Fondamenti di Informatica
Assegnamento 4

Istruzioni per lo svolgimento e la consegna

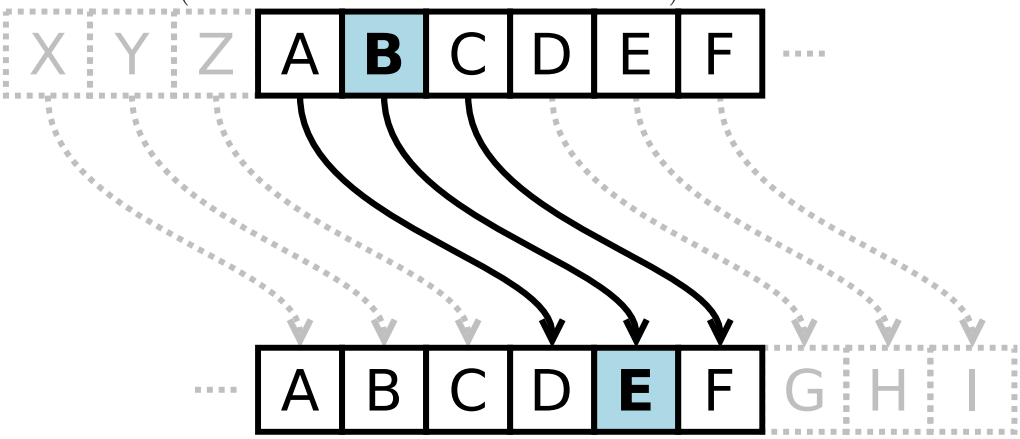
- La prima operazione da effettuare è modificare il file `studente.txt` (presente nella directory dove avete trovato questo `pdf`) inserendo il proprio nome e cognome e numero di matricola. Utilizzare un semplice editor di testo, salvando il file senza modificarne il nome.
- Nella stessa directory sono presenti i file necessari allo svolgimento dell'esercizio. **Per ogni esercizio dovrà essere modificato solamente il file `.c` corrispondente. Non devono essere modificati né spostati o eliminati i rimanenti file, pena la valutazione negativa dell'assegnamento**. Nel file `.c` dovranno essere mantenuti tutti e soli gli output a schermo forniti, in modo da mantenere la corrispondenza con l'output di esempio. Inoltre, è possibile definire funzioni ausiliarie all'interno del sorgente `.c`.
- Per compilare e generare l'eseguibile, da terminale entrate nella directory dove avete trovato questo `pdf` e lanciate il comando `make nome_esercizio`. Il `nome_esercizio` corrisponde al nome del sorgente, privato dell'estensione `.c`. Verrà generato l'eseguibile `nome_esercizio` che, lanciato da terminale (`./nome_esercizio`), vi permetterà di provare il vostro programma.
- Lanciando invece il comando (`./self_evaluation nome_esercizio`) eseguirete in maniera automatica alcuni test per verificare le soluzioni che avete implementato. I test sono studiati per verificare anche i casi particolari, in modo da gestire quelli che possono essere errori comuni in fase di implementazione. **Tenete presente che il correttore funziona solo all'interno di una distribuzione Linux a 64 bit (ad esempio, le macchine messe a disposizione nel laboratorio)**.
- La procedura di consegna dovrà iniziare lanciando il programma `./prepara_consegna.sh` presente nella directory dove avete trovato il presente `pdf`. Una volta lanciato, esso genererà un archivio di nome `consegna.tar.gz`: tale file sarà **l'unico che dovrà essere inviato** attraverso il sito `https://stem.elearning.unipd.it` per consegnare il vostro elaborato, seguendo anche le istruzioni che saranno fornite in aula dai docenti.

Considerate 2 aspetti:

- Se ci sono errori in compilazione/esecuzione, c'è qualcosa che rende errata/incompleta la vostra implementazione;
- Se non ci sono errori in compilazione/esecuzione, verificate che i risultati siano corretti (in alcuni casi è molto semplice fare il calcolo anche a mente).

1 Cifrario di Cesare (`caesar.c`)

Scrivere un programma C che simuli un sistema di crittografia basato sul cifrario di Cesare. Il cifrario di Cesare prevede l'uso di una chiave numerica per cui ogni lettera del testo in chiaro è sostituita, nel testo cifrato, dalla lettera che si trova un certo numero di posizioni (pari alla chiave) dopo nell'alfabeto (solo lettere maiuscole con `'A'`=65 e `'Z'`=90).



