Presentazione progetto Cypher-Playfair per Sistemi Operativi laboratorio in C.

Ho diviso il progetto secondo i compiti svolti dalle funzioni e ad ogni file di tipo '.c' ne corrisponde un altro di tipo '.h' con lo stesso nome, dove vengono dichiarati i file creati .

--ReadFile.c: contiene i metodi utili alla lettura dei file di testo .

La funzione readfile() inizia creando un buffer con stessa grandezza del file che vado ad aprire ,questo grazie alla funzione size() che attraverso la funzione fseek() sposta il puntatore alla fine del file e attraverso ftell() restituisce la sua posizione corrente. Poi aperto il file con fopen() , leggo le stringhe del file con la funzione fgets() e attraverso due 'if' controllo ogni carattere affiche siano in maiuscolo, altrimenti vengono convertiti o vengono eliminati come nel caso di '\r' e di '\n' , altrimenti attraverso la funzione ferror() ,genero l'errore . Infine, chiudo il file e restituisco il buffer creato all'inizio.

--Key.c: dato un file già presente nel progetto con i valori da aggiungere alla matrice, contiene tutte le funzioni utili a generare la matrice per la cifratura

La funzione creaMatrice() è quella che si occupa di generare la matrice 5x5 e al suo interno contiene tutte le altre funzioni del file utili allo scopo .Dopo aver passato alla funzione il percorso del file che contiene tutti i valori per la matrice e uno struct (nel nostro caso 'chiave') , creiamo un buffer all'interno della funzione richiamo in ordine :

getkey: salviamo inizialmente all'interno della variabile 'speciale' dello struct 'chiave' il valore alla posizione 30 del buffer(dove c'è il valore 'speciale' ovvero il carattere da utilizzare per spezzare i caratteri doppi o da aggiungere alla fine del messaggio) poi partendo dalla posizione 33(ossia dove inizia la chiave speciale all' interno del file) del buffer scorriamo il puntatore fino alla fine del file assegniamo i caratteri nella matrice con la funzione scorriMatrice() evito le ripetizioni .

scorriMatrice(): è dichiarato all' interno di getkey() e semplicemente controlla la variabile matrice dello 'struct' chiave.

Definito un contatore si controlla tutta la matrice e nel momento in cui trovo una ripetizione il puntatore aumenta e perciò passa alla variabile successiva, se la cella invece è vuota e il contatore è a 0, il carattere è salvato all'interno.

getAlfabeto(): simile a getkey() solo che qui aggiungo nella matrice i valori dell'alfabeto (che si trovano nel file con la chiave) ossia le lettere che non erano presenti nella chiave .Inoltre anche in questa funzione utilizzo scorriMatrice() per controllare la matrice e aggiungere i valori mancanti.

getMancante(): trova la lettera mancante dell'alfabeto non presente nella matrice . Attraverso un variabile boolean capiamo che se il carattere è uguale a carattere della matrice allora il valore diventa 'true' altrimenti se quel valore non è presente nella matrice ,esce dagli ultimi due cicli for e viene salvato all'interno della variabile 'mancante' dello struct chiave .

getSostituto(): salva all'interno della variabile sostituto dello struct 'chiave' il carattere alla posizione 27 del buffer .Questo non è altro che il carattere da utilizzare per rimpiazzare il carattere mancante nel messaggio da cifrare.

Tutti queste funzioni si trovano all' interno di creaMatrice() che una volta richiamate libera il buffer e gli da valore 'Null'.

--Keygen.c: crea il file contenente l'alfabeto, il carattere mancante, quello speciale e la chiave.

Il file keygen.c contiene unicamente la funzione creaChiave() che molto banalmente, passato il percorso del file, scrive i valori passati insieme al percorso all'interno del file, creando un nuovo file con la chiave .

--codifica.c: si divide il testo in coppie e poi attraverso le regole della Cypher-Playfair vengono codificate con l'utilizzo della matrice.

