**psh010209@gc.gachon.ac.kr, 010-7661-7035**

**SANGHUI PARK (박상희)**

# **개발 과제 #1 :**

|  |  |
| --- | --- |
|  | **Write a C program to compress a telephone book file** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Find duplicate data (consider that program already knows what is duplicate)  
Write to the file with compression key on in front of the file.**

* **Step 2: Outline a solution – 간단한 logic설명**

**For each line, read digits and check whether it is able to compress via provided compression key.**

**If it is possible, take compression key and save it at in front of the file.**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**For each line, Remove compression key, write to file.**

* **Step 4: write a pseudo code – 수도코드 작성**

**Read lines from input file**

**Write compression key to output file**

**For each lines,**

**Check line starts with compression key**

**If true,**

**Write string after the compression key to output file**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  *#include* <string.h>  int main() {  *// open file pointer*  FILE \*original = fopen("phone\_book.txt", "r");  FILE \*compressed = fopen("compressed.txt", "w");  *// exception handling*  *if* (original == NULL) {  printf("[ERROR] couldn't open phone\_book.txt!\nCheck disk S.M.A.R.T. or file permissions!\n");  *return* 1;  }  *// exception handling*  *if* (compressed == NULL) {  printf("[ERROR] couldn't open compressed.txt!\nCheck disk S.M.A.R.T. or file permissions!\n");  *return* 1;  }  *// set compression key*  char compressKey[10] = "77";  *// write the compression key at the beginning of the file.*  fprintf(compressed, "%s\n", compressKey);  *// define temporary character arrat for storing current line.*  char tmp[10] = "";  *// define a pointer to check for compressionKey and remove.*  *for* (char \*a = tmp; !feof(original); a = tmp) {  *// read line from file.*  fgets(a, 10, original);  *// debug.*  *// printf("%s %s", compressKey, tmp);*  *// check beginning matches with compressKey.*  *for* (int i = 0; i < strlen(compressKey); i++) {  *// if it match, increase a which means that character is excluded from the output file.*  *if* (compressKey[i] == tmp[i]) {  a++;  } *else* {  *// else print inadequate compressKey and out.*  printf("inadequate compressKey!!!\n");  *return* 1;  }  }  *//printf("output: %s \nNext:\n", a);*  *// print to file.*  fprintf(compressed, "%s", a);  *// make first byte null, so basically cleaning tmp.*  tmp[0] = 0;  }  *// close file pointer.*  fclose(original);  fclose(compressed);  *return* 0;  } |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **Compressing** |
|  | |

**//(필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)**

|  |  |
| --- | --- |
| **#2** | **Adding more indexes** |
|  | |

**psh010209@gc.gachon.ac.kr, 010-7661-7035**

**SANGHUI PARK (박상희)**

# **개발 과제 #2 :**

|  |  |
| --- | --- |
|  | **Write a C program that can decompress the file]** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Decompress the compressed file via compressionKey available at beginning at the file.**

* **Step 2: Outline a solution – 간단한 logic설명**

**Get the compression key,  
Remove line return from compressionKey,  
print compressionKey and then print the each line.**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**Get line, remove line return and save as compressionKey,**

