C++프로그래밍및실습

암호화 프로그램

최종 보고서

제출일자: 2024/12/22
제출자명: 정철원
제출자학번: 214771

1. 프로젝트 목표

1) 배경 및 필요성

데이터 보안과 개인 정보 보호가 중요한 이슈가 되면서, 정보를 안전하게 전송하고 보관하기 위한 암호화의 필요성이 높아지고 있습니다. 개인이나 기업이 데이터를 암호화하지 않고 전송하거나 저장할 경우, 해킹이나 데이터 유출로 인한 피해가 발생할 수 있습니다. 특히, 데이터를 파일 형태로 저장할 때 이를 암호화하여 보호하는 방법은 필수적입니다. 이 문제를 해결하기 위해 사용자들이 손쉽게 사용할 수 있는 간단한 암호화 프로그램이 필요합니다.

2) 프로젝트 목표

사용자가 입력한 텍스트를 암호화하여 파일에 저장하고, 이 파일에 저장된 데이터를 복호화하여 원본 데이터로 복원하는 프로그램을 구현하는 것을 목표로 합니다. 간단한 암호화 및 복호화 기능을 제공하여, 데이터 보안을 강화하고 사용자가 정보를 안전하게 전달할 수 있도록 돕습니다.

3) 차별점

일반적으로 복잡한 암호화 소프트웨어는 사용하기 어렵고 설정이 복잡한 경우가 많습니다. 본 프로그램은 누구나 쉽게 사용할 수 있도록 단순한 사용 절차를 제공하며, 입문자도 쉽게 암호화/복호화 기능을 이용할 수 있는 것이 특징입니다.

2. 기능 계획

1) 기능 1 : 기본 암호화 및 복호화 구현

- 설명 : 사용자가 입력한 텍스트를 일정한 규칙(예: 시저 암호)을 이용해 암호화하고 복호화하는 기능입니다. 특정 시프트 값을 입력하면, 각 문자가 해당 시프트 값만큼 이동하여 암호화되고, 암호화에 사용된 시프트 값을 입력하면, 암호화된 텍스트를 다시 원본으로 복호화할 수 있습니다.

(1) 세부 기능 1 : EncryptCaesar 함수

- 설명 : 입력된 텍스트를 암호화합니다.

(2) 세부 기능 2 : DecryptCaesar 함수

- 설명 : 암호화된 텍스트를 복호화합니다.

2) 기능 2 : 암호화된 파일의 다중 텍스트 저장

- 설명 : 파일에 여러 줄의 암호화된 텍스트를 저장할 수 있도록 기능을 확장합니다.

(1) 세부 기능 1 : 암호화된 데이터를 파일에 추가로 저장

- 설명 : 파일에 암호화된 데이터를 추가로 저장 가능합니다.

(2) 세부 기능 2 : 여러 줄의 데이터를 읽고 복호화하는 기능

- 설명 : 복호화 시 모든 데이터를 처리할 수 있습니다.

3) 기능 3 : 암호화 강도 조절 (랜덤 시프트 값)

- 설명 : 암호화 강도를 높이기 위해 랜덤한 시프트 값을 생성하여 데이터를 암호화합니다.

(1) 세부 기능 1 : 무작위 시프트 값을 생성하여 암호화하고 복호화에 사용할 수 있도록 함

- 설명 : 랜덤 시프트 값으로 강력한 암호화가 가능합니다.

4) 기능 4 : 암호화 파일 보호 (비밀번호 설정 및 확인)

- 설명 : 파일 접근 전 비밀번호를 설정하고 확인하여 보안을 강화합니다.

(1) 세부 기능 1 : 사용자가 비밀번호를 설정

- 설명 : 비밀번호 보호 기능을 통해 파일 접근 보안을 강화한다.

5) 기능 5 : 암호화된 데이터 삭제 기능

- 설명 : 파일에 저장된 암호화 데이터를 삭제하여 민감 정보를 안전하게 처리합니다.

(1) 세부 기능 1 : 암호화된 데이터를 저장한 파일 삭제

- 설명 : 암호화 데이터를 안전하게 삭제하여 민감 정보를 보호 가능합니다.

