Infix expression to postfix expression conversion

/\* This program converts infix expression to postfix expression.

 \* This program assume that there are Five operators: (\*, /, +, -,^)

in infix expression and operands can be of single-digit only.

 \* This program will not work for fractional numbers.

 \* Further this program does not check whether infix expression is

 valid or not in terms of number of operators and operands.\*/

#include<stdio.h>

#include<stdlib.h> /\* for exit() \*/

#include<ctype.h> /\* for isdigit(char ) \*/

#include<string.h>

#define SIZE 100

/\* declared here as global variable because stack[]

\* is used by more than one fucntions \*/

char stack[SIZE];

int top = -1;

/\* define push operation \*/

void push(char item)

{

if(top >= SIZE-1)

{

printf("\nStack Overflow.");

}

else

{

top = top+1;

stack[top] = item;

}

}

/\* define pop operation \*/

char pop()

{

char item ;

if(top <0)

{

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

getchar();

/\* underflow may occur for invalid expression \*/

/\* where ( and ) are not matched \*/

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

/\* define function that is used to determine whether any symbol is operator or not

(that is symbol is operand)

\* this fucntion returns 1 if symbol is opreator else return 0 \*/

int is\_operator(char symbol)

{

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

{

return 1;

}

else

{

return 0;

}

}

/\* define fucntion that is used to assign precendence to operator.

\* Here ^ denotes exponent operator.

\* In this fucntion we assume that higher integer value

\* means higher precendence \*/

int precedence(char symbol)

{

if(symbol == '^')/\* exponent operator, highest precedence\*/

{

return(3);

}

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

{

return(2);

}

else if(symbol == '+' || symbol == '-') /\* lowest precedence \*/

{

return(1);

}

else

{

return(0);

}

}

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

{

int i, j;

char item;

char x;

push('('); /\* push '(' onto stack \*/

strcat(infix\_exp,")"); /\* add ')' to infix expression \*/

i=0;

j=0;

item=infix\_exp[i]; /\* initialize before loop\*/

while(item != '\0') /\* run loop till end of infix expression \*/

{

if(item == '(')

{

push(item);

}

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

{

postfix\_exp[j] = item; /\* add operand symbol to postfix expr \*/

j++;

}

else if(is\_operator(item) == 1) /\* means symbol is operator \*/

{

x=pop();

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

{

postfix\_exp[j] = x; /\* so pop all higher precendence operator and \*/

j++;

x = pop(); /\* add them to postfix expresion \*/

}

push(x);

/\* because just above while loop will terminate we have

oppped one extra item

for which condition fails and loop terminates, so that one\*/

push(item); /\* push current oprerator symbol onto stack \*/

}

else if(item == ')') /\* if current symbol is ')' then \*/

{

x = pop(); /\* pop and keep popping until \*/

while(x != '(') /\* '(' encounterd \*/

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

else

{ /\* if current symbol is neither operand not '(' nor ')' and nor

operator \*/

printf("\nInvalid infix Expression.\n"); /\* the it is illegeal symbol \*/

getchar();

exit(1);

}

i++;

item = infix\_exp[i]; /\* go to next symbol of infix expression \*/

} /\* while loop ends here \*/

if(top>0)

{

printf("\nInvalid infix Expression.\n"); /\* the it is illegeal symbol \*/

getchar();

exit(1);

}

postfix\_exp[j] = '\0'; /\* add sentinel else puts() fucntion \*/

/\* will print entire postfix[] array upto SIZE \*/

}

/\* main function begins \*/

int main()

{

char infix[SIZE], postfix[SIZE]; /\* declare infix string and postfix string \*/

/\* why we asked the user to enter infix expression

\* in parentheses ( )

\* What changes are required in porgram to

\* get rid of this restriction since it is not

\* in algorithm

\* \*/

printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");

printf("\nEnter Infix expression : ");

gets(infix);

InfixToPostfix(infix, postfix); /\* call to convert \*/

printf("Postfix Expression: ");

puts(postfix); /\* print postfix expression \*/

return 0;

}

**Evaluation of postfix expression**

/\* This program is for evaluation of postfix expression

\* This program assume that there are only four operators

\* (\*, /, +, -) in an expression and operand is single digit only

\* Further this program does not do any error handling e.g.

\* it does not check that entered postfix expression is valid

\* or not.

\* \*/

#include <stdio.h>

#include <ctype.h>

#define MAXSTACK 100 /\* for max size of stack \*/

#define POSTFIXSIZE 100 /\* define max number of charcters in postfix expression \*/

/\* declare stack and its top pointer to be used during postfix expression

evaluation\*/

int stack[MAXSTACK];

int top = -1; /\* because array index in C begins at 0 \*/

/\* can be do this initialization somewhere else \*/

/\* define push operation \*/

void push(int item)

{

if (top >= MAXSTACK - 1) {

printf("stack over flow");

return;

}

else {

top = top + 1;

stack[top] = item;

}

}

/\* define pop operation \*/

int pop()

{

int item;

if (top < 0) {

printf("stack under flow");

}

else {

item = stack[top];

top = top - 1;

return item;

}

}

/\* define function that is used to input postfix expression and to evaluate it \*/

void EvalPostfix(char postfix[])

{

int i;

char ch;

int val;

int A, B;

/\* evaluate postfix expression \*/

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

ch = postfix[i];

if (isdigit(ch)) {

/\* we saw an operand,push the digit onto stack

ch - '0' is used for getting digit rather than ASCII code of digit \*/

push(ch - '0');

}

else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/') {

/\* we saw an operator

\* pop top element A and next-to-top elemnet B

\* from stack and compute B operator A

\*/

A = pop();

B = pop();

switch (ch) /\* ch is an operator \*/

{

case '\*':

val = B \* A;

break;

case '/':

val = B / A;

break;

case '+':

val = B + A;

break;

case '-':

val = B - A;

break;

}

/\* push the value obtained above onto the stack \*/

push(val);

}

}

printf(" \n Result of expression evaluation : %d \n", pop());

}

int main()

{

int i;

/\* declare character array to store postfix expression \*/

char postfix[POSTFIXSIZE];

printf("ASSUMPTION: There are only four operators(\*, /, +, -) in an expression and operand is single digit only.\n");

printf(" \nEnter postfix expression,\npress right parenthesis ')' for end expression : ");

/\* take input of postfix expression from user \*/

for (i = 0; i <= POSTFIXSIZE - 1; i++) {

scanf("%c", &postfix[i]);

if (postfix[i] == ')') /\* is there any way to eliminate this if \*/

{

break;

} /\* and break statement \*/

}

/\* call function to evaluate postfix expression \*/

EvalPostfix(postfix);

return 0;

}

References:

1. https://www.includehelp.com/c/evaluation-of-postfix-expressions-using-stack-with-c-program.aspx