Assignment 1

IT851: Information and Systems Security Lab

ANINDYA KUNDU

IT, 8th Semester

ID **510817020**

Repository:
github.com/meganindya/btech-assignments/information-and-systems-security/assg-1

08-03-2021

Part – A

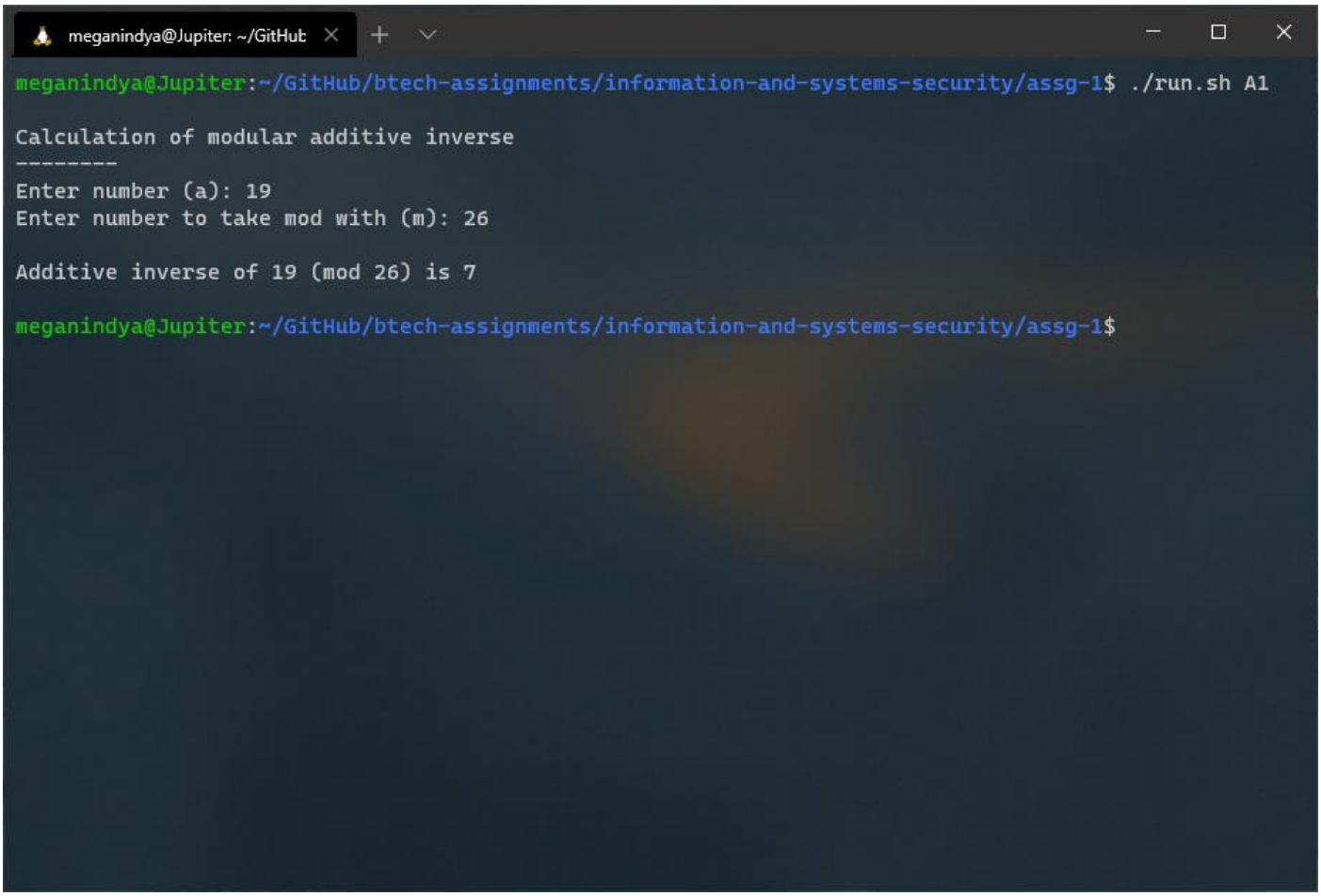
Implement the following in Modular Arithmetic:

1. Additive inverse of a number

Source: `A-1-additive-inverse.c`

```c
#include <stdio.h>

int mod_inv_add(int a, int m)
{
    return m - (a % m);
}

int main(int argc, char *argv[])
{
    int a, m;
    printf("\nCalculation of modular additive inverse\n--------\n");
    printf("Enter number (a): ");
    scanf("%d", &a);
    printf("Enter number to take mod with (m): ");
    scanf("%d", &m);
    printf("\nAdditive inverse of %d (mod %d) is %d\n\n", a, m, mod_inv_add(a, m));
}
```

