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

struct Stack
{
    int top;
    unsigned capacity;
    char *array;
};

struct Stack *createStack(unsigned capacity)
{
    struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (char *)malloc(stack->capacity * sizeof(char));
    return stack;
}

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

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

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

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

int isOperand(char ch)
```

```c
{
    return (ch >= '0' && ch <= '9') || (ch >= 'a' && ch <= 'z') || (ch >= 'A' &&
ch <= 'Z');
}

int precedence(char op)
{
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return 0;
}

void infixToPostfix(char *infixExpression, char *postfixExpression)
{
    struct Stack *stack = createStack(strlen(infixExpression) + 1);
    int i, k;

    for (i = 0, k = -1; infixExpression[i]; ++i)
    {
        if (isOperand(infixExpression[i]))
            postfixExpression[++k] = infixExpression[i];
        else if (infixExpression[i] == '(')
            push(stack, infixExpression[i]);
        else if (infixExpression[i] == ')')
        {
            while (!isEmpty(stack) && peek(stack) != '(')
                postfixExpression[++k] = pop(stack);
            if (!isEmpty(stack) && peek(stack) != '(')
            {
                printf("Invalid infix expression\n");
                return;
            }
            else
                pop(stack);
        }
        else
        {
            while (!isEmpty(stack) && precedence(infixExpression[i]) <=
precedence(peek(stack)))
                postfixExpression[++k] = pop(stack);
            push(stack, infixExpression[i]);
        }
    }
    while (!isEmpty(stack))
        postfixExpression[++k] = pop(stack);

    postfixExpression[++k] = '\0';
}

int main()
{
    char infixExpression[100];
    char postfixExpression[100];
    int choice;

    do
    {
        printf("\nMenu:\n");
        printf("1. Convert infix expression to postfix\n");
        printf("2. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("Enter an infix expression: ");
            scanf("%s", infixExpression);
            infixToPostfix(infixExpression, postfixExpression);
            printf("Postfix expression: %s\n", postfixExpression);
            break;
        case 2:
            printf("Exiting the program.\n");
            break;
        default:
            printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 2);

    return 0;
}
```

Output-

```
Menu:

1. Convert infix expression to postfix

2. Exit

Enter your choice: 1

Enter an infix expression: A+B/C*D

Postfix expression: ABC/D*+
```