Laurea In Ingegneria dell'Informazione

Esercitazioni Guidate di Tecniche della Programmazione

Note introduttive:

1) Nel titolo di ogni sezione di questo documento è specificato tra parentesi il nome del (o dei) file in cui è proposta una soluzione (se disponibile nella directory "`programmi`" di questa EG).
2) I programmi che scriveremo dovranno essere in accordo con la definizione standard **ANSI C** del linguaggio C.
    a. Se usate un sistema diverso dal DEV, provvedete a che la compilazione avvenga con il compilatore standard C.
    b. Ricordate che un programma C e' in un file con estensione "`.c`"
    c. Se usate il DEVC++, per configurare bene la compilazione bisogna
        i. andare nel menù "Tools", selezionare "Compiler Options", scegliere "Settings" e poi "C Compiler"; poi selezionare almeno "Support all ANSI Standard C Programs")
        ii. (se l'interfaccia è in italiano …) andare nel menu' "Strumenti", selezionare "Opzioni di compilazione", "Compilatore", "Generazione di Codice …", "Compilatore C" e poi far apparire "Yes" almeno accanto a "Supporto programmi ANSI standard C.

NB Ci sono svariati esercizi che sono proposti nello slide delle lezioni e che potrebbero essere affrontati, o confermati se li avete già fatti) durante la sessione di "laboratorio". Se li avete fatti fatemeli vedere. Se non li avete fatti provate a farli …

5. Esercitazione 5 – Parte 1

5.1. Esercizi introduttivi (PRIMA.C).

Scrivere (e provare) un programma che usa le seguenti funzioni (da definire opportunamente):

- funzione **`leggiArray()`** che riceve un array di interi e ne legge e assegna gli elementi
- funzione **`stampaArray()`** che riceve un array di interi e ne stampa gli elementi
- funzione **`sommaPrimiM()`** che riceve un array di interi **`arr`** e un valore **`m`** e restituisce la somma dei primi **`m`** elementi dell'array (o di tutti gli elementi se **`m>N`**)
- funzione **`trovaElemento()`** che riceve un array di interi **`arr`** un valore **`val`** e restituisce il puntatore ad un elemento val presente nell'array (o **`NULL`** se non c'è)

Il programma che fa uso delle funzioni sopra dichiarate e definite potrebbe eseguire le seguenti operazioni:

- si legge un array di **`N`** interi
- si chiede e legge da input un valore **`m`** e si stampa la somma dei primi m elementi dell'array (controllando che ci siano **`m`** elementi)
- si chiedono e leggono **`m`** e **`k`** e si stampa la somma degli **`m`** elementi dell'array che cominciano dal **`k`**-esimo (supponendo che ci siano)
- si chiede e legge un valore da trovare nell'array e si stampa se quel valore c'è o no nell'array
- si chiede e legge un valore da trovare nell'array e si stampa se quel valore c'è o no nell'array

5.2. Espressione Condizionale

C'è un operatore di cui non abbiamo parlato … l'operatore CONDIZIONALE: **`?:`**
Si tratta di un operatore TERNARIO, che cioè prende tre operandi, permettendo di ottenere una ***espressione condizionale***:

`(condizione) ? (exp1) : (exp2)`

Questa espressione si valuta così:
se la **`condizione`** vale **`TRUE`**,
    l'espressione ritorna il valore di **`exp1`**;
altrimenti l'espressione ritorna il valore di **`exp2`**

5.2.1. conditional expression problem 1

Scrivere la definizione della funzione **`isEven()`** ... (cioè *"è pari"*) per far funzionare bene il programma proposto in **`conditionalExpr1Problem.c`**

La funzione
- riceve un valore intero
- e restituisce 1 o 0, a seconda se il valore è pari o dispari.

La funzione deve usare un'espressione condizionale per determinare quale valore restituire.

Ah, quando guardi la soluzione, per confrontarla, c'è un errore nel programma ... correggilo. Ricorda che il compilatore prima si accorge dell'errore e poi, subito dopo, si ferma, indicando l'errore.

