11. Write a C program to evaluate an arithmetic operation

```c
#include <stdio.h>

int main() {
    int num1, num2, result;

    printf("Enter the first number: ");
    scanf("%d", &num1);

    printf("Enter the second number: ");
    scanf("%d", &num2);

    result = num1 + num2;
    printf("Sum: %d\n", result);

    result = num1 - num2;
    printf("Difference: %d\n", result);

    result = num1 * num2;
    printf("Product: %d\n", result);

    if (num2 != 0) {
        result = num1 / num2;
        printf("Division: %d\n", result);
    } else {
        printf("Cannot divide by zero.\n");
    }

    return 0;
}
```

```
/tmp/z8FS5CKxtl.o
Enter the first number: 10
Enter the second number: 5
Sum: 15
Difference: 5
Product: 50
Division: 2


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

12. Write a C program to balance symbols in a given expression

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

#define MAX 100

char stack[MAX];
int top = -1;

void push(char c) {
    stack[++top] = c;
}

char pop() {
    if (top == -1)
        return '\0';
    else
        return stack[top--];
}

int isMatchingPair(char character1, char character2) {
    if (character1 == '(' && character2 == ')')
        return 1;
    else if (character1 == '{' && character2 == '}')
        return 1;
    else if (character1 == '[' && character2 == ']')
        return 1;
    else
        return 0;
}

int isBalanced(char exp[]) {
    int i = 0;
    char popped_char;

    while (exp[i]) {
        if (exp[i] == '(' || exp[i] == '{' || exp[i] == '[')
            push(exp[i]);
        if (exp[i] == ')' || exp[i] == '}' || exp[i] == ']') {
            if (top == -1)
                return 0;
            else {
                popped_char = pop();
                if (!isMatchingPair(popped_char, exp[i]))
                    return 0;
            }
        }
        i++;
    }

    if (top == -1)
        return 1;
    else
        return 0;
}

int main() {
    char exp[MAX];
    printf("Enter an expression: ");
    scanf("%s", exp);

    if (isBalanced(exp))
        printf("The expression is balanced.\n");
    else
        printf("The expression is not balanced.\n");

    return 0;
}
```

```
Enter an expression: [({})]
The expression is balanced.


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

13.Write a recursive function in C to implement Tower of Hanoi Problem

```c
#include <stdio.h>

void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
    if (n == 1) {
        printf("Move disk 1 from rod %c to rod %c\n", from_rod, to_rod);
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    printf("Move disk %d from rod %c to rod %c\n", n, from_rod, to_rod);
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}

int main() {
    int n = 3; // Number of disks
    towerOfHanoi(n, 'A', 'C', 'B');  // A, B, and C are rod names
    return 0;
}
```

```
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C


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

14.Write a recursive function in C to find the factorial of a number

```c
#include <stdio.h>

int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

int main() {
    int number = 5;
    int result = factorial(number);
    printf("Factorial of %d = %d", number, result);
    return 0;
}
```

```
Factorial of 5 = 120

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

15. Implement a queue using an array

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

#define MAX_SIZE 100

struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
};

struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = -1;
    queue->rear = -1;
    return queue;
}

int isEmpty(struct Queue* queue) {
    if (queue->rear == -1)
        return 1;
    else
        return 0;
}

int isFull(struct Queue* queue) {
    if (queue->rear == MAX_SIZE - 1)
        return 1;
    else
        return 0;
}

void enqueue(struct Queue* queue, int value) {
    if (isFull(queue))
        printf("Queue is full\n");
    else {
        if (isEmpty(queue))
            queue->front = 0;
        queue->rear++;
        queue->items[queue->rear] = value;
    }
}

int dequeue(struct Queue* queue) {
    int item;
    if (isEmpty(queue)) {
        printf("Queue is empty\n");
        return -1;
    } else {
        item = queue->items[queue->front];
        queue->front++;
        if (queue->front > queue->rear) {
            queue->front = queue->rear = -1;
        }
        return item;
    }
}

int main() {
    struct Queue* queue = createQueue();

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);

    printf("Dequeued item: %d\n", dequeue(queue));
    printf("Dequeued item: %d\n", dequeue(queue));
```

```
Dequeued item: 10
Dequeued item: 20


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

16. Implement a queue using linked list

```
Queue Front: 30
Queue Rear: 40


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

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

struct Node {
    int data;
    struct Node* next;
};

struct Queue {
    struct Node *front, *rear;
};

struct Node* newNode(int data) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = data;
    temp->next = NULL;
    return temp;
}

struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

void enQueue(struct Queue* queue, int data) {
    struct Node* temp = newNode(data);

    if (queue->rear == NULL) {
        queue->front = queue->rear = temp;
        return;
    }

    queue->rear->next = temp;
    queue->rear = temp;
}

void deQueue(struct Queue* queue) {
    if (queue->front == NULL)
        return;

    struct Node* temp = queue->front;

    queue->front = queue->front->next;

    if (queue->front == NULL)
        queue->rear = NULL;

    free(temp);
}

int main() {
    struct Queue* queue = createQueue();

    enQueue(queue, 10);
    enQueue(queue, 20);
    deQueue(queue);
    enQueue(queue, 30);
    enQueue(queue, 40);
    deQueue(queue);

    printf("Queue Front: %d\n", queue->front->data);
    printf("Queue Rear: %d\n", queue->rear->data);

    return 0;
}
```