Password Generator And Encryptor/ Decryptor in C++

"Enhancing password security through encryption"

"Developed in C++"

Introduction

What is the project about ?

This project focuses on creating a secure tool in C++ that generates random passwords and provides encryption and decryption functionality to safeguard sensitive information.

Why is it important ?

- Protects against password theft.
- Ensures sensitive data remains secure in storage or transit.
- Helps to maintain privacy and prevent unauthorized access.

Objectives..,

Primary Goals **:**

Generate strong, random passwords.

Encrypt passwords or sensitive data to enhance security.

Decrypt encrypted data when needed for access.

To provide secure passwords and protect sensitive data.

Focus on Usability and Security :

Simple interface for users.

Implements reliable and efficient algorithms.

Tools and Technologies..,

Programming Language : C++

Libraries/Functions Used :

<random> : For generating random numbers.

<string> : For handling and manipulating strings.

<iomanip> : For formatted output(if required).

Encryption Algorithm :

Example : XOR encryption Caesar Cipher, or a custom implementation.

Password Generation Logic..,

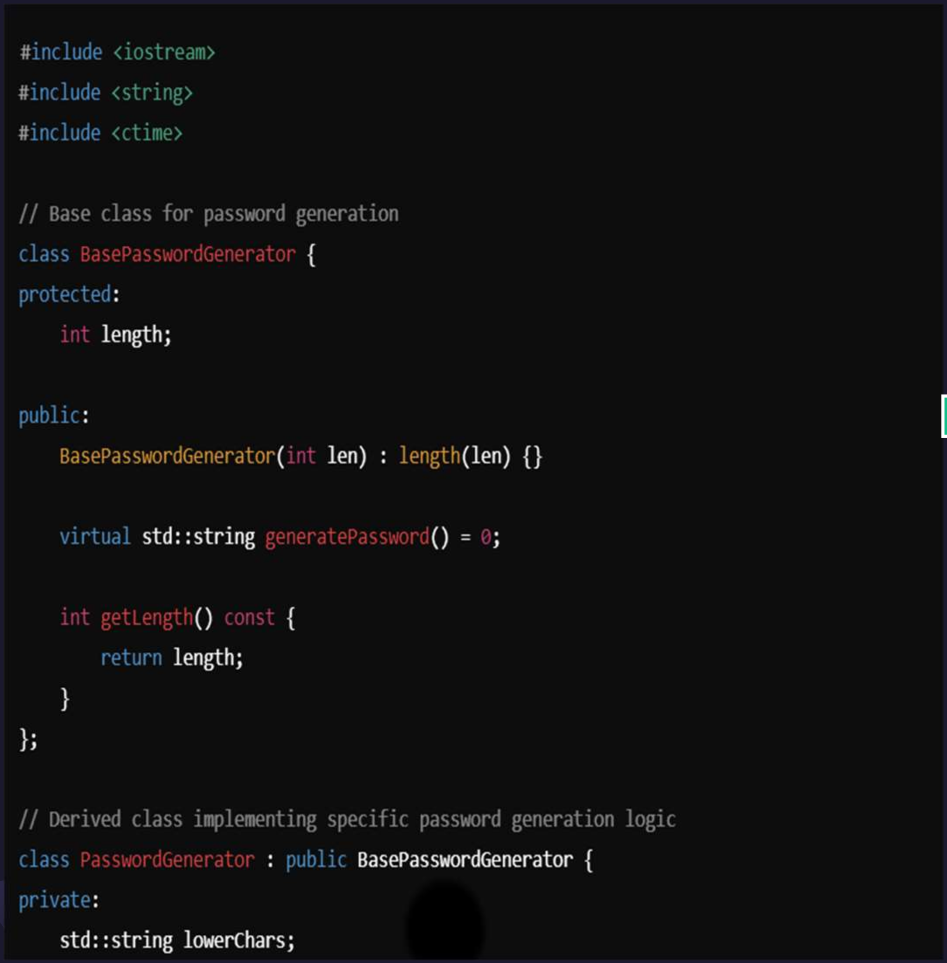
Steps involved :

Define character sets: uppercase, lowercase, digits, special symbols.