Soluzione in **`conditionalExpr1.c`**

5.2.2. conditional expression problem 2

Il programma in **`conditionalExpr1Problem.c`** risolve un'equazione di secondo grado, i cui coefficienti sono dati in input.
Aggiungere la definizione della funzione **`soluzioniComplesse()`**, che
- riceve i coefficienti e
- restituisce 1 se sono ammesse soluzioni complesse coniugate, 0 se sono ammesse soluzioni reali.

La funzione deve usare un'espressione condizionale per determinare quale valore restituire.

Oh, ci sono degli errori da correggere nel programma soluzione **`conditionalExpr2.c`**

5.2.3. conditional expression problem 3

Scrivere un programma in cui vengono letti (o assegnati mediante inizializzazione) tre numeri interi e viene usata una funzione maxTra3() per stamoare infine il massimo tra i tre numeri.

Il clou è la definizione della funzione maxTra3(), che

- riceve tre numeri interi
- e restituisce il massimo tra loro.

La funzione usa un'espressione condizionale per determinare quale valore restituire.

Provare il programma per diversi input, scrivendo in fondo al file quali test si sono eseguiti e per ciascuno quale giustificazione lo rende necessario (o non ridondante)

Soluzione +/- in **`conditionalExpr3.c`**

5.3. Trova e Conta (estensione di 5.1)

Estendere il programma svolto in 5.1, con una funzione **`trovaEcontaPO()`** che

- riceve almeno un array di interi `arr` ed un valore `val`
- e restituisce
  - il puntatore ad un elemento **`val`** presente nell'array (o `NULL` se non c'è)
  - **e, tramite un parametro di output `q`** il numero di occorrenze di **`val`** nell'array

(Per ***parametro di output***, qui, intendiamo che il parametro formale **`q`** sarà un puntatore e il parametro attuale corrispondente a **`q`** sarà l'indirizzo di una locazione di memoria che verrà modificata con il valore calcolato come risultato della funzione (si intende risultato non inviato via **`return`**).

5.4. Passi1

Un’azienda che sviluppa smartwatch per la fitness vuole analizzare i dati raccolti da un utente durante la giornata.

In particolare, durante una sessione di allenamento, lo smartwatch registra il numero di passi effettuati in ciascun minuto.

Gli ingegneri dell’azienda vogliono scrivere un programma in C che analizzi questi dati per ricavare:

- il numero massimo di passi fatti in un minuto (cioè l’intensità massima dell’attività),
- la media dei passi fatti in un minuto (cioè l’intensità media dell’allenamento).

Scrivere un programma che:

- Legga un array di N valori interi (N una costante), ciascuno rappresentante i passi eseguiti in un minuto; chiamiamo **`passiFatti`** questo array

- Chieda ad una funzione di calcolare il massimo e la media dei valori, usando almeno un parametro di output per restituire uno dei due risultati;

- Stampi i risultati sullo schermo.

Chiamare la funzione **`analizzaPassi()`**
Chiamare la costante che dimensiona l'array **`MAX_MINUTI`**

Soluzione in **`passi1.c`**

Suggerimenti seguono

Suggerimento

una possibile decisione, riguardo ad **`analizzaPassi()`** è di far restituire la media come risultato "normale" (con return, e il massimo tramite parametro di output)

Suggerimento
Una possibile decisione e` che **`analizzaPassi()`** debba

- ricevere un array **`passi[]`** di **`MAX_MINUTI`** elementi;

- ricevere l'indirizzo di una variabile della funzione chiamante, in cui verra` riversato il valore massimo (e questo è il parametro di output ... **`massimo`**)
E massimo è un puntatore … ok?

- calcolare il massimo in **`passi[]`** e restituirlo tramite il parametro di output

- calcolare la media in **`passi[]`** e restituirla con **`return`** (valore di ritorno della funzione)

Suggerimento
Nel programma, la chiamata potrebbe essere
```
mediaPassi = analizzaDati(passiFatti, &maxPassiFatti);
```

dove **`mediaPassi`**, **`passiFatti`**, **`maxPassiFatti`**
sono variabili della funzione chiamante

Suggerimento
intestazione
```
double analizzaPassi(int passi[MAX_MINUTI], int *massimo);
```

5.5. Passi2

Il programma precedente è vincolato all'uso "completo dell'array".
Se **`MAX_MINUTI`** era 6, abbiamo probabilmente fatto 6 iterazioni dell'istruzione di lettura.
Ma così il programma va bene solo se l'array è di 6 minuti.

