11. Write a C program to evaluate an arithmetic operation

```c
1   #include <stdio.h>
2
3   int main() {
4       int num1, num2, result;
5
6       printf("Enter the first number: ");
7       scanf("%d", &num1);
8
9       printf("Enter the second number: ");
10      scanf("%d", &num2);
11
12      result = num1 + num2;
13      printf("Sum: %d\n", result);
14
15      result = num1 - num2;
16      printf("Difference: %d\n", result);
17
18      result = num1 * num2;
19      printf("Product: %d\n", result);
20
21      if (num2 != 0) {
22          result = num1 / num2;
23          printf("Division: %d\n", result);
24      } else {
25          printf("Cannot divide by zero.\n");
26      }
27
28      return 0;
29  }
```

```
/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
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define MAX 100
5
6   char stack[MAX];
7   int top = -1;
8
9   void push(char c) {
10      stack[++top] = c;
11  }
12
13  char pop() {
14      if (top == -1)
15          return '\0';
16      else
17          return stack[top--];
18  }
19
20  int isMatchingPair(char character1, char character2) {
21      if (character1 == '(' && character2 == ')')
22          return 1;
23      else if (character1 == '{' && character2 == '}')
24          return 1;
25      else if (character1 == '[' && character2 == ']')
26          return 1;
27      else
28          return 0;
29  }
30
31  int isBalanced(char exp[]) {
32      int i = 0;
33      char popped_char;
34
35      while (exp[i]) {
36          if (exp[i] == '(' || exp[i] == '{' || exp[i] == '[')
37              push(exp[i]);
38          if (exp[i] == ')' || exp[i] == '}' || exp[i] == ']') {
39              if (top == -1)
40                  return 0;
41              else {
42                  popped_char = pop();
43                  if (!isMatchingPair(popped_char, exp[i]))
44                      return 0;
45              }
46          }
47          i++;
48      }
49
50      if (top == -1)
51          return 1;
52      else
53          return 0;
54  }
55
56  int main() {
57      char exp[MAX];
58      printf("Enter an expression: ");
59      scanf("%s", exp);
60
61      if (isBalanced(exp))
62          printf("The expression is balanced.\n");
63      else
64          printf("The expression is not balanced.\n");
65
66      return 0;
67  }
```

```
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
1   #include <stdio.h>
2
3 ▾ int factorial(int n) {
4 ▾     if (n == 0) {
5             return 1;
6 ▾     } else {
7             return n * factorial(n - 1);
8         }
9   }
10
11 ▾ int main() {
12        int number = 5;
13        int result = factorial(number);
14        printf("Factorial of %d = %d", number, result);
15        return 0;
16  }
```

```
Factorial of 5 = 120


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

15. Implement a queue using an array

```c
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define MAX_SIZE 100
5
6 * struct Queue {
7       int items[MAX_SIZE];
8       int front;
9       int rear;
0   };
1
2 * struct Queue* createQueue() {
3       struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
4       queue->front = -1;
5       queue->rear = -1;
6       return queue;
7   }
8
9 * int isEmpty(struct Queue* queue) {
0       if (queue->rear == -1)
1           return 1;
2       else
3           return 0;
4   }
5
6 * int isFull(struct Queue* queue) {
7       if (queue->rear == MAX_SIZE - 1)
8           return 1;
9       else
0           return 0;
1   }
2
3 * void enqueue(struct Queue* queue, int value) {
4       if (isFull(queue))
5           printf("Queue is full\n");
6 *     else {
7           if (isEmpty(queue))
8               queue->front = 0;
9           queue->rear++;
0           queue->items[queue->rear] = value;
1       }
2   }
3
4 * int dequeue(struct Queue* queue) {
5       int item;
6 *     if (isEmpty(queue)) {
7           printf("Queue is empty\n");
8           return -1;
9 *     } else {
0           item = queue->items[queue->front];
1           queue->front++;
2 *         if (queue->front > queue->rear) {
3               queue->front = queue->rear = -1;
4           }
5           return item;
6       }
7   }
8
9 * int main() {
0       struct Queue* queue = createQueue();
1
2       enqueue(queue, 10);
3       enqueue(queue, 20);
4       enqueue(queue, 30);
5
6       printf("Dequeued item: %d\n", dequeue(queue));
7       printf("Dequeued item: %d\n", dequeue(queue));
```

```
/tmp/...
Dequeued item: 10
Dequeued item: 20


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

16. Implement a queue using linked list

```c
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define MAX_SIZE 100
5
6 * struct Queue {
7       int items[MAX_SIZE];
8       int front;
9       int rear;
0   };
1
2 * struct Queue* createQueue() {
3       struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
4       queue->front = -1;
5       queue->rear = -1;
6       return queue;
7   }
8
9 * int isEmpty(struct Queue* queue) {
0       if (queue->rear == -1)
1           return 1;
2       else
3           return 0;
4   }
5
6 * int isFull(struct Queue* queue) {
7       if (queue->rear == MAX_SIZE - 1)
8           return 1;
9       else
0           return 0;
1   }
2
3 * void enqueue(struct Queue* queue, int value) {
4       if (isFull(queue))
5           printf("Queue is full\n");
6 *     else {
7           if (isEmpty(queue))
8               queue->front = 0;
9           queue->rear++;
0           queue->items[queue->rear] = value;
1       }
2   }
3
4 * int dequeue(struct Queue* queue) {
5       int item;
6 *     if (isEmpty(queue)) {
7           printf("Queue is empty\n");
8           return -1;
9 *     } else {
0           item = queue->items[queue->front];
1           queue->front++;
2 *         if (queue->front > queue->rear) {
3               queue->front = queue->rear = -1;
4           }
5           return item;
6       }
7   }
8
9 * int main() {
0       struct Queue* queue = createQueue();
1
2       enqueue(queue, 10);
3       enqueue(queue, 20);
4       enqueue(queue, 30);
5
6       printf("Dequeued item: %d\n", dequeue(queue));
7       printf("Dequeued item: %d\n", dequeue(queue));
```

```
Queue Front: 30
Queue Rear: 40


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

```c
1   #include <stdio.h>
2   #include <stdlib.h>
3
4 * struct Node {
5       int data;
6       struct Node* next;
7   };
8
9 * struct Queue {
10      struct Node *front, *rear;
11  };
12
13 * struct Node* newNode(int data) {
14      struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
15      temp->data = data;
16      temp->next = NULL;
17      return temp;
18  }
19
20 * struct Queue* createQueue() {
21      struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
22      queue->front = queue->rear = NULL;
23      return queue;
24  }
25
26 * void enQueue(struct Queue* queue, int data) {
27      struct Node* temp = newNode(data);
28
29 *     if (queue->rear == NULL) {
30          queue->front = queue->rear = temp;
31          return;
32      }
33
34      queue->rear->next = temp;
35      queue->rear = temp;
36  }
37
38 * void deQueue(struct Queue* queue) {
39      if (queue->front == NULL)
40          return;
41
42      struct Node* temp = queue->front;
43
44      queue->front = queue->front->next;
45
46      if (queue->front == NULL)
47          queue->rear = NULL;
48
49      free(temp);
50  }
51
52 * int main() {
53      struct Queue* queue = createQueue();
54
55      enQueue(queue, 10);
56      enQueue(queue, 20);
57      deQueue(queue);
58      enQueue(queue, 30);
59      enQueue(queue, 40);
60      deQueue(queue);
61
62      printf("Queue Front: %d\n", queue->front->data);
63      printf("Queue Rear: %d\n", queue->rear->data);
64
65      return 0;
66  }
```