Classic Cipher 실습

1) Affine Cipher 소스 코드, 동작 과정

[소스코드] Caesar_cipher.c

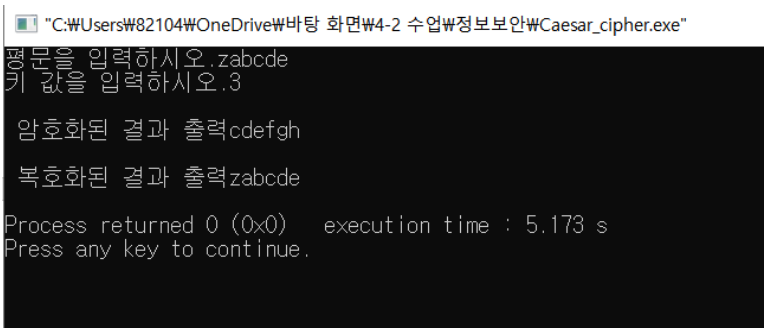
```c
#include <stdio.h>
char *CaesarCipher(char *, int, int); // 암호화 Encryption
char *CaesarCipher2(char *, int, int); // 복호화 Decryption
int main(void){
    int i = 0, key = 0, str_size = 0;
    char str[50] = { 0, };
    printf("평문을 입력하시오.");
    gets(str);
    printf("키 값을 입력하시오.");
    scanf("%d", &key);
    str_size = strlen(str);

    CaesarCipher(str, str_size, key);
    // 암호화 결과 출력
    printf("\n 암호화된 결과 출력");
    puts(str);
    // 암호화 시킨 것 다시 복호화.
    CaesarCipher2(str, str_size, key);
    printf("\n 복호화된 결과 출력");
    puts(str);
    return 0;
}
// 암호화
char *CaesarCipher(char *str, int str_size, int key){
    int i;
    for(int i = 0; i < str_size; ++i){
        // 문자가 알파벳 범위 안에 있으면 아스키코드에 해당하는 숫자로 변경시켜준다. A -> 0 Z ->
25
        // 대소문자 구분없이 처리되게 하기 위해서 이런 방식을 사용.
        if(str[i] > 'A' && str[i] <= 'Z'){
            str[i] -= 'A';
            if((str[i] + key) < 0)
                str[i] += 26;
            str[i] = (str[i] + key) % 26;
            // 다시 알파벳 형식으로 변경.
            str[i] += 'A';
        // 마찬가지 방식으로 소문자 처리.
        }else if(str[i] >= 'a' && str[i] <= 'z'){
            str[i] -= 'a';
            if((str[i] +key) < 0)
                str[i] += 26;
            str[i] = (str[i] + key) % 26;
            str[i] += 'a';
        }
    }
}
// 복호화
char *CaesarCipher2(char *str, int str_size, int key){
    int i;
    for(int i = 0; i < str_size; ++i){
        // 문자가 알파벳 범위 안에 있으면 아스키코드에 해당하는 숫자로 변경시켜준다. A -> 0 Z ->
25
        // 대소문자 구분없이 처리되게 하기 위해서 이런 방식을 사용.
        if(str[i] > 'A' && str[i] <= 'Z'){
            str[i] -= 'A';
```

```
            // 복호화의 경우 음수가 될 수 있으므로 이를 방지하고 cycle 처럼 처리되도록 하기 위해서
음수가 될 경우에는 26 를 더한다.
            if((str[i] - key) < 0)
                str[i] += 26;
            str[i] = (str[i] - key) % 26;
            // 다시 알파벳 형식으로 변경.
            str[i] += 'A';
        // 마찬가지 방식으로 소문자 처리.
        }else if(str[i] >= 'a' && str[i] <= 'z'){
            str[i] -= 'a';
            if((str[i] - key) < 0)
                str[i] += 26;
            str[i] = (str[i] - key) % 26;
            str[i] += 'a';
        }
    }
}
```

[동작 과정]

1. 평문을 입력받는다.
2. CaesarCipher 함수를 이용해서 암호화(Encryption)시킨다.
   - 대소문자를 구분해서 따로 처리해준다.
   - 쉽게 처리하기 위해서 대, 소문자 각각 -'A', -'a'로 숫자로 만들어 준 후에 + key를 해준다.
3. CaesarCipher2 함수를 이용해서 복호화(Decryption)시킨다.
   - 2번과 동일하게 처리한다. -를 해주기 때문에 음수값이 나올 수 있어 음수가 나올 경우 +26을 해서 cycle처럼 처리되도록 한다.
4. 평문을 입력받으면, 암호화를 시키고 출력, 다시 복호화를 하여 평문 그대로를 출력해주어 암호화, 복호화가 제대로 처리된 것을 확인할 수 있었다.