Sample run

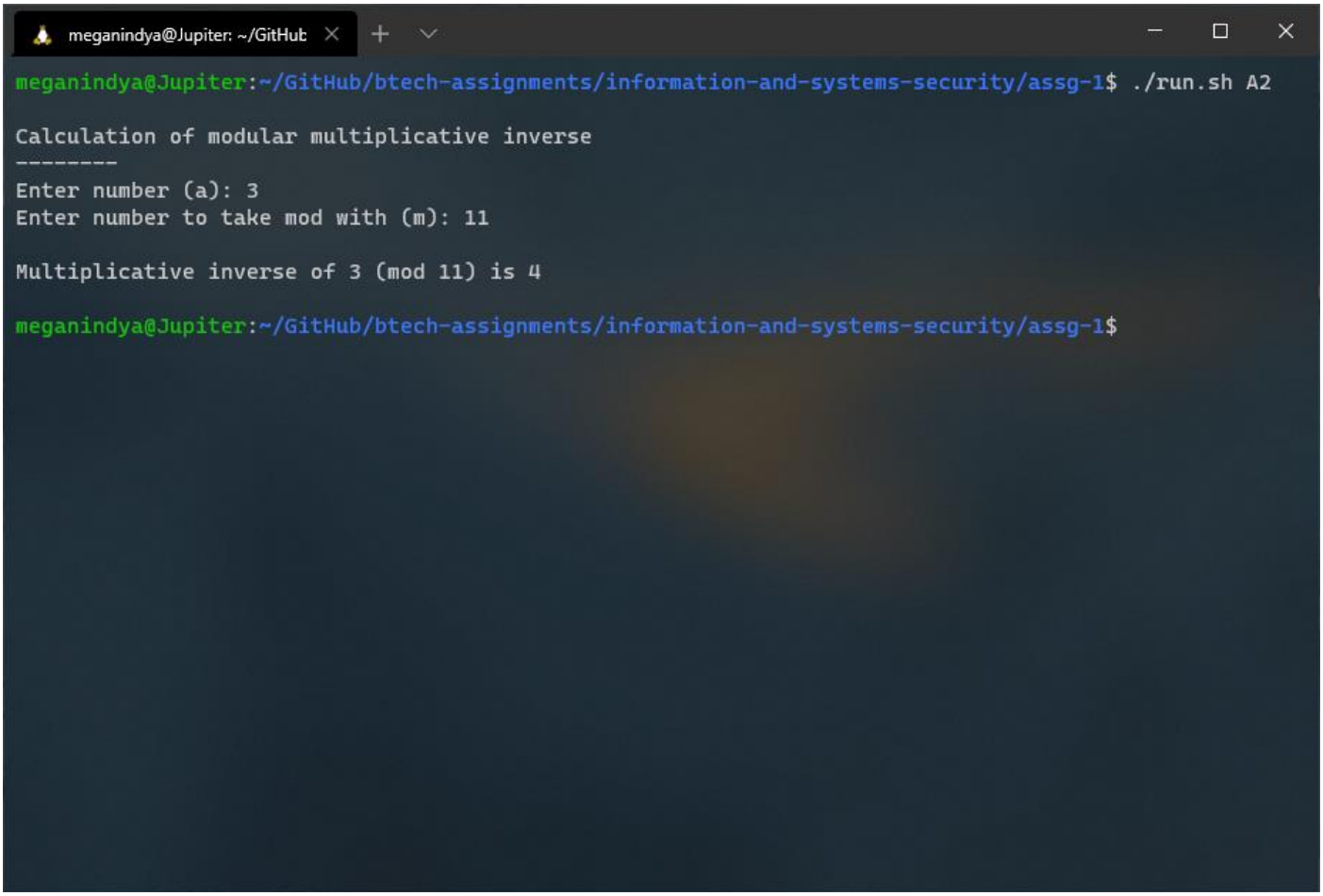


2. Multiplicative inverse of a number

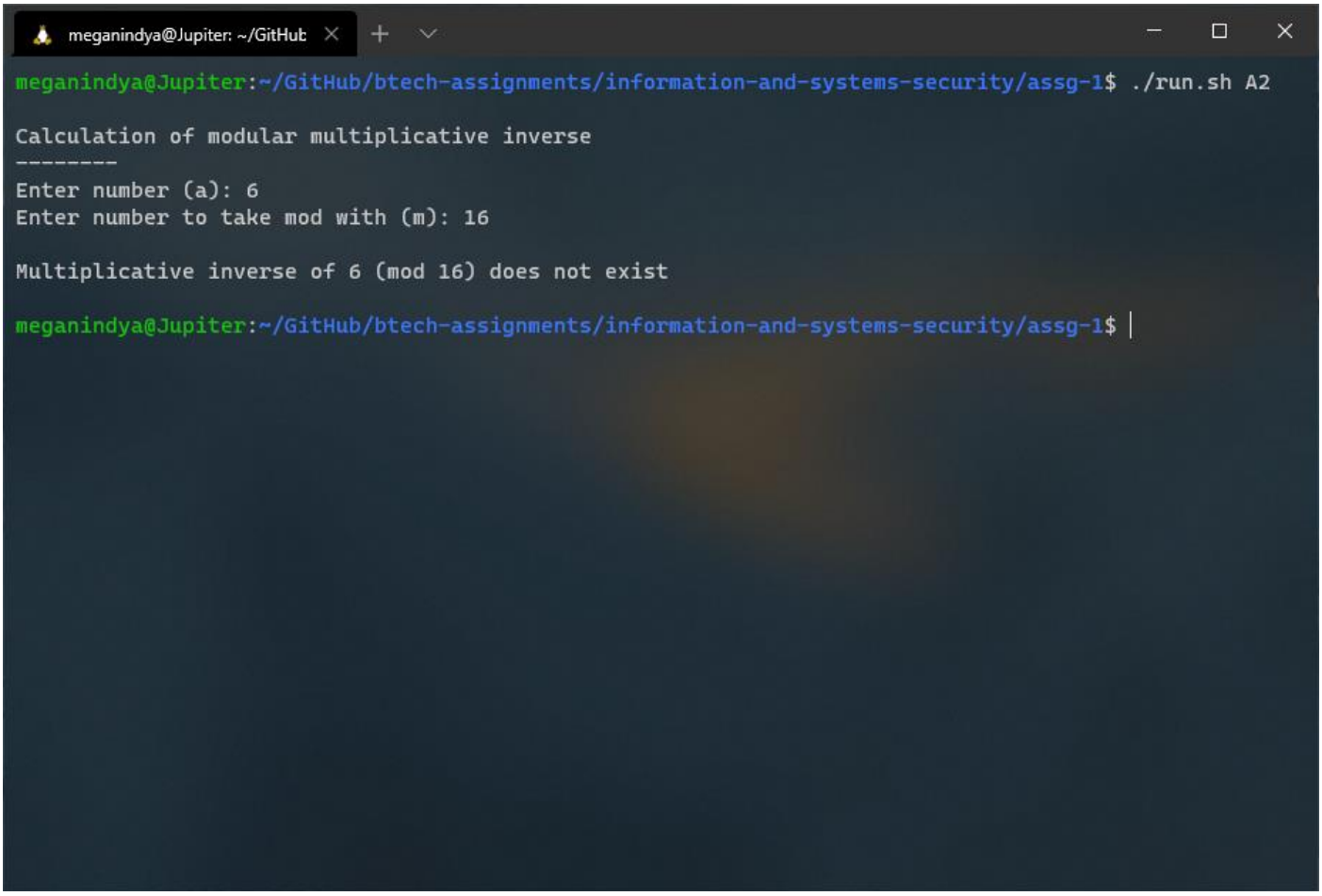
Source: **A-2-multiplicative-inverse.c**

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

int mod_inv_mul(int a, int m)
{
    for (int x = 1; x < m; x++)
    {
        if (a < 0)
        {
            if (((m - (abs(a) % m)) * (x % m)) % m == 1)
            {
                return x;
            }
        }
        else
        {
            if (((a % m) * (x % m)) % m == 1)
            {
                return x;
            }
        }
    }
    return -1;
}

int main(int argc, char *argv[])
{
    int a, m;
    printf("\nCalculation of modular multiplicative inverse\n--------\n");
    printf("Enter number (a): ");
    scanf("%d", &a);
    printf("Enter number to take mod with (m): ");
    scanf("%d", &m);
    int inv = mod_inv_mul(a, m);
    if (inv == -1)
    {
        printf("\nMultiplicative inverse of %d (mod %d) does not exist\n\n", a, m);
    }
    else
    {
        printf(
            "\nMultiplicative inverse of %d (mod %d) is %d\n\n", a, m, mod_inv_mul(a, m)
        );
    }
}
```

Sample run


meganindya@Jupiter: ~/GitHub
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$ ./run.sh A2
Calculation of modular multiplicative inverse

Enter number (a): 3
Enter number to take mod with (m): 11
Multiplicative inverse of 3 (mod 11) is 4
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$


meganindya@Jupiter: ~/GitHub
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$ ./run.sh A2
Calculation of modular multiplicative inverse

Enter number (a): 6
Enter number to take mod with (m): 16
Multiplicative inverse of 6 (mod 16) does not exist
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$

3. Inverse of an m×m matrix with m ≤ 3

Source: **A-3-matrix-inverse.c**

