Esercitazioni di Tecniche di Programmazione

Due avvertenze:

1) Le soluzioni agli esercizi, le versioni di programmi dati nel testo delle esercitazioni e quant'altro sono raggiungibili tramite la pagina web del corso. Nel titolo di ogni sezione è specificato tra parentesi il nome del (o dei) file in cui è proposta una soluzione (se disponibile ...).
2) I programmi che scriveremo dovranno essere in accordo con la definizione standard **ANSI C** del linguaggio C; perciò, prima di cominciare vogliamo assicurarci che l'ambiente di programmazione che usiamo "usi" anche lui la medesima definizione.
    a. SE si usa il Dev C++, nella versione 4.9.9.2 (lingua inglese) bisogna andare nel menu' "Tools", selezionare "Compiler Options", scegliere "Settings" e poi "C Compiler" (selezionare almeno "Support all ANSI Standard C Prorams")
    b. Se si usa il vecchio ed eroico Turbo C++, per essere sicuri di star usando ANSI C, bisogna assegnare opportunamente una certa opzione: aprire il menù OPTIONS, selezionare *Compiler* e poi *Source*. Nella finestra di scelta che appare, selezionate ANSI C.

6. Sesta esercitazione autoguidata: file binari e ricorsione

6.1. File binari (BINARI1.C)

Scrivere un programma che usa una variabile double `val` e un array di double `arrayDouble` per contenere certi valori double. Il proogramma deve leggere da input il nome di un file e memorizzare in tale file (gestito come FILE BINARIO), usando la funzione `fwrite()`. **E' richiesto** che il terzo parametro delle chiamate di `fwrite()` sia, in questo esercizio, sempre uguale ad 1.

- il contenuto di `val`;
- il contenuto di `arrayDouble`

6.2. File binari (BINARI2.C)

Scrivere un programma che usa una variabile double `val` e un array di double `arrayDouble` per contenere certi valori double. I valori contenuti in `val` e `arrayDouble` devono essere letti da un file binario (quello costruito nell'esercizio precedente), usando la funzione `fread()`. **E' richiesto** che il terzo parametro delle chiamate di `fread()` sia, in questo esercizio, sempre uguale ad 1.

6.3. File binari (BINARI3.C, BINARI4.C)

Ripetere gli esercizi dei due punti precedenti. Il limite imposto precedentemente sul terzo parametro delle funzioni `fread()` ed `fwrite()` e' cancellato ...

6.4. File binari (BINARI5.C)

Scrivere un programma che

- chieda e legga un numero intero `n`
- e legga poi `n` numeri double
- memorizzando tutti i dati letti (interi e double) in un file binario il cui nome e' stato anche letto da input (si suggerisce di farlo prima di leggere i dati numerici)

Il file prodotto conterra', come primo dato il valore `n` e conterra' successivamente tutti i double letti. Non serve usare array!

6.5. File binari (BINARI6.C)

Scrivere un programma che
- chieda e legga il nome di un file binario, analogo a quelli prodotto con il programma dell'esercizio precedente)
- legga dal file un numero intero `n`
- e legga i successivi n numeri double, memorizzandoli in un array dinamico di `n` double (cioe' allocato "esattamente")

6.6. Quadrilateri e file binari (QUADRI2.C)

Scrivere un programma capace di leggere una sequenza di quadrilateri da standard input (stdin, la tastiera) memorizzandoli in un file binario.
Il programma deve chiedere il nome del file binario in cui effettuare le memorizzazioni e poi deve leggere quadrilateri e memorizzarli fintantoche' l'utente intende continuare a fornire dati.

6.7. Quadrilateri e file binari (QUADRI3.C)

Scrivere un programma capace di leggere da un file costruito dal programma precedente.
Il programma deve chiedere il nome del file di quadrilateri da usare come file di input ed utilizzarlo per leggere i quadrilateri in esso contenuti. Ogni quadrilatero deve essere stampato in standard output (sul video).

Piu' giu' c'e' un suggerimento: non guardarlo.

Suggerimento:
ridefinire le funzioni di
- lettura di un punto (funzione che riceve l'indirizzo di un punto e una variabile file, gia' legata ad un file binario, e legge dal file unpunto con fread());
- e di lettura di un quadrilatero (funzione che riceve l'indirizzo di un quadrilatero e una variabile file, gia' legata ad un file binario, e legge dal file i quattro vertici del quadrilatero, usando la funzione di lettura punti da file).

