// PROGRAM TO CONVERT AN INFIX EXPRESSION TO POSTFIX EXPRESSION

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

//MACROS

#define ENTER '\0'

#define BLANK ' '

#define TAB '\t'

#define MAXLENGTH 64

#define empty (-1)

#define operator (-10)

#define operand (-20)

#define leftparen (-30)

#define rightparen (-40)

//SYMBOL PRECEDENCE

#define leftparenprec 0

#define addprec 1

#define subprec 1

#define multprec 2

#define divprec 2

#define remprec 2

#define expoprec 3

#define none 999

//MODULE DECLARATIONS

void read\_input(char[]);

void infix\_to\_postfix(char[],char[],char[],int\*);

void write\_output(char[],char[]);

void push(char[],int\*,char symbol);

char pop(char[],int\*);

int get\_type(char);

int get\_prec(char);

int white\_space(char);

void full\_stack();

void empty\_stack();

void main()

{

char infix[MAXLENGTH+1],stack[MAXLENGTH],postfix[MAXLENGTH+1];

static int top;

//static char\* symbols="()+-%\*/$";

char ch;

printf("\n\n\t\t\t<<<<<<<< NOTATIONS >>>>>>>>\n\n");

do

{

top=empty;

read\_input(infix);

infix\_to\_postfix(infix,postfix,stack,&top);

write\_output(infix,postfix);

printf("\n\nDo You Wish To Continue[y/n]:");

ch=getchar();

printf("\n");

}while(ch=='Y'||ch=='y');

}

void infix\_to\_postfix(char infix[],char postfix[],char stack[],int \*top)

{

int i,p,len,type,precedence;

char next;

i=p=0;

len=strlen(infix);

while(i<len)

{

if(white\_space(infix[i]))

{

type=get\_type(infix[i]);

switch(type)

{

case leftparen:

push(stack,top,infix[i]);

break;

case rightparen:

while((next=pop(stack,top))!='(')

postfix[p++]=next;

break;

case operand:

postfix[p++]=infix[i];

break;

case operator:

precedence=get\_prec(infix[i]);

while(\*top>empty&&precedence<=get\_prec(stack[\*top]))

postfix[p++]=pop(stack,top);

push(stack,top,infix[i]);

break;

}

}

i++;

}

while(\*top>empty)

postfix[p++]=pop(stack,top);

postfix[p]=ENTER;

}

int white\_space(char c)

{

if (c==BLANK || c==ENTER || c==TAB)

return 0;

else

return 1;

}

int get\_type(char symbol)

{

switch(symbol)

{

case '(':return(leftparen);

case ')':return(rightparen);

case '+':

case '-':

case '%':

case '\*':

case '/':

case '$':

return(operator);

default:return(operand);

}

}

int get\_prec(char symbol)

{

switch(symbol)

{

case '(':return(leftparenprec);

case '+':return(addprec);

case '-':return(subprec);

case '%':return(remprec);

case '\*':return(multprec);

case '/':return(divprec);

case '$':return(expoprec);

default: return(none);

}

}

void push(char stack[],int \*top,char symbol)

{

if(\*top>MAXLENGTH)

full\_stack();

else

stack[++(\*top)]=symbol;

}

char pop(char stack[],int \*top)

{

if(\*top<=empty)

{

empty\_stack();

return 0;

}

else

return(stack[(\*top)--]);

}

void full\_stack(void)

{

printf("\nSorry, Stack is Full !\n");

exit(1);

}

void empty\_stack(void)

{

printf("\nSorry, Stack is Empty !\n");

exit(2);

}

//READ

void read\_input(char infix[])

{

printf("\n Enter the Infix max upto %d Characters : ",MAXLENGTH);

gets(infix);

}

//WRITE

void write\_output(char infix[],char postfix[])

{

printf("\n Infix : %s",infix);

printf("\n Postfix : %s",postfix);

}