```c
#include <stdio.h>  // printf, scanf
#include <stdlib.h> // abs
#include <math.h>   // pow

int mod(int, int);
int mod_inv_mul(int, int);
int determinant_mat(int, int **);
void cofactor_mat(int, int **, int **);
void transpose_mat(int, int **);

int main()
{
    printf("\nCalculation of modular matrix inverse\n--------\n");

    printf("Enter the order of the matrix: ");
    /* Stores order of matrix. */
    int ord_mat;
    scanf("%d", &ord_mat);

    printf("Enter the elements of the %d × %d matrix:\n", ord_mat, ord_mat);
    /* Stores the matrix. */
    int **mat;
    mat = malloc((ord_mat) * sizeof *mat);
    for (int i = 0; i < ord_mat; i++)
    {
        mat[i] = malloc(ord_mat * sizeof *mat[i]);
    }
    for (int i = 0; i < ord_mat; i++)
    {
        for (int j = 0; j < ord_mat; j++)
        {
            scanf("%d", &mat[i][j]);
        }
    }

    /* Stores the number to take modulus with. */
    int m;
    printf("Enter the number to take mod with (m): ");
    scanf("%d", &m);

    /* Stores the determinant of the matrix. */
    int det = determinant_mat(ord_mat, mat);
    /* Stores the modular multiplicative inverse of the determinant. */
    int det_inv = mod_inv_mul(det, m);
    if (det_inv == -1)
    {
        printf("\nInverse of entered matrix is not possible\n\n");
        return 0;
    }
```

```
int **new_mat;
new_mat = malloc((ord_mat) * sizeof *new_mat);
for (int i = 0; i < ord_mat; i++)
{
    new_mat[i] = malloc(ord_mat * sizeof *new_mat[i]);
}

cofactor_mat(ord_mat, mat, new_mat);
transpose_mat(ord_mat, new_mat);

for (int i = 0; i < ord_mat; i++)
{
    for (int j = 0; j < ord_mat; j++)
    {
        new_mat[i][j] = mod(new_mat[i][j] * det_inv, m);
    }
}

printf("\nMatrix is\n");
for (int i = 0; i < ord_mat; i++)
{
    for (int j = 0; j < ord_mat; j++)
    {
        printf("%3d ", mat[i][j]);
    }
    printf("\n");
}

printf("\nModular matrix inverse is\n");
for (int i = 0; i < ord_mat; i++)
{
    for (int j = 0; j < ord_mat; j++)
    {
        printf("%3d ", new_mat[i][j]);
    }
    printf("\n");
}

printf("\nSide by side\n");
for (int i = 0; i < ord_mat; i++)
{
    for (int j = 0; j < ord_mat; j++)
    {
        printf("%3d ", mat[i][j]);
    }
    printf("    ");
    for (int j = 0; j < ord_mat; j++)
    {
        printf("%3d ", new_mat[i][j]);
    }
    printf("\n");
```

```
    }

    printf("\nOn multiplication\n");
    for (int i = 0; i < ord_mat; i++)
    {
        for (int j = 0; j < ord_mat; j++)
        {
            int sum = 0;
            for (int k = 0; k < ord_mat; k++)
            {
                sum += mat[i][k] * new_mat[k][j];
            }
            printf("%3d ", sum);
        }
        printf("\n");
    }

    printf("\nTaking modulus with %d\n", m);
    for (int i = 0; i < ord_mat; i++)
    {
        for (int j = 0; j < ord_mat; j++)
        {
            int sum = 0;
            for (int k = 0; k < ord_mat; k++)
            {
                sum += mat[i][k] * new_mat[k][j];
            }
            printf("%3d ", mod(sum, m));
        }
        printf("\n");
    }
    printf("\n");

    for (int i = 0; i < ord_mat; i++)
    {
        free(new_mat[i]);
    }
    free(new_mat);

    for (int i = 0; i < ord_mat; i++)
    {
        free(mat[i]);
    }
    free(mat);
}

/*
 * Returns the modulus of a number with another. Handles negative case.
 *
 * a: number to take modulus of (can be any integer)
 * m: number to take modulus with (natural number)
 */
```

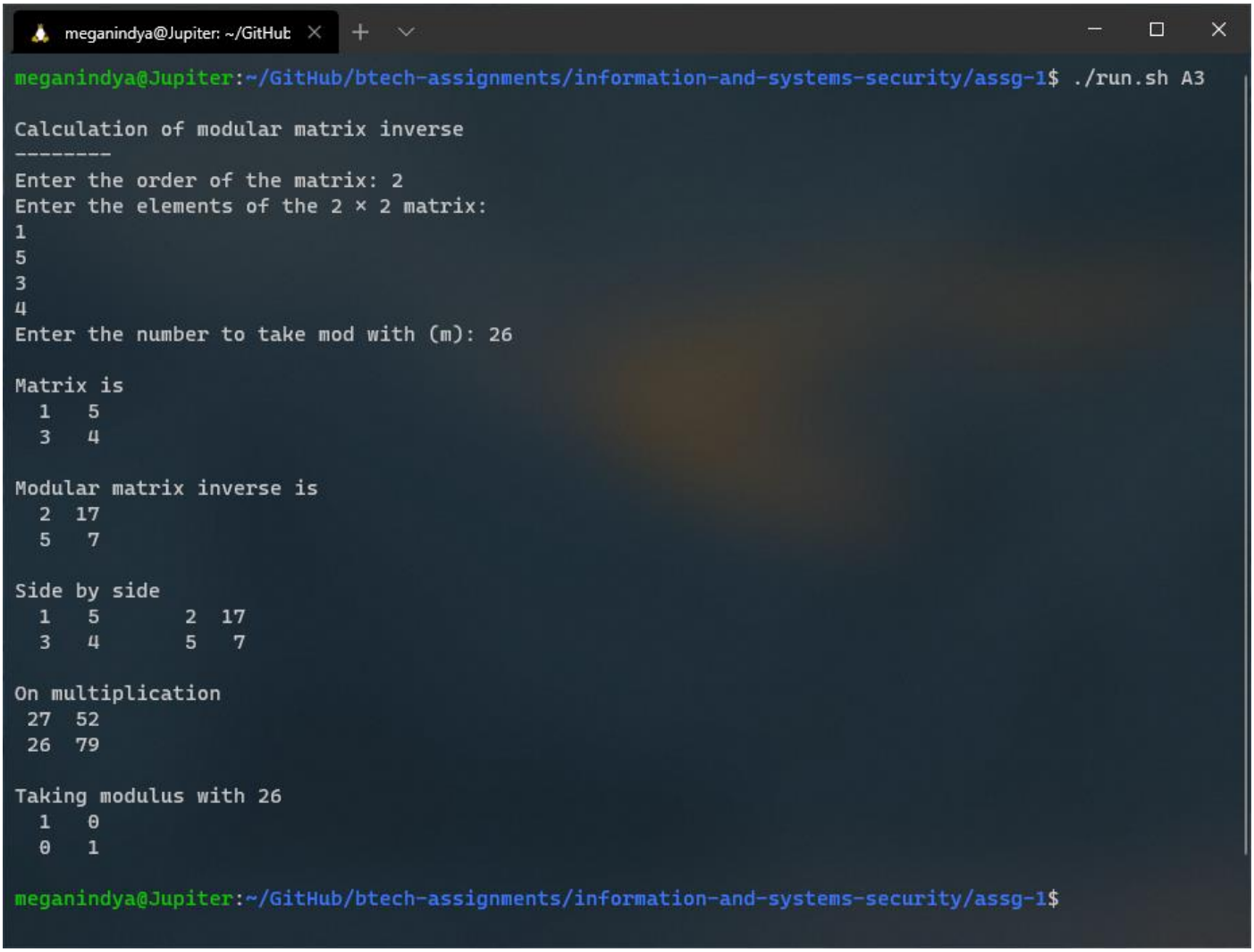
```c
int mod(int a, int m)
{
    return a < 0 ? m - (abs(a) % m) : a % m;
}

