* INFIX TO POSTFIX

/\*Infix to postfix\*/

#define MAX 50

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

struct stack

{

int top;

char items[MAX];

};

void push(struct stack \*,char);

char pop(struct stack \*);

char peek(struct stack \*);

void infixtopost(char[]);

int isOperand(char);

int empty(struct stack \*);

int icp(char);

struct stack st;

char postfix[MAX];

int main()

{

char infix[MAX];

st.top=-1;

printf("Enter infix expressoin");

scanf("%s",infix);

printf("Infix Expressions is %s",infix);

infixtopost(infix);

return 0;

}

int empty(struct stack \*st)

{

if(st->top==-1)

{

return 1;

}

else

{

return 0;

}

}

int isOperand(char c)

{

if((c>='a'&&c<='z')||(c>='A'&&c<='Z')||(c>='0'&&c<='9'))

{

return 1;

}

else

{

return 0;

}

}

int isp(char c)

{

switch(c)

{

case '(': return 0;

case '+':

case '-': return 1;

case '\*':

case '/': return 2;

case '^':

case '$': return 3;

}

}

int icp(char c)

{

switch(c)

{

case '+':

case '-': return 1;

case '\*':

case '/': return 2;

case '^':

case '$': return 3;

}

}

void infixtopost(char infix[])

{

int i,j=0;

char sym,topsym;

for(i=0;(sym=infix[i])!='\0';i++)

{

if(isOperand(sym))

{

postfix[j]=sym;

j++;

}

else if(sym=='(')

{

push(&st,sym);

}

else if(sym==')')

{

topsym = pop(&st);

while (topsym!='(' && !empty(&st))

{

postfix[j]=topsym;

j++;

topsym =pop(&st);

}

}

else

{

while(!empty(&st)&& icp<=isp(peek(&st)))

{

topsym=pop(&st);

postfix[j]=topsym;

j++;

}

push(&st,sym);

}

}

while(!empty(&st))

{

topsym=pop(&st);

postfix[j]=topsym;

j++;

}

postfix[j]='\0';

printf("\nAPostFix expression is %s",postfix);

}

void push(struct stack \*st,char a)

{

if(st->top==MAX-1)

{

printf("\nstack is overflow");

}

else

{

st->top++;

st->items[st->top]=a;

}

}

char pop(struct stack \*st)

{

char a;

if(empty(st))

return -1;

else

{

a=st->items[st->top];

st->top--;

return a;

}

}

char peek(struct stack \*st)

{

if(empty(&st))

return -1;

else

return st->items[st->top];

}

* INFIX TO PREFIX

/\*Infix to postfix\*/

#define MAX 50

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

struct stack

{

int top;

char items[MAX];

};

void push(struct stack \*,char);

char pop(struct stack \*);

char peek(struct stack \*);

void infixtopost(char[]);

int isOperand(char);

int empty(struct stack \*);

int icp(char);

int isp(char);

struct stack st;

char postfix[MAX];

char prefix[MAX];

int main()

{

char infix[MAX];

char infix1[MAX];

int i;

char sym;

st.top=-1;

printf("Enter infix expressoin");

scanf("%s",infix);

printf("Infix Expressions is %s",infix);

for(i=0;(sym=infix[i])!='\0';i++)

{

if(sym=='(')

{

push(&st,')');

}

if(sym==')')

{

push(&st,'(');

}

else

{

push(&st,sym);

}

}

i=0;

while (!empty(&st))

{

sym=pop(&st);

infix1[i]=sym;

i++;

}

infix1[i]='\0';

printf("Infix Expressions after reverse %s",infix1);

infixtopost(infix1);

return 0;

}

int empty(struct stack \*st)

{

if(st->top==-1)

{

return 1;

}

else

{

return 0;

}

}

int isOperand(char c)

{

if((c>='a'&&c<='z')||(c>='A'&&c<='Z')||(c>='0'&&c<='9'))

{

return 1;

}

else

{

return 0;

}

}

int isp(char c)

{

switch(c)

{

case '(': return 0;

case '+':

case '-': return 1;

case '\*':

case '/': return 2;

case '^':

case '$': return 3;

}

}

int icp(char c)

{

switch(c)

{

case '+':

case '-': return 1;

case '\*':

case '/': return 2;

case '^':

case '$': return 3;

}

}

void infixtopost(char infix[])

{

int i,j=0;

char sym,topsym;

for(i=0;(sym=infix[i])!='\0';i++)

{

if(isOperand(sym))

{

postfix[j]=sym;

j++;

}

else if(sym=='(')

{

push(&st,sym);

}

else if(sym==')')

{

topsym = pop(&st);

while (topsym!='(' && !empty(&st))

{

postfix[j]=topsym;

j++;

topsym =pop(&st);

}

}

else

{

while(!empty(&st)&& icp<=isp(peek(&st)))

{

topsym=pop(&st);

postfix[j]=topsym;

j++;

}

push(&st,sym);

}

}

while(!empty(&st))

{

topsym=pop(&st);

postfix[j]=topsym;

j++;

}

postfix[j]='\0';

printf("\nAPostFix expression is %s",postfix);

for(i=0;(sym=postfix[i])!='\0';i++)

{

push(&st,sym);

}

i=0;

while(!empty(&st))

{

topsym = pop(&st);

prefix[i]=topsym;

i++;

}

prefix[i]='\0';

printf("\nAPreFix expression is %s",prefix);

}

void push(struct stack \*st,char a)

{

if(st->top==MAX-1)

{

printf("\nstack is overflow");

}

else

{

st->top++;

st->items[st->top]=a;

}

}

char pop(struct stack \*st)

{

char a;

if(empty(st))

return -1;

else

{

a=st->items[st->top];

st->top--;

return a;

}

}

char peek(struct stack \*st)

{

if(empty(&st))

return -1;

else

return st->items[st->top];

}

* POST FIX EVAL

#define MAX 50

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

struct stack

{

int top;

double items[MAX];

};

void push(struct stack \*,double);

double pop(struct stack \*);

int empty(struct stack \*);

int isOperand(char);

double operate(char,double,double);

struct stack st;

int main()

{

char exp[MAX];

st.top=-1;

int i;

char sym;

double result;

double opnd1,opnd2,value;

printf("Enter expressoin");

scanf("%s",exp);

printf("\n Expressions is %s",exp);

for(i=0;(sym=exp[i])!='\0';i++)

{

if(isOperand(sym))

{

push(&st,(double)sym-'0');

}

else

{

opnd2=pop(&st);

opnd1=pop(&st);

value=operate(sym,opnd1,opnd2);

push(&st,value);

}

}

result=pop(&st);

printf("Result is %lf",result);

}

double operate(char sym,double op1,double op2)

{

switch(sym)

{

case '+': return (op1+op2);

case '-': return (op1-op2);

case '\*': return (op1\*op2);

case '/': return (op1/op2);

case '$': return (pow(op1,op2));

default : printf("Invalid");

exit(0);

}

}

int isOperand(char c)

{

if(c>='0'&&c<='9')

{

return 1;

}

else

{

return 0;

}

}

int empty(struct stack \*st)

{

if(st->top==-1)

{

return 1;

}

else

{

return 0;

}

}

void push(struct stack \*st,double a)

{

if(st->top==MAX-1)

{

printf("\nstack is overflow");

}

else

{

st->top++;

st->items[st->top]=a;

}

}

double pop(struct stack \*st)

{

double a;

if(empty(st))

return -1;

else

{

a=st->items[st->top];

st->top--;

return a;

}

}