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

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

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

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

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

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

int pop(struct Stack* stack) {
    if (isEmpty(stack)) return -1;
    return stack->array[stack->top--];
}
```

```c
int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return -1;
}

char* infixToPostfix(char* expression) {
    struct Stack* stack = createStack(strlen(expression));
    int i, k;
    for (i = 0, k = -1; expression[i]; ++i) {
        if (isdigit(expression[i])) {
            expression[++k] = expression[i];
        } else if (expression[i] == '(') {
            push(stack, expression[i]);
        } else if (expression[i] == ')') {
            while (!isEmpty(stack) && stack->array[stack->top] != '(') {
                expression[++k] = pop(stack);
            }
            pop(stack);
        } else {
            while (!isEmpty(stack) && precedence(expression[i]) <= precedence(stack->array[stack->top])) {
                expression[++k] = pop(stack);
            }
            push(stack, expression[i]);
        }
    }
```

```c
    while (!isEmpty(stack)) {
        expression[++k] = pop(stack);
    }
    expression[++k] = '\0';
    return expression;
}

void displayPostfix(char* expression) {
    printf("Postfix expression: %s\n", expression);
}

int main() {
    char expression[100];
    printf("Enter the infix expression: ");
    fgets(expression, sizeof(expression), stdin);
    printf("\n");

    char* postfixExpression = infixToPostfix(expression);
    displayPostfix(postfixExpression);
    return 0;
}
```

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

Postfix expression: (/ab((-+cd*e
```