**After that, forEach line, print compressionKey and print that line to output file.**

* **Step 4: write a pseudo code – 수도코드 작성**

**Read line, remove \n and save as compressionKey**

**forEach thisLine of lines,**

**print compressionKey(that doesn’t have \n)**

**print thisLine**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  int main() {  *// open files*  FILE \*comp = fopen("compressed.txt", "r");  FILE \*decomp = fopen("decompressed.txt", "w");  *// exception handling.*  *if* (comp == NULL) {  printf("[ERROR] couldn't open compressed.txt!\nCheck disk S.M.A.R.T. or file permissions!\n");  *return* 1;  }  *// exception handling.*  *if* (decomp == NULL) {  printf("[ERROR] couldn't open decompressed.txt!\nCheck disk S.M.A.R.T. or file permissions!\n");  *return* 1;  }  *// declare array for storing compression Key.*  char compressedKey[10];  *// get compression key from file.*  fgets(compressedKey, 10, comp);  *// if compressed key line contains line return at the end, change that to null.*  *for* (int i = 0; i < 10; i++) {  *if* (compressedKey[i] == '\n') {  compressedKey[i] = 0;  }  }  *// until end of the file.*  *for* (char a[10] = ""; !feof(comp); ) {  *// read line to a.*  fgets(a, 10, comp);  *//printf("%s%s", compressedKey, a);*  *// output compressed key and a to file.*  fprintf(decomp, "%s%s", compressedKey, a);  }  *// close files.*  fclose(comp);  fclose(decomp);  *return* 0;  } |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **Providing original and compressed and decompressed results.** |
|  | |

**psh010209@gc.gachon.ac.kr, 010-7661-7035**

**SANGHUI PARK (박상희)**

# **개발 과제 #3 :**

|  |  |
| --- | --- |
|  | **Cipher original.txt via caesarCipher with provided shift value** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Create a ciphertext from a provided plaintext original.txt**

* **Step 2: Outline a solution – 간단한 logic설명**

**caesarCipher is all about cycling through alphabet like circle,  
since modulation operation behaves like that, I came up with an approach with using modulation.**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**For all characters, check if it is alphabet, get current alphabet index and add shift value. And do a modular operation to 26 (total characters in alphabet.) the result will be cipherText’s alphabet index. change that into alphabet. And write to output.**

* **Step 4: write a pseudo code – 수도코드 작성**

**Ask User shift value**

**While EOF**

**A = getCharacterFromFile()  
 if A in alphabet  
 writeToFile(alphabetIndexToChar((getAlphabetIndex(A)+shift) % 26))**

