1) C program to implement infix, prefix and postfix notations for arithmetic expressions using stack

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
typedef struct {
   int top;
   char items[MAX];
} Stack;
void push(Stack* s, char c)
{
   if (s->top == (MAX - 1))
   {
      printf("Stack overflow\n");
   }
   else
   {
      s->items[++(s->top)] = c;
   }
}
char pop(Stack* s)
{
   if (s->top == -1)
   {
      printf("Stack underflow\n");
      return '\0';
   }
   else
   {
```

```
        return s->items[(s->top)--];
    }
}
char peek(Stack* s)
{
    if (s->top == -1)
    {
        printf("Stack is empty\n");
        return '\0';
    }
    else
    {
        return s->items[s->top];
    }
}
int precedence(char op)
{
    switch (op) {
        case '+':
        case '-': return 1;
        case '*':
        case '/': return 2;
        case '^': return 3;
        default: return 0;
    }
}
void infixToPostfix(const char* infix, char* postfix) {
    Stack s;
    s.top = -1;
    int k = 0;
    for (int i = 0; infix[i]; i++)
```

```
    {
        if (isalpha(infix[i])) {
            postfix[k++] = infix[i];
        }
        else if (infix[i] == '(')
        {
            push(&s, infix[i]);
        }
        else if (infix[i] == ')')
        {
            while (s.top != -1 && peek(&s) != '(')
            {
                postfix[k++] = pop(&s);
            }
            pop(&s);
        }
        else
        {
            while (s.top != -1 && precedence(peek(&s)) >= precedence(infix[i])) {
                postfix[k++] = pop(&s);
            }
            push(&s, infix[i]);
        }
    }
    while (s.top != -1) {
        postfix[k++] = pop(&s);
    }
    postfix[k] = '\0';
}
void reverse(char* str)
{
```

```
    int length = strlen(str);
    for (int i = 0; i < length / 2; i++)
    {
        char temp = str[i];
        str[i] = str[length - i - 1];
        str[length - i - 1] = temp;
    }
}
void infixToPrefix(const char* infix, char* prefix) {
    char infixReversed[MAX], postfixReversed[MAX];
    strcpy(infixReversed, infix);
    reverse(infixReversed);
    for (int i = 0; infixReversed[i]; i++)
    {
        if (infixReversed[i] == '(')
        {
            infixReversed[i] = ')';
        }
        else if (infixReversed[i] == ')')
        {
            infixReversed[i] = '(';
        }
    }
    infixToPostfix(infixReversed, postfixReversed);
    reverse(postfixReversed);
    strcpy(prefix, postfixReversed);
}
int evaluatePostfix(const char* postfix)
{
    Stack s;
    s.top = -1;
```

```
    for (int i = 0; postfix[i]; i++) {
        if (isdigit(postfix[i])) {
            push(&s, postfix[i] - '0'); // Convert char digit to int
        }
        else
        {
            int val2 = pop(&s);
            int val1 = pop(&s);
            switch (postfix[i]) {
                case '+': push(&s, val1 + val2); break;
                case '-': push(&s, val1 - val2); break;
                case '*': push(&s, val1 * val2); break;
                case '/': push(&s, val1 / val2); break;
            }
        }
    }
    return pop(&s);
}
int main()
{
    char infix[MAX] = "A+B*(C^D-E)^(F+G*H)-I";
    char postfix[MAX];
    char prefix[MAX];
    printf("Infix Expression: %s\n", infix);
    infixToPostfix(infix, postfix);
    printf("Postfix Expression: %s\n", postfix);
    infixToPrefix(infix, prefix);
    printf("Prefix Expression: %s\n", prefix);
    return 0;
}
```

OUT PUT:

```
/tmp/ZyY2LKeOA8.o
Infix Expression: A+B*(C^D-E)^(F+G*H)-I
Postfix Expression: ABCD^E-FGH*+^*+I-
Prefix Expression: +A-*B^-^CDE+F*GHI


=== Code Execution Successful ===
```

2)C program to check if the parentheses in an expression are balanced using a stack. Extend the program to handle multiple types of parentheses (e.g., {}, [], ()).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // Include for strlen function

#define MAX 100

// Stack structure
typedef struct {
    int top;
    char items[MAX];
} Stack;

// Stack functions
void push(Stack* s, char c) {
    if (s->top == (MAX - 1)) {
        printf("Stack overflow\n");
```

```
    } else {
        s->items[++(s->top)] = c;
    }
}

char pop(Stack* s) {
    if (s->top == -1) {
        printf("Stack underflow\n");
        return '\0';
    } else {
        return s->items[(s->top)--];
    }
}

char peek(Stack* s) {
    if (s->top == -1) {
        return '\0';
    } else {
        return s->items[s->top];
    }
}

// Function to check if parentheses are balanced
int isBalanced(const char* expr) {
    Stack s;
    s.top = -1;

    for (int i = 0; expr[i]; i++) {
        char ch = expr[i];
        switch (ch) {
            case '(':
```

```
            case '{':
            case '[':
                push(&s, ch);
                break;
            case ')':
                if (peek(&s) == '(') {
                    pop(&s);
                } else {
                    return 0; // Unbalanced
                }
                break;
            case '}':
                if (peek(&s) == '{') {
                    pop(&s);
                } else {
                    return 0; // Unbalanced
                }
                break;
            case ']':
                if (peek(&s) == '[') {
                    pop(&s);
                } else {
                    return 0; // Unbalanced
                }
                break;
        }
    }
    return s.top == -1; // Stack should be empty if balanced
}

int main() {
```

```
    char expr[MAX];

    printf("Enter an expression with parentheses: ");
    fgets(expr, sizeof(expr), stdin);

    // Remove newline character if present
    size_t length = strlen(expr);
    if (length > 0 && expr[length - 1] == '\n') {
        expr[length - 1] = '\0';
    }

    if (isBalanced(expr)) {
        printf("The expression has balanced parentheses.\n");
    } else {
        printf("The expression has unbalanced parentheses.\n");
    }

    return 0;
}
```