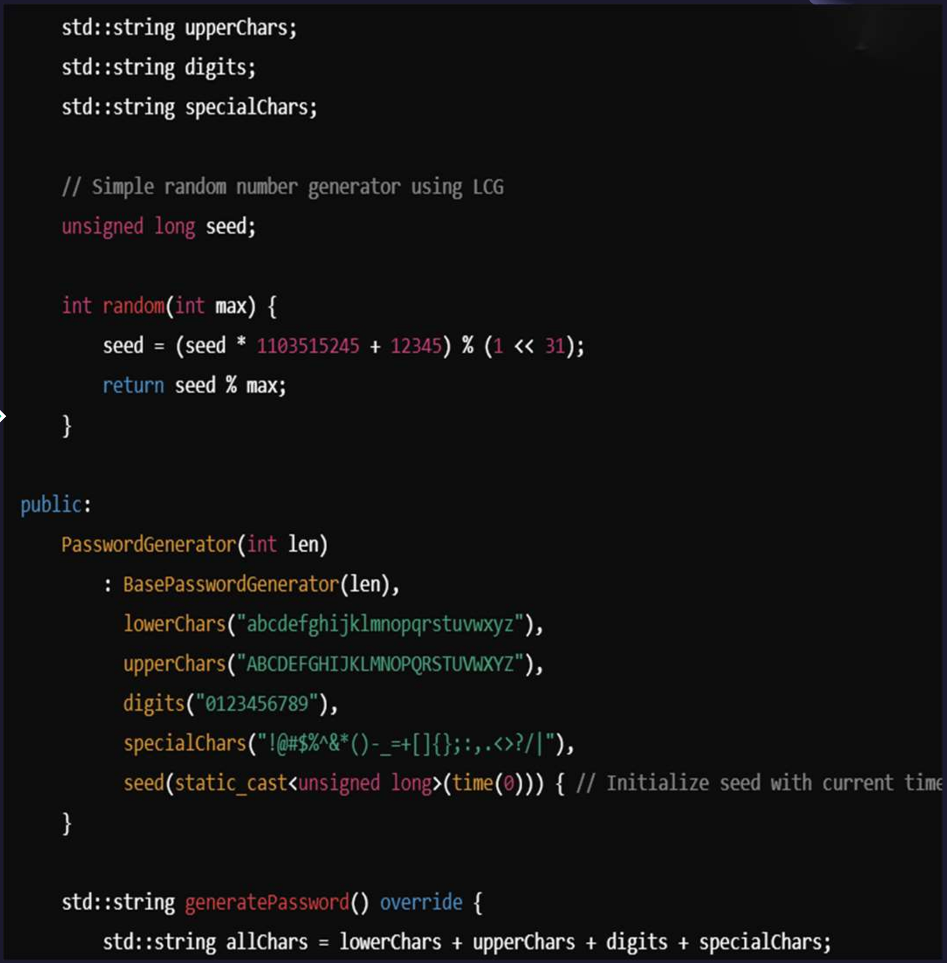
Use a random number generator to select characters from these sets.

Ensure the passwords meets strength criteria.

EXAMPLE OUTPUT : A random password like @d3f9h7#.



