Assignment IV

Name: R.D.RAJEESWAR
Roll No: 24CSM2R11
CSIS

Question 1:
i) Find a generator g of a cyclic group G of order n.

Program:

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

void compute_totient(mpz_t result, const mpz_t num) {
    mpz_t i, temp, gcd;
    mpz_inits(i, temp, gcd, NULL);

    mpz_set(temp, num);
    mpz_set_ui(i, 2);
    mpz_set(result, num);

    while (mpz_cmp(i, temp) <= 0) {
        if (mpz_divisible_p(temp, i)) {
            mpz_sub_ui(gcd, i, 1);
            mpz_div(result, result, i);
            mpz_mul(result, result, gcd);
            mpz_div(temp, temp, i);
        } else {
            mpz_add_ui(i, i, 1);
        }
    }

    mpz_clears(i, temp, gcd, NULL);
}

int is_generator(const mpz_t candidate, const mpz_t modulus, const mpz_t phi) {
    mpz_t result, power, divisor;
    mpz_inits(result, power, divisor, NULL);

    mpz_t divisors[1000];
    size_t num_divisors = 0;

    mpz_t iterator, remainder;
    mpz_inits(iterator, remainder, NULL);
    mpz_set_ui(iterator, 1);

    while (mpz_cmp(iterator, phi) <= 0) {
```

```c
        mpz_mod(remainder, phi, iterator);
        if (mpz_cmp_ui(remainder, 0) == 0) {
            mpz_init(divisors[num_divisors]);
            mpz_set(divisors[num_divisors], iterator);
            num_divisors++;
        }
        mpz_add_ui(iterator, iterator, 1);
    }

    for (size_t i = 0; i < num_divisors; i++) {
        if (mpz_cmp_ui(divisors[i], 1) != 0 && mpz_cmp(divisors[i], phi) != 0) {
            mpz_powm(power, candidate, divisors[i], modulus);
            if (mpz_cmp_ui(power, 1) == 0) {
                for (size_t j = 0; j < num_divisors; j++) {
                    mpz_clear(divisors[j]);
                }
                mpz_clears(result, power, divisor, NULL);
                return 0;
            }
        }
    }

    for (size_t j = 0; j < num_divisors; j++) {
        mpz_clear(divisors[j]);
    }
    mpz_clears(result, power, divisor, NULL);
    return 1;
}

int main() {
    mpz_t modulus, totient, candidate;
    mpz_inits(modulus, totient, candidate, NULL);

    gmp_printf("Enter the modulus n: ");
    gmp_scanf("%Zd", modulus);

    compute_totient(totient, modulus);
    gmp_printf("φ(n) = %Zd\n", totient);

    for (mpz_set_ui(candidate, 2); mpz_cmp(candidate, modulus) < 0; mpz_add_ui(candidate, candidate, 1)) {
        if (is_generator(candidate, modulus, totient)) {
```

```
            gmp_printf("Generator for %Zd : %Zd\n",modulus, candidate);
            break;
        }
    }

    mpz_clears(modulus, totient, candidate, NULL);
    return 0;
}
```

Explanation:

Assumptions:

1. **Order of the Group**: We assume that nnn is the order of the group GGG. This means that the group GGG has exactly nnn elements.
2. **Cyclic Group**: We assume that GGG is a cyclic group. Every cyclic group has at least one generator.

Working :

1. Check for Primitive Root: An integer ggg is a primitive root modulo nnn if and only if the smallest positive integer kkk such that gk≡1(modn)g^k \equiv 1 \pmod{n}gk≡1(modn) is exactly nnn. In other words, ggg generates all integers from 111 to n−1n-1n−1.
2. Euler's Totient Function: Calculate ϕ(n)\phi(n)ϕ(n), where ϕ\phiϕ is Euler's totient function. The value ϕ(n)\phi(n)ϕ(n) is used to determine the order of elements in the group. For a prime nnn, ϕ(n)=n−1\phi(n) = n - 1ϕ(n)=n−1. For composite nnn, ϕ(n)\phi(n)ϕ(n) can be computed from the prime factors of nnn.
3. Verify Generator: To verify that ggg is a generator, ensure that gk≢1(modn)g^k \not\equiv 1 \pmod{n}gk□≡1(modn) for all kkk that are proper divisors of ϕ(n)\phi(n)ϕ(n).

3. **Input and Output:**