OUT PUT:

```
/tmp/4H8Nt4qOmK.o
Enter an expression with parentheses: (a+b)*{+[/(e-f)]}
The expression has balanced parentheses.


=== Code Execution Successful ===
```

3)a program to evaluate a postfix expression using a stack. The program should handle basic arithmetic operators (+, -, *, /).

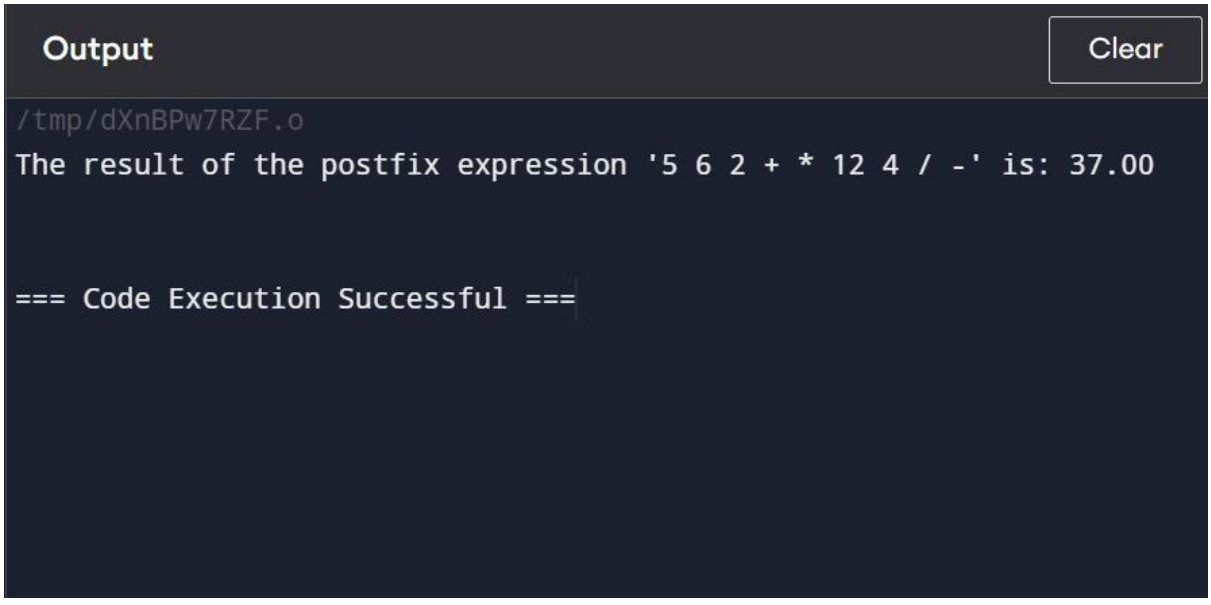
```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define MAX 100 // Maximum size of the stack

// Stack structure
typedef struct {
    float items[MAX];
    int top;
} Stack;

// Function to initialize the stack
void initStack(Stack *s) {
    s->top = -1;
}

// Function to check if the stack is empty
int isEmpty(Stack *s) {
    return s->top == -1;
}

// Function to push an item onto the stack
void push(Stack *s, float item) {
    if (s->top < MAX - 1) {
        s->items[++(s->top)] = item;
    } else {
        printf("Stack overflow\n");
        exit(EXIT_FAILURE);
    }
```

```
}

// Function to pop an item from the stack
float pop(Stack *s) {
    if (!isEmpty(s)) {
        return s->items[(s->top)--];
    } else {
        printf("Stack underflow\n");
        exit(EXIT_FAILURE);
    }
}

// Function to evaluate a postfix expression
float evaluatePostfix(const char *expression) {
    Stack stack;
    initStack(&stack);

    const char *p = expression;
    char buffer[20];

    while (*p) {
        // Skip whitespace
        if (isspace(*p)) {
            p++;
            continue;
        }

        // If the character is a digit or part of a number
        if (isdigit(p) || (*p == '-' && isdigit((p + 1)))) {
            // Read the number
            sscanf(p, "%s", buffer);
```

```
            push(&stack, atof(buffer));
            p += strlen(buffer); // Move the pointer forward
        } else {
            // It's an operator
            float b = pop(&stack);
            float a = pop(&stack);
            float result;

            switch (*p) {
                case '+':
                    result = a + b;
                    break;
                case '-':
                    result = a - b;
                    break;
                case '*':
                    result = a * b;
                    break;
                case '/':
                    if (b == 0) {
                        printf("Error: Division by zero\n");
                        exit(EXIT_FAILURE);
                    }
                    result = a / b;
                    break;
                default:
                    printf("Error: Unknown operator %c\n", *p);
                    exit(EXIT_FAILURE);
            }
            push(&stack, result);
        }
```

```
        p++; // Move to the next character
    }

    // The result will be the only item left in the stack
    return pop(&stack);
}

int main() {
    const char *expression = "5 6 2 + * 12 4 / -"; // Example postfix expression
    float result = evaluatePostfix(expression);
    printf("The result of the postfix expression '%s' is: %.2f\n", expression, result);
    return 0;
}
```

