1. Write a C++ program to generate the prime numbers within a given range.

```
    #include <iostream>
#include <vector>

// Function to generate prime numbers within a given range using Sieve of Eratosthenes
std::vector<int> generatePrimes(int start, int end) {
    std::vector<bool> isPrime(end + 1, true);
    std::vector<int> primes;

    for (int p = 2; p * p <= end; ++p) {
        if (isPrime[p]) {
            for (int i = p * p; i <= end; i += p) {
                isPrime[i] = false;
            }
        }
    }

    for (int p = std::max(2, start); p <= end; ++p) {
        if (isPrime[p]) {
            primes.push_back(p);
        }
    }

    return primes;
}

int main() {
    int start, end;

    // Input range
    std::cout << "Enter the starting range: ";
    std::cin >> start;
    std::cout << "Enter the ending range: ";
    std::cin >> end;

    // Generate and display prime numbers within the given range
    std::vector<int> primeNumbers = generatePrimes(start, end);

    std::cout << "\nPrime numbers between " << start << " and " << end << " are:" <<
std::endl;
    for (int prime : primeNumbers) {
        std::cout << prime << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Output:

```
PS C:\Users\stargaly galaxie> cd "d:\Desktop\System Security\System Security\" ; if ($?)
{ g++ 01.cpp -o 01 } ; if ($?) { .\01 }
Enter the starting range: 1
Enter the ending range: 20

Prime numbers between 1 and 20 are:
2 3 5 7 11 13 17 19
PS D:\Desktop\System Security\System Security>

Process finished with exit code 0
```

2. Write a C++ Program in Role-Based Access Control (RBAC) System .

```
#include <iostream>
#include <string>
#include <vector>
#include <map>

using namespace std;

class Role {
public:
    Role(string name) : name_(name) {}
    string name() const { return name_; }
    void add_permission(string permission) { permissions_.push_back(permission); }
    bool has_permission(string permission) const {
        for (const auto& perm : permissions_) {
            if (perm == permission) {
                return true;
            }
        }
        return false;
    }
private:
    string name_;
    vector<string> permissions_;
};

class User {
public:
    User(string name) : name_(name) {}
    string name() const { return name_; }
    void set_role(Role* role) { role_ = role; }
    bool has_permission(string permission) const {
        if (role_ != nullptr) {
            return role_->has_permission(permission);
        }
        return false;
    }
private:
    string name_;
    Role* role_ = nullptr;
};

class RBACSystem {
public:
    void add_role(Role* role) { roles_[role->name()] = role; }
    Role* get_role(string name) const {
        if (roles_.find(name) != roles_.end()) {
            return roles_.at(name);
        }
        return nullptr;
    }
    void add_user(User* user) { users_[user->name()] = user; }
    User* get_user(string name) const {
        if (users_.find(name) != users_.end()) {
```

```
            return users_.at(name);
        }
        return nullptr;
    }
private:
    map<string, Role*> roles_;
    map<string, User*> users_;
};

int main() {
    // Create roles
    Role* admin = new Role("admin");
    admin->add_permission("create_user");
    admin->add_permission("delete_user");
    admin->add_permission("update_user");

    Role* editor = new Role("editor");
    editor->add_permission("create_post");
    editor->add_permission("delete_post");
    editor->add_permission("update_post");

    // Create RBAC system
    RBACSystem rbac_system;
    rbac_system.add_role(admin);
    rbac_system.add_role(editor);

    // Create users
    User* alice = new User("alice");
    alice->set_role(admin);

    User* bob = new User("bob");
    bob->set_role(editor);

    // Check permissions
    cout << alice->name() << " has permission to create user: " <<
alice->has_permission("create_user") << endl;
    cout << alice->name() << " has permission to delete post: " <<
alice->has_permission("delete_post") << endl;
    cout << bob->name() << " has permission to create user: " <<
bob->has_permission("create_user") << endl;
    cout << bob->name() << " has permission to delete post: " <<
bob->has_permission("delete_post") << endl;

    // Clean up
    delete admin;
    delete editor;
    delete alice;
    delete bob;

    return 0;
}
```

Output:

```
 PS D:\Desktop\System Security\System Security> cd "d:\Desktop\System Security\System
Security\" ; if ($?) { g++ 2.cpp -o 2 } ; if ($?) { .\2 }
alice has permission to create user: 1
```

```
alice has permission to delete post: 0
bob has permission to create user: 0
bob has permission to delete post: 1
```

3. Write a C++ Program to decrypt the given plaintext messages with and selected decrypting Key using Caesar Cipher encryption technique.

