//1Q1: WAP in C programming to take a graph (Directed + Undirected) and show the degree of each vertex.

```
#include <stdio.h>

#define MAX 100

void calculateDegree(int graph[MAX][MAX], int V) {
    int degree[MAX] = {0};
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (graph[i][j] == 1) {
                degree[i]++;
            }
        }
    }
    for (int i = 0; i < V; i++) {
        printf("Degree of vertex %d is %d\n", i, degree[i]);
    }
}

int main() {
    int V, E, u, v;
    int graph[MAX][MAX] = {0};

    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    printf("Enter the number of edges: ");
    scanf("%d", &E);

    for (int i = 0; i < E; i++) {
        printf("Enter edge (u v): ");
        scanf("%d %d", &u, &v);
        graph[u][v] = 1;
        graph[v][u] = 1;  // For undirected graph
    }

    calculateDegree(graph, V);

    return 0;
}
```

Q2: WAP to calculate the modulus of two numbers (like A mod B).

```
#include <stdio.h>

int calculateModulus(int A, int B) {
    if (B == 0) {
        printf("Error: Division by zero is undefined.\n");
        return -1;
    }

    int quotient = A / B;
    int remainder = A - (quotient * B);

    return remainder;
}

int main() {
    int A, B, result;

    printf("Enter the value of A: ");
    scanf("%d", &A);

    printf("Enter the value of B: ");
    scanf("%d", &B);

    result = calculateModulus(A, B);

    if (result != -1) {
        printf("The result of %d mod %d is: %d\n", A, B, result);
    }

    return 0;
}
```

Q3: WAP to implement the Extended Euclidean Algorithm.

```c
#include <stdio.h>

// Function to implement the Extended Euclidean Algorithm
void extendedEuclidean(int a, int b) {
    int s0 = 1, s1 = 0, s;
    int t0 = 0, t1 = 1, t;
    int q, r;
    int original_a = a, original_b = b;

    // Ensure a is greater than or equal to b
    if (a < b) {
        int temp = a;
        a = b;
        b = temp;
    }

    // Print the table header
    printf("%5s %5s %5s %5s %5s %5s %5s %5s %5s\n", "q", "a", "b", "r", "s1",
"s0", "s", "t1", "t0", "t");

    while (b != 0) {
        q = a / b;
        r = a % b;

        // Print the current step
        printf("%5d %5d %5d %5d %5d %5d %5d %5d %5d\n", q, a, b, r, s1, s0,
s0 - q * s1, t1, t0, t0 - q * t1);

        // Update a and b
        a = b;
        b = r;

        // Update s and t
        s = s0 - q * s1;
        t = t0 - q * t1;

        // Move to next iteration
        s0 = s1;
        s1 = s;
        t0 = t1;
        t1 = t;
    }
```

```
    printf("GCD(%d, %d) = %d\n", original_a, original_b, a);
    printf("Coefficients x and y: %d, %d\n", s0, t0);
}

int main() {
    int a, b;

    // Input two numbers
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    // Calculate the Extended Euclidean Algorithm
    extendedEuclidean(a, b);

    return 0;
}
```

Q4: WAP to print the truth table for the tautology $(p \wedge q) \rightarrow (p \vee q)$.

```
#include <stdio.h>

int main() {
    int p, q;
    printf("p q | (p ∧ q) | (p ∨ q) | (p ∧ q) → (p ∨ q)\n");
    for (p = 0; p <= 1; p++) {
        for (q = 0; q <= 1; q++) {
            int and = p && q;
            int or = p || q;
            int implies = !and || or;
            printf("%d %d |    %d    |    %d    |       %d\n", p, q, and, or, implies);
        }
    }
    return 0;
}
```

Q5: WAP to find Addition, Subtraction, and Multiplication of two binary numbers.

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

```
int binary_addition(int a, int b) {
    int carry = 0;
    int result = 0;
    int position = 1;

    while (a > 0 || b > 0 || carry > 0) {
        int sum = (a % 2) + (b % 2) + carry;
        carry = sum / 2;
        result += (sum % 2) * position;
        a /= 2;
        b /= 2;
        position *= 2;
    }

    return result;
}

int binary_subtraction(int a, int b) {
    int borrow = 0;
    int result = 0;
    int position = 1;

    while (a > 0 || b > 0 || borrow > 0) {
        int diff = (a % 2) - (b % 2) - borrow;
        if (diff < 0) {
            diff += 2;
            borrow = 1;
        } else {
            borrow = 0;
        }
        result += diff * position;
        a /= 2;
        b /= 2;
        position *= 2;
    }

    return result;
}

int binary_multiplication(int a, int b) {
    int result = 0;
    int position = 1;
```

```
    while (b > 0) {
        if (b % 2 == 1) {
            result = binary_addition(result, a);
        }
        a <<= 1; // Shift left (multiply by 2)
        b >>= 1; // Shift right (divide by 2)
    }

    return result;
}

int main() {
    int a, b;

    printf("Enter first binary number: ");
    scanf("%d", &a);
    printf("Enter second binary number: ");
    scanf("%d", &b);

    printf("Addition: %d\n", binary_addition(a, b));
    printf("Subtraction: %d\n", binary_subtraction(a, b));
    printf("Multiplication: %d\n", binary_multiplication(a, b));

    return 0;
}
```