3. 진척사항

1) 기능 구현

(1) CaesarCipher 클래스

적용된 배운 내용 : 접근 제어(10주차), 클래스와 객체(9주차)

- **EncryptCaesar 함수**

```
// 시저 암호로 텍스트를 암호화하는 함수
string EncryptCaesar(const string &text, int shift){
    string encrypted_text = text;
    for (char &c : encrypted_text){
        if (isalpha(c)){ // 알파벳만 암호화
            char base = isupper(c) ? 'A' : 'a';
            c = (c - base + shift) % 26 + base;
        }
    }
    return encrypted_text;
}
```

입출력

- const string &text = 암호화할 문자열, 띄어쓰기와 숫자는 암호화되지 않으며, 영어 알파벳만 암호화됨
- int shift = 알파벳을 시피트할 값, shift가 3일 경우, 'A'는 'D로 'Z'는 'C'로 암호화됨
- encrypted_text = 암호화된 결과를 저장할 변수
- base = 기준이 되는 위치를 저장 (A 또는 a를 저장)

반환값

- 암호화된 문자열을 반환

결과

- 사용자가 입력한 문자열에서 영어 알파벳 문자는 입력한 시프트 값만큼 이동한 결과를 반환

설명

- for루프를 통해 문자열의 각 문자를 순회하고 isalpha 함수를 통해 문자가 알파벳인지 확인(알파벳만 암호화 대상임)

- isupper 함수를 통해 대소문자를 구분하고 삼항 연산자로 기준 문자를 저장함

- (c- base + shift) % 26 + base에서 c에서 base를 빼 문자의 상대적 위치를 구한 뒤 shift를 더해 26으로 나누어 알파벳의 범위를 유지함, 다시 base를 더해 시프트된 문자 값을 얻음

적용된 배운 내용

- 4주차(조건문 반복문), 6주차(함수), 7주차(문자열)

- **DecryptCaesar 함수**

```
// 시저 암호로 텍스트를 복호화하는 함수
string DecryptCaesar(const string &text, int shift){
    return EncryptCaesar(text, 26 - shift); // 복호화는 암호화의 반대
}
```

입출력

- const string &text : 복호화할 암호화된 문자열

- int shift : 암호화 시 사용된 시프트 값

반환값

- 복호화된 문자열을 반환

결과

- 시저 암호로 암호화된 문자열에서 시프트 값을 반대로 적용하여 원래 텍스트를 복구함

설명

- 암호화 시 사용한 시프트 값 shift를 26에서 뺀 값을 사용해 암호화와 동일한 방식으로 처리하면 복호화가 됨. 내부적으로 EncryptCaesar 함수를 재사용하여, 복호화와 암호화를 동일한 방식으로 처리

적용된 배운내용

- 6주차(함수), 7주차(문자열)

- **SetPassWord 함수**

```
public:
    // 비밀번호 설정 함수
    void SetPassword() {
        cout << "새 비밀번호를 설정하세요: ";
        getline(cin, password);
        cout << "비밀번호가 설정되었습니다." << endl;
    }
```

입출력

- password : 비밀번호 입력

결과

- password 멤버 변수에 사용자가 입력한 비밀번호가 저장됨

설명

- 사용자가 새 비밀번호를 입력하면, getline을 통해 문자열 전체가 password 변수에 저장됨
- 이후 프로그램 내에서 이 비밀번호를 사용해 접근을 제어함
- 비밀번호는 프로그램이 끝날때까지 유지됨

적용된 배운 내용

- 함수(6주차)

- **VerifyPassword 함수**

```
// 비밀번호 확인 함수
bool VerifyPassword() {
    if (password.empty()) return true; // 비밀번호가 설정되지 않은 경우 바로 접근

    string entered_password;
    cout << "비밀번호를 입력하세요: ";
    getline(cin, entered_password);

    if (entered_password == password) {
        cout << "비밀번호가 확인되었습니다." << endl;
        return true;
    } else {
        cout << "비밀번호가 일치하지 않습니다!" << endl;
        return false;
    }
}
```