```
Enter the modulus n: 277
φ(n) = 276
Generator for 277 : 5


...Program finished with exit code 0
Press ENTER to exit console.
```

ii) Determine the order of a roup element a.

Program:

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

void compute_totient(mpz_t result, const mpz_t num) {
    mpz_t i, temp, gcd;
    mpz_inits(i, temp, gcd, NULL);

    mpz_set(temp, num);
    mpz_set_ui(i, 2);
    mpz_set(result, num);

    while (mpz_cmp(i, temp) <= 0) {
        if (mpz_divisible_p(temp, i)) {
            mpz_sub_ui(gcd, i, 1);
            mpz_div(result, result, i);
            mpz_mul(result, result, gcd);
            mpz_div(temp, temp, i);
        } else {
            mpz_add_ui(i, i, 1);
        }
    }

    mpz_clears(i, temp, gcd, NULL);
}

void compute_divisors(mpz_t *divisors, size_t *count, const mpz_t n) {
    mpz_t i, mod, zero;
    mpz_inits(i, mod, zero, NULL);
    mpz_set_ui(zero, 0);

    mpz_set_ui(i, 1);
    *count = 0;

    while (mpz_cmp(i, n) <= 0) {
        mpz_mod(mod, n, i);
        if (mpz_cmp(mod, zero) == 0) {
            mpz_init(divisors[*count]);
            mpz_set(divisors[*count], i);
            (*count)++;
        }
        mpz_add_ui(i, i, 1);
```

```
    }

    mpz_clears(i, mod, zero, NULL);
}

void finding_order(mpz_t a, mpz_t n) {
    mpz_t phi, i, temp;
    mpz_inits(phi, i, temp, NULL);

    compute_totient(phi, n);

    mpz_t divisors[1000];
    size_t count;
    compute_divisors(divisors, &count, phi);


    for (size_t j = 0; j < count; j++) {
        mpz_powm(temp, a, divisors[j], n);
        if (mpz_cmp_ui(temp, 1) == 0) {
            gmp_printf("Order of %Zd in Z_%Zd^* is %Zd\n", a, n, divisors[j]);
            break;
        }
    }

    for (size_t j = 0; j < count; j++) {
        mpz_clear(divisors[j]);
    }
    mpz_clears(phi, i, temp, NULL);
}

int main() {
    mpz_t a, n;
    mpz_inits(a, n, NULL);

    gmp_printf("Enter the modulus n: ");
    gmp_scanf("%Zd", n);

    gmp_printf("Enter the element a: ");
    gmp_scanf("%Zd", a);

    finding_order(a, n);
```

```
    mpz_clears(a, n, NULL);

    return 0;
}
```

Explanation:

Assumptions:

1. **Group Structure**: The group GGG is well-defined and its structure is known. Specifically, the group must have a defined identity element and operation (e.g., addition or multiplication).
2. **Element of the Group**: The element aaa is an element of the group GGG. This means aaa adheres to the group's operation and the group's defining properties.
3. **Identity Element**: The identity element of the group is known or can be determined. In the context of cyclic groups, the identity element is often 1 (for multiplication) or 0 (for addition).
4. **Group Order**: The order of the group GGG is known. This is the number of elements in the group, which helps in limiting the search for the order of the element aaa.

Working:

1. Identify the Identity Element: Ensure that you know the identity element eee of the group.
2. Compute Successive Powers (or Multiples): Calculate successive applications of the group operation on aaa (e.g., a1,a2,a3,...a^1, a^2, a^3, \ldotsa1,a2,a3,...) until the result equals the identity element eee.
3. Find the Smallest Integer: The smallest positive integer kkk for which ak=ea^k = eak=e is the order of the element aaa.

Output:

```
Enter the modulus n: 21
Enter the element a: 124
Order of 124 in Z_21^* is 6


...Program finished with exit code 0
Press ENTER to exit console.
```

Question 2:

Compute the multiplicative inverse of a given element a in ℤn (the set of integers modulo n), if it exists.

Program:

```
#include <stdio.h>


typedef struct {
    int gcd;
    int x;
    int y;
} ExtendedGCDResult;
```

```
ExtendedGCDResult extended_gcd(int a, int b) {
    ExtendedGCDResult result;
    if (b == 0) {
        result.gcd = a;
        result.x = 1;
        result.y = 0;
        return result;
    }
    ExtendedGCDResult temp = extended_gcd(b, a % b);
    result.gcd = temp.gcd;
    result.x = temp.y;
    result.y = temp.x - (a / b) * temp.y;
    return result;
}

int mod_inverse(int a, int n) {
    ExtendedGCDResult result = extended_gcd(a, n);
    if (result.gcd != 1) {
        return -1;
    }
    return (result.x % n + n) % n;
}

int main() {
    int a, n;
    printf("Enter a and n: ");
    scanf("%d %d", &a, &n);
    int inverse = mod_inverse(a, n);
    if (inverse == -1) {
        printf("No multiplicative inverse exists.\n");
    } else {
        printf("The multiplicative inverse of %d modulo %d is %d.\n", a, n, inverse);
    }
    return 0;
}
```

Explanation:

1. **Assumptions:**
   1. **Group Structure**: The set Zn\mathbb{Z}_nZn forms a group under multiplication if nnn is a positive integer, and aaa is an element of this set.
   2. **Existence of Inverse**: For an element aaa to have a multiplicative inverse modulo nnn, aaa and nnn must be coprime. In other words, their greatest common divisor (gcd) must be 1: gcd(a,n)=1\text{gcd}(a, n) = 1gcd(a,n)=1
   3. **Positive Modulus**: nnn is a positive integer greater than 1.
   4. **Element aaa**: The element aaa is a valid integer within the set Zn\mathbb{Z}_nZn, i.e., 0≤a<n0 \leq a < n0≤a<n.

