Relazione Progetto Sistemi Operativi

Struttura del codice :

Per la realizzazione del programma playfair, sono state individuate le seguenti classi con i seguenti ruoli:

- main.c -> classe base del programma, qui vengono controllati il numero di parametri passati al programma e da qui partono tutte le operazioni che andranno eseguite.

- file.h -> file header, contenente la struct file che ha il compito di memorizzare le informazioni di ogni file, e di creare dei prototipi dei metodi che saranno accessibili dalla classi che implementeranno questa classe.
    - MyFile *checkArgv(char **argv, int argc);
        ha il compito di controllare i parametri passati andando a creare la struct file che conterrà ogni file su cui eseguire il playfair.
    - bool checkType(char *type);
        ha il compito di controllare il tipo di operazione passato.
    - void playfair(MyFile *mf);
        ha il compito di eseguire l'algoritmo playfair in base ai parametri che sono stati passati sui file presenti nella struct file.

- file.c -> file che implementa il file file.h e playfair.h , ha il compito di controllare la corretta sintassi dei vari file e la loro esistenza, ritornando eventuali errori, ha anche il compito di creare una struttura per memorizzare i vari file presenti, inoltre deve controllare il corretto tipo di operazione da eseguire (encode/decode) e controlla se viene passata una directory per l'output dei file creati.

- keyfile.h -> file header, contenente la struct keyFile che ha il compito di memorizzare le informazioni relative al file key passato dall'utente, e di creare dei prototipi dei metodi che saranno accessibili dalla classi che implementeranno questa classe.
    - RealKeyFile *checkKeyFile(char *keyFile);

ha il compito di creare la struct keyFile , controllando eventuali errori nel file file passato.

- keyfile.c -> file che implementa il file keyfile.h, ha il compito di creare la struct keyFile dal file keyFile passato dall'utente seguendo delle specifiche richieste per la sintassi del contenuto del file.

- playfair.h -> file header, contenente la struct playfair che ha il compito di memorizzare le informazioni relative ai parametri necessari per la realizzazione del programma , come la matrice usate per eseguire la crittografia e la decrittografia e di creare dei prototipi dei metodi che saranno accessibili dalla classi che implementeranno questa classe.

    - int createMatrix(RealKeyFile *realKeyFile);
        ha il compito di creare la matrice dato il il keyFile passato.

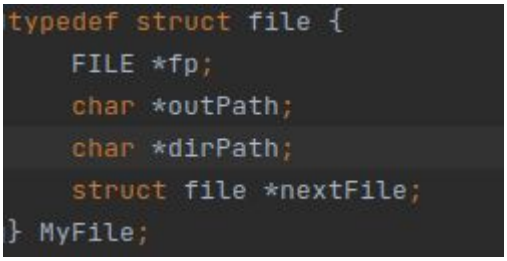
    - int encodeLine(char *line);
        metodo che ha il compito di eseguire la crittografia del testo passato.
    - int decodeLine(char *line);
        metodo che ha il compito di eseguire la decrittografia del testo passato, controllando eventuali errori di formattazione del testo.
    - void initPlayFair();
        metodo utilizzato per inizializzare la struct playFair , allocando lo spazio in memoria.
    - void setType(bool isEncode);
        metodo per settare il tipo di operazione che verrà eseguita.
    - bool getType()
        metodo per recuperare il tipo di operazione da eseguire.

- playfair.c -> file che implementa il file playfair.h , questa ha lo scopo di eseguire i vari algoritmi di crittografia e di generare la matrice dal keyFile, utilizza la struct playfair per immagazzinare la matrice, il tipo di operazione da eseguire e il keyFile che passato (convertito in struct keyFile).

Strutture dati utilizzate :

Per la realizzazione del programma, sono state individuate 3 strutture dati :

1)

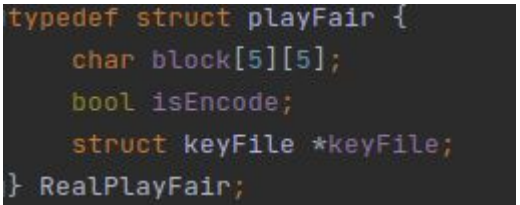


La seguente struttura ha il compito di immagazzinare dentro di sé le informazioni di cui abbiamo bisogno di un file passata da criptare / decriptare , come il puntatore del file (che viene creato quando apriamo per la prima volta il file per vedere se esiste e chiuso quando abbiamo letto tutto il suo contenuto) , l' outPath che contiene le informazioni necessarie per creare il percorso di destinazione del file di outPut , il dirPath contiene il percorso della directory di output del file da creare ed infine il nextFile, che e' un puntatore alla struttura del prossimo file (se il file successivo non e' valido o non e' presente, questo viene impostato come null), questa struttura viene definita nel file header file.h

2)