입출력

- password : 설정된 비밀번호

- enterd_password : 비밀번호 확인 사용자 입력 비밀번호

결과, 반환값

- 입력한 비밀번호가 설정된 비밀번호가 일치하면 메시지 출력 후 true를 반환

- 비밀번호가 일치하지 않다면 일치x 메시지 출력후 false를 반환

설명

- password.empty()를 통해 비밀번호가 설정되지 않았는지 확인

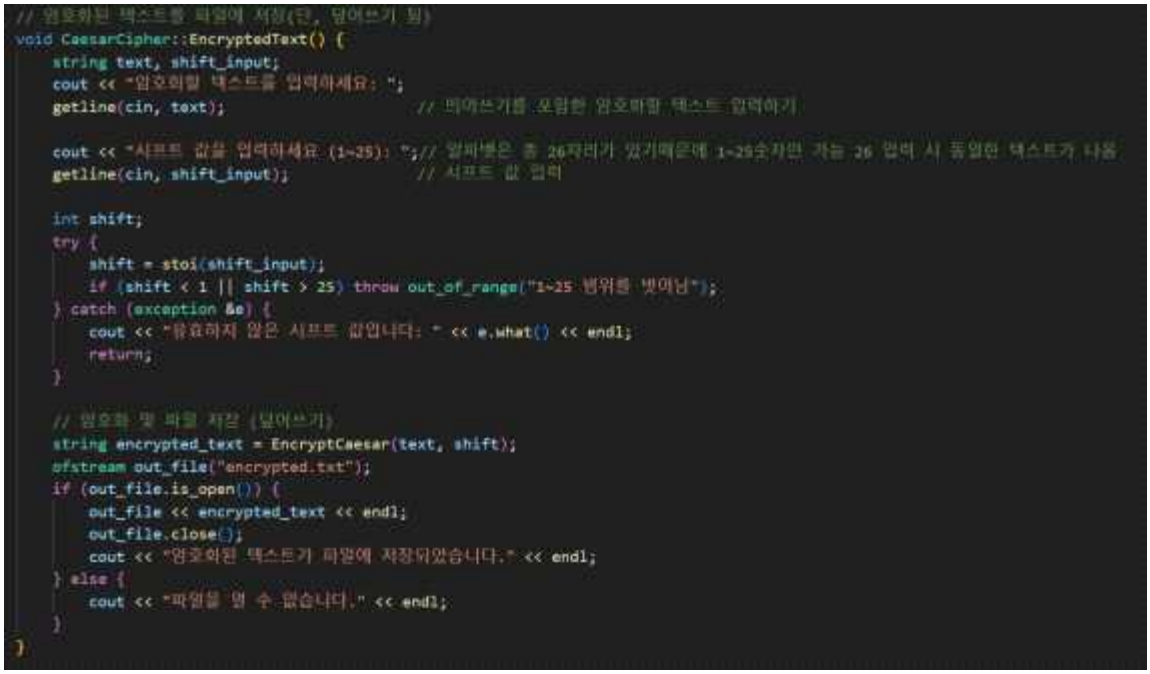
- 설정된 비밀번호가 있을 경우, 사용자가 입력한 값과 비교하여 일치 여부를 확인

- 비밀번호가 일치해야 기능 실행 가능

적용된 배운 내용

- 함수(6주차), 문자열(7주차), 조건문 (4주차)

- **EncryptedText 함수**



```
// 암호화된 텍스트를 파일에 저장(단, 덮어쓰기 됨)
void CaesarCipher::EncryptedText() {
    string text, shift_input;
    cout << "암호화할 텍스트를 입력하세요: ";
    getline(cin, text);                    // 띄어쓰기를 포함한 암호화할 텍스트 입력하기

    cout << "시프트 값을 입력하세요 (1~25): ";// 알파벳은 총 26자리가 있기때문에 1~25숫자만 가능 26 입력 시 동일한 텍스트가 나옴
    getline(cin, shift_input);             // 시프트 값 입력

    int shift;
    try {
        shift = stoi(shift_input);
        if (shift < 1 || shift > 25) throw out_of_range("1~25 범위를 벗어남");
    } catch (exception &e) {
        cout << "유효하지 않은 시프트 값입니다: " << e.what() << endl;
        return;
    }

    // 암호화 및 파일 저장 (덮어쓰기)
    string encrypted_text = EncryptCaesar(text, shift);
    ofstream out_file("encrypted.txt");
    if (out_file.is_open()) {
        out_file << encrypted_text << endl;
        out_file.close();
        cout << "암호화된 텍스트가 파일에 저장되었습니다." << endl;
    } else {
        cout << "파일을 열 수 없습니다." << endl;
    }
}
```

입출력

- text : 암호화 또는 복호화 대상 텍스트
- shift : 알파벳 이동 시 사용할 시프트 값
- shift_input : 알파벳 이동 시 사용할 시프트 값(문자열형)
- encrypted_text = 암호화된 결과를 저장할 변수

결과

- 사용자가 입력한 텍스트를 시저 암호 방식으로 암호화하고 encrypted.txt 파일에 저장, 기존 파일 내용은 덮어쓰기됨.