In codifica.c si trova la funzione getCoppieC , ossia la funzione centrale del file. Alla funzione vengono passati due percorsi e uno struct di tipo 'chiave'. Inizialmente creiamo un buffer di nome stinga e creiamo la directory in cui salveremo le coppie codificate attraverso la funzione creaDirectoryC, apriamo il file che corrisponde alla directory di

uscita per scrivere le coppie codificate, poi puliamo la directory e gli assegniamo il valore 'null'. Creiamo l'array coppie che conterranno le coppie create dal file di testo .Creiamo poi un ciclo for pero ogni valore del buffer e dichiariamo altre due funzioni : creaCoppie(),funzione definita in playfair_utils.h che divide il testo da codificare in coppie , e codifica() ,definita in codifica.c e applica le regole per la codifica sulle coppie di testo . Infine, scriviamo le coppie codificate con la funzione fprintf() e utiliziamo la funzione fflush() sul file per pulire il file , lo si chiude col fclose() e anche il buffer stringa viene pulito e gli viene assegnato il valore 'null'.

La funzione già citata , codifica(), applica le regole della decodifica sulle coppie .All'interno della funzione viene prima chiamata la funzione trovaPosizione() che trova la posizione nella matrice dei caratteri nell'array coppie.

Le regole sono da seguire sono : 1)se le due lettere sono in colonne e righe diverse, si prendono le due che fanno rettangolo con esse, cominciando da quella che si trova in linea con la prima lettera del gruppo chiaro;
2)se le due lettere si trovano sulla stessa riga, si sostituiscono con le due lettere che le seguono a destra (se una lettera si trova sulla quinta colonna, si prenderà come successiva la prima lettera a sinistra della stessa riga);
3)se le due lettere sono sulla stessa colonna, si prendono le due lettere sottostanti; (se una lettera è nell'ultima riga, si prenderà come seguente la lettera che sta nella prima riga della stessa colonna);

Il primo 'if' controlla che le due righe dei valori della coppia siano uguali ,se ciò è vero allora si entra all'interno del ciclo e se la colonna del primo valore è 4 allora secondo la regola questo verrà spostato al valore zero, lo stesso anche se la colonna del secondo valore della coppia è 4.Poi si controllano se il valore delle colonne è uguale e si gestisce anche il caso in cui queste abbiano valore 4 . Infine, si gestisce la prima regola per cui se le due lettere si trovano in colonne e righe diverse allora le due lettere si scambiano rispettivamente il valore delle colonne .

L'ultima funzione del file è creaDirectoryC() che crea un buffer con la stessa grandezza della parte del percorso dopo l'ultima '/' e la aumento di quattro poiché il file deve essere di tipo '.pf' e deve contenere lo '\0'. Copio poi la funzione basename() col il percorso all' interno del buffer creato precedentemente. Arrivato a questo punto devo eliminare l'estensione del file perciò uso un ciclo 'for' e lascio solo il nome del file, uscito dal ciclo aggiungo l'estensione per il file della codifica grazie alla funzione strcat() .Abbiamo il file ma non il percorso di uscita quindi creo un buffer capace di contenere il percorso(il +1 è per il '\0') e il file (di tipo .pf) creato prima , copio il percorso per l'uscita che ho passato al nuovo buffer ,aggiungo uno '/' e il file creato prima. Infine, libero lo spazio del buffer che conteneva il file di uscita, lo rendo 'null' e restituisco il nuovo percorso di uscita.

--decodifica.c: si divide il testo in coppie e poi attraverso le regole della Cypher-Playfair vengono decodificate con l'utilizzo della matrice.

