%c", p -> x) ;
        }
}
```

```
void push(struct node* ptr)
{
        if (top == MAX - 1)
                printf("\nStack is full.\n") ;
        else
        {
                top++ ;
                stack[top] = ptr ;
        }
}


struct node* pop()
{
        if (top == -1)
        {
                printf("Stack is empty\n") ;
                return -1 ;
        }
        else
        {
                struct node* ptr = stack[top] ;
                top-- ;
                return ptr ;
        }
}
```

```
int Evaluate(struct node* sr)
{
  int x,y,z;
        if (sr != NULL)
        {
       if (sr->x == '+'|| sr->x == '-' ||  sr->x == '/' ||  sr->x == '*' ||  sr->x == '^')
                {
                 x = Evaluate(sr -> left_child) ;
                 y = Evaluate(sr -> right_child) ;
                switch(sr->x)
                 {
                   case  '+' : z=x+y;      break;
                   case  '-' : z=x-y;      break;
                   case  '*' : z=x*y;      break;
                   case  '/' : z=x/y;      break;
                   case  '^' : z=pow(x,y); break;
                  }
                 return z;
                }
                return (sr -> x - 48) ;         }
}


void Delete_Tree(struct node * root)
{
   if(root!=NULL)
   {
        Delete_Tree(root->left_child);
        Delete_Tree(root->right_child);
        free(root);
   }
}
```

HOME OF EDUCATION

Navlakhi's

www.navlakhi.com
Home of Education

DATA Structures@ Navlakhi's

DATA Structures@ Navlakhi's