/*
 * Calculates the modular multiplicative inverse of a number.
 *
 * a: input number
 * m: modulus number
 *
 * returns: multiplicative inverse of a (mod m)
 */
int mod_inv_mul(int a, int m)
{
    for (int x = 1; x < m; x++)
    {
        if ((mod(a, m) * (x % m)) % m == 1)
        {
            return x;
        }
    }
    return -1;
}

/*
 * Calculates the determinant of a matrix.
 *
 * ord_mat: order of the matrix
 * mat: supplied matrix
 *
 * returns: determinant of mat
 */
int determinant_mat(int ord_mat, int **mat)
{
    if (ord_mat == 1)
    {
        return mat[0][0];
    }

    int det = 0;
    for (int z = 0; z < ord_mat; z++)
    {
        int sign = ((1 - (z & 1)) << 1) - 1;

        int **sub_mat;
        sub_mat = malloc((ord_mat - 1) * sizeof *sub_mat);
        for (int i = 0; i < ord_mat - 1; i++)
        {
            sub_mat[i] = malloc((ord_mat - 1) * sizeof *sub_mat[i]);
        }
```

```
        int kr = 0, kc = 0;
        for (int r = 1; r < ord_mat; r++)
        {
            for (int c = 0; c < ord_mat; c++)
            {
                if (c != z)
                {
                    sub_mat[kr][kc] = mat[r][c];
                    kc = (kc + 1) % ord_mat;
                }
            }
            kr++;
        }

        det += sign * mat[0][z] * determinant_mat(ord_mat - 1, sub_mat);

        for (int i = 0; i < ord_mat - 1; i++)
        {
            free(sub_mat[i]);
        }
        free(sub_mat);
    }

    return det;
}

/*
 * Calculates and fills the cofactors for a matrix.
 *
 * ord_mat: order of the matrix
 * mat: supplied matrix
 * cof_mat: matrix to fill cofactors in
 */
void cofactor_mat(int ord_mat, int **mat, int **cof_mat)
{
    for (int r = 0; r < ord_mat; r++)
    {
        for (int c = 0; c < ord_mat; c++)
        {
            int sign = ((1 - ((r + c) & 1)) << 1) - 1;

            int **sub_mat;
            sub_mat = malloc((ord_mat - 1) * sizeof *sub_mat);
            for (int i = 0; i < ord_mat - 1; i++)
            {
                sub_mat[i] = malloc((ord_mat - 1) * sizeof *sub_mat[i]);
            }

            int ki = 0, kj = 0;
            for (int i = 0; i < ord_mat; i++)
            {
                for (int j = 0; j < ord_mat; j++)
```

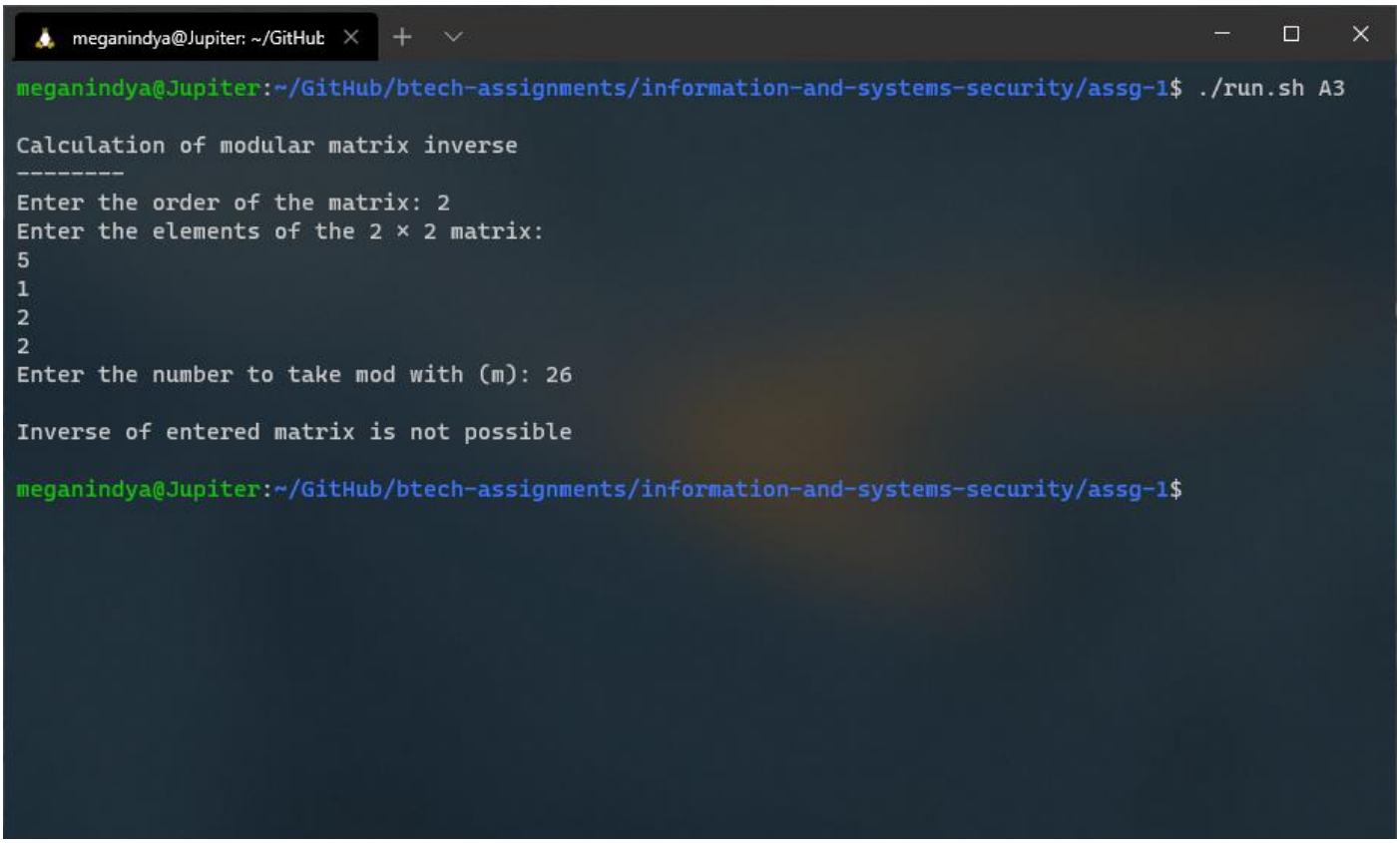
```
                {
                    if (i != r && j != c)
                    {
                        sub_mat[ki][kj] = mat[i][j];
                        kj = (kj + 1) % (ord_mat - 1);
                    }
                }
                if (i != r)
                    ki++;
            }

            cof_mat[r][c] = sign * determinant_mat(ord_mat - 1, sub_mat);

            for (int i = 0; i < ord_mat - 1; i++)
            {
                free(sub_mat[i]);
            }
            free(sub_mat);
        }
    }
}