6.8. Quadrilateri e file binari (QUADRI4.C)

Scrivere un programma capace di

- ricevere il nome, `n1`, di un file binario, contenente quadrilateri; (si suppone che questi quadrilateri siano non degeneri, cioe' aventi, quali vertici, quattro punti distinti)
- produrre un nuovo file (sempre binario, e il cui nome, `n2`, e' stato fornito in input) contenente i quadrilateri registrati nel file `n1` che siano quadrati.

Piu' giu' qualche suggerimento

Suggerimento:
`n1` viene usato come file di input, aperto in lettura; `n2` sara' il file di output, aperto in scrittura.
Si legge un quadrilatero da n1, si controlla e , se e' un quadrato, lo si registra in n2.
Dato che i due file rimangono aperti contemporaneamente, servono due variabili FILE da collegare ad essi.

Seguono altri suggerimenti …

Suggerimento 2 di 5:
Per controllare se un quadrilatero e' un quadrato sarebbe elegante usare una funzione `isQuadrato()` che, ricevendo un quadrilatero, restituisca 1 se si tratta di un quadrato e 0 altrimenti.

Suggerimento 3 di 5:
Un'ida della parte di algoritmo che si occupa di estrarre dati dal primo file e (decidere se) riversarli nel secondo:

```
mentre il file di input contiene dati
          - si legge un quadrilatero
          - si controlla se e' quadrato e,
            se e' quadrato, si registra nel file di output
```

Suggerimento 4 di 5:
Supponendo che quad sia il quadrilatero e f2 la variabile file associata al file di output, l'istruzione di registrazione del quadrilatero nel file potrebbe essere:

```
fwrite(quad, sizeof(TipoPunto), 4, f2)
```

Suggerimento 5:
Tra i vari controlli da inserire nel programma, ci sono :

- controllo se il file di input e' stato aperto regolarmente;
- controllo se il file di output e' stato aperto regolarmente;
- controllo se la registrazione di un quadrilatero nel file di output e' stata effettuata con successo (ricordarsi che una chiamata ad `fwrite()` restituisce il numero di registrazioni che sono state effettuate con successo, quindi se la chiamata doveva registrare 4 punti e lo ha fatto con successo, restituisce 4.

6.9. La funzione fattoriale(FATT.C)

Scrivere una funzione `int fatt(int)` che, ricevendo un numero intero `n` calcola e restituisce il valore di `n!`.

Scrivere inoltre un programma capace di testare questa funzione e verificare che, usando nella definizione della funzione il tipo int (per parametro e valore ritornato) la funzione produce risultati troppo grandi anche con numeri relativamente piccoli. (Fino a che valore di `n` si riesce ad arrivare senza andare in *overflow*?)

6.10. La funzione fattoriale(FATTLONG.C)

Ripetere l'esercizio precednte, ma usando il tipo long int per il risultato della funzione. Verificare che questa funzione permette di fare calcoli su numeri (un pò) più grandi rispetto alla precedente.

6.11. La funzione LeggiEInvertiInput() (INVINP.C)

Scrivere una funzione che esegue la lettura di una sequenza di carattero fornita da input e terminata dal carattere '.', producendo in output una stampa dei medesimi caratteri (escluso il punto) in ordine inverso rispetto a quello con ui sono stati inseriti in input. Ad esempio, se l'input fosse TOPI. l'output dovrebbe essere IPOT

Si vuole che la funzione implementi un algoritmo ricorsivo per risolvere il problema. Provare la funzione con un opportuno programma.

…

Suggerimento:
L'idea di quanto visto a lezione era la seguente: l'algoritmo da realizzare deve

- leggere il primo carattere che si presenta in input e memorizzarlo in una variabile `ch`;
  (ad esempio T viene messo in `ch`)
- chiedere che il resto dell'input venga letto e stampato al contrario
- se l'operazione indicata al punto precedente e' stata realizzata, ora in output ci sono, stampati in ordine inverso, i caratteri che erano forniti in input dopo quello che abbiamo memorizzato in `ch`
  ad esempio ora in output c'e' **IPO**
  quindi e' sufficiente stampare il carattere attualmente in ch e l'output assume la forma richiesta
  IPO**T**

…

Suggerimento 2 di 3:
Chiamiamo LEGGIEINVERTI la sequenza di operazioni che vogliamo implementare (insomma, l'algoritmo ...)

