Esercitazione di Laboratorio - 5

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

Oggi facciamo esercitazione su

- puntatori
- vettori di stringhe
- passaggio di parametri per valore e riferimento
- programmi con argomenti da linea di comando
- ricorsione

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 **es05** della vostra home directory, dove metteremo i file di oggi.

Esercizi su funzioni e procedure

- La maggior parte degli esercizi di oggi riguarderanno la scrittura di funzioni o procedure che non possono essere testate isolatamente
- Quindi per ogni esercizio in cui si chiede una funzione, dovrete scrivere anche un programma con un main *minimo* che testi l'uso della funzione implementata.
- Se la funzione richiede qualche tipo di dato (ad esempio, un array) il main dovrà prepararlo secondo le specifiche.

Esercizi proposti

1) Scrivere una funzione che ricevuto un array di interi, restituisca la posizione dell'ultima occorrenza di un elemento pari nell'array.

```
int ultimo_pari(int arr[], int dim);
```

2) Scrivere una procedura che ricevuto un array di caratteri modifichi il vettore in modo che ogni vocale venga sostituita dal simbolo '$'.

```
void dollarize(char arr[], int dim);
```

3) Scrivere una funzione che verifichi se un vettore è ordinato in senso descrescente, ritornando 1 in caso lo sia o 0 altrimenti.

```
int decrescente(int arr[], int dim);
```

4) Scrivere una funzione che ricevuta una stringa, verifichi se è palindroma, ritornando 1 in caso lo sia, 0 altrimenti.

```
int palindroma(const char* stringa);
```

5) Scrivere una procedura che ricevuta una stringa, la inverta.

```
void inverti(char* stringa);
```

Esercizi proposti

6) Che operazione esegue la funzione seguente?

```
void foo(const char* pa, const char* pb, char* pc) {
    while(*pa)
        *pc++ = *pa++;
    while(*pb)
        *pc++ = *pb++;
    *pc=0;
}
```

Specificare i requisiti richiesti ai dati di input (pa, pb e pc) perché l'esecuzione vada a buon fine.

Scrivere un programma che crei due stringhe casuali di 20 caratteri e una stringa non inizializzata di 40, le passi alla funzione foo e quindi le stampi a video.

Esercizi proposti

7) Scrivere una funzione che riceva una matrice bidimensionale di interi e azzeri ogni elemento il cui contenuto è maggiore o uguale alla somma delle sue coordinate.
```
void azzera(int mat[][5], int numRighe);
```

8) Scrivere un programma che prende un numero da linea di comando e stampi il suo quadrato.
Suggerimento: usare la funzione **`atoi`** per convertire una stringa in un numero (**`man atoi`** per il manuale).

9) Scrivere un programma che prende da linea di comando tre argomenti in quest'ordine: un numero, un carattere fra **+**, – , * e **/** e un altro numero, e stampi il risultato della corrispondente operazione (o un messaggio di errore se qualcosa non va).
Suggerimento: usare la funzione **`atoi`** per convertire una stringa in un numero.

Esercizi proposti

10) Scrivere una funzione ricorsiva che presa una stringa, sostituisca tutte le occorrenze del carattere A con il carattere B.

```
void replace(char *s, char A, char B);
```

Suggerimento: la funzione verifichi se il primo carattere è da sostituire e quindi ricorra sulla parte restante del vettore, individuata da s+1 (vedi “aritmetica dei puntatori”).

Suggerimento 2: pensate bene a qual'è la condizione di terminazione della ricorsione.

Esercizi proposti

11) La funzione di Ackermann è una delle più semplici funzioni totalmente computabili a non essere ricorsiva primitiva.
http://it.wikipedia.org/wiki/Funzione_di_Ackermann
In pratica, la funzione cresce *più velocemente* di qualsiasi funzione ricorsiva primitiva (compreso qualsiasi esponenziale).

La funzione è definita ricorsivamente per casi (sui naturali):

- $A(m,n) = n+1$ *(se m=0)*
- $A(m,n) = A(m-1,1)$ *(se m>0 e n=0)*
- $A(m,n) = A(m-1, A(m, n-1))$ *(se m>0 e n>0)*

Scrivere una funzione che calcoli la funzione di Ackermann.

```
unsigned long Ackermann(unsigned long m, unsigned long n);
```

Quanto vale Ackermann(3,10)? Quanto vale Ackermann(4,1)?
Quanto tempo ci mette a calcolare questi valori?

Avvertenza: non andate oltre questi limiti (soprattutto su m) o l'esecuzione potrebbe non terminare in tempo per il fine settimana...