Assignment Number

Problem Statement

Program in C to convert an infix algebraic expression to the postfix form.

Theory

The problem of converting an infix algebraic expression to its postfix form (or to the Reverse Polish notation) is very old and can be dated back to 1960s. This type of expression has the most efficient use in stack based computing systems, where by following some simple rules, an expression in postfix form can easily be evaluated. The algorithm used in the conversion here is famously known as the shunting-yard algorithm, invented by Edsger Dijkstra. This algorithm works by making use of an auxiliary stack to keep track of operands at hand, adding them to the resulting expression whenever applicable.

Algorithm

Input : An arithmetic expression, balanced in terms of paranthesis, say E
Output : The postfix equivalent of E
Data Structure Used : A stack, used as an auxiliary data structure for the conversion.
Steps :

```
Step 1 : While( E is not empty )
    a) Set token = E.scan_symbol()  // scan_symbol is a procedure that
                                    // scans an item from the expression from left to right
    b) If(token = OPERAND)
```

```
        Then
        i.  Print token
        EndIf
    c)  Else If( token = '(' )
        Then
        i.  Push(token)  // Push is a procedure that pushes the argument
                          // element to the top of the stack
        EndIf
    d)  Else If( token = ')' )
        Then
        i.  While( (token = Pop()) is not '(')  // Pop is a procedure that
                                    // retrieves and removes the element at the
                                    // top of the stack
            1.  Print token
            EndWhile
        EndIf
    e)  Else
        i.  While( priority(token) <= priority(Peek()) And stack is not empty)
            // Peek() is an operation that retrives, but does not remove,
            // the element at the top of the stack
            // priority() is an operation that returns the relative priority
            // of the given operator in algebraic sense
            1.  Set x = Pop()
            2.  Print x
            EndWhile
        ii.  Push(token)
        EndIf
Step 2 : While( stack is not empty )
        i.  Set x = Pop()
        ii.  Print x
        EndWhile
```

Source Code

```
#include <ctype.h>
#include <stdio.h>
#define MAX 100

typedef struct stack {
    int data[MAX];
    int top;
} stack;

int priority(char);
void init(stack *);
int empty(stack *);
int full(stack *);
char pop(stack *);
void push(stack *, char);
char top(stack *);

int main() {
    stack s;
    char x;
    int token;
    init(&s);
    printf("\nEnter infix expression : ");
    while ((token = getchar()) != '\n') {
        if (isalnum(token))
            printf(" %c ", token);
        else if (token == '(')
            push(&s, '(');
        else {
            if (token == ')')
                while ((x = pop(&s)) != '(')
                    printf(" %c ", x);
            else {
                while (priority(token) <= priority(top(&s)) && !empty(&s)) {
                    x = pop(&s);
                    printf(" %c ", x);
                }
                push(&s, token);
            }
        }
    }
    while (!empty(&s)) {
        x = pop(&s);
        printf(" %c ", x);
    }
}
//--------------------------------------------
int priority(char x) {
    if (x == '(')
        return (0);
    if (x == '+' || x == '-')
        return (1);
    if (x == '*' || x == '/' || x == '%')
        return (2);
    return (3);
}
//--------------------------------------------
void init(stack *s) {
    s->top = -1;
}
//--------------------------------------------
int empty(stack *s) {
    if (s->top == -1)
        return (1);
    else
        return (0);
}
//--------------------------------------------
```

```
int full(stack *s) {
    if (s->top == MAX - 1)
        return (1);
    else
        return (0);
}
//---------------------------------------------
void push(stack *s, char x) {
    s->top = s->top + 1;
    s->data[s->top] = x;
}
//---------------------------------------------
char pop(stack *s) {
    int x;
    x = s->data[s->top];
    s->top = s->top - 1;
    return (x);
}
//---------------------------------------------
char top(stack *s) {
    return (s->data[s->top]);
}
//---------------------------------------------
```

Input and Output

Set 1 :

```
Enter infix expression : a+b/2*(c-d)/e
 a  b  2  /  c  d  -  *  e  /  +
```

Set 2 :

```
Enter infix expression : b+2*(3-2)/a
 b  2  3  2  -  *  a  /  +
```

Set 3 :

```
Enter infix expression : a%b*e/d*f
 a  b  %  e  *  d  /  f  *
```

Discussion

1. This program does not support multicharacter variables and constants.
2. Due to the implementation, it also does not support floating point numbers.
3. No check is performed to justify the validity of the given expression, the user is trusted to provide correctly formatted expression every time. However, that might not always be the case.