설명

- cin을 사용하지 않고 getline만을 사용하기에 stoi함수를 통해 문자열을 int형 정수로 바꿔줌
- getline을 통해 암호화할 텍스트와 시프트 값을 입력받음
- EncryptCaesar 함수를 호출하여 암호화 결과를 반환받아 출력
- ofstream 객체를 통해 파일을 열고, 암호화된 텍스트를 파일에 작성 (만약 ios::app 없이 파일을 열면 기존 내용은 덮어쓰기됨.)
- 파일 저장이 완료되면 성공 메시지를 출력하고, 파일을 열 수 없으면 에러 메시지를 출력

- 잘못된 시프트 값 입력 시 예외 처리를 통해 오류 메시지를 출력

적용된 배운 내용

- 조건문(4주차), 함수(6주차), 파일시스템, 예외(14주차)

- **AddEncryptedText 함수**

```
// 암호화된 텍스트를 추가로 파일에 저장
void CaesarCipher::AddEncryptedText() {
    string text, shift_input;
    cout << "추가로 암호화할 텍스트를 입력하세요: ";
    getline(cin, text);

    cout << "시프트 값을 입력하세요 (1~25): ";
    getline(cin, shift_input);

    int shift;
    try {
        shift = stoi(shift_input);
        if (shift < 1 || shift > 25) throw out_of_range("1~25 범위를 벗어남");
    } catch (exception &e) {
        cout << "유효하지 않은 시프트 값입니다: " << e.what() << endl;
        return;
    }

    string encrypted_text = EncryptCaesar(text, shift);
    ofstream out_file("encrypted.txt", ios::app); // 추가 모드
    if (out_file.is_open()) {
        out_file << encrypted_text << endl;
        out_file.close();
        cout << "암호화된 텍스트가 파일에 추가되었습니다." << endl;
    } else {
        cout << "파일을 열 수 없습니다." << endl;
    }
}
```

입출력

- mode : 암호화 복호화 모드를 선택하는 정수

- shift : 알파벳 이동 시 사용할 시프트 값

- shift_input : 알파벳 이동 시 사용할 시프트 값(문자열형)

- encrypted_text = 암호화된 텍스트를 입력하는 변수

결과

- 입력받은 텍스트를 암호화하여 encrypted.txt 파일에 추가 저장, 기존 파일 내용은 유지됨

설명

- cin을 사용하지 않고 getline만을 사용하기에 stoi함수를 통해 문자열을 int형 정수로 바꿔줌
- getline을 통해 추가 암호화할 텍스트를 입력받고, shift 값을 입력받음
- EncryptCaesar 함수를 호출하여 입력된 텍스트를 암호화함
- ofstream 객체를 ios::app 모드로 열어, 기존 파일 내용 뒤에 암호화된 텍스트를 추가로 저장함
- 파일 저장이 완료되면 성공 메시지를 출력하고, 파일을 열 수 없으면 에러 메시지를 출력함
- 잘못된 시프트 값 입력 시 예외 처리를 통해 오류 메시지를 출력

적용된 배운 내용

- 조건문(4주차), 함수(6주차), 파일시스템, 예외(14주차)

- **RandomEncryptText 함수**

```
// 랜덤 강도 암호화
void RandomEncryptText() {
    string text;
    cout << "암호화할 텍스트를 입력하세요: ";
    getline(cin, text);

    srand(time(0));
    int random_shift = rand() % 25 + 1; // 1 ~ 25 사이의 무작위 값
    cout << "랜덤 시프트 값: " << random_shift << endl;

    string encrypted_text = EncryptCaesar(text, random_shift);
    ofstream out_file("encrypted.txt", ios::app);
    if (out_file.is_open()) {
        out_file << encrypted_text << endl;
        out_file.close();
        cout << "암호화된 텍스트가 파일에 저장되었습니다." << endl;
    } else {
        cout << "파일을 열 수 없습니다." << endl;
    }
}
```

입출력

- text : 암호화할 텍스트

- random_shift : 무작위로 설정된 1~25사이의 시프트 값

결과

- 랜덤 시프트 값 출력후 encrypted.txt 파일에 암호화된 문자 저장

- 후에 암호화된 텍스트가 파일에 저장되었음을 츨략

설명

- rand()함수를 사용하여 1~25사이의 랜덤 시프트 값을 생성

- 암호화된 결과는 기존 파일에 추가 저장

적용된 배운 내용

- 함수(6주차), 조건문 (4주차), 파일시스템(14주차)

