PLAYFAIR CYPHER

INTRODUZIONE

Il programma *playfair* si occupa principalmente di:

- Codificare uno o più file data la matrice generata a partire da un keyfile
- Decodificare uno o più file data la matrice generata a partire da un keyfile

I file all'interno di *filedistance* che rendono ciò possibile sono:

- *commands.c* (e rispettivamente *commands.h*): in commands sono presenti le funzioni che permettono di processare i comandi forniti dall'utente. Al suo interno sono presenti le seguenti funzioni:
  - *process_commands*: questa funzione si occupa di generare un keyfile e, in base al comando, avviare la funzione di encode o decode su tutti i file forniti dall'utente.
  - *encode*: questa funzione si occupa di avviare il processo di codifica di un file dato un *keyfile* e generare il file codificato in formato .pf.
  - *decode*: questa funzione si occupa di avviare il processo di decodifica di un file dato un *keyfile* e generare il file codificato in formato .dec.
- file_utils.c (e rispettivamente *file_utils.h*): in file_utils sono presenti le funzioni che mi aiutano nella generazione del file codificato/decodificato. Al suo interno sono presenti le seguenti funzioni:
  - *get_message*: questa funzione si occupa di rendere il messaggio cifrato una stringa, dove ogni due lettere vi è uno spazio.
  - *get_decoded_message*: questa funzione si occupa di rendere il messaggio decifrato una stringa.
  - *create_file*: questa funzione si occupa di generare un file in formato .pf dato il messaggio cifrato e la directory in cui salvarlo.
  - *create_decode_file*: questa funzione si occupa di generare un file in formato .dec dato il messaggio decifrato e la directory in cui salvarlo.
  - *gen_pf_path*: questa funzione si occupa di generare il path in cui si andrà poi a salvare il messaggio cifrato. All'interno di questa funzione utilizzo la macro *basename* dalla libreria *libgen.h*, che mi permette di ritornare la stringa che segue l'ultimo "/", ovvero il nome del file da cifrare.
  - *gen_dec_path*: questa funzione si occupa di generare il path in cui si andrà poi a salvare il messaggio decifrato. All'interno di questa funzione utilizzo la macro *basename* dalla libreria *libgen.h*, che mi permette di ritornare la stringa che segue l'ultimo "/", ovvero il nome del file da decifrare.
  - *remove_ext*: questa funzione si occupa semplicemente di rimuovere l'estensione dal nome del file. L'ho trovata su stackoverflow al seguente link: https://stackoverflow.com/questions/43163677/how-do-i-strip-a-file-extension-from-a-string-in-c/43163761.
  -
- *keyfile_read.c* (e rispettivamente *keyfile_read.h*): all'interno di questo file si trovano 3 strutture importanti per il funzionamento di questo programma:

```
typedef struct s_alphabet {
    char *alphabet;
    int *flag;
}al;
```

  - In questa prima struct vado a salvare l'alfabeto, fornito nel keyfile, in un array di char chiamato *alphabet*, e, creo un array di interi chiamato *flag*, che ha la stessa grandezza dell'array *alphabet*; in questo modo alla lettera *alphabet*[n] corrisponde il valore *flag*[n]: questo array di interi è inizialmente composto da soli 0, che, successivamente, diventeranno 1, in base alla presenza o meno della lettera corrispondente all'interno della chiave.

```
typedef struct key {
    char *key;
    int size;
    int *flag;
}k;
```

- In questa seconda struct vado invece a salvare la chiave, sempre fornita nel keyfile, in un array di char chiamato *key*, la sua grandezza, e, un array di interi chiamato *flag*, che ha la stessa grandezza dell'array *key*; in questo modo alla lettera *key*[n] corrisponde il valore *flag*[n]: come prima l'array è inizialmente composto da soli 0, che poi diventeranno 1 in base alla ripetizione di una stessa lettera.

```
typedef struct keyfile {
    al *alphabet;
    char missing_char;
    char missing_alphabet_letter;
    char special_char;
    k *key;
    char **matrix;
}kf;
```

- In questa terza struct, infine, vengono salvate sia le due strutture dati sopra, insieme al carattere mancante, la lettera mancante dell'alfabeto, il carattere speciale e la matrice 5x5 generata.