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

// Base class for password generation
class BasePasswordGenerator {
protected:
    int length;

public:
    BasePasswordGenerator(int len) : length(len) {}

    virtual std::string generatePassword() = 0;

    int getLength() const {
        return length;
    }
};

// Derived class implementing specific password generation logic
class PasswordGenerator : public BasePasswordGenerator {
private:
    std::string lowerChars;
```



```cpp
    std::string upperChars;
    std::string digits;
    std::string specialChars;

    // Simple random number generator using LCG
    unsigned long seed;

    int random(int max) {
        seed = (seed * 1103515245 + 12345) % (1 << 31);
        return seed % max;
    }

public:
    PasswordGenerator(int len)
        : BasePasswordGenerator(len),
          lowerChars("abcdefghijklmnopqrstuvwxyz"),
          upperChars("ABCDEFGHIJKLMNOPQRSTUVWXYZ"),
          digits("0123456789"),
          specialChars("!@#$%^&*()-_=+[]{};:,.<>?/|"),
          seed(static_cast<unsigned long>(time(0))) { // Initialize seed with current time
    }

    std::string generatePassword() override {
        std::string allChars = lowerChars + upperChars + digits + specialChars;
```

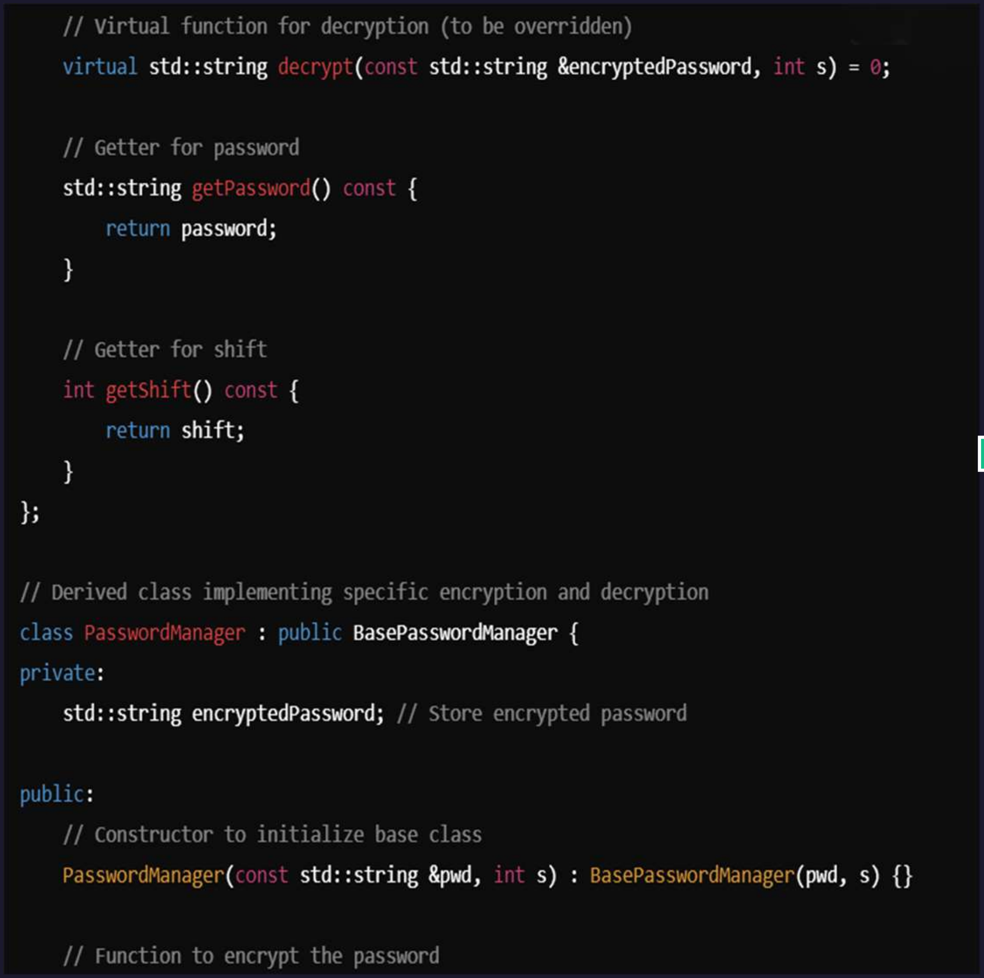
Encryption and Decryption Logic..,

Encryption Algorithm :

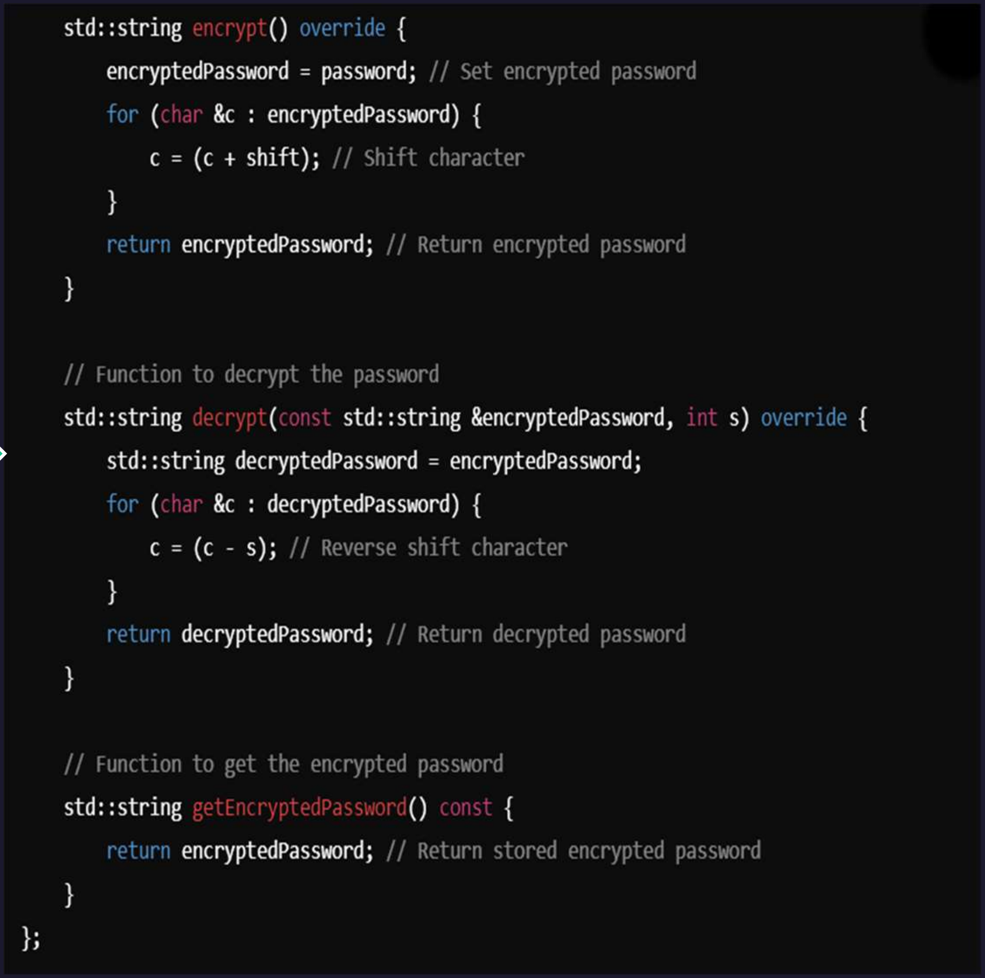
- Convert each character to its ASCII value.
- Perform a bitwise XOR operation with a key.
- Convert the result back to characters.

Decryption :

- Apply the XOR operation again using the same key to retrieve the original data.