**Else**

**writeToFile(A)**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  *#define* \_\_ENCRYPT\_\_  *// check whether this letter is lowercase*  char isLowerCase(char letter) { *return* (letter <= 'z' && letter >= 'a'); }  *// check whether this letter is uppercase*  char isUpperCase(char letter) { *return* (letter <= 'Z' && letter >= 'A'); }  *// check whether this letter is alphabet*  char isInAlphabet(char letter) { *return* isUpperCase(letter) || isLowerCase(letter); }  *#ifdef* \_\_ENCRYPT\_\_  *// encrypt single letter with specified shift. returns encrypted character.*  char encryptLetter(char letter, int shift) {  char lower = isLowerCase(letter);  *// since it is encryption, shift value should be added. since it is looping,*  *// it should calculated with the remainder*  *return* (((letter - (lower ? 'a':'A')) + shift) % 26) + (lower ? 'a':'A');  }  *// encrypt file. utilizing encrypt:etter function*  void encrypt(FILE \*input, FILE\* output, int shift) {  *// for every character in input*  *for* (char a = fgetc(input); !feof(input); a = fgetc(input)) {  *// check is it an alphabet*  *if* (isInAlphabet(a)) {  *// then encrypt and write to output*  fputc(encryptLetter(a, shift), output);  } *else* {  *// just write it.*  fputc(a, output);  }  }  }  *#else*  *// decrypt single letter with specified shift. returns decrypted character.*  char decryptLetter(char letter, int shift) {  char lower = isLowerCase(letter);  *// since it is decryption, shift value should be subtracted. since it is looping,*  *// it should calculated with the remainder, 26 was added because of C's interpretation*  *// of modulo in negative numbers.*  *return* (((letter - (lower ? 'a':'A')) + 26 - shift) % 26) + (lower ? 'a':'A');  }  *// decrypt file. utilizing decryptLetter function*  void decrypt(FILE \*input, FILE\* output, int shift) {  *// for every character in input*  *for* (char a = fgetc(input); !feof(input); a = fgetc(input)) {  *// check is it an alphabet*  *if* (isInAlphabet(a)) {  *// then decrypt and write to output*  fputc(decryptLetter(a, shift), output);  } *else* {  *// just write it.*  fputc(a, output);  }  }  }  *#endif*  int main() {  *// open file.*  *#ifdef* \_\_ENCRYPT\_\_  FILE \*input = fopen("original.txt", "r");  FILE \*output = fopen("cypher.txt", "w");  *#else*  FILE \*input = fopen("cypher.txt", "r");  FILE \*output = fopen("decrypted.txt", "w");  *#endif*  *// exception handling*  *if* (input == NULL || output == NULL) {  printf("[ERROR] couldn't open file\nCheck disk S.M.A.R.T. or file permissions!\n");  *return* 1;  }  *// ask user shift value.*  printf("Shift: ");  int shift = 0;  scanf("%d", &shift);  *#ifdef* \_\_ENCRYPT\_\_  *// call encrypt function*  encrypt(input, output, shift);  *#else*  decrypt(input, output, shift);  *#endif*  *// close file pointer*  fclose(input);  fclose(output);  } |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **Encryption demonstration** |
|  | |

