Fondamenti di Informatica
I Appello - 06/02/2023

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 Stock residui (`residual_stocks.c`)

Il materiale nel magazzino di un impianto chimico è gestito a **stock**.
Ad inizio giornata, un certo numero di stock viene distribuito nella fabbrica per la lavorazione.
Ogni stock contiene materiale sigillato in buste, contenti al massimo un'unità di materiale (`1.00`).
Per ogni stock è previsto che sia possibile aprire al massimo una busta di materiale alla volta.
Alla fine di ogni giorno, gli stock tornano in magazzino e il materiale residuo (quello nelle buste aperte) deve essere sigillato nuovamente nelle buste per legge. Al fine di minimizzare gli sprechi, le buste aperte vengono raccolte e il materiale residuo viene sigillato in una stazione separata.
Tale operazione è detta finalizzazione. Il sistema è rappresentato utilizzando una lista concatenata:

```c
// Struttura dati che rappresenta uno stock
// come un nodo di una lista concatenata
struct Stock {
    double value;
    struct Stock* next;
};
```

Il valore all'interno del nodo rappresenta il quantitativo di materiale nello stock misurato in buste. Ad esempio se uno stock contiene 6 buste e mezza di materiale, allora ha un campo `value=6.50`.
Il campo `next` contiene il riferimento allo stock successivo se presente, altrimenti conterrà `NULL`.
Per poter simulare il funzionamento dell'impianto chimico è necessario sviluppare e testare le seguenti funzioni C.

```c
// 1. Rimuove la parte non intera dal singolo stock
//    in input e restituisce il valore del residuo
double floor_stock(struct Stock* stock);
```

la funzione riceve come input un puntore a `Stock` e dovrà:

- aggiornare il valore nello stock mantenendo solo le buste sigillate (parte intera di `value`);
- restituire il valore del materiale residuo (parte razionale di `value`);

per calcolare la parte intera utilizzare la funzione `floor()`, che prende in input un valore `double` e lo restituisce arrotondato all'intero inferiore.

```c
// 2. Finalizza gli stock in input arrotondando i valori all'interno di ogni
//    nodo della lista e crea una seconda lista in cui ne salva i residui
struct Stock* residual_stocks(struct Stock *head);
```

la funzione riceve in input il puntore al nodo iniziale di una lista concatenata di `Stock` e dovrà:

- richiamare la funzione `floor_stock()` per ogni nodo della lista;
- utilizzare l'output della funzione `floor_stock()` per creare una seconda lista di `Stock` allocata dinamicamente che contenga i residui risultanti da ogni nodo, nel corretto ordine;
- restituire come output il puntatore al primo elemento della lista creata a partire dai residui;
- la lista di residui e di stock dovranno avere la stessa lunghezza, se uno stock ha residuo pari a zero, inserire un nodo con residuo pari a `0.00`;
- nel caso la lista di stock in input sia vuota, restituire `NULL`.

```c
// 3. Calcola il totale dei valori presenti nella lista in input
double compute_total(struct Stock *head);
```

la funzione riceve in input il puntore al nodo iniziale di una lista concatenata di `Stock` e dovrà restituire la somma dei valori presenti nella lista di residui.

Le tre funzioni richieste sono pensate in modo che `residual_stocks()` e `compute_total()` siano richiamate in questo ordine dal main (già sviluppato), mentre `floor_stock()` possa essere utilizzata per sviluppare `residual_stocks()`.

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

```
gcc -o residual_stocks residual_stocks.c residual_stocks_main.obj -lm
```

Esempio 1

Lista in input:

```
1.63 -> NULL
```

Output:

```
Total residual: 0.63
Finalized stocks:
1.00 -> NULL
Residual stocks:
0.63 -> NULL
```

Esempio 2

Lista in input:

```
1.00 -> 2.00 -> 3.11 -> NULL
```

Output:

```
Total residual: 0.11
Finalized stocks:
1.00 -> 2.00 -> 3.00 -> NULL
Residual stocks:
0.00 -> 0.00 -> 0.11 -> NULL
```

Esempio 3

Lista in input:

```
1.15 -> 12.88 -> 4.59 -> 77.24 -> NULL
```

Output:

```
Total residual: 1.86
Finalized stocks:
1.00 -> 12.00 -> 4.00 -> 77.00 -> NULL
Residual stocks:
0.15 -> 0.88 -> 0.59 -> 0.24 -> NULL
```

Esempio 4

Lista in input:

```
0.20 -> 0.50 -> 0.99 -> NULL
```

Output:

```
Total residual: 1.69
Finalized stocks:
0.00 -> 0.00 -> 0.00 -> NULL
Residual stocks:
0.20 -> 0.50 -> 0.99 -> NULL
```

2 Caratteri adiacenti ripetuti (`repreated_char.c`)

Date le seguenti definizioni:

- due caratteri sono **adiacenti** se uno dei due si trova immediatamente prima o dopo l'altro all'interno della stringa in input;
- due caratteri sono **ripetuti** se contengono la stessa lettera sia essa minuscola o maiuscola.

Scrivere un programma C che conti il numero di coppie di caratteri adiacenti ripetuti all'interno di una stringa in input. Ogni stringa è composta esclusivamente da lettere o spazi (`[' ', 'A'-'Z', 'a'-'z']`), non saranno presenti spazi ripetuti. Il programma si basa su due funzioni:

```
// 1. Confronta due caratteri per capire se contengono
//    la stessa lettera (maiuscola o minuscola)
bool is_same_letter(char c1, char c2);
```

la funzione riceve in input due caratteri (`char`) e dovrà restituire `true` se i due caratteri contengono la stessa lettera sia essa maiuscola o minuscola, `false` altrimenti. Infatti, questa funzione non è case-sensitive, perciò, ad esempio, i caratteri `'A'` e `'a'` devono essere considerati la stessa lettera.

```
// 2. Conta il numero di coppie di caratteri adiacenti ripetuti nella stringa
int repeated_char(char *str);
```

la funzione riceve in input un array di caratteri e dovrà restituire il numero di coppie di caratteri che rispettano entrambe le definizioni date di caratteri **adiacenti** e **ripetuti**.
L'operazione si conclude una volta raggiunta la fine della stringa (`'\0'`).

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

```
gcc -o repeated_char repeated_char.c repeated_char_main.obj
```

Esempio 1

Stringa in input:

```
alba
```

Output:

```
0
```

Esempio 2

Matrice in input

```
Fatto il misfatto
```

Output:

```
2
```

Esempio 3

Stringa in input:

```
aAa
```

Output:

```
2
```

Esempio 4

Stringa in input:

```
AAAaaaiuto
```

Output:

```
5
```