Ogni messaggio nel cifrario è composto da due stringhe, con una delle due possibilmente vuota: una codificata e l'altra in chiaro. Dato un array di messaggi e la sua dimensione, sviluppare una libreria che:

- identifichi se è possibile ricavare la chiave del cifrario
- se la chiave è disponibile si proceda alla decodifica di tutti i messaggi
- altrimenti ne richieda una nuova e proceda a codificare tutti i messaggi

La libreria deve essere composta dalle seguenti funzioni:

```c
// Codifica le informazioni in chiaro presenti in messages con la chiave key
void encode(struct cypher messages[], unsigned int size, int key);

// Decodifica le informazioni codificate in messages con la chiave key per rendere in chiaro
void decode(struct cypher messages[], unsigned int size, int key);

// Calcola la chiave di codifica per il cifrario di Cesare dato un messaggio
// contenente sia codifica che decodifica, se necessario gestire errori, ritornare 0
int get_key(struct cypher message);

// Calcola la chiave di codifica per il cifrario di Cesare dato un array di messaggi
// se non ci sono messaggi contenenti sia codifica che decodifica, ritorna 0
int find_key(struct cypher messages[], unsigned int size);

// Verifica un array di messaggi per codificarlo o decodificarlo
void check_messages(struct cypher messages[], unsigned int size); // [implementata]

// Restituisce una chiave di cifratura valida per il cifrario di Cesare
int ask_key(int size); // [implementata]
```

Le informazioni sulla struttura del cifrario sono contenute nel file header `caesar.h`:

```c
// Numero di chiavi presenti nell'alfabeto del cifrario di Cesare
#define NUM_KEYS 26
#define MAX_LEN 50

// Struttura di un messaggio da cifrare/decifrare
struct cypher
{
    char encrypted[MAX_LEN];
    char decrypted[MAX_LEN];
};
```

Il programma di test è già implementato e compilato, fornito in forma di file oggetto `caesar_main.obj`. Per riuscire ad utilizzare usare il comando `gcc`, è necessario linkare il file come segue:

```
gcc -o caesar caesar.c caesar_main.obj
```

Per inserire in input un messaggio vuoto usare il simbolo `@`.

Esempio 1

Input:

```
5
@
IRHB
I
F
@
XJ
@
VLRO
@
CXQEBO
```

Output:

```
Cypher size:
>>> Input Messagge #1 <<<
Decrypted: Encrypted:
>>> Input Messagge #2 <<<
Decrypted: Encrypted:
>>> Input Messagge #3 <<<
Decrypted: Encrypted:
>>> Input Messagge #4 <<<
Decrypted: Encrypted:
>>> Input Messagge #5 <<<
Decrypted: Encrypted:
>>> Output Messagge #1 <<<
Decrypted: LUKE
Encrypted: IRHB
>>> Output Messagge #2 <<<
Decrypted: I
Encrypted: F
>>> Output Messagge #3 <<<
Decrypted: AM
Encrypted: XJ
>>> Output Messagge #4 <<<
Decrypted: YOUR
Encrypted: VLRO
>>> Output Messagge #5 <<<
Decrypted: FATHER
Encrypted: CXQEBO
```

Esempio 2

Input:

```
5
HIGH
@
LOW
@
RIGHT
@
LEFT
@
GOAHEAD
@
```

Output:

```
Cypher size:
>>> Input Messagge #1 <<<
Decrypted: Encrypted:
>>> Input Messagge #2 <<<
Decrypted: Encrypted:
>>> Input Messagge #3 <<<
Decrypted: Encrypted:
>>> Input Messagge #4 <<<
Decrypted: Encrypted:
>>> Input Messagge #5 <<<
Decrypted: Encrypted:
>>> Output Messagge #1 <<<
Decrypted: HIGH
Encrypted: XYWX
>>> Output Messagge #2 <<<
Decrypted: LOW
Encrypted: BEM
>>> Output Messagge #3 <<<
Decrypted: RIGHT
Encrypted: HYWXJ
>>> Output Messagge #4 <<<
Decrypted: LEFT
Encrypted: BUVJ
>>> Output Messagge #5 <<<
Decrypted: GOAHEAD
Encrypted: WEQXUQT
```

2 Monopoli (`monopoly.c`)

Scrivere un programma C che simuli una partita a Monopoli, tenendo traccia del denaro a disposizione per ogni singolo giocatore.

Per ogni giocatore le informazioni sono:

- pedina
- nome
- denaro a disposizione
- case a disposizione
- alberghi a disposizione
- condizione di gioco (inprigionato o no)

Le pedine a disposizione dei giocatori sono:

- Car
- Scottish terrier
- Hat
- Iron
- Battleship
- Penguin
- Thimble
- Rubber ducky

I turni di gioco sono rappresentati da una stringa, ogni carattere rappresenta il turno di un giocatore. All'inizio tutti i giocatori partono dallo stesso capitale, ma possiedono un numero diverso di case e hotel. Se il giocatore ha guadagnato denaro, la lettera sarà minuscola, se invece il giocatore ha perso denaro, la lettera sarà maiuscola. La prima lettera della sequenza è sempre minuscola.
Dato un array di giocatori e la stringa contenente i turni di gioco:

- analizzare la stringa carattere per carattere
- ricavare il personaggio che sta giocando il suo turno
- a meno che il giocatore non sia in prigione
- determinare se il giocatore perde o vince denaro
- se perde, calcolare la somma persa (2*case - alberghi)
- se vince, calcolare la somma vinta (2*alberghi + case)
- aggiornare il saldo del giocatore
- ad ogni turno mostrare il giocatore con il capitale più elevato: pedina, nome, denaro
- concluso il gioco (la stringa) mostrare il vincitore

A tal fine implementare le seguenti funzioni:

```c
// Restituisce la pedina corrispondente al carattere in input
alterego get_turn(char c);

// Restituisce la cifra da pagare se il giocatore ha un debito
int neg_score(int houses, int hotels);

// Restituisce la cifra se il giocatore deve riscuotere un credito
int pos_score(int houses, int hotels);

// Gestisce i turni di gioco e al termine stampa il vincitore
void game_evolution(struct figure players[], char game[]);
```

A supporto sono fornite nel file header `monopoly.h` le seguenti funzioni e tipi:

```c
// Pedine che rappresentano i giocatori sulla plancia di gioco
typedef enum alterego {
    CAR = 0,
    SCOTTISH_TERRIER,
    HAT,
    IRON,
    BATTLESHIP,
    PENGUIN,
    THIMBLE,
    RUBBER_DUCKY,
    LAST_TOKEN,
} alterego;

// Struttura relativa ad un personaggio in gioco
struct figure
{
    alterego token;
    char name[20];
    int money;
    int houses;
    int hotels;
    bool jail;
};

// Stampa il puntenggio a partire da token, nome e denaro di un giocatore
void print_score(alterego token, char name[], int money); // [implementata]

// Stampa le informazioni del giocatore in input e lo dichiara vincitore
void print_winner(struct figure player); // [implementata]
```

Il programma di test è già implementato e compilato, fornito in forma di file oggetto `monopoly_main.obj`. Per riuscire ad utilizzare usare il comando gcc, è necessario linkare il file come segue:

```
gcc -o monopoly monopoly.c monopoly_main.obj
```

Esempio 1

Input:

```
chccShssCHCsshhHcCSSs
```

Output:

```
Number of players:
>>> Player #1 <<<
Token ID: Name: Hotels: Houses: In jail:
>>> Player #2 <<<
Token ID: Name: Hotels: Houses: In jail:
>>> Player #3 <<<
Token ID: Name: Hotels: Houses: In jail:
Game evolution string:
Token: CAR
Player: Harry
Money: 5
Token: HAT
Player: Hermione
Money: 6
Token: CAR
Player: Harry
Money: 10
Token: CAR
Player: Harry
Money: 15
Token: CAR
Player: Harry
Money: 15
Token: CAR
Player: Harry
Money: 15
Token: CAR
Player: Harry
Money: 15
Token: CAR
Player: Harry
Money: 15
Token: CAR
Player: Harry
Money: 15
Token: CAR
Player: Harry
Money: 15
Token: CAR
Player: Harry
Money: 15
Token: SCOTTISH TERRIER
Player: Ron
Money: 17
Token: SCOTTISH TERRIER
Player: Ron
Money: 24
Token: SCOTTISH TERRIER
Player: Ron
Money: 24
Token: SCOTTISH TERRIER
Player: Ron
Money: 24
Token: SCOTTISH TERRIER
Player: Ron
Money: 24
Token: SCOTTISH TERRIER
Player: Ron
Money: 24
Token: SCOTTISH TERRIER
Player: Ron
Money: 24
Token: SCOTTISH TERRIER
Player: Ron
Money: 20
Token: SCOTTISH TERRIER
Player: Ron
Money: 16
Token: SCOTTISH TERRIER
Player: Ron
Money: 23

//////////////////////////////////////
////////////// WINNER /////////////////
//////////////////////////////////////
Token: SCOTTISH TERRIER
Player: Ron
Money: 23
//////////////////////////////////////
```

Esempio 2

Input:

```
pppppppR
```

Output:

```
Number of players:
>>> Player #1 <<<
Token ID: Name: Hotels: Houses: In jail:
>>> Player #2 <<<
Token ID: Name: Hotels: Houses: In jail:
>>> Player #3 <<<
Token ID: Name: Hotels: Houses: In jail:
>>> Player #4 <<<
Token ID: Name: Hotels: Houses: In jail:
>>> Player #5 <<<
Token ID: Name: Hotels: Houses: In jail:
>>> Player #6 <<<
Token ID: Name: Hotels: Houses: In jail:
Game evolution string:
Token: PENGUIN
Player: Phoebe
Money: 0
Token: PENGUIN
Player: Phoebe
Money: 0
Token: PENGUIN
Player: Phoebe
Money: 0
Token: PENGUIN
Player: Phoebe
Money: 0
Token: PENGUIN
Player: Phoebe
Money: 0
Token: PENGUIN
Player: Phoebe
Money: 0
Token: PENGUIN
Player: Phoebe
Money: 0
Token: PENGUIN
Player: Phoebe
Money: 0
Token: RUBBER DUCKY
Player: Joey
Money: 1

//////////////////////////////////////
////////////// WINNER /////////////////
//////////////////////////////////////
Token: RUBBER DUCKY
Player: Joey
Money: 1
//////////////////////////////////////
```