**//(필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)**

**psh010209@gc.gachon.ac.kr, 010-7661-7035**

**SANGHUI PARK (박상희)**

# **개발 과제 #4 :**

|  |  |
| --- | --- |
|  | **Decrypt cipher.txt which was encrypted via caesarCipher with provided shift value** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Create a plaintext from a provided ciphertext cipher.txt**

* **Step 2: Outline a solution – 간단한 logic설명**

**caesarCipher is all about cycling through alphabet like circle,  
since modulation operation behaves like that, I came up with an approach with using modulation.**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**For all characters, check if it is alphabet, get current alphabet index and subtract shift value. And do a modular operation to 26 (total characters in alphabet.) the result will be plainText’s alphabet index. change that into alphabet. And write to output.**

* **Step 4: write a pseudo coode – 수도코드 작성**

**Ask User shift value**

**While EOF**

**A = getCharacterFromFile()  
 if A in alphabet  
 writeToFile(alphabetIndexToChar((getAlphabetIndex(A)-shift) % 26))**

**Else**

**writeToFile(A)**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  *//#define \_\_ENCRYPT\_\_*  *// check whether this letter is lowercase*  char isLowerCase(char letter) { *return* (letter <= 'z' && letter >= 'a'); }  *// check whether this letter is uppercase*  char isUpperCase(char letter) { *return* (letter <= 'Z' && letter >= 'A'); }  *// check whether this letter is alphabet*  char isInAlphabet(char letter) { *return* isUpperCase(letter) || isLowerCase(letter); }  *#ifdef* \_\_ENCRYPT\_\_  *// encrypt single letter with specified shift. returns encrypted character.*  char encryptLetter(char letter, int shift) {  char lower = isLowerCase(letter);  *// since it is encryption, shift value should be added. since it is looping,*  *// it should calculated with the remainder*  *return* (((letter - (lower ? 'a':'A')) + shift) % 26) + (lower ? 'a':'A');  }  *// encrypt file. utilizing encrypt:etter function*  void encrypt(FILE \*input, FILE\* output, int shift) {  *// for every character in input*  *for* (char a = fgetc(input); !feof(input); a = fgetc(input)) {  *// check is it an alphabet*  *if* (isInAlphabet(a)) {  *// then encrypt and write to output*  fputc(encryptLetter(a, shift), output);  } *else* {  *// just write it.*  fputc(a, output);  }  }  }  *#else*  *// decrypt single letter with specified shift. returns decrypted character.*  char decryptLetter(char letter, int shift) {  char lower = isLowerCase(letter);  *// since it is decryption, shift value should be subtracted. since it is looping,*  *// it should calculated with the remainder, 26 was added because of C's interpretation*  *// of modulo in negative numbers.*  *return* (((letter - (lower ? 'a':'A')) + 26 - shift) % 26) + (lower ? 'a':'A');  }  *// decrypt file. utilizing decryptLetter function*  void decrypt(FILE \*input, FILE\* output, int shift) {  *// for every character in input*  *for* (char a = fgetc(input); !feof(input); a = fgetc(input)) {  *// check is it an alphabet*  *if* (isInAlphabet(a)) {  *// then decrypt and write to output*  fputc(decryptLetter(a, shift), output);  } *else* {  *// just write it.*  fputc(a, output);  }  }  }  *#endif*  int main() {  *// open file.*  *#ifdef* \_\_ENCRYPT\_\_  FILE \*input = fopen("original.txt", "r");  FILE \*output = fopen("cypher.txt", "w");  *#else*  FILE \*input = fopen("cypher.txt", "r");  FILE \*output = fopen("decrypted.txt", "w");  *#endif*  *// exception handling*  *if* (input == NULL || output == NULL) {  printf("[ERROR] couldn't open file\nCheck disk S.M.A.R.T. or file permissions!\n");  *return* 1;  }  *// ask user shift value.*  printf("Shift: ");  int shift = 0;  scanf("%d", &shift);  *#ifdef* \_\_ENCRYPT\_\_  *// call encrypt function*  encrypt(input, output, shift);  *#else*  decrypt(input, output, shift);  *#endif*  *// close file pointer*  fclose(input);  fclose(output);  } |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **Decryption demonstration** |
|  | |