OUT PUT:

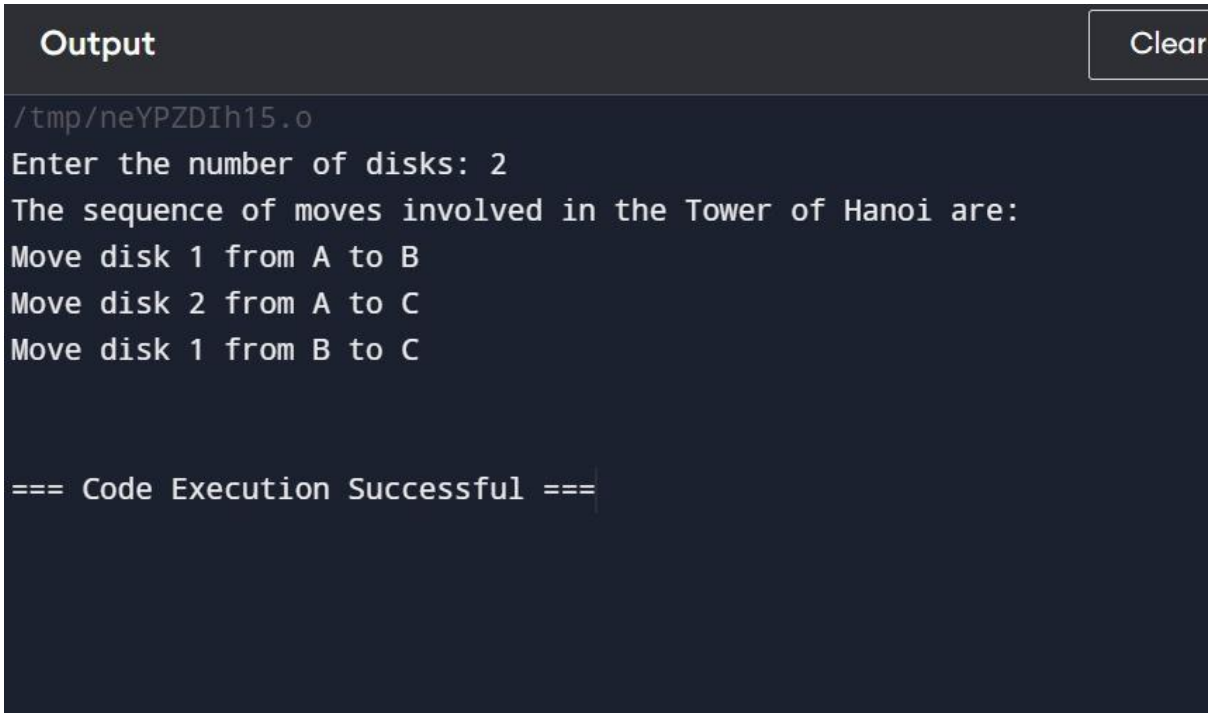


4) C program to solve the Tower of Hanoi problem using recursion.

```
#include <stdio.h>

// Function to solve the Tower of Hanoi problem
void towerOfHanoi(int n, char source, char target, char auxiliary) {
    // Base case: If there is only one disk to move
```

```
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", source, target);
        return;
    }

    // Move n-1 disks from source to auxiliary, using target as auxiliary
    towerOfHanoi(n - 1, source, auxiliary, target);

    // Move the nth disk from source to target
    printf("Move disk %d from %c to %c\n", n, source, target);

    // Move the n-1 disks from auxiliary to target, using source as auxiliary
    towerOfHanoi(n - 1, auxiliary, target, source);
}

int main() {
    int n; // Number of disks

    // Input: Number of disks
    printf("Enter the number of disks: ");
    scanf("%d", &n);

    // Solve the Tower of Hanoi problem
    printf("The sequence of moves involved in the Tower of Hanoi are:\n");
    towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods

    return 0;
}
```

OUT PUT:


Output
Clear
/tmp/neYPZDIh15.o
Enter the number of disks: 2
The sequence of moves involved in the Tower of Hanoi are:
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
=== Code Execution Successful ===