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.
    d. NB le immagini in queste dispense sono prese dalla versione 4.9.9.2 del DEV. La versione 5.11 è probabilmente la più recente e può differire solo in qualche dettaglio.

Esercitazione 2

2.1. Pari e dispari (dispari.c)

Scrivere un programma che, ricevuto un numero intero, stampi un commento in cui si dice se il numero è pari o dispari.

2.2. Massimo one Vs. one (max2.c)

Scrivere un programma che, ricevuti due numeri intero, riveli chi è il massimo.
Provare il programma su diversi dati di test. Per ogni test, giustificarne le motivazioni (ad esempio come commento all'interno del codice).
A proposito: hai provato con diversi dati di test il programma del punto 2.1?

2.3. Segno di un numero intero

Scrivere un programma che, ricevuto un numero intero, stampi un commento in cui si dice se il numero è positivo, negativo o nullo.

Suggerimento: si può usare il programma in `negativo.c`, visto a lezione, modificandolo opportunamente.

2.4. Tempo

Scrivere un programma che,
ricevuta una coppia di interi,
il primo rappresentante il mese e il secondo l'anno di una data,
stampi un commento indicante qual è il mese successivo.

Ad esempio, se in input c'è
`11 2026`
viene stampato **12/2026** … ma se c'è
`12 2027`
… viene stampato **1/2028.**

Nella directory dei programmi per questa EG ci sono due file .c relativi a questo esercizio: uno è quello con cui confrontate la vostra soluzione.
L'altro è una sfida … non va bene … presenta qualche difetto … provate a correggerlo ;)
(ci sono dei commenti dentro che possono aiutare)

Suggerimento in fondo …

Suggerimento: un'ipotesi di algoritmo

0) servono mese ed anno … almeno questi sono sicuri
1) input mese e anno
2) se i dati sono illegittimi
    2.1) stampare un messaggio (e poi il programma finisce)
    altrimenti
    2.2) se il mese è 12,
        2.2.1) anno vale … e mese vale …
        ALTRIMENTI
        2.2.2) mese vale … (ma anno non cambia)
3) ora possiamo stampare mese/anno
4) FINE PROGRAMMA

2.5. Ricerca del massimo. (max3.c, max3-alg3.c, max3-maxparz.c

Scrivere un programma che,
ricevuti in input tre numeri interi,
assegni ad una variabile max il massimo tra i tre, per poi stampare questa informazione.

Provare a ripercorrere le soluzioni diverse viste a lezione.

2.6. Massimo tra 4 (max4.c, max4-lazy.c)

Scrivere un programma C che
- dati quattro numeri interi,
- rivela il più grande, usando la tecnica del massimo parziale.

Farlo! Poi si torna qui.

Suggerimento, in fondo alla pagina.

SUGGERIMENTO
a) inizializzazione **`maxParz`**
…
d) se **`num4`** ...
e) ora in **`maxParz`** c'è il valore massimo tra quelli dati in input e si stampa

Bisogna provare il programma con diversi dati di prova.

Queste prove si chiamano “test” e i “dati di test” vanno pensati bene, per evitare di testare il programma sempre sullo stesso tipo di istanza.

Bisogna invece cercare di fare test che rappresentino diverse istanze (distribuzioni di dati di input) del problema, in modo i test non siano ridondanti.

Solo cosi`, quando abbiamo fatto i test che avevamo progettato, ~~abbiamo la certezza che il programma sia corretto~~ abbiamo buone probabilità che il programma non sia terribilmente scorretto.

Per ogni test eseguito,
giustificare, su carta, la motivazione del test,
`/* o scriverla nello stesso file del programma, alla fine del file in una zona commentata */`

Nello scrivere una giustificazione, immaginate di far leggere queste motivazioni a un collega … saranno queste giustificazioni abbastanza chiare e conseguenti?

POI scrivere il programma analogo al precedente, in cui però i dati di input sono tutti negativi, per definizione del problema.
Farlo: poi si torna qui.

Prima l'algoritmo e poi il programma, ok?

POI torniamo qui e parliamo di inizializzazione: quando farla effettivamente e quando ce la possiamo risparmiare.