**//(필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)**

**psh010209@gc.gachon.ac.kr, 010-7661-7035**

**SANGHUI PARK (박상희)**

# **개발 과제 #5 :**

|  |  |
| --- | --- |
|  | **Encrypt plaintext using columnTransposition** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Get plain text from input, read password as a precedence of which row should be read first.**

**Make it as a one line and export that into cypher.txt**

* **Step 2: Outline a solution – 간단한 logic설명**

**Read input, make a matrix and input data into matrix.**

**Read matrix, read password and parse to check the precedence**

**And transposition matrix and write output to the file.**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**Read input, make a matrix and input data into matrix.**

**Read matrix, read password and parse to check the precedence**

**If it is current precedence, output to file**

* **Step 4: write a pseudo code – 수도코드 작성**

**Read plaintext**

**Allocate memory for matrix**

**Save matrix.**

**For 1 to password length as a**

**Check which index of the password is a**

**If a is current index,**

**Write current column to output.**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  *#include* <string.h>  int main() {  *// get the length of the password*  printf("Enter length of password: ");  int passwordSize = 0;  scanf("%d", &passwordSize);  *// ask user to enter password seperated by spaces*  printf("Please Enter password seperated by spaces: ");  int \*passwordList = (int \*)malloc(sizeof(int) \* passwordSize);  *for* (int i = 0; i < passwordSize; i++) {  scanf("%d", passwordList+i);  }    *// open file*  FILE \*input = fopen("original.txt", "r");  FILE \*output = fopen("cypher.txt", "w");  *// exception handling*  *if* (input == NULL || output == NULL) {  printf("[ERROR] couldn't open file\nCheck disk S.M.A.R.T. or file permissions!\n");  *return* 1;  }    *// all the file has at least 1 line.*  int lineCount = 1;    *// for every line return was found, add lineCount*  *for* (char a = getc(input); !feof(input); a = getc(input)) {  *if* (a == '\n') lineCount++;  }  *// display user a linecount of the original file*  printf("LineCount: %d\n\n", lineCount);  *// due to linecount, current location is at the end.*  *// seeking to the begining.*  fseek(input, 0, SEEK\_SET);  *// define a row to store all the columns.*  char \*\*row;  row = (char \*\*)malloc(sizeof(char \*) \* passwordSize);  *// exception handling*  *if* (row == NULL) *return* 1;  *// for every rows create columns.*  *for* (int i = 0; i < passwordSize; i++) {  char \*column;  column = (char \*)malloc(sizeof(char) \* lineCount);    *// exception handling*  *if* (column == NULL) *return* 1;  *//initializing columns.*  *for* (int j = 0; j < lineCount; j++) {  column[j] = ' ';  }  row[i] = column;  }      *// for every line*  *for* (int i = 0; i < lineCount; i++) {  char \*a;    *//having extra size just in case it has multiple spaces.*  a = malloc(sizeof(char) \* (3\*passwordSize));    *// exception handling.*  *if* (a == NULL) *return* -1;  fgets(a, 3\*passwordSize, input);  int k = 0;  *// for this line,*  *for* (int j = 0; j < strlen(a); j++) {  *// if it is space or line feed, skip.*  *if* (a[j] == ' ' || a[j] == '\n' || a[j] == '\r') {  *continue*;  *// if this is end of file, get out of loop.*  } *else* *if* (feof(input)) {  *break*;  } *else* {  *// else add to matrix.*  row[k][i] = a[j];  k++;  }  }  free(a);  }  *// display to user.*  *for* (int i = 0; i < lineCount; i++) {  *for* (int j = 0; j < passwordSize; j++) {  printf("%c ", row[j][i]);  }  printf("\n");  }  printf("\n");    *// create variable for storing cipherText*  char \*cipherText;  cipherText = (char \*)malloc(sizeof(char) \* passwordSize \* lineCount);  *// variable to store which index of cipherText is currently being read.*  int cipherTextCount = 0;  *// loop 1-passwordSize to loop through columns.*  *for* (int i = 1; i <= passwordSize; i++) {  int o;  *// for all column,*  *for* (o = 0; o < passwordSize; o++) {  *// if i is column of i-th in password, break.*  *if* (i == passwordList[o]) *break*;  }  *// for current column,*  *for* (int j = 0; j < lineCount; j++) {  *// if it is space (which was initialization value, skip.)*  *if* (row[o][j] == ' ') {  *continue*;  }  *// add current index in the matrix to cipherText*  cipherText[cipherTextCount] = row[o][j];  cipherTextCount++;  }  }  *// show cipher text to screen and write cipher text to file*  printf("%s\n", cipherText);  fprintf(output, "%s", cipherText);  *// free ciphertext*  free(cipherText);  *// free all columns*  *for* (int i = 0; i < passwordSize; i++) {  free(row[i]);  }  *// free all un-freed memories.*  free(row);  free(passwordList);  *// close file pointer*  fclose(input);  fclose(output);  } |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **Encryption Example** |
|  | |

**//(필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)**

**psh010209@gc.gachon.ac.kr, 010-7661-7035**

**SANGHUI PARK (박상희)**

# **개발 과제 #6 :**

|  |  |
| --- | --- |
|  | **Decrypt the ciphertext encrypted via matrix transpose** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Get ciphertext from input, read password as a precedence of which row should be read first on the encryption stage**

**Make it plain text and write it to decrypted.txt**

* **Step 2: Outline a solution – 간단한 logic설명**

**Read input, make a matrix size of password length and strlen(inputline) / password length**

**And by password’s precedence order, write cipherText into matrix.**

**Read matrix, and print it to the file.**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**Read input, make a matrix and input data into matrix.**