2. **Working :**
   1. **Check Coprimality**: Verify that aaa and nnn are coprime using the greatest common divisor (gcd) function.
   2. **Use Extended Euclidean Algorithm**: Compute the inverse using the Extended Euclidean Algorithm. This algorithm not only finds the gcd of two numbers but also finds coefficients (including the multiplicative inverse) that satisfy Bézout's identity: a·x+n·y=gcd(a,n)a \cdot x + n \cdot y = \text{gcd}(a, n)a·x+n·y=gcd(a,n) For the inverse, this equation simplifies to: a·x≡1(modn)a \cdot x \equiv 1 \pmod{n}a·x≡1(modn) where xxx is the multiplicative inverse.

3.Input and Output:

```
Enter a and n: 13
123
The multiplicative inverse of 13 modulo 123 is 19.


...Program finished with exit code 0
Press ENTER to exit console.
```

Question 3:

Factorize the large integer n using the congruence of squares.

```
#include <gmp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

mpz_t random_number(mpz_t n)
{
  mpz_t r;
  mpz_init(r);
  mpz_urandomb(r, gmp_randstate_t, n);
  return r;
}

int is_prime(mpz_t n)
{
    return mpz_probab_prime_p(n, 25);
}

void congruence_of_squares(mpz_t n)
{
    mpz_t x, y, d;
    mpz_inits(x,y,d,NULL);
    mpz_set_ui(x, 2);
    mpz_set_ui(y, 2);
    while (1)
    {
        mpz_add_ui(x, x, 1);
        mpz_mod(x, x, n);
        mpz_set(y, x);
        mpz_add_ui(y, y, 1);
        mpz_mod(y, y, n);
        mpz_sub(d, x, y);
        mpz_abs(d, d);
        mpz_gcd(d, d, n);
        if (mpz_cmp(d, n) != 0 && mpz_cmp_ui(d, 1) != 0)
        {
            gmp_printf("Non-trivial factor found: %Zd\n", d);
            break;
        }
    }
    mpz_clears(x,y,d,NULL);
}

void prime_factorization(mpz_t n)
{
    if (is_prime(n))
    {
        gmp_printf("Prime number: %Zd\n", n);
        return;
    }
    congruence_of_squares(n);
```

```
    mpz_t factor;
    mpz_init(factor);
    mpz_t d;
    mpz_init(d);
    mpz_tdiv_q(factor, n, d);
    prime_factorization(factor);
    prime_factorization(d);
    mpz_clear(factor);
}
int main()
{
    mpz_t n;
    mpz_init(n);
    gmp_printf("enter the number:");
    gmp_scanf("%Zd",n)
    gmp_printf("Prime factorization of %Zd:\n", n);
    prime_factorization(n);
    mpz_clear(n);
    return 0;
}
```

Assumptions:

1. Integer nnn: The integer nnn to be factored is a composite number (i.e., it has factors other than 1 and itself).
2. Factor Size: The method is more effective if nnn has small prime factors. It may not be efficient for numbers with large prime factors.
3. Coprime Conditions: The chosen values in the method must be carefully selected to ensure that the factors are found

Explanation:

1. **Initialization**:
   - **GMP Library**: Used for handling large integers and modular arithmetic.
   - **Random State**: Initializes the random state for generating random values.
2. **Congruence of Squares Function (congruence_of_squares)**:
   - **Random Values**: Randomly selects integers xxx and yyy.
   - **Compute Squares**: Computes x2mod nx^2 \mod nx2modn and y2mod ny^2 \mod ny2modn.
   - **Difference**: Calculates the difference between these squares and takes the absolute value.
   - **GCD Calculation**: Computes the GCD of nnn and the difference. If the GCD is a non-trivial factor, it prints the factor and terminates.
3. **Main Function**:
   - **Input**: Reads the integer nnn from the user.
   - **Factorization**: Calls congruence_of_squares to attempt to factorize nnn.

input

```
Enter a positive integer: 102546
Prime factors: 2        3       3       3       3       3       211

...Program finished with exit code 0
Press ENTER to exit console.
```