Lo schema base da realizzare e' il seguente:

- lettura di un carattere in `ch`
- LEGGIEINVERTI
  (cosi' viene letto e invertito il resto dell'input (quello dopo il carattere memorizzato in ch)
- stampa `ch`

Suggerimento 3:
Ovviamente se il carattere letto e memorizzato in `ch` durante l'esecuzione di LEGGIEINVERTI e' `'.'`, non deve essere richiesto di leggere e invertire "il resto dell'input" (non c'e'!), ne' di stampare il carattere `ch`.
Queste due operazioni vanno eseguite solo se il carattere letto e' diverso da `'.'`.

Quindi il caso in cui `ch!='.'` e' un CASO RICORSIVO, in cui l'algoritmo LEGGIEINVERTI decide di eseguire se stesso sul resto dell'input e poi stampare ch.

Invece il caso in cui `ch=='.'` e' un CASO BASE (in cui non c'e' attivazione ricorsiva e, in questo caso particolare, non bisogna fare nulla).

```
- lettura ch
- se ch!='.'
     o LEGGIEINVERTI
     o stampa ch
  (altrimenti niente)
```

6.12. La funzione palin() (PALIN1.C)

Scrivere una funzione palin() che, ricevendo una parola (stringa di caratteri) e due indici interi, i, j, restituisca 1 se la porzione di parola compresa tra i caratteri di indice i e j e' palindroma.

Provare la funzione con un opportuno programma.

…

Suggerimento:
Una parola e' palindroma se e' leggibile indifferentemente da sinistra a destra o da destra a sinistra (anilina, asor-rosa).

…

Suggerimento 2 di 5:
Sia p la parola (un array di N caratteri, occupato dal carattere di indice 0 al carattere di indice `k` (con `strlen(p)` uguale a `k+1` e `p[strlen(p)]=='\0'` e quindi escluso dalle nostre considerazioni).

La porzione `p[0]---p[k]` (cioe' la stringa)

- non e' palindroma se `p[0]!=p[k]`
- **potrebbe** essere palindroma se `p[0]==p[k]`(e in tal caso sarebbe effetivamente palindroma **solo se** fosse palindroma la porzione `p[1]---p[k-1]`)

….

Suggerimento 3 di 5:
Dal suggerimento precedente viene uno schema come il seguente

`palin(p, i, j)` **e'** (ritorna come risultato)
- 0 se `p[0]!=p[k]`
- il valore di `palin(p, i+1, j-1)` se `p[0]==p[k]`

…

Suggerimento 4 di 5:
Lo schema visto al punto precedente vale se i<j;
infatti,
- se `i==j` non c'e' nulla da controllare! E se siamo arrivati ad eseguire la valutazione di `palin(p,i,j)` con `i==j` vuol dire che tutte le coppie `p[i], p[j]` con i valori precedenti di `i` e `j` erano a caratteri uguali. A questo punto la stringa si rivela palindroma e possiamo restituire 1 senza ulteriori attivazioni ricorsive;

- se `i>j`, vale un discorso analogo: tutte le coppie `p[i], p[j]` con i valori precedenti di i e j (cioe' con valori per cui `i<j`) erano a caratteri uguali. Continuando a fare verifiche su coppie di caratteri `p[i], p[j]` con `i > j` controlleremmo coppie gia' controllate (e che erano uguali). Quindi anche in questo caso la stringa si e' rivelata palindroma e possiamo restituire 1 senza ulteriori attivazioni ricorsive.

Suggerimento 5:
Dal suggerimento precedente viene uno schema come il seguente

`palin(p, i, j)` **e'** (ritorna come risultato)
- se `i<j`
  - **`0`** se `p[0]!=p[k]`
  - **`palin(p, i+1, j-1)`** se `p[0]==p[k]`
- altrimenti **`1`**

6.13. cosa fa?

Osservare il seguente programma:

- la main() contiene due errori, facilmente evidenziabili. Quali sono?
- cosa fa la funzione `cosaFa()` e, quindi, cosa fa il programma, una volta corretto? Giustificare la risposta **e poi** vedere se è giusta, eseguendo il programma.

```
#include <stdio.h>
#define MAX 6

int cosaFa(int a[], int k, int dim) {
      if (k < dim)
            return (a[k] + cosaFa(a, k+1, dim));
      else return 0;
}

int main(){
      int io[MAX], j;
      printf("\nInserire le %d cifre del numero di matricola (uno spazio tra
l'una e l'altra): ", MAX);

      for(j=0; j<=MAX; j++)
            scanf("%d",io+j);

      printf("ecco qua: %d\n", cosaFa(io,2,MAX);
      printf("\nFINE\n");
return 0;
}
```