**Read password, by precendence, read ciphertext and write it to matrix**

* **Step 4: write a pseudo code – 수도코드 작성**

**Read ciphertext**

**Allocate memory for matrix**

**For 1 to password length as a**

**Check which index of the password is a**

**Write the plain text length of strlen(cipherText) / passwordLength to the column index number**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  *#include* <string.h>  int main() {  *// open file*  FILE \*input = fopen("cypher.txt", "r");  FILE \*output = fopen("decrypted.txt", "w");  *// exception handling*  *if* (input == NULL || output == NULL) {  printf("[ERROR] couldn't open file\nCheck disk S.M.A.R.T. or file permissions!\n");  *return* 1;  }  *// get file size*  fseek(input, 0, SEEK\_END);  *// adding 1 because it counts from 0*  long long int size = ftell(input) + 1;  *// go back to beginning*  fseek(input, 0, SEEK\_SET);  *// satilizing ciphertext - creating memory section*  char \*rawCipherText = (char \*)malloc(sizeof(char) \* size);  char \*cipherText = (char \*)malloc(sizeof(char) \* size);  *// exception handling*  *if* (rawCipherText == NULL || cipherText == NULL) *return* -1;  *//get cipher text*  fgets(rawCipherText, size, input);  *// for all characters*  *for* (int i = 0, k = 0; i < size; i++) {  *// ignore line return, carrige return and spaces.*  *if* (rawCipherText[i] == ' ' || rawCipherText[i] == '\r' || rawCipherText[i] == '\n') {  *continue*;  } *else* {  *// for every charcter that passed the test goes to satilized ciphertext variable*  cipherText[k] = rawCipherText[i];  k++;  *// setting NULL at end in advance.*  cipherText[k] = 0;  }  }  *// rawciphertext is no longer required. freeing memory*  free(rawCipherText);  *// check length of the ciphertext*  int length = strlen(cipherText);  *// get length of password.*  int passwordLength = 0;  printf("Enter the length of the password : ");  scanf("%d", &passwordLength);  *// get password from user*  *// create passwordList variable to store password*  int \*passwordList = (int \*)malloc(sizeof(int) \* passwordLength);  *// exception handling*  *if* (passwordList == NULL) *return* -1;  *// show prompt*  printf("Enter password seperated by spaces : ");  *// and save it to passwordList.*  *for* (int i = 0; i < passwordLength; i++) {  scanf("%d", passwordList+i);  }  *// check there is null characters at the end to meet the passwordLength*  *// else lasts will have extended columns.*  char isNullCipher = (length % passwordLength == 0);    *// basic line counts for non-extended columns*  int baseLineCount = (int)(length / passwordLength);  *// line counts for extended columns*  int lineCount = isNullCipher ? (int)(length / passwordLength) : (int)(length / passwordLength) + 1;  *// the section to store columns and rows.*  char \*\*row = (char \*\*)malloc(sizeof(char \*) \* passwordLength);  *// exception handling*  *if* (row == NULL) *return* -1;  *// initializing and allocating columns to rows.*  *for* (int i = 0; i < passwordLength; i++) {  char \*column = (char \*)malloc(sizeof(char) \* lineCount);  *// exception handling*  *if* (column == NULL) *return* -1;  *//initialzing columns.*  *for* (int j = 0; j < lineCount; j++) {  column[j] = ' ';  }  row[i] = column;  }  *// variable to store which index of the ciphertext is currently reading.*  int currentCipherIndex = 0;  *// precendence loop. 1 to passwordLength*  *for* (int i = 1; i <= passwordLength; i++) {  int k = -1;  *// check which column has most precedence*  *for* (int j = 0; j < passwordLength; j++) {  *// if this is it, save current index and break the loop.*  *if* (passwordList[j] == i) {  k = j;  *break*;  }  }  *// if it was not found. terminate the program.*  *if* (k == -1) *return* -1;  *// get cipher text and put it until the line limit reqched.*  *for* (int j = 0; j < baseLineCount; j++) {  row[k][j] = cipherText[currentCipherIndex];  currentCipherIndex++;  }  *// this logic is only for extended columns which requires more rows*  *// than other columns.*  *if* (!isNullCipher && passwordLength - i < (length % passwordLength)) {  row[k][lineCount - 1] = cipherText[currentCipherIndex];  currentCipherIndex++;  }  }  *// show and export it to text file. for each rows.*  *for* (int i = 0; i < lineCount; i++) {  *// display each columnns and*  *for* (int j = 0; j < passwordLength; j++) {  printf("%c ", row[j][i]);  fprintf(output, "%c ", row[j][i]);  }  *// line return.*  printf("\n");    fprintf(output, "\n");    }    *// free all the columns.*  *for* (int i = 0; i < passwordLength; i++) {  free(row[i]);  }  *// rows and all the variables that was required.*  free(row);  free(cipherText);  free(passwordList);  *// close file pointer.*  fclose(input);  fclose(output);  } |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **Decryption Example** |
|  | |

**//(필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)**