Questa struttura ha il compito di memorizzare le informazioni relative al keyFile passato come argomento al programma, avremo l'array di caratteri alfabeto, che ci indicherà le lettere che saranno utilizzate per la creazione della matrice del playFair, il char_mancante che ci indica il carattere da utilizzare per rimpiazzare nel messaggio da criptare il carattere che non e' presente nell'alfabeto o da utilizzare nel caso che se dividendo per coppie di due caratteri il messaggio da criptare , l'ultima coppia di valori sia composta da un solo valore allora viene aggiunto questo carattere alla fine prima di criptare il messaggio.
Char_speciale ha il ruolo di dividere eventuali coppie di valori uguali (esempio : char_assente=X coppia di valori=GG risultato= GX G).
Char_assente e' il valore dell'alfabeto che rimane escluso dall'alfabeto passato e che verrà sostituito dal char_mancante.
Chiave e' il parametro su cui verrà basata la costruzione della matrice.
Questa struttura viene definita nel file header keyfile.h

3)



Questa struttura ha il compito di memorizzare le informazioni necessarie per la realizzazione del playFair , come la matrice su cui si basa questo algoritmo, il tipo di operazione da eseguire rappresentata da una variabile booleana isEncode (true per encode , false per decode), e il keyFile che contiene le informazioni per la realizzazione della matrice e per la preparazione del messaggio da criptare/decriptare.
Questa struttura viene definita nel file header playfair.h

Funzionamento del programma:

Per prima cosa il programma controlla il numero di argomenti passati, se queste sono minori di 4, il programma mostra un errore ha schermo indicando l'errore specifico e mostra un esempio di comandi comandi eseguibili poi termina il programma.

* **GESTIONE DEGLI ERRORI**

  Per gestire gli mostrare gli errori che si possono avere durante l'esecuzione del programma si usa perror , che permette specificamente la stampa di errori in c, inoltre viene utilizzato errno che e' una variabile in base al valore assegnato , modifica il tipo di errore di ritorno di perror.

Nel caso che il numero di parametri passati sia corretto, viene successivamente controllato il tipo di operazione da eseguire, controllando se si tratti di un encode o decode, nel caso che il parametro passato sia errato viene mostrato l'errore, questo parametro viene memorizzato come boolean dentro la struttura playFair.
Successivamente si passa ad analizzare i parametri passati, per prima cosa si controlla il keyFile , se il suo contenuto rispetta la sintassi richiesta (altrimenti ritorna l'opportuno errore) e memorizzate le sue informazioni dentro la struct keyFile e con il keyFile generato verrà generata la matrice per l'esecuzione del playFair.
Successivamente si controlla se il parametro successivo sia una directory e in quel caso verrà memorizzato in ogni file della struct file che si creerà successivamente, una volta controllata eseguito il controllo si passa a controllare i vari file da criptare / decriptare , per ognuno di questi prima viene provato se esiste e in tal caso viene creata una struct file che conterrà tutti i file passati che esistano realmente (per il primo file valido, verrà creata una variabile di appoggio che ci permetterà di avere un punto di riferimento per il file da cui partire per eseguire l'encode / decode).
Una volta controllati tutti i file , se nessuno di questi e' valido viene mostrato un errore e terminato il programma , altrimenti viene eseguire il playFair sui file corretti , partendo dal primo file controllato.
Per prima cosa si crea un array di char che ha il compito di ospitare la porzione di testo che andrà letta dal file , per sicurezza , visto che il suo

contenuto andrà manipolato e potrà cambiare di dimensione, gli viene allocata in memoria una dimensione doppia a quella che verrà letta dal file (attraverso delle costanti definite nel file header) , per evitari eventuali buffer overflow, si apre/crea il file di output e successivamente si legge il file riga per riga .
Qui abbiamo due possibili scenari, dati dal tipo di operazione richiesta :

-Encode

Nel caso in cui venga richiesto di eseguire l'encode, per prima cosa si va a rimuovere i caratteri che non sono lettere dalla linea letta (inclusi anche i spazi vuoti), sostituiti eventuali caratteri assenti con il char_mancante, spezzate le coppie di valori uguali con il char_speciale e aggiunto l'eventuale char_mancante alla fine se l'ultima coppia di valori sia composta da una sola lettera, successivamente si aggiunge uno spazio vuoto tra ogni coppia di valori ed eseguito l'encode della linea a coppie di due valori.
L'encode della linea di testo, viene eseguita sempre sullo stesso array di char, per il fatto che si andrà a passare il suo puntatore tra le varie funzione.
Qui viene criptato grazie alla matrice contenuta all'interno della struct playFair, secondo delle regole prestabilite.
Una volta criptata l'intera linea, questa viene stampata nel file di output.
Questa operazione viene eseguita per ogni linea del file e per ogni file, passando al file successivo grazie alla struct file che ha un puntatore alla struct del file successivo.

-Decode

Qui per prima cosa vengono rimossi gli spazi contenuti nel messaggio e e controllato se questo sia formato da coppie di valori, nel caso che non lo sia molto probabilmente il file e' stato alterato e quindi il suo messaggio in chiaro non sara piu recuperabile , stessa cosa se e' presente il char_assente , cosa che porterebbe alla stampa di un errore specifico e alla fine della decriptazione del file.
Nel caso che non si abbia uno di questi errori, viene eseguita la decriptazione a coppie di due grazie alla matrice del playFair ed utilizzando delle regole prestabilite.
Una volta finito , vengono aggiunti gli spazi tra le coppie di caratteri e stampato nel file di output.
Questa operazione viene eseguita per ogni linea del file e per ogni file, passando al file successivo grazie alla struct file che ha un puntatore alla struct del file successivo.