```
#include <iostream>
#include <string>

// Function to decrypt a message using Caesar Cipher
std::string decryptCaesarCipher(const std::string& message, int key) {
    std::string decryptedMessage = "";

    for (char ch : message) {
        if (isalpha(ch)) {
            char base = (isupper(ch)) ? 'A' : 'a';
            decryptedMessage += static_cast<char>((ch - base - key + 26) % 26 + base);
        } else {
            decryptedMessage += ch; // Preserve non-alphabetic characters
        }
    }

    return decryptedMessage;
}

int main() {
    // Input ciphertext message
    std::string ciphertext;
    std::cout << "Enter the ciphertext message: ";
    std::getline(std::cin, ciphertext);

    // Input the Caesar cipher key for decryption
    int decryptKey;
    std::cout << "Enter the Caesar cipher decryption key (an integer): ";
    std::cin >> decryptKey;

    // Decrypt the message using the Caesar cipher
    std::string decryptedMessage = decryptCaesarCipher(ciphertext, decryptKey);

    // Display the results
    std::cout << "\nCaesar Cipher Decryption:" << std::endl;
    std::cout << "Ciphertext: " << ciphertext << std::endl;
    std::cout << "Decryption Key: " << decryptKey << std::endl;
    std::cout << "Decrypted Message: " << decryptedMessage << std::endl;

    return 0;
}
```

Output:

```
PS C:\Users\stargaly galaxie> cd "d:\Desktop\System Security\System Security\" ; if ($?)
{ g++ 03.cpp -o 03 } ; if ($?) { .\03 }
Enter the ciphertext message: saurav
Enter the Caesar cipher decryption key (an integer): 5
```

```
Caesar Cipher Decryption:
Ciphertext: saurav
Decryption Key: 5
Decrypted Message: nvpmvq
PS D:\Desktop\System Security\System Security>
```

4. Write a C++ program to generate Pseudo Random numbers in a range.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main() {
    int lower_bound = 10;
    int upper_bound = 20;

    // Seed the random number generator
    srand(time(nullptr));

    // Generate a random number in the range [lower_bound, upper_bound]
    int random_number = rand() % (upper_bound - lower_bound + 1) + lower_bound;

    cout << "Random number between " << lower_bound << " and " << upper_bound << ": " <<
random_number << endl;

    return 0;
}
```

Output:

```
PS D:\Desktop\System Security\System Security> cd "d:\Desktop\System Security\System
Security\" ; if ($?) { g++ 4.cpp -o 4 } ; if ($?) { .\4 }

Random number between 10 and 20: 11
```

5. Write a C++ to Encrypt and Decrypt the given plaintext messages using XOR operation.

```
#include <iostream>
#include <cstring>

using namespace std;

void encryptDecrypt(char inpString[]) {
    char xorKey = 'P';
    int len = strlen(inpString);
    for (int i = 0; i < len; i++) {
        inpString[i] = inpString[i] ^ xorKey;
    }
}

int main() {
    char sampleString[] = "Hello, world!";
```

```
    cout << "Original string: " << sampleString << endl;

    encryptDecrypt(sampleString);
    cout << "Encrypted string: " << sampleString << endl;

    encryptDecrypt(sampleString);
    cout << "Decrypted string: " << sampleString << endl;

    return 0;
}
```

Output:

```
PS D:\Desktop\System Security\System Security> cd "d:\Desktop\System Security\System
Security\" ; if ($?) { g++ 06.cpp -o 06 } ; if ($?) { .\06 }
Original string: Hello, world!
Encrypted string: ↑5<<?|p'?"<4q
Decrypted string: Hello, world!
```

6. Write a program in C++ for RSA algorithm taking p and q randomly.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>

using namespace std;

int main() {
    // Generate two random prime numbers
    srand(time(nullptr));
    int p = rand() % 100 + 1;
    int q = rand() % 100 + 1;

    // Calculate n and phi
    int n = p * q;
    int phi = (p - 1) * (q - 1);

    // Choose an integer e such that 1 < e < phi(n) and gcd(e, phi(n)) = 1
    int e = 2;
    while (e < phi) {
        if (__gcd(e, phi) == 1) {
            break;
        }
        e++;
    }

    // Calculate d as d ≡ e^-1 (mod phi(n))
    int d = 1;
    while (true) {
        if ((d * e) % phi == 1) {
            break;
        }
        d++;
    }

    // Print the public and private keys
    cout << "Public key: {" << e << ", " << n << "}" << endl;
    cout << "Private key: {" << d << ", " << n << "}" << endl;

    return 0;
}
```

Output:

```
"D:\Program Files\Python\Python310\python.exe" D:\Desktop\A.I\A.I\06.py
Public key: {7, 143}
Private key: {103, 143}
```

7. Write a C++ program to encrypt the given plaintext messages using Rail fence encryption technique.

```
#include <iostream>
#include <string>

std::string railFenceEncrypt(const std::string& plaintext, int rails) {
    int len = plaintext.length();
    char fence[rails][len];

    // Initialize fence matrix
    for (int i = 0; i < rails; i++)
        for (int j = 0; j < len; j++)
            fence[i][j] = '\0';

    // Fill the fence matrix
    bool directionDown = false;
    int row = 0, col = 0;
    for (int i = 0; i < len; i++) {
        if (row == 0 || row == rails - 1)
            directionDown = !directionDown;
        fence[row][col++] = plaintext[i];
        directionDown ? row++ : row--;
    }