Oltre a queste 3 struct, all'interno di questo file sono presenti le seguenti funzioni:

- *read_all*: questa funzione si occupa di leggere tutto il keyfile e associa ad ogni elemento presente nella struct *keyfile* il valore corretto.
- *find_repetition*: questa funzione si occupa di trovare lettere ripetute all'interno della chiave. Quando una lettera viene ripetuta il suo valore all'interno di *flag* passa da 0 a 1.
- *clear_alphabet*: questa funzione si occupa di trovare nell'alfabeto le lettere già presenti nella chiave. Quando una lettera viene trovata, il suo valore all'interno di *flag* passa da 0 a 1.
- *find_missing_letter*: questa funzione si occupa di trovare la lettera che manca all'interno dell'alfabeto fornito dal keyfile.
- *free_alphabet*: questa funzione si occupa di liberare la memoria occupata dalla struct *alphabet*.
- *free_key*: questa funzione si occupa di liberare la memoria occupata dalla struct *key*.
- *free_keyfile*: questa funzione si occupa di liberare la memoria occupata dalla struct *keyfile*.
- *check_keyfile*: questa funzione si occupa semplicemente di assicurarsi che, dato un file, esso abbia gli argomenti corretti per generare un keyfile.

- *keyfile_gen.c* (e rispettivamente *keyfile_gen.h*): in keyfile_gen si trovano tutte le funzioni che permettono il completamento della struct *keyfile* (e di conseguenza anche delle struct *alphabet* e *key*). Al suo interno sono presenti le seguenti funzioni:
  - *write_key*: questa funzione permette di scrivere nella matrice 5x5 i caratteri non ripetuti della chiave.
  - *write_alphabet*: questa funzione permette di scrivere nella matrice 5x5 le lettere dell'alfabeto non presenti nella chiave.
  - *fill_matrix*: questa funzione semplicemente riempie la matrice 5x5 richiamando i due metodi descritti sopra.
  - *complete_keyfile*: questa funzione si occupa di allocare correttamente la memoria della struct *keyfile* e di generare poi la struct completa.

- *matrix_utils.c* (e rispettivamente *matrix_utils.h*): questo file contiene delle funzioni utili nella gestione delle matrici. Al suo interno sono presenti le seguenti funzioni:
  - *initialize_matrix*: questa funzione si occupa di inizializzare la memoria di una matrice.
  - *free_matrix*: questa funzione si occupa di liberare correttamente la memoria occupata da una matrice
  - *print_matrix*: questa funzione si occupa di stampare a console una matrice (in realtà questa funzione non viene mai utilizzata all'interno del programma, ma è stata comunque utile durante la stesura del programma quindi ho deciso di lasciarla).

- *message_utils.c* (e rispettivamente *message_utils.h*): all'interno di questo file si trova l'altra struttura importante per il funzionamento di questo programma:

```
typedef struct message{
    int size;
    char **pairs;
    char **encoded_pairs;
}sm;
```

  - In questa struct vado a salvare come prima cosa quella che è la lunghezza di un messaggio una volta applicate le regole di codifica, oppure semplicemente la lunghezza del messaggio da decodificare. Successivamente dentro alla matrice *pairs* (la cui grandezza è (size/2)x2) vado a salvare su ogni riga la coppia di lettere del messaggio decodificato, mentre, dentro alla matrice *encoded_pairs* (la cui grandezza è di nuovo (size/2)x2), vado a salvare su ogni riga la coppia di lettere del messaggio codificato.

  Oltre a questa struct, in questo file sono presenti le seguenti funzioni:
  - *find_m_size*: questa funzione mi permette, dato un messaggio da codificare, di sapere quale sarà la lunghezza finale del messaggio una volta applicate tutte le regole di codifica. Questa funzione mi è molto utile perché mi permette di allocare esattamente la memoria necessaria per cifrare un messaggio.
  - *remove_missing_alphabet_letter*: questa funzione mi permette, dato un messaggio da codificare, di rimuovere da quel messaggio la lettera mancante dell'alfabeto, ogni volta che compare, e di sostituirla con il carattere mancante.
  - *find*_position: questa funzione mi permette, data una lettera, di trovare la sua posizione all'interno della matrice 5x5.
  - *free_message*: questa funzione mi permette di liberare correttamente la memoria occupata dalla struct *message*.