[결과]



2) Gronsfeld Cipher 소스 코드, 동작 과정

[소스코드] Gronsfeld_cipher.c

```c
#include <stdio.h>
void Vigenere(int, char *, int, char *, int);
int main(void){
    int Key_Size = 0, Str_Size = 0, select = 1;
    char str[50] = { 0, }, key[16] = { 0, };
    printf("암호문 또는 평문을 입력 : ");
    gets(str);
    printf("암호는 1 번, 복호는 2 번 선택 : ");
    scanf("%d", &select);
    // scanf()에서 '\n'이 표준 입출력의 버퍼에 남아있을 수 있으므로 fflush()를 통해 버퍼를
비운다.
    fflush(stdin);
    printf("키 값 입력(4 자) : ");
    gets(key);
    // 자리에 맞는 문자열 - 키 값을 대응시켜 연산시키기 위해서 문자열의 길이를 이용한다.
    Str_Size = strlen(str);
    Key_Size = strlen(key);
    Vigenere(select, str, Str_Size, key, Key_Size);
    printf("\n 암호화 또는 복호화되 결과 출력 :");
    puts(str);
    return 0;
}
void Vigenere(int select, char *str, int Str_Size, char *key, int Key_Size){
    int i = 0, j = 0;
    for(i = 0; i < Str_Size; ++i){
        // 할당된 문자열의 위치를 고려해서. 넘으면 Key_Size 에 대한 나머지를 구해서 cycle 의 자리를
찾는다.
        j = i % Key_Size;
        // 암호화(Encryption)
        if(select == 1){
            // 대소문자 나눠서 처리하기 편하게 아스키코드로 빼서 숫자로 계산하고 이후에
처리해주었다.
            if(str[i] >= 'a' && str[i] <= 'z'){
                str[i] -= 'a';
                // gronsfeld 에서는 숫자를 입력했기 때문에 -'a'대신에 -'0'으로 이동거리를
맞춰준다.
                key[j] -= '0';
                if(str[i] + key[j] < 0){
                    str[i] += 26;
                }
                str[i] = (str[i] + key[j]) % 26;
                str[i] += 'a';
                key[j] += '0';
            }else if(str[i] >= 'A' && str[i] <= 'Z'){
                str[i] -= 'A';
                // gronsfeld 에서는 숫자를 입력했기 때문에 -'a'대신에 -'0'으로 이동거리를
맞춰준다.
                key[j] -= '0';
                if(str[i] + key[j] < 0){
                    str[i] += 26;
                }
                str[i] = (str[i] + key[j]) % 26;
                str[i] += 'A';
                key[j] += '0';
            }
        }
        // 복호화(Decryption)
        if(select == 2){
            // 대소문자 나눠서 처리하기 편하게 아스키코드로 빼서 숫자로 계산하고 이후에
처리해주었다.
            if(str[i] >= 'a' && str[i] <= 'z'){
                str[i] -= 'a';
```

```
                // gronsfeld 에서는 숫자를 입력했기 때문에 -'a'대신에 -'0'으로 이동거리를
맞춰준다.
                key[j] -= '0';
                if(str[i] - key[j] < 0){
                    str[i] += 26;
                }
                str[i] = (str[i] - key[j]) % 26;
                str[i] += 'a';
                key[j] += '0';
            }else if(str[i] >= 'A' && str[i] <= 'Z'){
                str[i] -= 'A';
                // gronsfeld 에서는 숫자를 입력했기 때문에 -'a'대신에 -'0'으로 이동거리를
맞춰준다.
                key[j] -= '0';
                if(str[i] - key[j] < 0){
                    str[i] += 26;
                }
                str[i] = (str[i] - key[j]) % 26;
                str[i] += 'A';
                key[j] += '0';
            }
        }
    }
}
```

[동작과정]

1.

- 암호문 또는 평문을 입력받는다.
- 암호화는 1번, 복호화는 2번을 선택하여 원하는 동작을 수행할 수 있다.
- 키 값을 입력받는다.

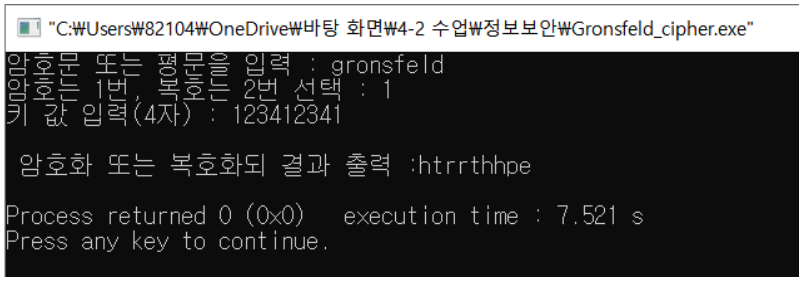
2. Vigenere 함수를 이용해서 암호화/복호화를 수행한다. i는 문자열의 길이만큼 루프를 돌리고, 내부에 j라는 변수를 이용(j = I % Key_Size)해서 cycle에 맞는 암호화 및 복호화를 수행하도록 한다.