- **DeleteEncryptedFile 함수**

```
// 파일 삭제 함수
void DeleteEncryptedFile() {
    if (remove("encrypted.txt") == 0) {
        cout << "파일이 삭제되었습니다." << endl;
    } else {
        cout << "파일을 삭제할 수 없습니다." << endl;
    }
}
```

입출력

- 별도의 입력 없음

결과

- encrypted.txt 파일이 삭제됨
- 파일 삭제 후 "파일이 삭제되었습니다." 출력

설명

- remove 함수를 사용해 파일을 삭제
- 파일이 삭제되면 성공 메시지를 출력하고, 실패 시 오류 메시지 출력

적용된 배운 내용

- 함수(6주차), 조건문 (4주차), 파일시스템(14주차)

- **DecryptFile 함수**

```cpp
// 복호화 함수
void CaesarCipher::DecryptFile() {
    ifstream in_file("encrypted.txt");
    if (in_file.is_open()) {
        string encrypted_text, shift_input;
        cout << "복호화할 시프트 값을 입력하세요 (1~25): ";
        getline(cin, shift_input);

        int shift;
        try {
            shift = stoi(shift_input);
            if (shift < 1 || shift > 25) throw out_of_range("1~25 범위를 벗어남");
        } catch (exception &e) {
            cout << "유효하지 않은 시프트 값입니다: " << e.what() << endl;
            return;
        }

        while (getline(in_file, encrypted_text)) {
            string decrypted_text = DecryptCaesar(encrypted_text, shift);
            cout << "복호화된 텍스트: " << decrypted_text << endl;
        }
        in_file.close();
    } else {
        cout << "파일을 열 수 없습니다." << endl;
    }
}
```

입출력

- shift : 알파벳 이동 시 사용할 시프트 값

- shift_input : 알파벳 이동 시 사용할 시프트 값(문자열형)

- encrypted_text = 암호화된 텍스트를 입력하는 변수

- decrypted_text = 복호화된 결과를 저장할 변수

결과

- encrypted.txt 파일에 저장된 모든 텍스트를 한 줄씩 읽어와 복호화한 결과를 출력

설명

- 복호화할 시프트 값을 입력받음

- ifstream 객체로 파일을 열고, 모든 텍스트를 한 줄씩 읽음

- 각 줄에 대해 DecryptCaesar 함수로 복호화한 결과를 출력

- 파일을 열 수 없으면 에러 메시지를 출력

- 잘못된 시프트 값 입력 시 예외 처리를 통해 오류 메시지를 출력

적용된 배운 내용

- 조건문(4주차), 함수(6주차), 파일시스템, 예외(14주차)

(2) main 함수

```
int main() {
    CaesarCipher cipher;
    string choice_input;

    while (true) {
        cout << endl << "=== 암호화 프로그램 ===" << endl;
        cout << "1. 암호화된 텍스트 저장(덮어쓰기)" << endl;
        cout << "2. 암호화된 텍스트 추가 저장" << endl;
        cout << "3. 암호화된 텍스트 랜덤 강도로 저장" << endl;
        cout << "4. 암호화된 파일 삭제" << endl;
        cout << "5. 비밀번호 설정" << endl;
        cout << "6. 암호화된 파일 읽기 및 복호화" << endl;
        cout << "0. 종료" << endl;
        cout << "선택: ";
        getline(cin, choice_input);

        int choice;
        try {
            choice = stoi(choice_input);
        } catch (exception &e) {
            cout << "유효하지 않은 선택입니다." << endl;
            continue;
        }

        switch (choice) {
        case 1:
            if (cipher.VerifyPassword()) cipher.EncryptedText();
            break;
        case 2:
            if (cipher.VerifyPassword()) cipher.AddEncryptedText();
            break;
        case 3:
            if (cipher.VerifyPassword()) cipher.RandomEncryptText();
            break;
        case 4:
            if (cipher.VerifyPassword()) cipher.DeleteEncryptedFile();
            break;
        case 5:
            if (cipher.VerifyPassword()) cipher.SetPassword();
            break;
        case 6:
            if (cipher.VerifyPassword()) cipher.DecryptFile();
            break;
        case 0:
            cout << "프로그램을 종료합니다." << endl;
            return 0;
        default:
            cout << "잘못된 선택입니다." << endl;
        }
    }
}
```