A) valutare il valore di verità delle seguenti affermazioni:

a. *"Tutte le variabili devono essere inizializzate da noi, coscientemente, dato che non ci fidiamo che nella memoria assegnata a quelle variabili ci siano soltanto zeri"*.

b. "*Visto che ho una sequenza di numeri interi, potrei sempre inizializzare* `maxParz` *con 0 (cioè assegnare a* `maxParz` *il suo primo valore nel programma, prima di tutto) e poi cercarmi con calma il massimo*"

c. "*Visto che il primo uso che faccio di* `maxParz` *è la sua assegnazione ad un valore (il primo della sequenza), è inutile inizializzare* `maxParz` *(il suo primo valore sarà con l'assegnazione al primo numero della sequenza …)*"

Rispondere e poi guardare dopo.

Risposte

a. *“Tutte le variabili devono essere inizializzate da noi, coscientemente, dato che non ci fidiamo che nella memoria assegnata a quelle variabili ci siano soltanto zeri”.*
Most certainly TRUE
Ma bisogna comunque rifletterci …

b. “*Visto che ho una sequenza di numeri interi, potrei sempre inizializzare* **`maxParz`** *con 0 e poi cercarmi con calma il massimo*”
FALSE!
Cioè magari va bene, se diamo al programma una sequenza di numeri tutti positivi … ma se ci fosse qualche numero negativo nella sequenza?

c. “*Visto che il primo uso che faccio di* **`maxParz`** *è la sua assegnazione ad un valore (il primo della sequenza), è inutile inizializzare* **`maxParz`** *prima (il suo primo valore significativo arriverà con l’assegnazione al primo numero della sequenza …)*”
TRUE
Very clever, master

2.7. Minimalismi

Ripetere 2.2, 2.5, 2.6 … ma per trovare il minimo (invece del massimo).

Potete … dovete … riusare i programmi che avete già scritti … modificandoli quel tanto che basta.

2.8. Che triangolo è?

Scrivere un programma che,

- richiesto all'utente l'inserimento di 3 interi `a`, `b` e `c`,
- legge i tre interi
- e stampa un messaggio che dice all'utente se il triangolo con lati `a`, `b` e `c` è equilatero, isoscele o scaleno.

Provare a scrivere l'algoritmo; poi guardare più sotto un primo suggerimento.

SUGGERIMENTO

Un primo suggerimento è che questo programma è piuttosto buono per esercitarsi sulle espressioni logiche …
Quali proprietà dei tre segmenti (gli interi) esprimono il fatto che essi formano un triangolo equilatero? Scaleno? Isoscele?
Provare a scrivere le espressioni logiche corrispondenti e poi usarle nel programma.

Ad esempio (a,b,c tutti uguali) è TRUE se il triangolo è equilatero …
Le altre sono più' difficili ;)

Seguono altri suggerimenti

Suggerimento2
"tre lati uguali" → equilatero
"solo due lati uguali" → isoscele
"tre lati diversi" → scaleno
Come si possono esprimere queste proprietà della triade di lati (con quali espressioni logiche)?
Ad esempio,

la prima proprietà è espressa da `(a e b sono uguali) AND (a e c sono uguali)`

la seconda … ?
"due lati sono uguali" si puo' scrivere come "`(a==b) OR (a==c) OR (b==c)`"
Ma questa espressione non cattura la proprietà per cui
"*solo due lati sono uguali, e l'altro è diverso*".

Suggerimento3
Nel programma stampare le seguenti espressioni logiche insieme con il tipo di triangolo che esse denotano se sono vere. Poi stampare l'output del programma.

```
(a==b) && (b==c)  ..... equilatero
(a!=b) && (a!=c) && (c!=b)  ..... ?
((a!=b) && (b==c)) || ((a!=b) && (a==c)) || ((c!=b) && (a==b))
(a==b) && (b!=c)
(b==c) && (b!=a)
(a==c) && (c!=b)
```

Qualcuna di queste espressioni andrà combinata per ottenere qualcosa … Inoltre nel programma con la soluzione suggerita per questo esercizio, in certi casi c'è una stampa strana. Perché'?

2.9. Sì, ma è un triangolo?

Tre numeri possono essere le misure dei lati di un triangolo, a patto che ciascuno sia positivo e minore della somma degli altri due.
Scrivere un programma che,

- chiesto all'utente di inserire 3 interi a, b e c,
- li legge
- e stampa un messaggio che dice all'utente se i tre numeri possono rappresentare oppure no un triangolo.

2.10. Minimi, medie, massimi. Oppure no: solo medie

Scrivere un programma che calcola la media di 5 numeri interi. Stampare sia la media intera che quella con parte frazionaria.

2.11. A volte ritornano, e mordono

Scrivere un programma in cui

- viene chiesta e ricevuta una data,
  - la data è rappresentata come una sequenza di tre interi,
    - giorno **`g`**,
    - mese **`m`**,
    - anno **`a`**,
- e viene stampato un commento che dice al caro utente qual è il giorno successivo

(ad esempio se il caro utente introduce 31, 8 e 2023, il messaggio stampato deve essere del tipo "1/9/2023").

2.12. Ambiguif, ambiguif2, ambiguif3

Analizzare il comportamento del programma in `ambiguif.c`
Cioe` … farlo eseguire almeno tre volte su dati di prova significativi.

Quando decidi un dato di prova da usare (un numero in questo caso) scrivi una motivazione per usare proprio quel dato.
La motivazione falla vedere ad un tuo collega, e discutila con lui: *è chiara? Ha senso? Il test in pratica ne ripete uno già fatto?*

POI
correggere il programma in `ambiguif.c`, in modo che, ricevuto un intero n
1) venga stampato "ma è grande!" se n è maggiore di 100,
2) venga stampato "ma è negativo!" se n è negativo,
e comunque venga stampato "bella giornata!"

(Un suggerimento più avanti)

(mi raccomando salva il lavoro in un file diverso da ambiguif.c …)

POI
Come sopra, ma cambiando un po' il codice, in modo che il secondo if sia messo nella parte ELSE del primo if, così evitiamo di avere ancora istruzioni if annidate nella parte TRUE di altre istruzioni if. (Discusso nella lezione 6)

.

SUGGERIMENTO

Le graffe, Luke, usa le graffe.
In modo da forzare l'interpretazione dell'else come parte del primo if e non del secondo.

2.13. Pesi

Scrivere un programma che
- riceva in input un "peso" in chilogrammi
- e stampi 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”).

Tre suggerimenti seguono … a debita distanza

SUGGERIMENTO 1
Usare delle costanti per i limiti delle categorie. Non è detto che sia proprio indispensabile: può dipendere da come è scritto il codice. MA potrebbe essere utile nel caso in cui si voglia scrivere un programma simile ma con categorie definite diversamente: basterebbe cambiare i pesi limite in una piccola parte del programma, invece di cambiare qui e là in tutto il programma.