3. 대, 소문자에 따라 따로 -'A', -'a'를 하는 방식으로 따로 처리해주고, 숫자로 만들어서 연산한다. 복호화의 경우 key값을 빼서 음수가 되는 경우에는 + 26을 통해 양수를 만들어준다

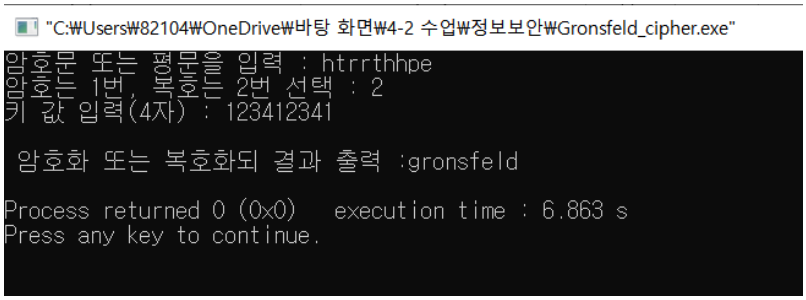
4. 결과를 출력한다.

[결과]

- 암호화(Encryption)



- 복호화(Decryption)



3) Rail Fence Cipher 소스 코드, 동작 과정

[소스코드] Rail_fence_cipher_text.c

```c
#include <stdio.h>

void Encryption(char *str, int key, int size);
void Decryption(char *str, int key, int size);
int main(void){
        char text[100];
        int size = 0, select = 0, key = 0;
   printf("암호는 1 번, 복호는 2 번 선택 : ");
   scanf("%d", &select);
   if(select == 1){
       printf("평문 입력 : ");
       fflush(stdin);
       scanf("%s", text);
       printf("키 입력 : ");
       scanf("%d", &key);
       size = strlen(text);
       Encryption(text, key, size);
   }else if(select == 2){
       printf("평문 입력 : ");
       fflush(stdin);
       scanf("%s", text);
       printf("키 입력 : ");
       scanf("%d",&key);
       size = strlen(text);
       Decryption(text, key, size);
   }
        return 0;
}
void Encryption(char *str, int key, int size){
        // Rail Matrix
        char R[key][size];
        // 빈공간 표현
        for(int i = 0; i < key; ++i)
      for(int j = 0; j < size; ++j)
         R[i][j] = '_';
        // 지그재그로 원하는 위치에 encoding 시키기 위해
        int flag = 1;
        for(int i = 0, j = 0; j < size; i+=flag, ++j){
                if(i == 0)
                        flag = 1;
                else if(i == key-1)
                        flag = -1;
                R[i][j] = str[j];
        }
        // 출력
```

```
        printf("암호문 : ");
        for(int i = 0; i < key; ++i){
                for(int j = 0; j < size; ++j){
                        if(R[i][j] != '_'){
                                printf("%c", R[i][j]);
                        }
                }
        }
        printf("\n");
}
void Decryption(char *str, int key, int size){
        // Rail Matrix
        char R[key][size];
        // 빈공간 표현
        for(int i = 0; i < key; ++i)
      for(int j = 0; j < size; ++j)
         R[i][j] = '_';
        // 지그재그로 원하는 위치에 decoding 시키기 위해
        int flag = 1;
        for(int i = 0, j = 0; j < size; i+=flag, ++j){
                if(i == 0)
                        flag = 1;
                else if(i == key-1)
                        flag = -1;
                R[i][j] = '@';
        }
        // '@'로 표시된 공간을 ciphertext를 채워준다.
        int k = 0;
        for(int i = 0; i < key; ++i){
                for(int j = 0; j < size; ++j){
                        if(R[i][j] == '@'){
                                R[i][j] = str[k];
                                k += 1;
                        }
                }
        }
        // 출력
        printf("복호문 : ");
        for(int i = 0; i < size; ++i){
                for(int j = 0; j < key; ++j){
                        if(R[j][i] != '_'){
                                printf("%c", R[j][i]);
                        }
                }
        }
        printf("\n");
}
```

[동작 과정]

1. 평문을 입력받는다. key값(rail)을 입력받는다.

2. Rail Fence cipher의 원리를 이용한 Encryption 함수를 이용해서 암호화를 수행한다.

   - Rail Matrix를 구성하고, '_'를 이용해 표시해준다.

   - for 루프를 구성하여 key만큼의 길이만큼 지그재그로 처리해주기 위해서 flag라는 임의의 변수를 이용하여 저장해준다

   - '_'가 아닌 위치에 이전에 저장해주었던 것을 출력해준다.

3. Decryption

- 2번과 동일하게 '_'를 이용해 표시해준다.

- for 루프를 이용하여 암호화에서는 저장해주었지만 여기서는 자리('@')를 표시해준다.

- for 루프를 이용하여 '@'로 표시된 공간에 복호화시킨 코드를 채워준다.

- 출력

4. 결과를 확인할 수 있다.

[결과]