1. Stack operation

```
#include <stdio.h>
#define MAX_SIZE 5

int stack[MAX_SIZE];
int top = -1;

int is_empty() {
    return top == -1;
}

int is_full() {
    return top == MAX_SIZE - 1;
}

void push(int item) {
    if (is_full()) {
        printf("Stack is full. Cannot push item.\n");
    } else {
        top++;
        stack[top] = item;
        printf("Pushed %d to the stack.\n", item);
    }
}

void pop() {
    if (is_empty()) {
        printf("Stack is empty. Cannot pop item.\n");
    } else {
        int item = stack[top];
        top--;
        printf("Popped item: %d\n", item);
    }
}

void peek() {
    if (is_empty()) {
        printf("Stack is empty.\n");
    } else {
        printf("Top item: %d\n", stack[top]);
    }
}

void display() {
    if (is_empty()) {
        printf("Stack is empty.\n");
    } else {
```

```
        printf("Stack contents:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d\n", stack[i]);
        }
    }
}

int main() {
    push(10);
    push(20);
    push(30);
    push(40);
    push(50);
    push(60);  // This will give a "Stack is full" message

    display();  // Prints the stack contents: 50 40 30 20 10

    pop();  // Pops the top item: 50
    peek();  // Prints the top item: 40

    display();  // Prints the stack contents: 40 30 20 10

    pop();  // Pops the top item: 40
    pop();  // Pops the top item: 30

    display();  // Prints the stack contents: 20 10

    return 0;
}
```

2. Infix to postfix expression

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

#define MAX_SIZE 100

// Stack operations
char stack[MAX_SIZE];
int top = -1;

void push(char item) {
    if (top >= MAX_SIZE - 1) {
        printf("Stack Overflow\n");
    } else {
        stack[++top] = item;
```

```
    }
}

char pop() {
    if (top < 0) {
        printf("Stack Underflow\n");
        return '\0';
    } else {
        return stack[top--];
    }
}

int isOperator(char symbol) {
    if (symbol == '+' || symbol == '-' || symbol == '*' || symbol == '/')
        return 1;
    return 0;
}

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

void infixToPostfix(char infix[], char postfix[]) {
    int i, j;
    char symbol;
    char next;

    push('(');
    strcat(infix, ")");

    i = 0;
    j = 0;
    symbol = infix[i];

    while (symbol != '\0') {
        if (symbol == '(') {
            push(symbol);
        } else if (isdigit(symbol) || isalpha(symbol)) {
            postfix[j++] = symbol;
        } else if (isOperator(symbol) == 1) {
            next = pop();
            while (isOperator(next) == 1 && precedence(next) >= precedence(symbol)) {
                postfix[j++] = next;
                next = pop();
```

```
            }
            push(next);
            push(symbol);
        } else if (symbol == ')') {
            next = pop();
            while (next != '(') {
                postfix[j++] = next;
                next = pop();
            }
        } else {
            printf("Invalid infix expression\n");
            return;
        }
        i++;
        symbol = infix[i];
    }
    if (top > 0) {
        printf("Invalid infix expression\n");
        return;
    }
    postfix[j] = '\0';
}

int main() {
    char infix[MAX_SIZE], postfix[MAX_SIZE];

    printf("Enter infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```

3. Evaluation of postfix expression

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

#define MAX_SIZE 100

// Stack operations
float stack[MAX_SIZE];
```

```
int top = -1;

void push(float item) {
    if (top >= MAX_SIZE - 1) {
        printf("Stack Overflow\n");
    } else {
        stack[++top] = item;
    }
}

float pop() {
    if (top < 0) {
        printf("Stack Underflow\n");
        return 0;
    } else {
        return stack[top--];
    }
}

float evaluatePostfix(char postfix[]) {
    int i;
    char symbol;
    float operand1, operand2, result;

    i = 0;
    symbol = postfix[i];

    while (symbol != '\0') {
        if (isdigit(symbol)) {
            push(symbol - '0');
        } else {
            operand2 = pop();
            operand1 = pop();

            switch (symbol) {
                case '+':
                    result = operand1 + operand2;
                    break;
                case '-':
                    result = operand1 - operand2;
                    break;
                case '*':
                    result = operand1 * operand2;
                    break;
                case '/':
                    result = operand1 / operand2;
                    break;
                case '^':
```

```
                result = pow(operand1, operand2);
                break;
            default:
                printf("Invalid postfix expression\n");
                return 0;
        }
        push(result);
    }
    i++;
    symbol = postfix[i];
  }
  return pop();
}

int main() {
  char postfix[MAX_SIZE];
  float result;

  printf("Enter postfix expression: ");
  scanf("%s", postfix);

  result = evaluatePostfix(postfix);
  printf("Result: %.2f\n", result);

  return 0;
}
```