/*
 * Transposes a square matrix in place.
 *
 * ord_mat: order of the matrix
 * mat: supplied matrix
 */
void transpose_mat(int ord_mat, int **mat)
{
    for (int r = 0; r < ord_mat; r++)
    {
        for (int c = 0; c <= ((ord_mat - 1) >> 1); c++)
        {
            int temp = mat[r][c];
            mat[r][c] = mat[c][r];
            mat[c][r] = temp;
        }
    }
}
```

[P.T.O.]

Sample run


meganindya@Jupiter: ~/GitHub
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$ ./run.sh A3
Calculation of modular matrix inverse

Enter the order of the matrix: 2
Enter the elements of the 2 × 2 matrix:
1
5
3
4
Enter the number to take mod with (m): 26
Matrix is
1 5
3 4
Modular matrix inverse is
2 17
5 7
Side by side
1 5 2 17
3 4 5 7
On multiplication
27 52
26 79
Taking modulus with 26
1 0
0 1
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$


meganindya@Jupiter: ~/GitHub
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$ ./run.sh A3
Calculation of modular matrix inverse

Enter the order of the matrix: 2
Enter the elements of the 2 × 2 matrix:
5
1
2
2
Enter the number to take mod with (m): 26
Inverse of entered matrix is not possible
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$

4. Implement Diffie Hellman Key Exchange (DHKE) Protocol, with small integer values for testing.

Source: **A-4-diffie-hellman.c**

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <sys/wait.h>
#include <math.h>

/*
 * Returns value of a ^ b (mod P).
 */
long long int power(long long int a, long long int b, long long int P)
{
    return b == 1 ? a : (((long long int)pow(a, b)) % P);
}

int main(int argc, char *argv[])
{
    printf("\n");

    long long int P = 23; // prime number P
    printf("Prime Number (P): %lld\n", P);

    long long int G = 9; // primitve root for P, G
    printf("Primitive Root (G): %lld\n\n", G);

    // pipe 1 to send from parent to child
    int fd1[2]; // used to store two ends of pipe 1

    // pipe 2 to send from child to parent
    int fd2[2]; // used to store two ends of pipe 2

    if (pipe(fd1) == -1 || pipe(fd2) == -1)
    {
        fprintf(stderr, "Pipe Failed");
        return 1;
    }

    pid_t pid = fork();
    if (pid < 0)
    {
        fprintf(stderr, "fork Failed");
        return 1;
    }

    // parent process
    else if (pid > 0)
    {
        // private key of parent
```

```
        long long int a;
        FILE *fp = fopen("A-4-a.txt", "r");
        fscanf(fp, "%lld", &a);
        fclose(fp);

        printf("Private key of process A (a): %lld\n", a);
        // generated key of parent
        long long int x = power(G, a, P), y;
        printf("Generated public key from process A is x = G ^ a (mod P) = %lld\n", x);

        // close reading end of pipe 1
        close(fd1[0]);

        // write x and close writing end of pipe 1
        write(fd1[1], &x, sizeof(x));
        close(fd1[1]);

        // wait for child to send
        wait(NULL);

        // read from child
        read(fd2[0], &y, sizeof(y));

        // secret key for parent
        long long int ka = power(y, a, P);
        printf("Shared secret key at A (ka): %lld\n", ka);

        // close reading end of pipe 2
        close(fd2[0]);

        printf("\n");
        return 0;
    }

    // child process
    else
    {
        // private key of child
        long long int b;
        FILE *fp = fopen("A-4-b.txt", "r");
        fscanf(fp, "%lld", &b);
        fclose(fp);

        printf("Private key of process B (b): %lld\n", b);
        // generated key of child
        long long int x, y = power(G, b, P);
        printf("\nGenerated public key from process B is y = G ^ b (mod P) = %lld\n", y);

        // read from parent
        read(fd1[0], &x, sizeof(x));

        // secret key for child
```

```c
        long long int kb = power(x, b, P);
        printf("\nShared secret key at B (kb): %lld\n", kb);

        // close both reading ends
        close(fd1[0]);
        close(fd2[0]);

        // write y and close writing end of pipe 2
        write(fd2[1], &y, sizeof(y));
        close(fd2[1]);

        exit(0);
    }
}
```