Possiamo cambiare la costante per adattare il programma a funzionare con UN ALTRO numero DI DATI, ma uno solo!

E ogni volta che il numero di minuti cambia, dobbiamo porre mano al programma e ricompilarlo. *Vabbe`, non e` terribile,*

… ma è terribile!

Invece in questo esercizio cerchiamo di fare in modo che il programma non abbia bisogno di essere modificato per funzionare con qualsiasi numero di dati, fino ad un certo valore massimo ...

Modificare il programma fatto prima, in modo che il funzionamento sia simile ma il numero di dati inseriti nell'array **`passiFatti[]`** sia deciso durante l'esecuzione della fase di input.

Insomma, se vengono inseriti 8 dati, i calcoli verranno fatti solo su quegli otto dati; se vengono inseriti 88 dati, i calcoli verranno fatti solo su quegli 88 dati, e così via.
Ovviamente più di **`MAX_MINUTI`** dati non si potranno inserire in input.

Soluzione in **`passi2.c`**

Suggerimenti seguono

Suggerimento

- definire **`MAX_MINUTI`** come 100
- usare una funzione **`int inputDati()`** che
  - esegua le operazioni di input su **`passiFatti[MAX_MINUTI]`**
  - e restituisca il numero di dati inseriti (**`numDati`**).
- usare una versione modificata di **`analzzaDati()`** in cui la scansione dell'array **`passiFatti`** sia limitata al numero di dati effettivamente inseriti nell'array.

Suggerimento
Per prima cosa riscrivi la **`main()`** limitatamente a quel che serve: il programma fornisce un output del tutto simile a quello precedente.
In particolare

- se non già fatto, la lettura dei dati venga effettuata facendo una chiamata di funzione **`leggiMinuti()`**

- la chiamata della funzione **`analizzaMinuti()`** va riscritta:
  se non ti viene in mente come
  1) chiudi gli occhi e pensa alla situazione in cui il prof ti dice "la funzione dovrà analizzare solo i dati immessi",
  2) riscrivila uguale a com'era prima; poi vedrai che succede e comunque ci sono altri suggerimenti utili

Suggerimento
intestazione

```
int leggiMinuti(int passi[MAX_MINUTI]) {...}
```

così` la chiamata potra` essere
```
numDati = leggiMinuti(passiFatti);
```

e la funione chiamante (**`main()`**) avra` in **`numDati`** il numero di dati effettivamente inseriti nel suo array **`passiFatti[]`**

Suggerimento
l'array e` grande, così` il ciclo di lettura dati potrebbe essere introdotto da **`"dammi i dati sui passi e termina l'inserimento con -1"`**
Qui non possiamo usare 0 come terminatore ... uno potrebbe aver fatto 0 passi in un minuto ...

Suggerimento
la funzione **`analizzaDati()`** dovrebbe avere un parametro addizionale, per fermare il proprio lavoro quando ha analizzato tutti e soli i dati immessi.

Definiamo **`quantiDati`** questo parametro formale ...

Suggerimento
l'intestazione potrebbe essere

```
double analizzaDati(int passi[MAX_MINUTI], int quantiDati, int *massimo);
```

Suggerimento
la chiamata potrebbe essere
```
mediaPassi = analizzaDati(passiFatti, numDati, &maxPassiFatti);
```

dove **mediaPassi, passiFatti, maxPassiFatti** e **numDati** sono variabili della funzione chiamante.

5.6. Passi3

Modificare il programma precedente in modo che entrambe **`leggiMinuti()`** e **`analizzaDati()`** sino funzioni **`void`**

Soluzione in **`passi3.c`**

Suggerimenti seguono

Suggerimento
`Intestazione`

```
void leggiMinuti(int passi[MAX_MINUTI], int *quantiLetti) {...}
```

Suggerimento
la chiamata potrebbe essere

```
leggiMinuti(passiFatti, &numDati);
```

Suggerimento
l'intestazione potrebbe essere
```
void analizzaDati(int passi[MAX_MINUTI], int quantiDati, int *massimo, double media);
```

Suggerimento
la chiamata potrebbe essere

```
analizzaDati(passiFatti, numDati, &maxPassiFatti, double mediaPassi);
```

5.7. Passi4

Con le nozioni acquisite durante la lezione online sui file, realizzare una nuova versione del programma precedente, che riceve i dati per l'array **`passiFatti[]`** da un file **`passiFatti.txt`**.

Nessuna soluzione offerta ☹

5.8. Simulazione di strcpy(): strCopia() (`strCopia.c`).

Vogliamo scrivere una funzione che possa sostituire **`strcpy()`**.

Scrivere un programma in cui
- viene letta da input una stringa, **`stringaLetta`**, di caratteri alfanumerici, senza spazi bianchi all'interno,
- la stringa letta viene copiata ina stringa **`stringaCopiata`**,
- e poi entrambe le stringhe vengono stampate.

Dopo aver scritto il programma, contenente la chiamata a **`strCopia()`**, inserire il prototipo della funzione in cima, e la definizione della funzione in fondo.

Eseguire il programma in modo simulato, disegnando la mappa di memoria e il RDA della chiamata.

5.9. Concatenazione di stringhe (`concatenaStringhe.c`).

Scrivere e provare un programma che usi tre stringhe dimensionate staticamente (al più di 99 caratteri significativi). Il programma deve leggere due stringhe (la prima priva di spazi bianchi) e concatenarle in una terza stringa, che poi viene stampata. Definire ed usare le funzioni **`stringlung`** (che calcola la lunghezza di una stringa ricevuta come argomento) e **`concatena`** (che riceve tre stringhe e riempie la terza con la concatenazione delle prime due. Prova a progettare il programma (algoritmo per la **`main`**, prototipi per le funzioni, algoritmi per le funzioni. Dopo averci almeno provato, vedi la soluzione (i commenti, gli algoritmi, il codice, proposta in **`stringhe1.c`**).

5.10. (`sequenza.c`).

Scrivere e provare un programma che legga una sequenza di numeri interi, terminata da 0, stampi il massimo tra tali valori, la loro media e stampi anche i valori che superano la media.
I valori dati in input non saranno più di **`N`** (ad esempio con **`N=10`**).

Si userà un array di 10 locazioni intere.
I valori dati in input verranno memorizzati negli elementi dell'array, a partire da quello di indice 0. Dato che i numeri sono al massimo N, molto spesso solo una parte dell'array sarà occupata dai valori letti, mentre altre componenti saranno non significative.

Per distinguere tra gli elementi significativi e quelli non, utilizziamo una variabile addizionale **`numeroDati`**, intera, che contiene il numero di dati che sono stati effettivamente inseriti da input prima dello 0.

Dopo la lettura dei dati, gli elementi significativi dell'array saranno quelli che vanno da indice 0 a indice **`numeroDati-1`**.

Soluzione proposta in **`sequenza.c`**.

5.11. Allocazione dinamica (`ardouble0.c`, `arrdoubl.c`)

Scrivere e provare un programma che chieda il nome di un file testo contenente numeri reali e memorizzi tali numeri in un array di double allocato dinamicamente in base al numero effettivo di dati da memorizzarvi.

Un suggerimento per la soluzione potrebbe essere:

- il programma legge il nome del file in una variabile nomefile;
- apre un file di testo di nome nomefile
- conta i numeri ivi contenuti
- alloca un blocco di tanti double quanti ne servono, puntato dalla variabile **`arrayDouble`**
- rilegge i numeri dal file, memorizzandoli nell'array appena allocato
- e poi stampa i valori contenuti nell'array, se non si tratta di un array vuoto (**`NULL`**)

5.12. ARRAY DI STRINGHE "ESATTE" (presenz1.c)

Scrivere e provare un programma che legge da input N stringhe prive di spazi bianchi e di lunghezza al massimo **`MAXLUNG`**, memorizzandole in un array di stringhe. La memorizzazione delle stringhe deve essere "esatta" (cioè ciascuna stringa deve occupare solo il numero di caratteri ad essa necessario). (**`N`**=5, **`MAXLUNG`**=50 ad esempio).
Il programma deve poi chiedere in input una stringa e dire se essa è presente nell'array (usando una funzione `presente` appositamente definita, che riceve un array di stringhe e una stringa e restituisce 1 se la stringa è presente nell'array e 0 altrimenti). Se la stringa cercata non è presente nell'array, la si ristampa rivoltata.

Soluzione proposta in **`presenz1.c`**.

5.13. ANCORA ARRAY DI STRINGHE "ESATTE" (`presenz2.c`)

Ripetere l'esercizio precedente, ma stavolta bisogna poter eseguire numerose ricerche: l'utente viene invitato a chiedere diverse stringhe da cercare nell'array di stringhe; quando viene inserita la stringa "STOP" il programma termina.

5.14. ANCORA ARRAY DI STRINGHE "ESATTE" (presenz3.c)

Ripetere l'esercizio precedente, ma stavolta bisogna usare una funzione **`duplicato`**, da chiamare per assegnare direttamente gli elementi dell'array di stringhe che stiamo leggendo: la funzione duplicato riceve una stringa, ne alloca un duplicato (allocando solo il numero di caratteri necessario) e restituisce tale duplicato (come risultato della funzione).

5.15. ARRAY DI STRINGHE USATO PARZIALMENTE (strmenu.C)

Scrivere un programma che soddisfi i seguenti requisiti:

- il programma deve gestire un array di stringhe nel quale possono essere memorizzate da 0 a **`N`** stringhe (es. **`N`**=40);
- il programma mostra un menù di scelte possibili (aggiunta di una stringa nell'array, sostituzione di una stringa presente nell'array con un'altra
- , stampa delle stringhe contenute nell'array) e chiede all'utente di scegliere tra le varie opzioni;
- in base alla scelta, il programma attiva l'opportuna sua funzione;
- il programma usa le funzioni
    - **`aggiunta`** (che, ricevendo un array di stringhe, il numero di stringhe presenti nell'array e una stringa, aggiunge la stringa nella prima posizione libera dell'array e aggiorna il numero di stringhe presenti);
    - **`ricerca`** (che, ricevendo un array di stringhe e una stringa, restituisce l'indice dell'elemento dell'array in cui c'è la stringa (oppure –1));
    - **`sostituzione`** (che, ricevendo un array di stringhe e due stringhe, s1, s2, cerca la prima occorrenza di s1 nell'array e la sostituisce con s2);
    - **`stampa`** (che, ricevendo un array di stringhe, stampa tali stringhe sul monitor).
    - per gestire l'array di stringhe, oltre all'array vero e proprio di puntatori a carattere (come **`char * stringhe[N]`**) sarà necessario usare una variabile supplementare intera **`numeroStringhe`**, che contenga il numero di stringhe che sono state effettivamente aggiunte nell'array in un dato momento dell'esecuzione del programma.

La funzione **`aggiungi`** riceve l'indirizzo della variabile **`numeroStringhe`**, in modo da poter usare il suo valore e poterlo modificare. questa funzione deve usare quel valore per memorizzare nel primo elemento libero dell'array la nuova stringa; inoltre deve poterlo modificare perché' dopo l'aggiunta il numero complessivo di stringhe nell'array è cresciuto di 1.

Le altre funzioni dovranno ricevere anche **`numeroStringhe`** come argomento, in modo da poter limitare la scansione dell'array alle sole componenti effettivamente occupate.

5.16. CALCOLI SU UN ARRAY ESATTO

Scrivere un programma che legge da input un valore n e poi n valori di tipo double, memorizzandoli in un array di esattamente n componenti. Su tale array vanno calcolati (per poi visualizzarli sullo schermo, il valore medio del contenuto, il valore massimo e il valore minimo.

5.17. CALCOLI SU UN ARRAY ESATTO - 2

Verificare che l'esercizio precedente poteva essere risolto senza usare l'array (calcolando i valori richiesti durante le operazioni di input).

5.18. CALCOLI SU UN ARRAY ESATTO - 3

(Se non già fatto nell'esercizio 1.6), scriver una seconda versione di quel programma, in cui si calcolano tutti i valori richiesti con l'esecuzione di un unico ciclo di scansione dell'array dinamico impiegato.

5.19. CALCOLI SU UN ARRAY ESATTO - 3

(Se non già fatto nell'esercizio 1.6 o 1.7), scrivere una seconda versione di quel programma, in cui si calcolano tutti i valori richiesti mediante l'uso di funzioni apposite per il calcolo della media, del minimo e del massimo.