Il file decodifica.c presenta tutto il processo di decodifica ed è per quai tutti gli aspetti simile al file della codifica. La funzione centrale è getCoppieD( ) . Alla funzione vengono passati due percorsi e uno struct di tipo 'chiave'. Inizialmente creiamo un buffer di nome stinga e creiamo la directory in cui salveremo le coppie decodificate attraverso la funzione creaDirectoryD, apriamo il file che corrisponde alla directory di uscita per scrivere le coppie decodificate, poi puliamo la directory e gli assegniamo il valore 'null'. Creiamo l'array coppie che conterranno le coppie del file di testo .Creiamo poi un ciclo 'for' per ogni valore del buffer e dichiariamo altre due funzioni : creaCoppie(),funzione definita in playfair_utils.h che divide il testo da decodificare in coppie e decodifica (sempre di decodifica.c) per applicare le regole della decodifica. Infine, scriviamo le coppie decodificate con la funzione fprintf() e utiliziamo la funzione fflush() sul file per pulire il file , lo si chiude col fclose() e anche il buffer stringa viene pulito e gli viene assegnato il valore 'null'.

La funzione decodifica() contiene le regole per la decodifica e come per la funzione codifica() dichiara al suo interno trovaPosizione(). Le regole da seguire sono :1) se le due lettere sono in colonne e righe diverse, si prendono le due che fanno rettangolo con esse, cominciando da quella che si trova in linea con la prima lettera del gruppo chiaro;

2)se le due lettere si trovano sulla stessa riga, si sostituiscono con le due lettere che le precedono a sinistra (se una lettera si trova sulla prima colonna, si prenderà come precedente l'ultima lettera a destra della stessa riga);

3)se le due lettere sono sulla stessa colonna, si prendono le due lettere sopra; (se una lettera è nella prima riga, si prenderà come precedente la lettera che collocata nell'ultima riga della stessa colonna).

Il metodo è simile a codifica(), qui però il caso particolare lo troviamo se i valori sono uguali a zero e non a quattro e quindi lo spostamento delle coppie avviene il contrario.

La funzione creaDirectoryD( ) è uguale alla funzione per la codifica , gli unici cambiamenti sono : l' aggiunta di uno spazio in più dato che il file sarà '.dec' e perciò l' aggiunta con strcat di '.dec'.

--playfair_utils.c : file contenete funzioni utili per la codifica e decodifica.

Le funzioni sono già state tutte e due citate ossia trovaPosizione() e creaCoppie(),la prima semplicemente scorre la matrice e quando trova i valori delle coppie salva i le coordinate su delle variabili (questo grazie a due cicli 'if ') .CreaCoppie() invece è più complessa: passato un buffer con il file , l'array per le coppie, lo struct chiave e un valore int . Inizialmente controlla che i valori della stringa non siano uguali al valore della variabile mancante altrimenti cambiano il valore con quello della variabile sostituto. Poi pongo i valori della stringa nella coppia e passo a fare i controlli: se i due valori sono uguali allora il secondo valore della coppia diventa quello della variabile speciale di chiave e quanto anche se il secondo valore della coppia è '\0',se si entra in questi due controlli si aumenta il valore di int a uno altrimenti a 2 poiché la coppia non deve essere modificate.

Infine, abbiamo il file main.c che si occupa di controllare che il il comando sia corretto

Innanzitutto, si controlla se gli argomenti siano almeno 5 , altrimenti si stampa gli elementi da aggiungere per far partire il comando .Il secondo controllo serve per la codifica , se l' argomento alla seconda posizione è uguale a 'encode' allora richiamiamo la funzione creaMatrice() e poi getcoppieC che codifica il file .Il terzo invece è uguale al terzo ma qui controlliamo che il secondo argomento sia uguale a 'decode' allora richiamiamo la funzione creaMatrice() e poi getCoppieD. Il quarto è uguale ma per 'creaChiave' che richiama creaChiave().

In questo progetto ho imparato come programmare in C, a gestire la memoria dinamica , ho imparato ad usare i puntatori e a sfruttare la programmazione modulare cosi da dividere il progetto in base ai compiti svolti

.