입출력

- choice : 사용자가 입력하는 기능 번호(int형)
- choice_input : 사용자가 입력하는 기능 번호(문자열)
- cipher : CaesarCipher 클래스 객체 생성

결과

- 사용자의 선택에 따라 각 기능이 실행됨
- 0 입력 시 프로그램이 종료됨

설명

- 메뉴는 무한 루프 안에서 반복적으로 표시됨
- 사용자가 입력한 메뉴 번호는 stoi()를 통해 문자열에서 정수로 변환됨
- switch문을 통해 선택된 기능이 실행됨
- 사용자가 입력한 값이 잘못된 경우, dafault 케이스가 실행되어 오류 메시지를 출력
- 잘못된 모드(특수문자 or 문자) 선택 시 예외 처리를 통해 오류 메시지를 출력

적용된 배운 내용

- 조건문, 반복문(4주차), 함수(6주차), 클래스와 객체(7주차) 파일시스템, 예외(14주차)

2) 프로그램 실행 방법

- 사용자가 1~6번까지의 기능을 무한 루프를 통해 입력하여 암호화하고자 하는 문자를 파일에 암호화하여 저장하거나 저장된 암호화 문자를 복호화할 수 있음. 이때, 파일에 덮어쓸 수도, 추가 저장할 수도 있음. 사용자가 직접 시프트값을 입력하지 않고 랜덤 시프트값을 받아 암호화할 수도 있으며, 모든 기능에 접근하는데 비밀번호를 설정할 수도 있음. 파일을 삭제할 수도 있음

4. 테스트 결과

(1) CaesarCipher 클래스

- **EncryptCaesar 함수 (암호화)**

-대소문자를 구문하고 공백, 특수문자, 숫자는 암호화 하지 않음, 암호화 한뒤 파일에 저장, 덮어쓰기

```
선택: 1
암호화할 텍스트를 입력하세요: Hello World!!
시프트 값을 입력하세요 (1~25): 2
암호화된 텍스트가 파일에 저장되었습니다.
```

```
Jgnnq Yqtnf!!
```

- **DecryptCaesar 함수 (복호화)**

-대소문자를 구문하고 공백, 특수문자, 숫자는 복호화 하지 않음

```
선택: 6
복호화할 시프트 값을 입력하세요(1~25): 2
복호화된 텍스트: Hello World!!
```

- **SetPassWord 함수 (비밀번호 설정)**

```
선택: 5
새 비밀번호를 설정하세요: 1234
비밀번호가 설정되었습니다.
```

- **VerifyPassword 함수 (비밀번호 확인)**

-일치했을 때

```
선택: 5
비밀번호를 입력하세요: 1234
비밀번호가 확인되었습니다.
새 비밀번호를 설정하세요: 12345
비밀번호가 설정되었습니다.
```

-일치하지 않았을 때

```
선택: 5
비밀번호를 입력하세요: 123
비밀번호가 일치하지 않습니다!
```

- **EncryptedText 함수 (암호화 문자 파일저장, 덮어쓰기)**

```
선택: 1
암호화할 텍스트를 입력하세요: Hello World!!
시프트 값을 입력하세요 (1~25): 2
암호화된 텍스트가 파일에 저장되었습니다.
```

```
Jgnnq Yqtnf!!
```

-예외 발생

```
선택: 1
암호화할 텍스트를 입력하세요: hello
시프트 값을 입력하세요 (1~25): 30
유효하지 않은 시프트 값입니다: 1~25 범위를 벗어남
```

- **AddEncryptedText 함수 (암호화 문자 파일 추가저장)**

```
선택: 2
비밀번호를 입력하세요: 12345
비밀번호가 확인되었습니다.
추가로 암호화할 텍스트를 입력하세요: cheolwon
시프트 값을 입력하세요 (1~25): 2
암호화된 텍스트가 파일에 추가되었습니다.
```

```
Jgnnq Yqtnf!!
ejgqnyqp
```

-예외 발생

```
선택: 2
추가로 암호화할 텍스트를 입력하세요: hello
시프트 값을 입력하세요 (1~25): 30
유효하지 않은 시프트 값입니다: 1~25 범위를 벗어남
```

