Esercitazione di Laboratorio - 6

Pagina del corso :
http://didawiki.cli.di.unipi.it/doku.php/fisica/inf/start

Oggi facciamo esercitazione su

- Tipi definiti dall'utente
  - Enumerazione
  - Struct
- Liste
- Allocazione dinamica

Obiettivo: nessun warning!

E' molto importante avere programmi che compilino senza nemmeno uno warning.
Da ora in poi, quindi:

1) useremo `int main()` invece di
   `int main(int argc, char* argv[])`
   quando gli argomenti non vengono usati;
2) assegneremo sempre il valore di ritorno di `scanf` ad una variabile intera;
3) non useremo più la funzione `trunc` ed al suo posto useremo `floor` (o `ceil`).

Il makefile da usare

- Anche oggi useremo lo stesso makefile della scorsa volta

```
# makefile
CC=gcc
CFLAGS=-Wall -g -O -pedantic -Wformat=2 -Wextra -lm
```

ossia quello modificato con l'aggiunta dell'opzione **-lm** (meno elle emme) per la libreria matematica.

- Copiatelo dalla cartella della scorsa volta in una nuova cartella **es06** della vostra home directory, dove metteremo i file di oggi.

Richieste implicite degli esercizi

- Come la volta scorsa, se un esercizio richiede di scrivere una funzione, *implicitamente* vi richiede di scrivere anche un main minimo per preparare dei dati di test e verificare il corretto funzionamento delle procedure scritte.
- Inoltre, dove c'è bisogno di allocazione dinamica della memoria, dovete assicurarvi di deallocare SEMPRE *tutta* la memoria allocata: un esercizio funzionante ma che non deallochi tutta la memoria allocata sarà considerato scorretto.
- Ogni volta che trattate dinamicamente con i puntatori, state molto attenti alla produzione di garbage e ai riferimenti pendenti!

Strutture dati

- In alcuni degli esercizi che seguono (quando richiesto) si utilizzi la seguente definizione di tipo:

```
typedef struct El {
    int info;
    struct El *next;
}  ElementoListaInt;
typedef ElementoListaInt* ListaDiInteri;
```

- Ricordate che dovete includere la libreria `stdlib.h` per la gestione della memoria dinamica.
- Usate la seguente funzione per stampare le vostre liste:

```
void StampaLista(ListaDiInteri lista){
    while (lista != NULL)  {
        printf("%d -->", lista->info);
        lista = lista->next;
    }
    printf("//\n");
}
```

Esercizi proposti

1) Definire un nuovo tipo di dato capace di rappresentare i dipendenti di una ditta: di tali dipendenti interessa il cognome, il numero degli anni di anzianità maturati e lo stipendio.
   Si supponga che la dimensione massima del cognome sia di 40 caratteri.
   Scrivere le seguenti funzioni/procedure:

   - **aggiornaStipendio** che, dato un dipendente, aumenti del 1% il suo stipendio per ogni anno di anzianità accumulato (attenzione l'interesse da calcolare e' un interesse composto).

   - **precede** che dati due impiegati a e b verifichino che il cognome del dipendente a preceda il cognome del dipendente b in ordine alfabetico.

   Suggerimento: per confrontare le stringhe utilizzare la funzione **`strcmp`** dichiarata in **`string.h`**

   Scrivere quindi un programma che richieda all'utente di inserire i dati di due impiegati, utilizzi le due procedure di cui sopra e ne stampi l'esito.

Esercizi proposti

2) Definire un nuovo tipo di dato capace di rappresentare una data. Scrivere poi delle opportune funzioni/procedure che

   - ricevuta una data la aggiorni al giorno successivo (ignorando gli anni bisestili);

   - ricevute due date verifichino che la prima preceda la seconda.

3) Definire una funzione **creaLista** che legga da terminale i numeri che compongono una lista *ListaDiInteri* (vedi slide 5) e restituisca al programma chiamante un puntatore al primo elemento della lista. La lista è di lunghezza indefinita, la lettura termina quando l'utente inserisce un numero negativo. Se l'utente inserisce come primo numero un numero negativo, la funzione dovrà restituire una lista vuota (un puntatore inizializzato a NULL).
   Usare questa funzione per testare le funzioni dei prossimi esercizi.

Esercizi proposti

4) Definire una funzione **deallocaLista** che riceve una *ListaDiInteri* e la dealloca completamente. Usare questa funzione per la deallocazione delle liste nei prossimi esercizi.

5) Definire una funzione (fornirne due versioni, una iterativa e una ricorsiva) **lunghezzaLista** che data una *ListaDiInteri*,restituisca la sua lunghezza.

6) Definire una funzione (sia iterativa che ricorsiva) **primoPari** che data una *ListaDiInteri*, restituisca il puntatore al primo elemento pari nella lista (restituisce NULL se la lista è vuota o non contiene elementi pari).

7) Definire una funzione (sia iterativa che ricorsiva) **minimoPari** che data una *ListaDiInteri*, restituisca il puntatore al minimo elemento pari nella lista (restituisce NULL se la lista è vuota o non contiene elementi pari).

Esercizi proposti

8) Definire una procedura (sia iterativa che ricorsiva) **inserisciQuarto** che data una *ListaDiInteri* ed un intero P, inserisca P dopo il terzo elemento di lista. Quest'ultima viene lasciata inalterata se non contiene almeno tre elementi.

9) Definire una procedura **inserisciDopoUltimoMaggiore** che data una *ListaDiInteri* ed un intero P, inserisca P dopo l'ultimo elemento di lista il cui campo info sia maggiore di P.
Se lista non contiene alcun elemento maggiore di P (o se la lista è vuota), la procedura lo inserisce in testa.

10) Definire una procedura (sia iterativa che ricorsiva) **elimina** che ricevuta una *ListaDiInteri* e un intero X, elimini i primi X elementi e ritorni il puntatore alla testa della lista modificata.

Esercizi proposti

11) Definire una funzione **ordinaLista** che modifica una *ListaDiInteri* data ordinandola in modo crescente.
La funzione non deve usare allocazione dinamica della memoria (malloc e free).

12) Definire una procedura **elementiDi** che data una *ListaDiInteri*, restituisca una nuova *ListaDiInteri* ordinata e senza ripetizioni contenente tutti e soli gli elementi che compaiono nella lista data.
La lista originale deve restare immutata.

13) Definire una procedura **merge** che date due *ListaDiInteri* ordinate, restituisca una nuova *ListaDiInteri* ordinata contenente tutti gli elementi delle due liste.
Le liste originali devono restare immutate.