Source: `A-4-a.txt`

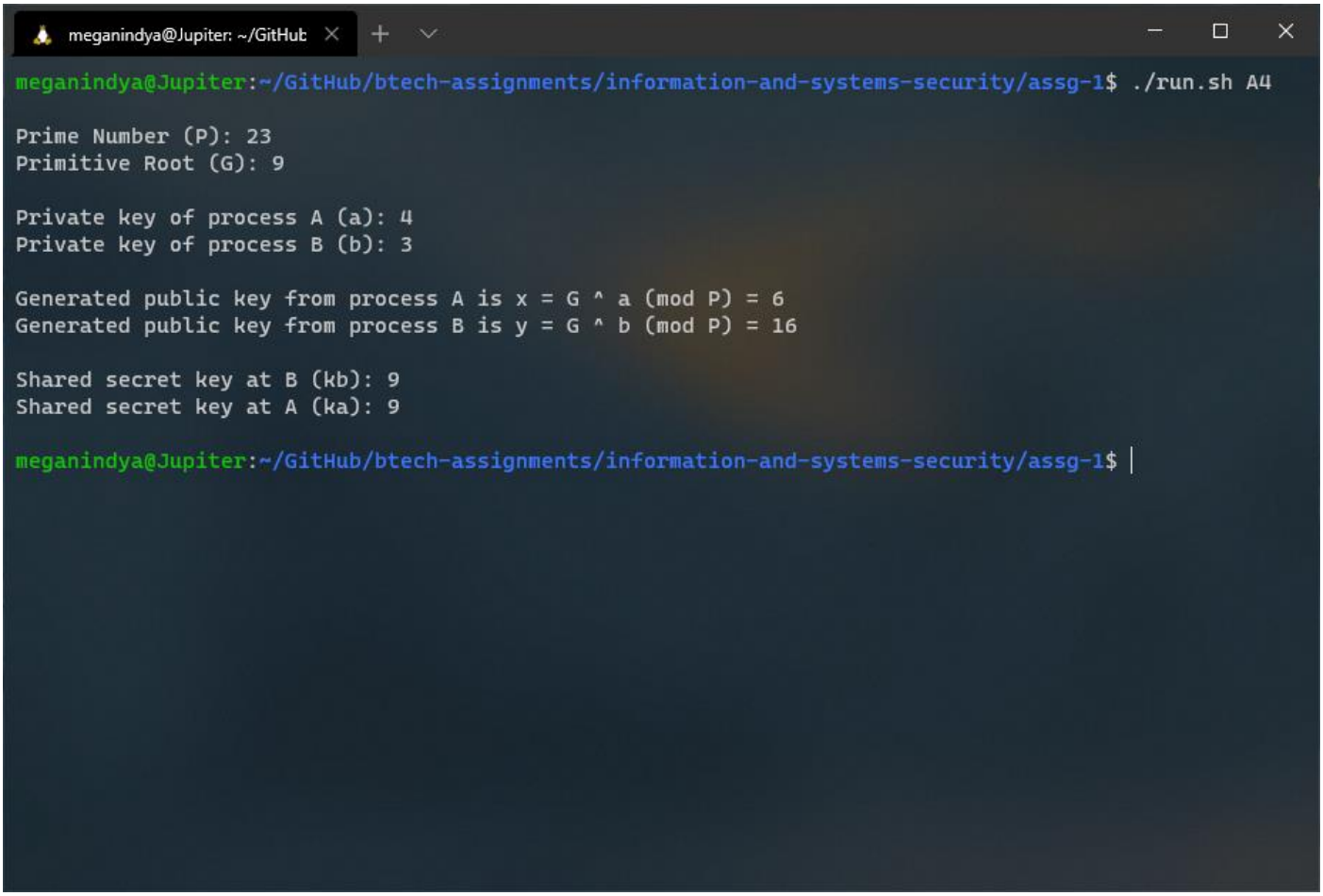
```
4
```

Source: `A-4-b.txt`

```
3
```

Sample run

Entry process is forked, then the two processes dynamically input data from files and interchange data via pipes.



Part – B

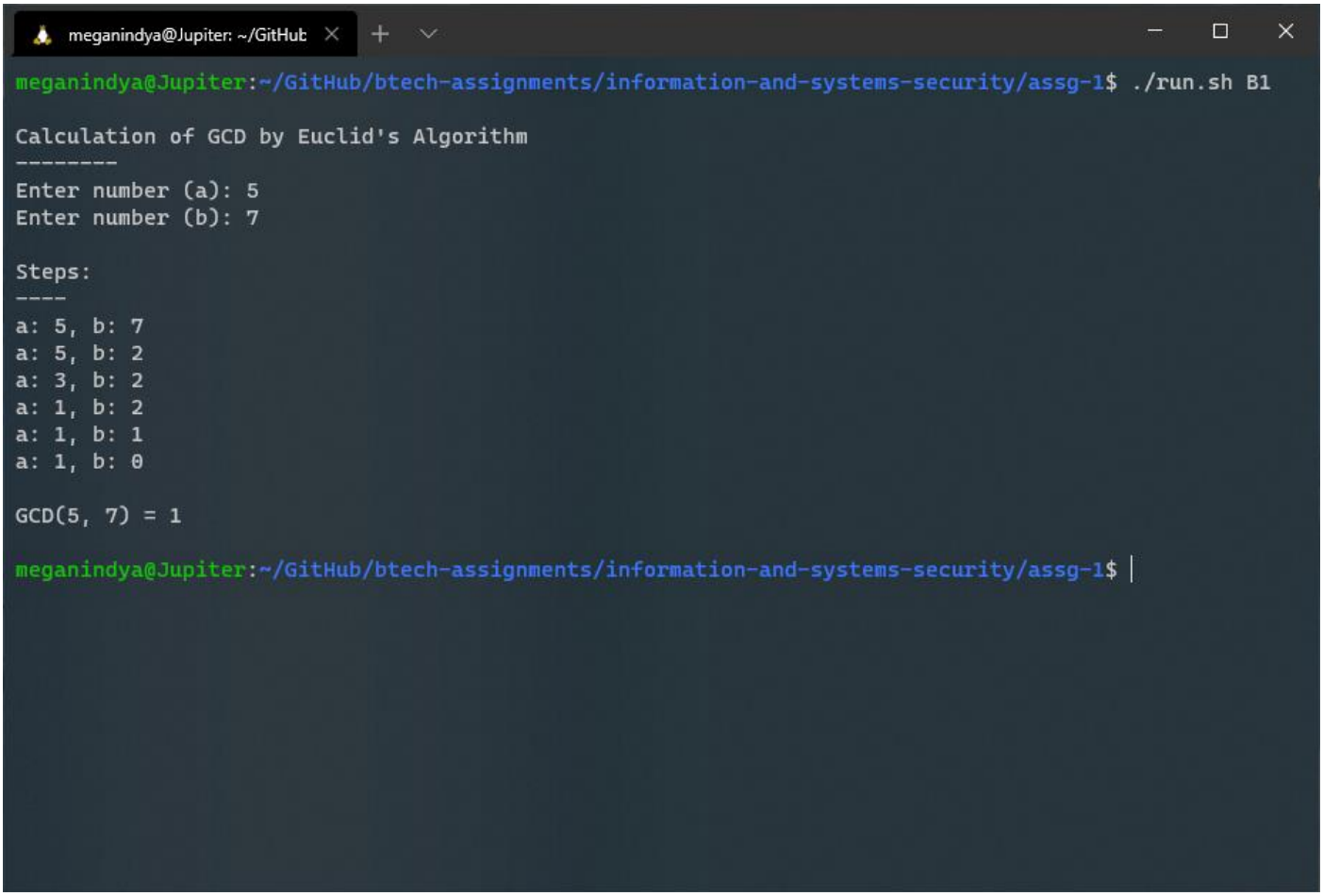
1. Implement the Euclidean Algorithm below, to find GCD of two numbers:

Source: `B-1-euclid-gcd-algorithm.c`

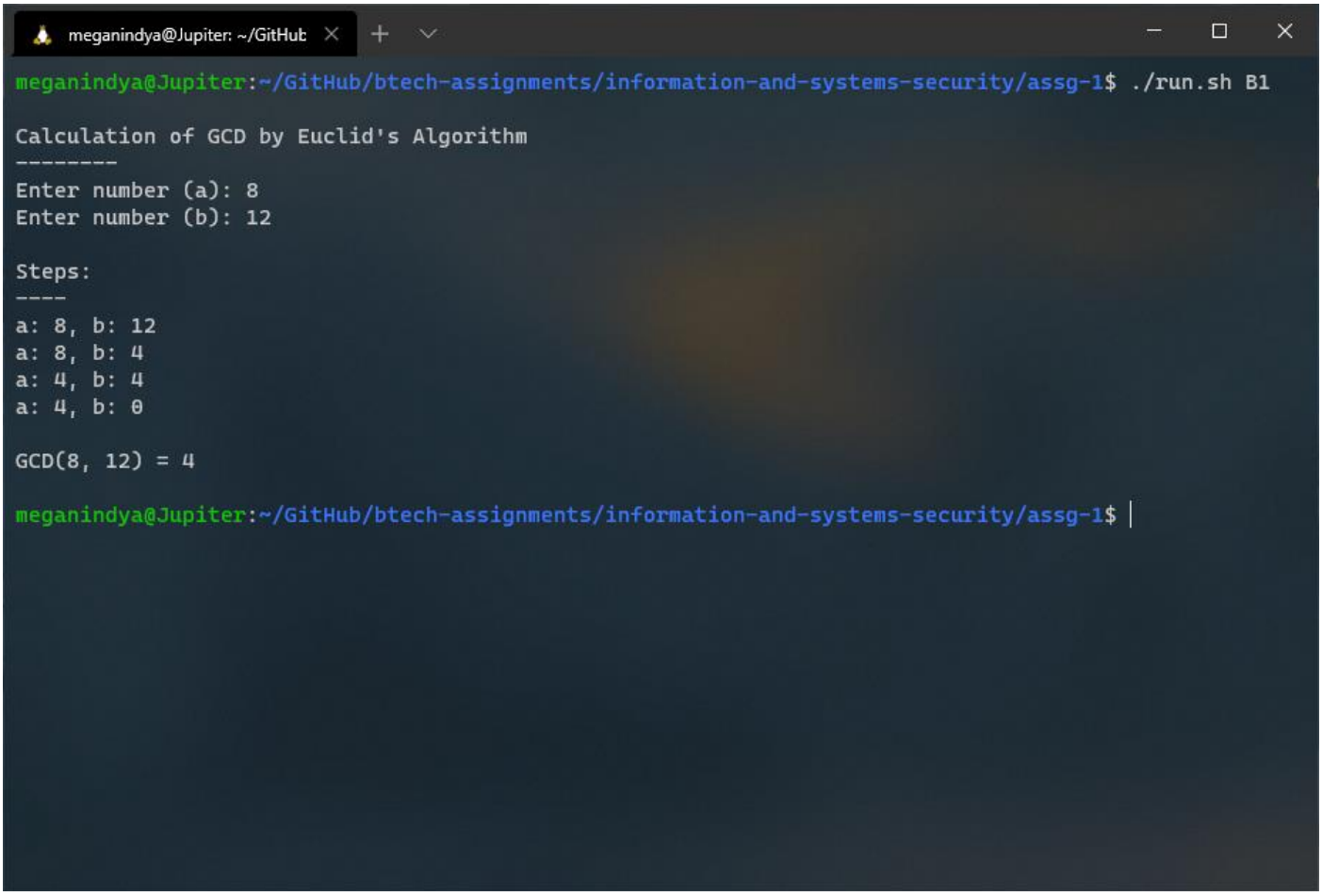
```c
#include <stdio.h>

int gcd(int a, int b)
{
    printf("\nSteps:\n----\n");
    if (a == 0)
    {
        printf("a: %d, b: %d\n", a, b);
        return b;
    }
    while (b != 0)
    {
        printf("a: %d, b: %d\n", a, b);
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }
    printf("a: %d, b: %d\n", a, b);
    return a;
}

int main(int argc, char *argv[])
{
    int a, b;
    printf("\nCalculation of GCD by Euclid's Algorithm\n--------\n");
    printf("Enter number (a): ");
    scanf("%d", &a);
    printf("Enter number (b): ");
    scanf("%d", &b);
    printf("\nGCD(%d, %d) = %d\n\n", a, b, gcd(a, b));
}
```

[P.T.O.]

Sample run


meganindya@Jupiter: ~/GitHub
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$ ./run.sh B1
Calculation of GCD by Euclid's Algorithm

Enter number (a): 5
Enter number (b): 7
Steps:
----
a: 5, b: 7
a: 5, b: 2
a: 3, b: 2
a: 1, b: 2
a: 1, b: 1
a: 1, b: 0
GCD(5, 7) = 1
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$


meganindya@Jupiter: ~/GitHub
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$ ./run.sh B1
Calculation of GCD by Euclid's Algorithm

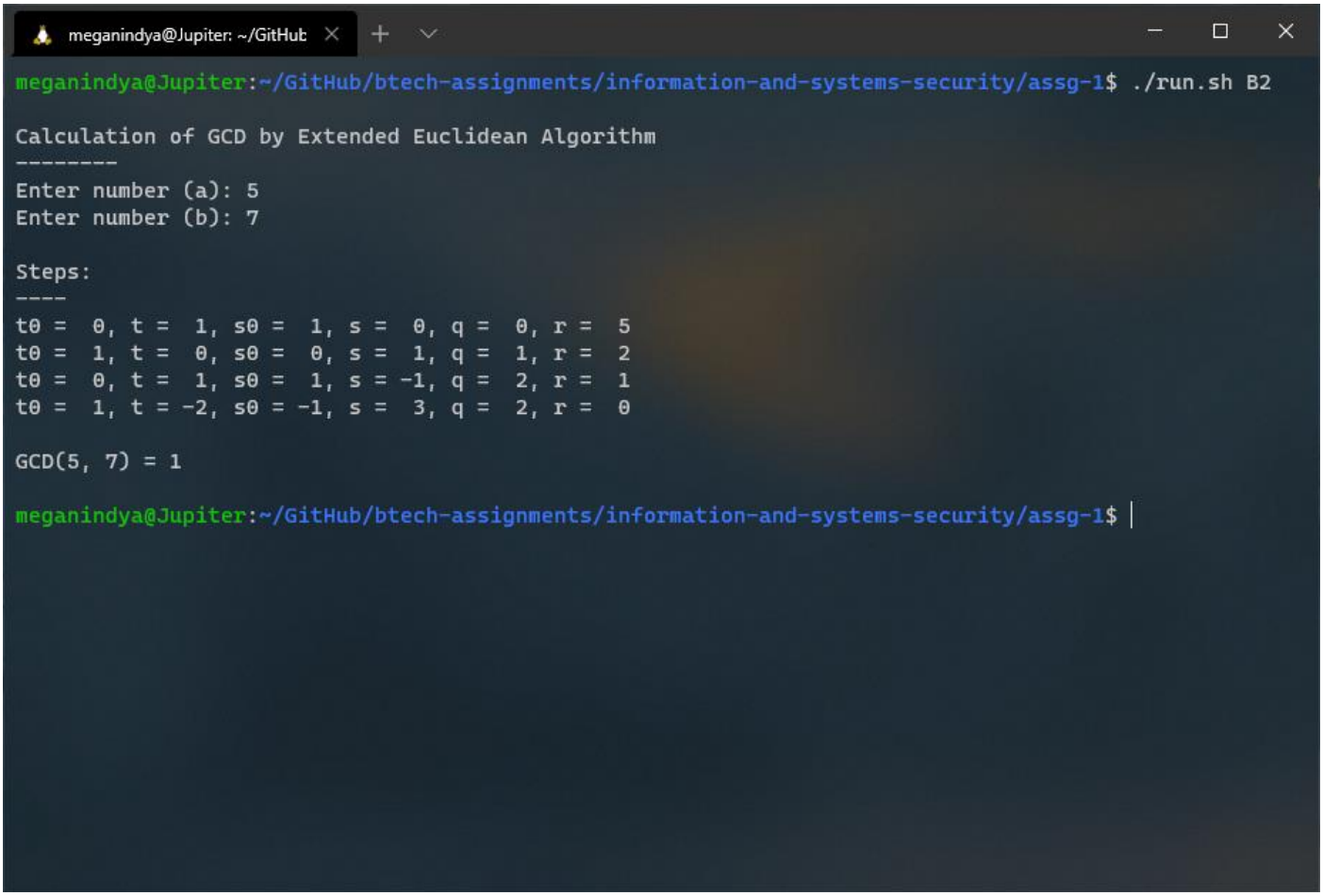
Enter number (a): 8
Enter number (b): 12
Steps:
----
a: 8, b: 12
a: 8, b: 4
a: 4, b: 4
a: 4, b: 0
GCD(8, 12) = 4
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$

2. Given two integers *a* and *b*, the following algorithm computes GCD (a, b) as well as b^{-1} mod a, when a and b are co-prime to each other. This is called the Extended Euclidean Algorithm.

