HW1 Readme

1-2思路

- 首先做輸入的處理，用**ROT13**解密得到pattern，利用**KMP演算法**比對文章，最後再輸出答案。
  接下來會說明我各個函式的功能

CODE 分析

輸入處理函式

```
void ProcessInput( char* ciphertext , char* article )
{
    fgets( ciphertext , 500 , stdin ) ;
    fgets( article , 500 , stdin ) ;
    for( int i = 0 ; i < 52 ; i++ )
    {
        fgets( answer_dics[ i ].secret_phrases , 50 , stdin ) ;
        int len_dic = strlen( answer_dics[ i ].secret_phrases )  ;
        answer_dics[ i ].secret_phrases[ len_dic - 1 ] = '\0' ;
    }   // put answer dic in struct
    int lena = strlen( article ) , lent = strlen( ciphertext ) ;
    // lenth_of_string
    article[ lena - 1 ] = '\0' ;
    ciphertext[ lent - 1 ] = '\0' ;
}
#因為用fgets的關係，會吃到\n，所以我有用\0把它蓋掉，避免麻煩，這樣也好操作。
```

ROT解密函式

```
void FindKeyByROT( char* ciphertext  ) // ROT  Implementation
{
    int lenc = strlen( ciphertext ) ;
    for( int i = 0 ; i < lenc  ; i++ )
    {
        int origin_text = ( int ) ciphertext[ i ] ;
        origin_text  -= 13 ;
        if( origin_text < 97 && islower( ciphertext[ i  ] ) )
        {
            origin_text = 123 - ( 97 - origin_text ) ;
            ciphertext[ i ] = ( char ) origin_text ;
        }
        else if( origin_text < 65 && isupper( ciphertext[  i  ] ) )
        {
            origin_text = 91 - ( 65 - origin_text ) ;
            ciphertext[ i ] = ( char ) origin_text ;
        }
        else
        {
            ciphertext[ i ] = ( char ) origin_text ;
        }
    }
    for( int i = 0 ; i < 5 ; i++ )
    {
        g_pattern[ i ] = ciphertext[ i ] ;
    }
}
# 利用強制轉型兩次來移動字元。
# 這邊分了大小寫的case來處理。
# 123 97 65 91 這些數字用來計算彼此ascii的差距。
```

計算 lps 的函式 : 用在 KMP 演算法

```
void CalculateLps(char* pat, int lenp)
{
    int len = 0; // 就是k 要盡量找最大
    g_lps[0] = 0; // 這邊用0 ， 也可以用-1
    int i = 1;
    while (i < lenp) // 從index = 1 找到 lenp - 1
    {
        if (pat[i] == pat[len]) //一樣
        {
            len++;
            g_lps[i] = len;
            i++;
        }
        else // 如果不一樣
        {
            if (  len != 0  ) // k 慢慢變小 去對應
            {
                len = g_lps[len - 1];
            }
            else //沒得對應只好變0
            {
                g_lps[i] = 0;
                i++;
            }
        }
    }
}
#  這邊的 k 是依據定義所設 。
#  f(j) = largest k < j such that p0p1...pk = pj-k...pj
   if such a k >= 0  exsists , otherwise f(j) = 0
```

KMP 實作 : 用來找文章到底有哪符合pattern ，並且將位置(position) 記錄下來。

```
void PatternMatch(char* pat, char* txt) // KMP Implementation
{
    int i = 0; // i 代表 article 目前的位置
    int j = 0; // j 代表 所尋找 pattern 目前位置
    int lenp = strlen(pat);
    int lent= strlen(txt);

    CalculateLps(  pat  , lenp );

    while (i < lent) // 一直找文章到結束為止
    {
        if (pat[j] == txt[i]) // 對到了，各往右一次。
        {
            j++;
            i++;
        }
        if (j == lenp ) // 找到了，這邊我開一個陣列去存position
        {
            g_key_index[ g_num_key_index ] = i - j ;
            g_num_key_index++ ;
            j = g_lps[j - 1];
        }
        else if (i < lent && pat[j] != txt[i])//smart shift
        {
            if (j != 0)
            {
                 j = g_lps[ j - 1 ] ;
            }
            else // 也就是 j == 0
            {
                 i = i + 1 ;
            }
        }
    }
}
#    這邊就是 KMP 的實作，算好 LPS 後就開始 smart shift
```

