```
//S Harshini-185001058


#include<stdio.h>
#include<bits.h>
#include<stdlib.h>
#include"treestackADT.h"
#include"charstackADT.h"
void main()
{
            char in[50];
            printf("\nenter an expression");
            scanf("%s",in);
            postfix(in);
}

/* contents of file charstackDT

void postfix(char in[])
{
 int l=strlen(in);
 int j=0;
 char post[20]={0};

 for(int k=0;k<l;k++)
 {
            if(in[k]=='(')
                        ppush(in[k]);
            else
            {
                        if(ppeek()=='(' && in[k]==')')
                                    {
                                                if(ppeek())
                                                            ppop();
                                     }
            }
}

 for(int i=0;i<l;i++)
 {
    if(in[i]=='(')
      ppush('(');
    else if(in[i]==')')
 {
   while(ppeek()!='(')
     {
       post[j]=ppeek();
       j++;
       ppop();
     }
   ppop();
 }
 else if(in[i]=='+' || in[i]=='-')
 {
   if((ppeek()=='+')||(ppeek()=='-')||(ppeek()=='*')||(ppeek()=='/' ))
   {
     while(ppeek()!='(')
                {
                            post[j]=ppeek();
         j++;
         ppop();
                }
                ppush(in[i]);
   }
   else
     ppush(in[i]);
 }
else if((in[i]=='*')||(in[i]=='/'))
{
 if((ppeek()=='*')||(ppeek()=='/'))
 {
  while((ppeek()!='+')&&(ppeek()!='-')&&(ppeek()!='('))
  {
         post[j]=ppeek();
         j++;
         ppop();
         }
         ppush(in[i]);
 }
 else
  ppush(in[i]);
}
else
 {
  post[j]=in[i];
  j++;
```

```
   }

  }
printf("\n%s",post);
int p;
for(int i=0;i<strlen(post);i++)
{
            p=isoper(post[i]);
            if(p==0)
                        operand(post[i]);
            else
                        operator(post[i]);
}

printf("\n infix exp is");
inorder(stack[topv]);
printf("\n prefix exp is");
preorder(stack[topv]);
printf("\n postfix exp is");
postorder(stack[topv]);

}

*/




/*contents of file treestackADT
struct node
{
 char data;
 struct node *next;
}*top=NULL;
void display()
{
  printf("CONTENTS ARE:-");
  for( struct node *temp=top; temp!=NULL ; temp=temp->next)
    printf("%c\t", temp->data  );
  printf("\n");
}
void ppush(char x)
{
            struct node *new;
            new=(struct node*)malloc(sizeof(struct node));
            new->data=x;
            if(top==NULL)
                        new->next=NULL;
            else
                        new->next=top;
            top=new;
}
void ppop()
{
 if(top==NULL)
  printf("\nstack is empty");
 else
  {
    struct node *temp;
            temp=(struct node*)malloc(sizeof(struct node));
    temp=top;
    top=temp->next;
   free(temp);
  }

}
char ppeek()
{
 if(top==NULL){
  return 0;}
 else
  return top->data;
}




struct et
{
            char val;
            struct et *left,*right;
};
struct et *stack[30];
struct et *node;
int topv=-1;

void push(struct et* node)
```

```
{
        stack[++topv]=node;
}
struct et * pop()
{
        return(stack[topv--]);
}
void inorder(struct et *node)
{
   if(node!=NULL)
   {
   inorder(node->left);
   printf("%c",node->val);
   inorder(node->right);
   }
}
void preorder(struct et *node)
{
   if(node!=NULL)
   {
   printf("%c",node->val);
        preorder(node->left);
   preorder(node->right);
   }
}
void postorder(struct et *node)
{
   if(node!=NULL)
   {
        postorder(node->left);
   postorder(node->right);
   printf("%c",node->val);
   }
}

void operand(char a)
{

        node=(struct et*)malloc(sizeof(struct et));
        node->val=a;
        node->left=NULL;
        node->right=NULL;
        push(node);
}
void operator(char b)
{
	node=(struct et*)malloc(sizeof(struct et));
	node->val=b;
	node->right=pop();
	node->left=pop();
	push(node);
}
int isoper(char c)
{
	if(c=='+' || c=='-' || c=='*' || c=='/')
		return 1;
	return 0;
}
*/


/*SAMPLE INPUT/OUTPUT

enter an expression((2+5)*(3-6)/(7*8))

25+36-*78*/
 infix exp is2+5*3-6/7*8
 prefix exp is/*+25-36*78
 postfix exp is25+36-*78*/

 enter an expression(7-(((3+2)*(6+1))/(5+6)))

732+61+*56+/-
 infix exp is7-3+2*6+1/5+6
 prefix exp is-7/*+32+61+56
 postfix exp is732+61+*56+/-

enter an expression((3+2)*(2+5))

32+25+*
 infix exp is3+2*2+5
 prefix exp is*+32+25
 postfix exp is32+25+*


*/
```