```cpp
    // Virtual function for decryption (to be overridden)
    virtual std::string decrypt(const std::string &encryptedPassword, int s) = 0;

    // Getter for password
    std::string getPassword() const {
        return password;
    }

    // Getter for shift
    int getShift() const {
        return shift;
    }
};

// Derived class implementing specific encryption and decryption
class PasswordManager : public BasePasswordManager {
private:
    std::string encryptedPassword; // Store encrypted password

public:
    // Constructor to initialize base class
    PasswordManager(const std::string &pwd, int s) : BasePasswordManager(pwd, s) {}

    // Function to encrypt the password
```



```cpp
    std::string encrypt() override {
        encryptedPassword = password; // Set encrypted password
        for (char &c : encryptedPassword) {
            c = (c + shift); // Shift character
        }
        return encryptedPassword; // Return encrypted password
    }

    // Function to decrypt the password
    std::string decrypt(const std::string &encryptedPassword, int s) override {
        std::string decryptedPassword = encryptedPassword;
        for (char &c : decryptedPassword) {
            c = (c - s); // Reverse shift character
        }
        return decryptedPassword; // Return decrypted password
    }

    // Function to get the encrypted password
    std::string getEncryptedPassword() const {
        return encryptedPassword; // Return stored encrypted password
    }
};
```

Benefits/Advantages and Applications..,

Advantages/Benefits :

1. Simple to implement .
2. Provides basic data security .
3. Stronger than the user chosen passwords .
4. Difficult to guess .

Applications :

1. Secure account setups .
2. File encryption .
3. Personal data protection .

Conclusion..,

Successfully implemented a tool for generating secure passwords and encrypting/decrypting them.

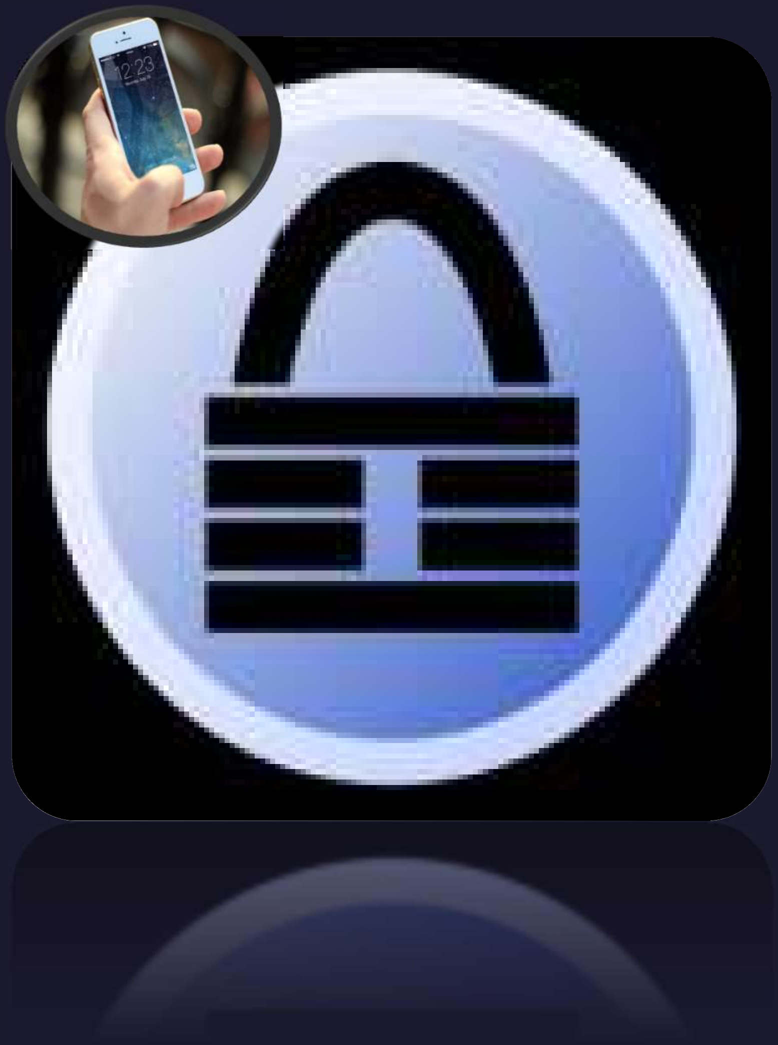
Demonstrates the power and the flexibility of C++ for secure application development.

Future enhancements :

Implementing advanced encryption methods like AES.

Adding a graphical user interface(GUI).

Storing encrypted passwords securely in a database.



Thank you.