a) Evaluate a given postfix expression using stack.

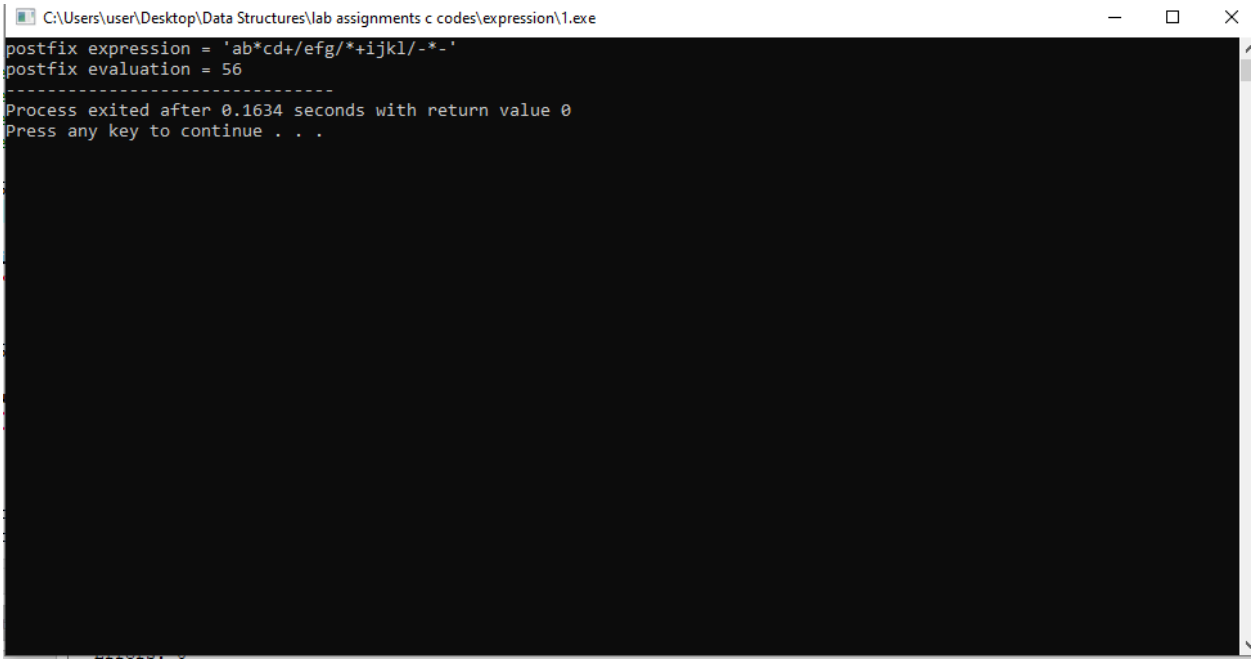
Source Code:

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

struct Stack
{
	int top;
	unsigned cap;
	int* arr;
};

struct Stack* create( unsigned cap )
{
	struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));
	if (!stack)
	{
		return NULL;
	}
	stack->top = -1;
	stack->cap = cap;
	stack->arr = (int*) malloc(stack->cap * sizeof(int));
	if (!stack->arr)
	{
		return NULL;
	}
	return stack;
}

int isEmpty(struct Stack* stack)
```

```
{
	return stack->top == -1 ;
}

char peek(struct Stack* stack)
{
	return stack->arr[stack->top];
}

char pop(struct Stack* stack)
{
	if (!isEmpty(stack))
	{
		return stack->arr[stack->top--] ;
	}
	return '$';
}

void push(struct Stack* stack, char op)
{
	stack->arr[++stack->top] = op;
}

int evaluate(char* exp)
{
	struct Stack* stack = create(strlen(exp));
	int i;
	if (!stack)
	{
		return -1;
	}
	for (i = 0; exp[i]; ++i)
	{
		if (isdigit(exp[i]))
```

```
            {
                push(stack, exp[i] - '0');
            }
            else
            {
                int val1 = pop(stack);
                int val2 = pop(stack);
                switch (exp[i])
                {
                case '+': push(stack, val2 + val1); break;
                case '-': push(stack, val2 - val1); break;
                case '*': push(stack, val2 * val1); break;
                case '/': push(stack, val2/val1); break;
                }
            }
        }
        return pop(stack);
}

int main()
{
        char exp[] = "ab*cd+/efg/*+ijkl/-*-";
        printf("postfix expression = 'ab*cd+/efg/*+ijkl/-*-'\n");
        printf ("postfix evaluation = %d", evaluate(exp));
        return 0;
}
```

Result:



b) Convert a valid infix expression into postfix notation.

Source Code:

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

struct Stack
{
	int top;
	unsigned cap;
	int* arr;
};

struct Stack* create( unsigned cap )
{
	struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));
	if (!stack)
```

```
	{
		return NULL;
	}
	stack->top = -1;
	stack->cap = cap;
	stack->arr = (int*) malloc(stack->cap * sizeof(int));
	return stack;
}

int isEmpty(struct Stack* stack)
{
	return stack->top == -1 ;
}

char peek(struct Stack* stack)
{
	return stack->arr[stack->top];
}

char pop(struct Stack* stack)
{
	if (!isEmpty(stack))
	{
		return stack->arr[stack->top--] ;
	}
	return '$';
}

void push(struct Stack* stack, char op)
{
	stack->arr[++stack->top] = op;
}
```

```
int isOperand(char ch)
{
	return (ch >= 'a' && ch <= 'z') ||
		(ch >= 'A' && ch <= 'Z');
}

int characters(char ch)
{
	switch (ch)
	{
	case '+':
	case '-':
		return 1;

	case '*':
	case '/':
		return 2;

	case '^':
		return 3;
	}
	return -1;
}

int convert(char* exp)
{
	int i, k;
	struct Stack* stack = create(strlen(exp));
	if(!stack)
	{
		return -1 ;
	}
	for (i = 0, k = -1; exp[i]; ++i)
```

```
        {
                if (isOperand(exp[i]))
                {
                        exp[++k] = exp[i];
                }
                else if (exp[i] == '(')
                {
                        push(stack, exp[i]);
                }
                else if (exp[i] == ')')
                {
                        while (!isEmpty(stack) && peek(stack) != '(')
                                exp[++k] = pop(stack);
                        if (!isEmpty(stack) && peek(stack) != '(')
                                return -1;
                        else
                                pop(stack);
                }
                else
                {
                        while (!isEmpty(stack) &&
                                characters(exp[i]) <= characters(peek(stack)))
                                exp[++k] = pop(stack);
                        push(stack, exp[i]);
                }
        }
        while (!isEmpty(stack))
                exp[++k] = pop(stack );

        exp[++k] = '\0';
        printf( "%s", exp );
}
```

```
int main()
{
        char exp[] = "a*b/(c+d)+e*(f/g)-i*(j-k/l)";
        convert(exp);
        return 0;
}
```

Result: