

## TERZA Esercitazione Autoguidata

Attivare il TC e posizionarsi sul proprio dischetto. Ciò può esser fatto usando il comando **Change Dir** del menù **File**, oppure attivando il comando **DOS shell** (stesso menù), spostandosi su A: e inserendo EXIT per tornare al TC.

Le soluzioni proposte per gli esercizi (indicate spesso tra parentesi nei titoli delle varie sezioni) sono reperibili via web, tramite il **sito del corso** (zona "esercitazioni autoguidate")

### 3.1. MYST

Analizzare il seguente diagramma di flusso e interpretarlo (cioè capire che cosa fa l'algoritmo da esso rappresentato). (NB le prime due scatole sono "di input"; quella prima dello stop è "di output").

Accanto al diagramma è presentato un programma (parzialmente scritto) che lo codifica in C. Completare il programma, in modo che rappresenti effettivamente una codifica dell'algoritmo descritto dal diagramma di flusso. Scritto il programma in un file, provarlo per una serie di (coppie di) input. Ad es. per {5, 3}, {3, 4}, {10,0}, {0,6} (La soluzione è in MYST.C)

<pre>#include&lt;stdio.h&gt;  int main() {     int n, m, p;      scanf("%d", &amp;m);     scanf("...", &amp;n);      for (...; n&gt;0; n...)         p=p+m;      printf("... .. e' %d\n", p);     printf("FINE\n");     return 0; }</pre>	<pre>graph TD     Start([start]) --&gt; M[/M/]     M --&gt; N[/N/]     N --&gt; P0[P = 0]     P0 --&gt; Ngt0{N &gt; 0}     Ngt0 -- SI --&gt; Pplus[P = P + M]     Pplus --&gt; Nminus[N = N - 1]     Nminus --&gt; Ngt0     Ngt0 -- NO --&gt; P[/P/]     P --&gt; Stop([stop])</pre>
---	--

Adesso bisogna provare a realizzare un programma analogo, ma **usando il costrutto while**. In questa seconda versione, fare in modo che il programma stampi un output come nell'esempio sotto (non viene stampata solo p, ma anche n e m). Una soluzione è in MYST2.C.

```
fornire il primo numero: 5
fornire il secondo numero: 12
il ... 5 ... 12 ... 60
FINE
```

### 3.2. INTERRUZIONE FORZATA DELL'ESECUZIONE

Se un errore in un programma porta ad un ciclo che non termina, è necessario interrompere manualmente l'esecuzione. Ad esempio, il precedente programma non termina se si omette la modifica di  $n$  ( $n=n-1$  o  $n--$ ).

In generale, per interrompere l'esecuzione si devono premere i tasti **<ctrl>-Pausa** (insieme) e poi invio. In questo caso particolare (in cui il programma si trova fermo su istruzioni di input) può essere necessario ripetere <ctrl>-Pausa seguito da invio. Provare a togliere la modifica di  $n$  ed eseguire il programma. Il ciclo (se  $n$  è diverso da 0, non termina mai! E quindi il programma non termina e non ci restituisce mai il controllo. Per bloccare l'esecuzione fare come detto sopra. Poi rimettere l'espressione di modifica di  $n$  ed eseguire ancora il programma).

### 3.3. ALTRI ESERCIZI DI BASE (CICLO1.C ... CICLO4.C)

**Scrivere un programma** che legge una sequenza di dieci numeri interi, stampando per ognuno il numero successivo. Programma visto a lezione (CICLO1.C, CICLO2.C)

**Scrivere un programma** che legge numeri interi, stampando per ognuno il successivo. Quando viene fornito in input il valore 50, il programma termina. Programma visto a lezione (CICLO3.C, CICLO4.C)

### 3.4. MINIME, MEDIE E MASSIME (MINIMO.C, MEDIA.C, MAX999.C)

**Scrivere un programma** che legge una sequenza di 8 numeri e stampa il minimo. Ricordare la tecnica del massimo parziale ... Soluzione proposta in MINIMO.C.

**Scrivere un programma** che legge una sequenza di 5 numeri e ne calcola e stampa la media. Ricordare la tecnica di accumulazione. Si sommano i numeri e si divide... (MEDIA.C).

**Scrivere un programma** che legge una sequenza di numeri, interrompendosi quando viene letto 999, per stampare il massimo dei valori letti (escluso 999). Soluzione proposta in MAX999.C.

### 3.5. A1 e A2 (A1.C, A2.C)

Scrivere un programma che stampa i numeri da 1 a 15, uno per riga, sul video. Il consiglio è di usare un contatore ...

**Avete usato il for/while?**

**Adesso allora rifate il programma con un while/for.**

Soluzioni proposte in A1.C e A2.C

### 3.6. MCD (MCD.C)

Scrivere un programma che riceve due interi e calcola e stampa il relativo massimo comun divisore.

Algoritmo: 1) leggere  $n$ ,  $m$   
2) mentre  $n$  diverso da  $m$   
    2.1) se  $m > n$  allora diminuisci  $m$  di  $n$  ( $m=m-n$ ),  
        altrimenti diminuisci  $n$  di  $m$   
3) ora in  $n$  (ed  $m$ !) c'è il MCD tra  $n$  e  $m$ .

Provare ad ottenere il seguente output (immaginando di aver dato 36 e 14 come input:

il massimo comun divisore tra 36 e 14 e' 4

### 3.7. PESI (PESI.C)

Scrivere un programma che riceve in input un "peso" in chilogrammi e stampa sul video la corrispondente "categoria". Le categorie dei pesi sono individuate come segue:

categoria	Intervallo di peso
A	peso<=50.0
B	50.0 < peso <= 125.0
C	125.0 < peso <= 200.0
D	peso > 200.0

Se il valore in input non corrisponde a nessuno di quelli previsti bisogna scrivere sul video che il valore non è ammissibile (eseguendo quella che si dice una "segnalazione di errore").

Se serve, ecco un suggerimento sull' algoritmo:

- 1) leggere da input un peso
- 2) se il dato è inammissibile  
scrivere segnalazione e terminare  
altrimenti calcolare e stampare la categoria

La soluzione è in PESI.C.

Il programma può essere generalizzato in modo da permettere che durante l'esecuzione vengano inseriti diversi pesi e per ciascuno venga stampata la categoria.

Si può pensare di iterare le seguenti azioni "mentre" l'utente (cioè chi chiede l'esecuzione del programma) manifesta l'intenzione di inserire nuovi pesi.

```
mentre l'utente vuole inserire un nuovo peso
- lettura di un peso
- stampa della categoria (o segnalazione di errore).
```

Già ora si può provare a scrivere il programma corrispondente.

Ma se serve qualche suggerimento, si può leggere la seguente discussione.

Per far "manifestare l'intenzione di inserire nuovi dati" bisogna evidentemente chiedere all'utente input aggiuntivi: ad es. scrivere 1 per inserire nuovi pesi oppure 0 per smettere di inserire pesi.

L'algoritmo che potrebbe risultare è il seguente:

```
1) leggere un peso
2) stampa categoria o errore
3) stampa di una richiesta
   ('si vogliono inserire altri pesi?')
4) lettura della risposta in una
   variabile scelta
ripetere 1-4 mentre scelta è uguale a 1
```

L'algoritmo, nel suo stato attuale, va raffinato:

**cosa c'è in scelta quando il test scelta==1 viene eseguito la prima volta?**

Ci vuole una inizializzazione (da aggiungere in cima all'algoritmo scritto sopra come "passo 0)")

**0) scelta := 1**

Scrivere il programma che codifica l'algoritmo 0-4 e permette all'utente di inserire diversi pesi, specificando ogni volta se vuole o no inserire un altro peso. Provare il programma, anche passo-passo, tenendo sotto controllo il valore delle variabili usate. (PESI2.C).

Questo tipo di problemi si risolve molto naturalmente con l'uso di un costrutto `do-while`:  
eliminare il passo 0 e  
ripetere 1-4 **fino al verificarsi di** SCELTA<>'S' ...)  
Scrivere il programma corrispondente. La soluzione è in PESI3.C

### **3.8. DEBUGGING di un programma (SOMPPSNO.C, SOMPS.C)**

Considerare il programma contenuto nel file SOMPSNO.C. Lo scopo del programma è (sarebbe ...),

- dato n in input,
- leggere una sequenza di n numeri interi, negativi e positivi,
- calcolando
  - la somma di tutti i valori positivi forniti in input (sommaPos),
  - la somma di tutti i valori negativi forniti in input (sommaNeg)
  - e la somma complessiva (somma).
- i valori di sommaPos, sommaNeg e somma vengono poi stampati.

Provare qualche esecuzione del programma. Il programma non funziona bene. Può essere compilato con successo, ma in realtà, quando viene eseguito, non produce i risulta attesi, perché è infarcito di errori logici, cioè di errori nella costruzione logica dell'algoritmo stesso.

Eseguire "passo passo" il programma, per verificare i valori contenuti nelle variabili durante l'esecuzione e ipotizzare delle correzioni. Salvare il programma in un nuovo file e correggerlo.

La versione corretta è in SOMPS.C.

### **3.9. MEDIA (MED999WH.C)**

Scrivere un programma che legge una sequenza di numeri terminata da 999. 999 non fa parte della sequenza di numeri su cui calcolare la media. Ricordare la tecnica di accumulazione. Inoltre stavolta bisogna mantenere anche l'informazione su quanti sono i numeri letti in input. Si fa la somma di tutti i numeri, contemporaneamente li si conta e alla fine si divide la somma per la quantità dei numeri della sequenza.

(MED999WH.C, con while e MED999FO.C, con for).

### **3.10. Pressione in una bottiglia (PRESS.C)**

La pressione  $p(T)$  in una bottiglia di coca cola e' espressa, in atmosfere, dall'equazione:

$$p(T) = 0.00105 * T^2 + 0.0042 * T + 1.325$$

dove T è la temperatura dell'ambiente in cui si trova la bottiglia

Quando la pressione nella bottiglia raggiunge o supera il valore 3.2 atm. la bottiglia esplode.

Scrivere un programma che descriva le variazioni di pressione nella bottiglia al variare della temperatura. Assumere che inizialmente la temperatura sia  $T=26$  gradi e aumentare  $T$  di 1 grado alla volta, fino a quando la bottiglia esplose.

### Suggerimento sull' algoritmo:

- inizialmente  $T=26$  e  $PRESSIONE = p(T)$
- mentre  $PRESSIONE <$  valore critico di 3.2 atmosfere
  - facciamo crescere la temperatura di 1 grado
  - modifichiamo conseguentemente il valore di  $PRESSIONE$

NB: usciamo dal ciclo quando la condizione  $PT < 3.2$  NON è più vera, cioè quando la bottiglia è esplosa. Non resta che comunicare l'evento con un opportuno messaggio.

La soluzione proposta è in `PRESS.C`.

### 3.11. Tabella Celsius/Fahreneit/Kelvin (TABELLA2.C)

Sapendo che, dato un valore in gradi Celsius,  $C$ , le corrispondenti temperature Fahreneit e Kelvin sono  $F=C*9/5+32$  e  $K= C+273.16$ , scrivere un programma che legge da input tre valori  $t_i$ ,  $ps$  e limite e stampa la tabella contenente i valori Celsius da  $t_i$  fino a non oltre limite, calcolati con passo  $ps$ , insieme con incorrispondenti valori nelle altre due scale.

```
fornire temperatura d'inizio, passo e limite: 13 .8 23
-----+-----+-----
celsius   | fahreneit | kelvin
13.000    | 55.400    | 286.160
13.800    | 56.840    | 286.960
14.600    | 58.280    | 287.760
15.400    | 59.720    | 288.560
...
21.800    | 71.240    | 294.960
22.600    | 72.680    | 295.760
-----+-----+-----
FINE
```

### 3.12. Indiani e olandesi a Manna Hatta (INDIANI.C, INDIANI2.C)

Nel 1628 il *Direttore generale dei commerci con la nuova Olanda*, Minnewit, acquistò l'isola di Manna Hatta dagli indiani pellerossa Rackagawawanc. Gli indiani erano all'epoca insediati nella zona di Brooklin e quindi poco interessati all'isola, per cui accettarono di cederla agli olandesi per sessanta *guilders*, pari a circa \$24.00. L'isola adesso si chiama Manhattan (a New York).

Se gli indiani avessero depositato in banca i 24 dollari ricavati dalla vendita, e la banca garantisse un interesse annuo del 3%, quanto ci sarebbe adesso sul conto?

Scrivere un programma che, dato in input l'anno attuale, calcola e stampa lo stato attuale del conto. Usare delle costanti simboliche per l'anno della vendita, il prezzo di vendita e l'interesse. Per calcolare il deposito attuale, iterare un'operazione di rivalutazione (al 3%) del deposito, per ogni anno dal 1628 all'anno attuale (dato in input).

Poi si potrebbe modificare il programma in modo che venga stampata in output la sequenza di tutti i valori (anno per anno) assunti dal conto degli indiani affaristi. Però la sequenza di stampe sarebbe oggettivamente troppo lunga. Si può provare a stampare la quantità di denaro presente sul conto per ogni anno, ma ci si accorge subito che solo le ultime stampe (una ventina) rimangono visibili sullo schermo e le altre precedenti sono perdute e non possono venir esaminate. Farlo ora! E salvare il programma in un nuovo file.

Un modo per ovviare a questo problema di “scorrimento veloce di output” (e di troppo output), consiste nel selezionare meglio gli output prodotti dal programma.

Ad esempio si potrebbe modificare il programma (e salvarlo in un nuovo file) in modo che la tabella di rivalutazione del capitale sia mostrata stampando la consistenza del deposito una volta ogni 30 anni. In questo modo il numero di stampe diminuisce e possiamo apprezzare meglio in una schermata video l'evoluzione di questo conto corrente.

Per realizzare quest'ultima soluzione, usare un'altra costante simbolica per l'intervallo 30.

### 3.13. Calcolo della radice di un numero con il metodo di Newton

Dato un numero reale  $a$ , calcolarne la radice quadrata ( $\sqrt{a}$ ) corrisponde a calcolare la radice (soluzione) della funzione  $f(x) = x^2 - a$ . Il metodo di Newton permette di farlo, calcolando approssimazioni sempre più precise della soluzione, a partire da una approssimazione iniziale  $X_0$ .

Data una approssimazione  $X_n$ , si calcola l'approssimazione successiva  $X_{n+1}$  come

$$X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)}$$

che, nel caso della nostra  $f(x)$  diventa

$$X_{n+1} = \frac{X_n + a/X_n}{2}$$

**Scrivere un programma** che, ricevuti in input

Un numero reale	a
Una approssimazione iniziale	start
Un coefficiente di approssimazione	eps

calcoli un'approssimazione  $X$  della radice di  $a$  tale che  $|X^2 - a| < eps$ .

Il valore assoluto  $|Y - a|$  è calcolabile con **fabs(Y-a)** (fabs() è una funzione di math.h)

### Una possibile soluzione (RADICEW.C, RADICEF.C)

L'algoritmo da usare in questo caso, itera la produzione (il calcolo) di una nuova approssimazione a partire da quella disponibile attualmente, fino a che non si presenta il caso in cui l'approssimazione disponibile è soddisfacente.

- Leggere i valori EPS, START e A
- $X\_ATTUALE \leftarrow START$  (l'approssimazione attualmente disponibile e' questa).
- Mentre  $|X\_ATTUALE^2 - A| > EPSILON$  (bisogna calcolare nuove approssimazioni)
  - $X\_NUOVA \leftarrow$  calcolo della successiva approssimazione
  - $X\_ATTUALE \leftarrow X\_NUOVA$  (ora e' questa l'approssimazione migliore disponibile)

**NB:** Quando usciamo dal ciclo,  $X\_ATTUALE$  contiene un'approssimazione del valore cercato. Questa approssimazione ha permesso di uscire dal ciclo, quindi NON realizza la condizione  $|X\_ATTUALE^2 - A| > EPSILON$  (se l'avesse realizzata, staremmo ancora dentro al ciclo, iterando ...)

- Stampa dell'approssimazione ottenuta.

## Una seconda opportunità (RADICEW2 . C)

Nella soluzione precedente abbiamo stabilito di interrompere il calcolo di nuove approssimazioni quando  $|X\_ATTUALE^2 - A| \leq \text{EPSILON}$ .

Una condizione alternativa si può basare sul calcolo *della distanza tra le ultime due approssimazioni calcolate*, chiamiamole APPR1 e APPR2. Quando le approssimazioni sono ad una distanza almeno  $\text{eps}$  (essendo il procedimento convergente) qualunque ulteriore approssimazione sarà migliore di quanto abbiamo richiesto. In altre parole, il programma ha raggiunto lo scopo quando  $|\text{APPR1} - \text{APPR2}| \leq \text{EPSILON}$ .

### Suggerimento sull'algoritmo:

- a) chiamiamo `ultimAppr` l'ultima approssimazione calcolata;
- b) a partire da questa calcoliamo (mentre serve) una nuova approssimazione, `nuovAppr`. Controlliamo se la distanza tra `ultimAppr` e `nuovAppr` è soddisfacente. Se lo è abbiamo finito; se non lo è, spostiamo `nuovAppr` in `ultimAppr` (perché ora è lei l'ultima approssimazione calcolata e iteriamo b).

Usiamo una variabile intera `soddisfacente`, come *flag*: inizialmente vale 0 e continua a valere zero mentre le approssimazioni calcolate non sono soddisfacenti. Non appena l'approssimazione `nuovAppr` calcolata è soddisfacente, il flag diventa 1 e possiamo smettere di calcolare. Quindi il ciclo che itera il calcolo di nuove approssimazione ha come condizione che sia (`soddisfacente==0`). Quando `soddisfacente` è 1, si esce dal ciclo e `nuovAppr` contiene l'approssimazione richiesta.