與輸出結果相關的函式

```
void PrintFinalAnswer( char* ciphertext , int sum_lps )
{
    printf("%s\n", ciphertext) ;
    printf("%s\n", g_pattern) ;
    for( int i = 0 ; i < 5 ; i++ )
    {
        sum_lps += g_lps[ i ] ;
    } // 算sum_lps
    printf("%d\n", sum_lps ) ;
    for( int i = 0 ; i < g_num_key_index ; i++ )
    {
        g_key_index[ i ] += sum_lps ;
        printf("%d\n",g_key_index[ i ] ) ;
    } // 將 sum_lps 加進key_index裡
    for( int i = 0 ; i < g_num_key_index ; i++ )
    {
       int index_answer_dic = g_key_index[ i ] ;
       index_answer_dic = MyChangingFunction( index_answer_dic ) ;
       PrintAnswerDic( index_answer_dic ) ;
       printf(" ") ;
    }
}
#     MyChangingFunction
      這個函式把找到postion + sum_lps 的值轉換成
      存在answer_dic裡面的index ，方便輸出。
#     PrintAnswerDic
      這邊我寫了一個函式，用字元(%c)的方式在輸出answer_dic
      原因是用fgets讀進來的資料並非全都要輸出。
```

```
int MyChangingFunction( int key_index_value ) // ChangeValueToAnswerDicIndex
{
    int index_answer_dic = -1 ;
    if( key_index_value >= 65 && key_index_value <= 90 )
    {
        index_answer_dic = key_index_value-65 ;
    }
    else
    {
        index_answer_dic = key_index_value - 97 +26 ;
    }
    return index_answer_dic ;
}
#這邊也有大小寫的問題 ， 因此我分兩個case處理 。
```

```
void PrintAnswerDic( int index_answer_dic ) // print out answer dic
{
    int lenth =strlen( answer_dics[ index_answer_dic ].secret_phrases ) ;
    for( int i = 2 ; i < lenth  ; i++ )
    {
        printf("%c",answer_dics[ index_answer_dic ].secret_phrases[ i ] ) ;
    }
}
#前面的A~z和逗號不用輸出，從 index = 2 開始輸出。
```

1-1 思路

- 利用**struct**存取資料 ， **if else**判斷後輸出 。

```
#include <stdio.h>
#include <string.h>
struct vaccineInfo
{
    char Id[ 11 ] ;
    char Name[ 26 ] ;
    char Vac_Brand[ 31 ] ; // vaccine brand ;
    int Age ;
    char City[ 21 ] ;
}vaccineInfos[ 20 ];
int main( )
{
    int num_of_data = 0 ;
    scanf("%d",&num_of_data) ;
    getchar( ) ;
    for( int i = 0 ; i < num_of_data ; i++ )
    {
        scanf("%s",vaccineInfos[ i ].Id) ;
        scanf("%s",vaccineInfos[ i ].Name) ;
        scanf("%s",vaccineInfos[ i ].Vac_Brand) ;
        scanf("%d",&vaccineInfos[ i ].Age ) ;
        scanf("%s",vaccineInfos[ i ].City ) ;
    }
    char qry[ 31 ] ; //query
    scanf("%s",qry) ;
    for( int i = 0 ; i < num_of_data ; i++ )
    {
        if( vaccineInfos[ i ].Vac_Brand[ 0 ] == qry[ 0 ] )
        {
            printf("%s %s %s %d %s\n",vaccineInfos[ i ].Id,vaccineInfos[ i ].Name,vac
        } //這邊的判斷是用每個疫苗品牌的開頭字元
    }
    return 0 ;
```

```
34  }
```