- **RandomEncryptText 함수 (랜덤 시프트 암호화)**

```
선택: 3
비밀번호를 입력하세요: 12345
비밀번호가 확인되었습니다.
암호화할 텍스트를 입력하세요: Jeong
랜덤 시프트 값: 5
암호화된 텍스트가 파일에 저장되었습니다.
```

```
Jgnnq Yqtnf!!
ejgqnyqp
Ojtsl
```

- **DeleteEncryptedFile 함수 (파일 삭제)**

```
선택: 4
비밀번호를 입력하세요: 1234
비밀번호가 확인되었습니다.
파일이 삭제되었습니다.
```

- **DecryptFile 함수 (파일 복호화)**

```
선택: 6
비밀번호를 입력하세요: 12345
비밀번호가 확인되었습니다.
복호화할 시프트 값을 입력하세요(1~25): 2
복호화된 텍스트: Hello World!!
복호화된 텍스트: cheolwon
복호화된 텍스트: Mhrqj
```

```
선택: 6
비밀번호를 입력하세요: 12345
비밀번호가 확인되었습니다.
복호화할 시프트 값을 입력하세요(1~25): 5
복호화된 텍스트: Ebiil Tloia!!
복호화된 텍스트: zeblitlk
복호화된 텍스트: Jeong
```

-예외 발생

```
선택: 6
복호화할 시프트 값을 입력하세요 (1~25): 30
유효하지 않은 시프트 값입니다: 1~25 범위를 벗어남
```

(2) main 함수

- 메뉴 출력

```
=== 암호화 프로그램 ===
1. 암호화된 텍스트 저장(덮어쓰기)
2. 암호화된 텍스트 추가 저장
3. 암호화된 텍스트 랜덤 강도로 저장
4. 암호화된 파일 삭제
5. 비밀번호 설정
6. 암호화된 파일 읽기 및 복호화
0. 종료
선택:
```

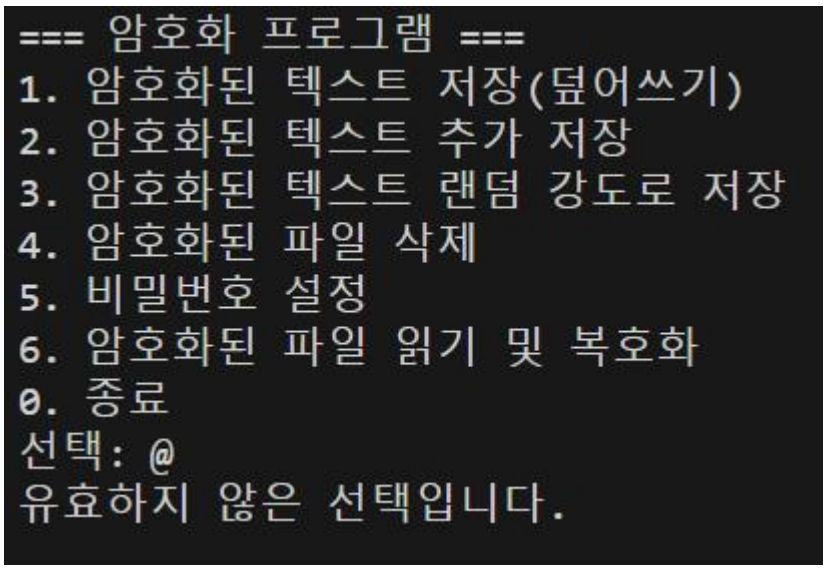
-0~6 값을 선택 안했을때

```
선택: 8
잘못된 선택입니다.
```

-0을 선택 했을 때

```
=== 암호화 프로그램 ===
1. 암호화된 텍스트 저장(덮어쓰기)
2. 암호화된 텍스트 추가 저장
3. 암호화된 텍스트 랜덤 강도로 저장
4. 암호화된 파일 삭제
5. 비밀번호 설정
6. 암호화된 파일 읽기 및 복호화
0. 종료
선택: 0
프로그램을 종료합니다.
PS C:\CPP2409-P>
```

-특수문자나 문자를 입력할 때



5. 계획 대비 변경 사항

1) 전체적인 기능 여럿 추가 (11/17)

- 이전 : 기존에는 암호화, 복호화, 파일 저장 기능 밖에 없었다

- 이후 : 암호화 강도 조절, 암호화 파일 비밀번호 설정, 암호화 파일 삭제 기능을 추가하였다.