- *secretmessage_encode.c* (e rispettivamente *secretmessage_encode.h*): in questo file sono presenti tutte le funzioni che mi permettono di codificare correttamente un file. Al suo interno sono presenti le seguenti funzioni:
  - *read_message*: questa funzione mi permette di leggere il messaggio presente all'interno di un file e, restituirlo, senza spazi vuoti, e con la lettera non presente nell'alfabeto sostituita con il carattere mancante.
  - *init_message*: questa funzione mi permette di allocare correttamente lo spazio della struct *message* e assegna il corretto valore a *size*.
  - *fill_pairs*: questa funzione mi permette di riempire la matrice *pairs* all'interno della struct con il messaggio letto a cui però sono stati aggiunti il carattere speciale e il carattere mancante quando ve ne era la necessità.
  - *fill_encoded_pairs*: questa funzione si occupa di riempire la matrice *encoded_pairs* partendo dal messaggio da cifrare, applicando ogni volta le funzioni corrette, seguendo le regole di playfair.
  - *different_r_c*: questa funzione si occupa di inserire nella matrice *encoded_pairs*, nel caso in cui la coppia di lettere prese in considerazione abbia righe e colonne differenti, le lettere corrette.
  - *same_row*: questa funzione si occupa di inserire nella matrice *encoded_pairs*, nel caso in cui la coppia di lettere prese in considerazione sia sulla stessa riga, le lettere corrette.
  - *same_column*: questa funzione si occupa di inserire nella matrice *encoded_pairs*, nel caso in cui la coppia di lettere prese in considerazione sia sulla stessa colonna, le lettere corrette.
  - *create_message*: questa funzione si occupa di utilizzare le altre funzioni all'interno di questo file per generare correttamente un messaggio codificato.

- *secretmessage_decode.c* (e rispettivamente *secretmessage_decode.h*): in questo file sono presenti tutte le funzioni che mi permettono di decodificare correttamente un file. Al suo interno sono presenti le seguenti funzioni:
  - *read_encoded_message*: questa funzione mi permette di leggere il messaggio codificato presente all'interno di un file e, restituirlo, senza spazi vuoti.
  - *init_message_D*: questa funzione mi permette di allocare correttamente lo spazio della struct *message* e assegna il corretto valore a *size*.
  - *fill_encoded_pairs*: questa funzione mi permette di riempire la matrice *encoded_pairs* all'interno della struct con il messaggio codificato letto.
  - *fill_pairs_D*: questa funzione si occupa di riempire la matrice *pairs* partendo dal messaggio da decifrare, applicando ogni volta le funzioni corrette, seguendo le regole di playfair.
  - *different_r_c_D*: questa funzione si occupa di inserire nella matrice *pairs*, nel caso in cui la coppia di lettere prese in considerazione abbia righe e colonne differenti, le lettere corrette.
  - *same_row_D*: questa funzione si occupa di inserire nella matrice *pairs*, nel caso in cui la coppia di lettere prese in considerazione sia sulla stessa riga, le lettere corrette.
  - *same_column_D*: questa funzione si occupa di inserire nella matrice *encoded_pairs*, nel caso in cui la coppia di lettere prese in considerazione sia sulla stessa colonna, le lettere corrette.
  - *create_encoded_message*: questa funzione si occupa di utilizzare le altre funzioni all'interno di questo file per generare correttamente un messaggio decodificato.
  - *check_odd*: questa piccola funzione si occupa di verificare che il messaggio da decifrare non sia di lunghezza dispari, poiché, in quel caso, non sarebbe possibile decifrarlo.

- *utils.c* (e rispettivamente *utils.h*): all'interno di questo file sono presenti tutte le funzioni che in un modo o nell'altro mi sono utili ai fini della lettura dei file. Al suo interno sono presenti le seguenti funzioni:
  - *find_size*: questa funzione si occupa, dato un file, di trovare le sue dimensioni.
  - *read*: questa funzione si occupa, dato un file, di leggerlo e salvarlo all'interno di un array di char.
  - *read_file*: questa funzione si occupa, dato il path di un file, di aprirlo, e applicare le due funzioni sopra per generare un array di char della giusta dimensione e che contenga ciò che era scritto all'interno del file, ed infine resituirlo.
  - *remove_spaces*: questa funzione si occupa, dato un array di char, di rimuovere tutti gli spazi vuoti presenti al suo interno.
  - *check_exist*: questa funzione si occupa, dato un file, di assicurarsi che esso esista.