```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct stack
{
    int size;
    int top;
    char *arr;
};

int stackTop(struct stack* sp){
    return sp->arr[sp->top];
}

int isEmpty(struct stack *ptr)
{
    if (ptr->top == -1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int isFull(struct stack *ptr)
{
    if (ptr->top == ptr->size - 1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

```

```
40: void push(struct stack* ptr, char val){
41:     if(isFull(ptr)){
42:         printf("Stack Overflow! Cannot push %d to the stack\n", va
43:     }
44:     else{
45:         ptr->top++;
46:         ptr->arr[ptr->top] = val;
47:     }
48: }
49:
50: char pop(struct stack* ptr){
51:     if(isEmpty(ptr)){
52:         printf("Stack Underflow! Cannot pop from the stack\n");
53:         return -1;
54:     }
55:     else{
56:         char val = ptr->arr[ptr->top];
57:         ptr->top--;
58:         return val;
59:     }
60: }
61: int precedence(char ch){
62:     if(ch == '*' || ch=='/')
63:         return 3;
64:     else if(ch == '+' || ch=='-')
65:         return 2;
66:     else
67:         return 0;
68: }
69:
70: int isOperator(char ch){
71:     if(ch=='+' || ch=='-' ||ch=='*' || ch=='/')
72:         return 1;
73:     else
74:         return 0;
75: }
76: char* infixToPostfix(char* infix){
77:     struct stack * sp = (struct stack *) malloc(sizeof(struct stac
78:     sp->size = 10;
```

```
 79:     sp->top = -1;
 80:     sp->arr = (char *) malloc(sp->size * sizeof(char));
 81:     char * postfix = (char *) malloc((strlen(infix)+1) * sizeof(ch
 82:     int i=0; // Track infix traversal
 83:     int j = 0; // Track postfix addition
 84:     while (infix[i]!='\0')
 85:     {
 86:         if(!isOperator(infix[i])){
 87:             postfix[j] = infix[i];
 88:             j++;
 89:             i++;
 90:         }
 91:         else{
 92:             if(precedence(infix[i])> precedence(stackTop(sp))){
 93:                 push(sp, infix[i]);
 94:                 i++;
 95:             }
 96:             else{
 97:                 postfix[j] = pop(sp);
 98:                 j++;
 99:             }
100:         }
101:     }
102:     while (!isEmpty(sp))
103:     {
104:         postfix[j] = pop(sp);
105:         j++;
106:     }
107:     postfix[j] = '\0';
108:     return postfix;
109: }
110: int main()
111: {
112:     char * infix = "x-y/z-k*d";
113:     printf("postfix is %s", infixToPostfix(infix));
114:     return 0;
115: }
116:
```