- 사유 : 암호화, 복호화 기능들로만은 보안 프로그램의 세밀성을 전부 표현할 순 없을거 같고, 프로젝트를 진행하다보니 다양한 기능을 구현해보고 싶은 욕심이 생겼기 때문이다.

2) 마지막 주차에 클래스 생성 및 최종 실행 과정 검토예정 (12/01)

- 이전 : 클래스를 생성하고 오류가 발생하진 않았지만 내가 계획한 대로 제대로 작동하는지 검토하는 단계를 계획에 포함시키지 않았음

- 이후 : 마지막 최종 보고서 제출 전에 클래스를 생성해 코드를 간략화 하고, 최종 검토를 통해 내가 의도한 대로 코드가 잘 실행하는지 검토할 예정

- 사유 : 코드를 간략화해 더 보기 쉽고, 유지보수가 쉬운 코드를 만들고, 내가 계획한대로 프로젝트를 잘 실행했는지 검토하기 위함.

3) 버퍼 문제 해결 및 클래스 개념 도입(12/15)

- 이전 : 버퍼 문제가 발생해 암호화된 문자를 파일에 저장할 때 맨 앞에 있던 문자 하나가 생략되어 출력되는 문제가 발생하였음, 클래스를 도입하지 않아 main함수가 쓸때없이 길고 지저분하였음

- 이후 : 버퍼 문제로 발생된 문제를 해결하였음, 클래스를 도입하여 main함수를 더욱 깔끔하게 볼 수 있음

- 설명 : 버퍼 문제가 보통 cin과 getline을 병행하여 사용해 나타나는데 cin.ignore로도 해결이 되지않아 cin을 사용하지 않고 getline만을 사용하여 코드를 작성함, stoi 함수를 사용하여 문자열을 int형으로 변환하여 이 문제를 해결함

- 사유 : 코드를 간략화해 더 보기 쉽고, 유지보수가 쉬운 코드를 만들기 위함

4) 예외 처리를 통해 오류 대응(12/22)

- 이전 : 이전에는 모드 선택시 숫자를 제외한 특수문자나 문자를 입력하면 에러를 출력하지 않았음

- 이후 : 특수문자나 문자를 입력해도 똑같이 오류를 출력함

- 사유 : 코드로 인해 프로그램이 더 견고해지고 사용자 입력 오류나 파일 작업 실패에 안정적으로 대응하기위함

6. 프로젝트 일정

업무		11/3	11/17	11/24	12/1
제안서 작성		완료			
기능1	세부기능1,2	완료			
기능2	세부기능1		완료		
	세부기능2			완료	

업무		12/1	12/8	12/15	12/22
기능3	세부기능1	완료			
기능4	세부기능1		완료		
기능5 및 최종검토	세부기능1			완료	

7. 느낀점

이번 프로젝트를 통해 암호화의 기본 개념과 프로그램 설계 과정을 직접 경험할 수 있었습니다. 특히, 시저 암호라는 간단한 알고리즘을 사용했지만, 이를 활용하여 다양한 기능(텍스트 암호화, 복호화, 파일 저장, 비밀번호 보호 등)을 구현하면서 실생활에서 암호화 기술이 어떻게 활용되는지 체감할 수 있었습니다.

프로젝트를 진행하며 직면했던 가장 큰 어려움은 기능 추가와 코드의 복잡성 사이에서 균형을 맞추는 일이었습니다. 기능을 추가할수록 코드가 복잡해지고 디버깅이 어려워졌지만, 이를 해결하기 위해 코드를 기능별로 나누고 주석과 문서를 통해 가독성을 높이려고 노력했습니다. 또한, 암호화와 복호화 과정에서 발생할 수 있는 입력 데이터의 유효성 검증 및 파일 입출력 오류를 처리하며, 사용자가 프로그램을 더 직관적으로 사용할 수 있도록 개선하는 데 집중했습니다.

이번 프로젝트는 단순히 암호화 프로그램을 만드는 것을 넘어, 문제를 해결하는 과정에서의 논리적 사고와 프로그래밍 기술을 발전시키는 계기가 되었습니다. 프로젝트를 마친 후, 코드가 단순히 동작하는 것을 넘어 사용성과 확장성을 고려하는 것이 얼마나 중요한지 깨달았습니다. 또한, 암호화 기술이 실생활에서 얼마나 중요한지 다시 한번 실감하며, 이러한 기술을 활용한 프로그램을 직접 설계하고 구현했다는 점에서 큰 성취감을 느낄 수 있었습니다.