    // Read from fence and generate ciphertext
    std::string ciphertext;
    for (int i = 0; i < rails; i++)
        for (int j = 0; j < len; j++)
            if (fence[i][j] != '\0')
                ciphertext.push_back(fence[i][j]);

    return ciphertext;
}

int main() {
    // Input plaintext and number of rails
    std::string plaintext;
    int rails;

    std::cout << "Enter the plaintext: ";
    std::getline(std::cin, plaintext);

    std::cout << "Enter the number of rails: ";
    std::cin >> rails;

    // Encrypt the plaintext using Rail Fence Cipher
    std::string ciphertext = railFenceEncrypt(plaintext, rails);

    // Display the results
    std::cout << "\nRail Fence Cipher Encryption:" << std::endl;
    std::cout << "Plaintext: " << plaintext << std::endl;
    std::cout << "Ciphertext: " << ciphertext << std::endl;

    return 0;
}
```

output:

```
PS C:\Users\stargaly galaxie> cd "d:\Desktop\System Security\System Security\" ; if ($?)
{ g++ 07.cpp -o 07 } ; if ($?) { .\07 }
Enter the plaintext: my name is saurav
Enter the number of rails: 3

Rail Fence Cipher Encryption:
Plaintext: my name is saurav
Ciphertext: maiavynm ssua e r
```

8. Write a program to implement the Diffie-Hellman key exchange algorithm.

```
#include <iostream>
#include <cmath>

// Function to calculate modular exponentiation (base^exp % modulus)
long long modPow(long long base, long long exp, long long modulus) {
    long long result = 1;
    base = base % modulus;

    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % modulus;
        }
        exp = exp >> 1;
        base = (base * base) % modulus;
    }

    return result;
}

// Function to perform the Diffie-Hellman key exchange
long long diffieHellman(long long base, long long private_key, long long prime) {
    return modPow(base, private_key, prime);
}

int main() {
    // Public parameters (usually chosen and shared in advance)
    long long base, prime;

    std::cout << "Enter the prime number (shared): ";
    std::cin >> prime;

    std::cout << "Enter the primitive root modulo prime (shared): ";
    std::cin >> base;

    // User 1's private key
    long long private_key_A;
    std::cout << "User 1: Enter your private key: ";
    std::cin >> private_key_A;

    // User 2's private key
    long long private_key_B;
    std::cout << "User 2: Enter your private key: ";
    std::cin >> private_key_B;

    // Calculate public keys
    long long public_key_A = diffieHellman(base, private_key_A, prime);
    long long public_key_B = diffieHellman(base, private_key_B, prime);

    // Shared secret key calculation
```

```
    long long secret_key_A = diffieHellman(public_key_B, private_key_A, prime);
    long long secret_key_B = diffieHellman(public_key_A, private_key_B, prime);

    // Display results
    std::cout << "\nPublic Keys:" << std::endl;
    std::cout << "User 1's Public Key: " << public_key_A << std::endl;
    std::cout << "User 2's Public Key: " << public_key_B << std::endl;

    std::cout << "\nShared Secret Keys:" << std::endl;
    std::cout << "User 1's Shared Secret Key: " << secret_key_A << std::endl;
    std::cout << "User 2's Shared Secret Key: " << secret_key_B << std::endl;

    return 0;
}
```

output:

```
PS C:\Users\stargaly galaxie> cd "d:\Desktop\System Security\System Security\" ; if ($?)
{ g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter the prime number (shared): 7
Enter the primitive root modulo prime (shared): 3
User 1: Enter your private key: 32
User 2: Enter your private key: 23

Public Keys:
User 1's Public Key: 2
User 2's Public Key: 5

Shared Secret Keys:
User 1's Shared Secret Key: 4
User 2's Shared Secret Key: 4
```

9. Write a C++ Program to encrypt the given plaintext messages with a randomly selected Key using Caeser Cipher encryption technique.

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>

// Function to encrypt a message using Caesar Cipher
std::string encryptCaesarCipher(const std::string& message, int key) {
    std::string encryptedMessage = "";

    for (char ch : message) {
        if (isalpha(ch)) {
            char base = (isupper(ch)) ? 'A' : 'a';
            encryptedMessage += static_cast<char>((ch - base + key) % 26 + base);
        } else {
            encryptedMessage += ch; // Preserve non-alphabetic characters
        }
    }

    return encryptedMessage;
}

int main() {
    // Seed for random number generation
    std::srand(std::time(0));

    // Input plaintext message
    std::string plaintext;
```

```
    std::cout << "Enter the plaintext message: ";
    std::getline(std::cin, plaintext);

    // Generate a random key in the range [1, 25]
    int key = std::rand() % 25 + 1;

    // Encrypt the message using the random key
    std::string encryptedMessage = encryptCaesarCipher(plaintext, key);

    // Display the results
    std::cout << "\nCaesar Cipher Encryption:" << std::endl;
    std::cout << "Plaintext: " << plaintext << std::endl;
    std::cout << "Key: " << key << std::endl;
    std::cout << "Encrypted Message: " << encryptedMessage << std::endl;

    return 0;
}
```

output:

```
PS C:\Users\stargaly galaxie> cd "d:\Desktop\System Security\System Security\" ; if ($?)
{ g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter the plaintext message: life is bad

Caesar Cipher Encryption:
Plaintext: life is bad
Key: 18
Encrypted Message: daxw ak tsv
```