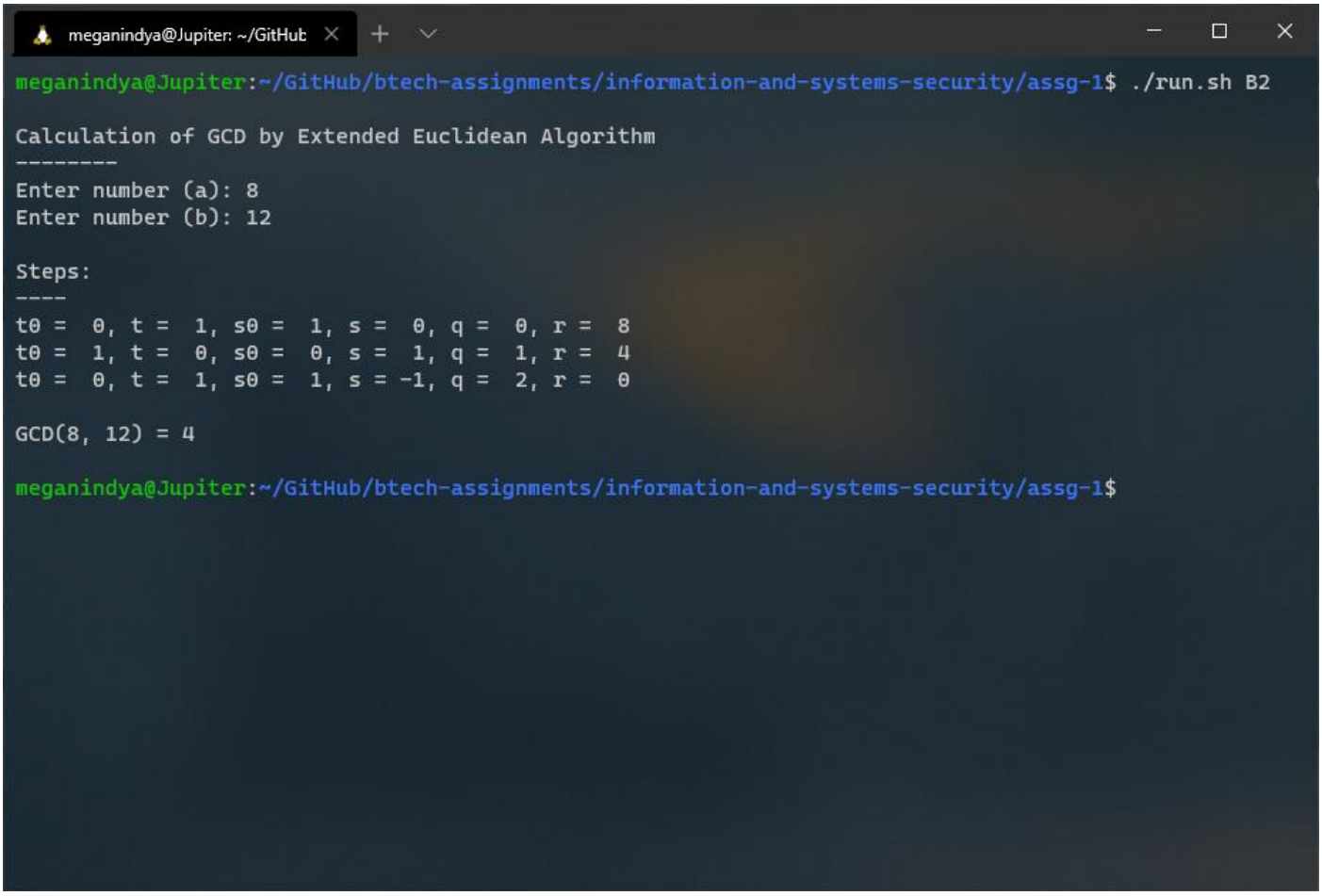
Source: **B-2-extended-euclid-algorithm.c**

```c
#include <stdio.h>

int gcd(int a, int b)
{
    printf("\nSteps:\n----\n");

    int t0 = 0;
    int t = 1;
    int s0 = 1;
    int s = 0;
    int q = a / b;
    int r = a - q * b;

    printf(
        "t0 = %2d, t = %2d, s0 = %2d, s = %2d, q = %2d, r = %2d \n",
        t0, t, s0, s, q, r);

    while (r > 0)
    {
        int temp = t0 - q * t;
        t0 = t;
        t = temp;
        temp = s0 - q * s;
        s0 = s;
        s = temp;
        a = b;
        b = r;
        q = a / b;
        r = a - q * b;

        printf(
         "t0 = %2d, t = %2d, s0 = %2d, s = %2d, q = %2d, r = %2d \n", t0, t, s0, s, q, r);
    }
    r = b;

    return r;
}

int main(int argc, char *argv[])
{
    int a, b;
    printf("\nCalculation of GCD by Extended Euclidean Algorithm\n--------\n");
    printf("Enter number (a): ");
    scanf("%d", &a);
    printf("Enter number (b): ");
    scanf("%d", &b);
    printf("\nGCD(%d, %d) = %d\n\n", a, b, gcd(a, b));
}
```

Sample run


meganindya@Jupiter: ~/GitHub
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$ ./run.sh B2
Calculation of GCD by Extended Euclidean Algorithm

Enter number (a): 5
Enter number (b): 7
Steps:
----
t0 = 0, t = 1, s0 = 1, s = 0, q = 0, r = 5
t0 = 1, t = 0, s0 = 0, s = 1, q = 1, r = 2
t0 = 0, t = 1, s0 = 1, s = -1, q = 2, r = 1
t0 = 1, t = -2, s0 = -1, s = 3, q = 2, r = 0
GCD(5, 7) = 1
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$


meganindya@Jupiter: ~/GitHub
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$ ./run.sh B2
Calculation of GCD by Extended Euclidean Algorithm

Enter number (a): 8
Enter number (b): 12
Steps:
----
t0 = 0, t = 1, s0 = 1, s = 0, q = 0, r = 8
t0 = 1, t = 0, s0 = 0, s = 1, q = 1, r = 4
t0 = 0, t = 1, s0 = 1, s = -1, q = 2, r = 0
GCD(8, 12) = 4
meganindya@Jupiter:~/GitHub/btech-assignments/information-and-systems-security/assg-1$