

NAME : NITHISH.G

REG NO : 19BCS0012

COURSE : DATA STRUCTURE

Prof : MARY MEKALA A

**SLOT : L37+L38**

1. Implement a c program to check whether the expression has a balanced parenthesis using stack.

PROGRAM:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define MAX 30

int top=-1;

int stack[MAX];

void push(char);

char pop();

int match(char a,char b);

int check(char []);

int main()

{

char exp[MAX];

int valid;

printf("Enter an algebraic expression : ");

gets(exp);

valid=check(exp);

if(valid==1)

printf("Valid expression\n");

else

printf("Invalid expression\n");

return 0;

}

int check(char exp[] )

{

int i;

char temp;

for(i=0;i<strlen(exp);i++)

{

if(exp[i]=='(' || exp[i]=='{' || exp[i]=='[')

push(exp[i]);

if(exp[i]==')' || exp[i]=='}' || exp[i]==']')

if(top==-1) /\*stack empty\*/

{

printf("Right parentheses are more than left parentheses\n");

return 0;

}

else

{

temp=pop();

if(!match(temp, exp[i]))

{

printf("Mismatched parentheses are : ");

printf("%c and %c\n",temp,exp[i]);

return 0;

}

}

}

if(top==-1) /\*stack empty\*/

{

printf("Balanced Parentheses\n");

return 1;

}

else

{

printf("Left parentheses more than right parentheses\n");

return 0;

}

}

int match(char a,char b)

{

if(a=='[' && b==']')

return 1;

if(a=='{' && b=='}')

return 1;

if(a=='(' && b==')')

return 1;

return 0;

}

void push(char item)

{

if(top==(MAX-1))

{

printf("Stack Overflow\n");

return;

}

top=top+1;

stack[top]=item;

}

char pop()

{

if(top==-1)

{

printf("Stack Underflow\n");

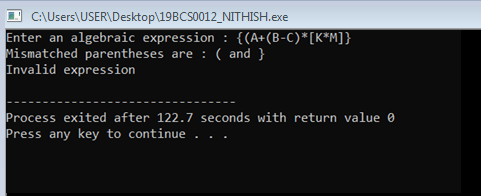
exit(1);

}

return(stack[top--]);

}

OUTPUT:



1. Implement a C program to convert an infix expression to postfix expression using stack.

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define SIZE 100

char stack[SIZE];

int top = -1;

void push(char item)

{

if(top >= SIZE-1)

{

printf("\nStack Overflow.");

}

else

{

top = top+1;

stack[top] = item;

}

}

char pop()

{

char item ;

if(top <0)

{

printf("stack under flow: invalid infix expression");

getchar();

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

int is\_operator(char symbol)

{

if(symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol =='-')

{

return 1;

}

else

{

return 0;

}

}

int precedence(char symbol)

{

if(symbol == '^')

{

return(3);

}

else if(symbol == '\*' || symbol == '/')

{

return(2);

}

else if(symbol == '+' || symbol == '-')

{

return(1);

}

else

{

return(0);

}

}

void InfixToPostfix(char infix\_exp[], char postfix\_exp[])

{

int i, j;

char item;

char x;

push('(');

strcat(infix\_exp,")");

i=0;

j=0;

item=infix\_exp[i];

while(item != '\0')

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

postfix\_exp[j] = item;

j++;

}

else if(is\_operator(item) == 1)

{

x=pop();

while(is\_operator(x) == 1 && precedence(x)>= precedence(item))

{

postfix\_exp[j] = x;

j++;

x = pop();

}

push(x);

push(item);

}

else if(item == ')')

{

x = pop();

while(x != '(')

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

else

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

i++;

item = infix\_exp[i];

}

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

postfix\_exp[j] = '\0';

}

int main()

{

char infix[SIZE], postfix[SIZE];

printf("\nEnter Infix expression : ");

gets(infix);

InfixToPostfix(infix,postfix);

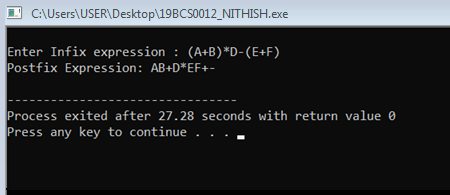
printf("Postfix Expression: ");

puts(postfix);

return 0;

}

OUTPUT:



1. Implement a c program to perform evaluation of prefix / postfix expression using stack

PROGRAM:

#include<stdio.h>

#include<conio.h>

int stack[10];

int top = -1;

void push(int x)

{

stack[++top] = x;

}

int pop()

{

return stack[top--];

}

int main()

{

char exp[10];

char \*e;

int op1,op2,result,num;

printf("Enter the postfix expression:");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isdigit(\*e))

{

num = \*e - 48;//ascii value

push(num);

}

else

{

op1 = pop();

op2 = pop();

switch(\*e)

{

case '+':

{

result = op1 + op2;

break;

}

case '-':

{

result = op1 - op2;

break;

}

case '\*':

{

result = op1 \* op2;

break;

}

case '/':

{

result= op2 / op1;

break;

}

}

push(result);

}

e++;

}

printf("\nThe result of expression %s = %d\n\n",exp,pop());

getch();

}

OUTPUT:

