Fondamenti di Informatica
Simulazione

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://esami.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 Liste tandem (`list_tandem.c`)

Lo svolgimento di questo esercizio richiede la comprensione preliminare dei concetti:

- **tandem**: stringa che, ad eccezione dell'eventuale carattere centrale, è costituita esattamente dalla stessa sottostringa ripetuta 2 volte (sono tandem `"anan"` e `"ingeing"`);
- **palindromo**: stringa che, ad eccezione dell'eventuale carattere centrale, è costituita esattamente dalla stessa sottostringa specchiata (sono palindromi `"anna"` e `"ingegni"`).

Inoltre, è importante notare che le stringhe composte da 0 (vuota) o 1 caratteri sono per definizione sia tandem che palindromi.
Per semplicità in questo esercizio considereremo solo caratteri minuscoli (`['a'-'z']`).
Ora, data una lista concatenata di `char`:

```c
// Struttura dati che rappresenta un nodo della lista di char
struct CharNode {
    char value;
    struct CharNode* next;
};
```

sviluppare e testare le seguenti funzioni C:

```c
// 1. Verifica se due nodi contengono lo stesso carattere
bool compare_char(struct CharNode* node_1, struct CharNode* node_2);
```

la funzione riceve come in input 2 puntori a `CharNode` e dovrà restituire `true` se i nodi contengono lo stesso carattere, `false` altrimenti;

```c
// 2. Calcola la lunghezza della lista in input
int list_len(struct CharNode *head);
```

la funzione riceve in input il puntore al nodo iniziale di una lista concatenata di `char` e dovrà restituire il numero di nodi che compone la lista;

```c
// 3. Verifica se la lista in input costituisce un tandem
bool is_tandem(struct CharNode *head);
```

la funzione riceve in input il puntore al nodo iniziale di una lista concatenata di `char` e dovrà restituire `true` se la lista rappresenta un tandem, `false` altrimenti;

```c
// 4. Se la lista in input costituisce un tandem la trasforma in un palindromo
bool tandem_2_palindrome(struct CharNode *head, char palindrome[]);
```

la funzione riceve in input il puntore al nodo iniziale di una lista concatenata di `char` e, nel caso questa sia un tandem, dovrà trasformarla in una stringa palindroma.
Il processo dovrà avvenire mantenendo inalterata la prima ripetizione della sottostringa e invertendo la seconda (ad esempio `"otot"` → `"otto"`, oppure `"radra"` → `"radar"`).
Il risultato dovrà essere salvato in `palindrome`, un array di caratteri preallocato (non c'è bisogno di memoria dinamica), anch'esso fornito come parametro della funzione.
Inoltre, la funzione restituisce `true` se la conversione è avvenuta correttamente, `false` altrimenti (quando la lista in input non rappresenta un tandem).

Le quattro funzioni richieste sono pensate in modo che `compare_char()` possa essere utilizzata per sviluppare `is_tandem()`, che a sua volta può essere integrata per completare `tandem_2_palindrome()`.
In fine, `list_len()` potrebbe risultare utile sia in `is_tandem()` che in `tandem_2_palindrome()`.

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

```
gcc -o list_tandem list_tandem.c list_tandem_main.obj
```

Esempio 1

Lista dati in input:

```
m -> a -> d -> m -> a -> NULL
```

Output:

```
madam
```

Esempio 2

Lista dati in input:

```
j -> NULL
```

Output:

```
j
```

Esempio 3

Lista dati in input:

```
n -> o -> n -> o -> NULL
```

Output:

```
noon
```

Esempio 4

Lista dati in input:

```
t -> a -> n -> d -> e -> m -> NULL
```

Output:

```
Not a tandem!
```

2 Conteggio elementi pari (`count_even.c`)

Scrivere una funzione C che conti il numero di elementi pari all'interno di una matrice quadrata di interi. Ai fni di questo esercizio, 0 è considerato un numero pari.
La funzione da sviluppare è la seguente:

```c
// Conta gli elementi pari all'interno di una matrice quadrata di interi
int count_even(int *matrix, int size);
```

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

```
gcc -o count_even count_even.c count_even_main.obj
```

Esempio 1

Matrice in input:

```
1 2
3 4
```

Output:

```
2
```

Esempio 2

Matrice in input

```
0
```

Output:

```
1
```

Esempio 3

Stringa in input:

```
-1 -2 -3
-4 -5 -6
-7 -8 -9
```

Output:

```
4
```

Esempio 4

Stringa in input:

```
 1  2  1  2
-2 -1 -2 -1
 1  2  1  2
-2 -1 -2 -1
```

Output:

```
8
```