SUGGERIMENTO 2
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
```

SUGGERIMENTO 3

0) un peso viene dato in input (variabile peso);
   al peso viene data una categoria, tra A B C e D (assumiamo di calcolare questa categoria e poi alla fine stampare un output che dice qual'è la categoria).
   Allora ci serve una variabile "categoria".
   I valori limite (50, 125 ...) sono buoni candidati ad essere costanti.
   Si può immaginare che questi valori, in un programma grande, sarebbero usati molte volte - non come succede qui ... quindi definirli come costanti può essere meglio.
   Perciò abbiamo anche LIMITEA, LIMITEB ... etc. per le quattro categorie.

1) chiedere e leggere peso

2) se il peso è minore uguale LIMITEA allora assegniamo 'A' a `categoria`

   altrimenti ...
   se peso <= LIMITEB, … vuol dire che peso è compreso tra LIMITE A e LIMITEB
   (NB siamo nella parte "else" di "peso minore uguale LIMITEA,
   perciò peso NON è minore uguale LIMITEA …)
   e quindi 'B' è il valore da assegnare a categoria;

   altrimenti ...
   se peso <= LIMITEC, ...
   (NB siamo nella parte "else" di "peso <= LIMITEB,
   perciò peso NON è minore uguale LIMITEB … cioè è maggiore di LIMITEB)
   … finire il ragionamento ed assegnare categoria ...
   altrimenti ...
   se peso <= LIMITED
   il peso e' maggiore di LIMITEC (senno' non saremmo qui …),
   allora la categoria è 'D'

3) fine

La soluzione è in `PESI.C`.

Troppo facile? Dopo il prossimo esercizio c'è un'anticipazione della prossima EG

2.14. Lascia o raddoppia?

Ripetere 2.5, 2.6 … ma per trovare i primi due valori massimi nella sequenza: ad esempio, i due valori massimi in (34, 18, 19, 16) sono 34 (massimo) e 19 (secondo più grande).

In questo caso assumere che i valori sono tutti positivi.

Provare a scrivere l'algoritmo prima di guardare i suggerimenti che seguono (tre, progressivi)

SUGGERIMENTO 1

- Si usano max1 e max2 (primo e secondo, rispettivamente)
- Inizializzazione di max1 e max2 con due dei valori
- Poi si controllano gli altri valori, uno per uno
    - Se max1<num
        Max1 cambia

        ALTRIMENTI (cioè quando max1 era ancora il più grande)
        Se max2<num,
        max2 cambia

SUGGERIMENTO 2

- Si usano max1 e max2 (primo e secondo, rispettivamente)
- Inizializzazione di max1 e max2 con due dei valori, esempio n1 ed n2
- Poi si controllano gli altri valori, uno per uno … n3, … n4 …
    - Se max1<num
        - Max1 cambia
        - ? deve cambiare altro?

      ALTRIMENTI (cioè quando max1 era ancora il più grande)
      Se max2<num,
      max2 cambia (solo lui)

SUGGERIMENTO 3

- Si usano max1 e max2 (primo e secondo, rispettivamente)
- Inizializzazione di max1 e max2 con due dei valori, esempio n1 ed n2
- Poi si controllano gli altri valori, uno per uno … n3, … n4 …
    - Se max1<num
        - Max2 cambia: max2=max1
        - Max1 = num

      prima bisogna assegnare max2 con l'attuae contenuto di max1 – era il primo, adesso e' il secondo);
      poi si puo' assegnare num a max1.
      Se facciamo queste assegnazioni nell'ordine inverso perdiamo quello che era il massimo parziale e non possiamo assegnarlo al secondo parziale.

      ALTRIMENTI (cioe' quando max1 era anocra il piu' grande)
      Se max2<num,
      max2 cambia (solo lui)

E, per i/le più impazienti …

2.15. (Anticipazione della prossima EG) Pesi

Il programma `pesi.c` 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 (cioe' chi chiede l'esecuzione del programma)
> manifesta l'intenzione di inserire nuovi pesi,
> *ricevere un nuovo peso e calcolarne e stamparne la categoria*.

Scritto meglio …

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 addizionali: 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:

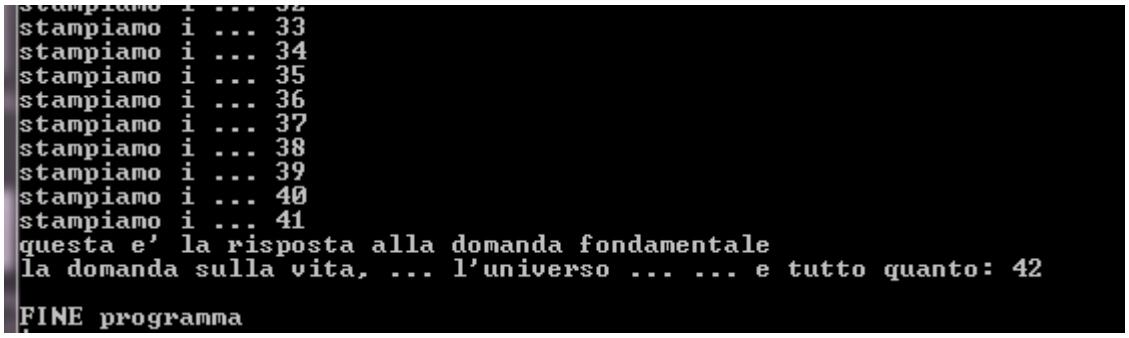
cosa c'è in `scelta` quando il test `scelta==1` viene eseguito la prima volta?

Ci vuole una inizializzazione (da aggiungere in cima all'algoritmo scritto sopra

```
scelta = 1
```

2.16. (Anticipazione della prossima EG) 42, 100

Sulla scorta di quanto visto a lezione, scrivere un programma, che stampa i primi 42 numeri interi, producendo il seguente output



Poi realizzare un programma che stampa i primi cento numeri, facendo meno modifiche possibile al precedente …

2.17. (Anticipazione della prossima EG) Successivo, 10 volte e per un numero indeterminato di volte

Scrivere un programma che legge una sequenza di dieci numeri interi, stampando per ognuno il numero successivo.

Poi scrivere il medesimo programma, in cui però

- Viene segnalato il numero d'ordine del numero che si sta leggendo (numero d'ordine nella sequenza di valori letti
- Viene usata una costante per rappresentare QUANTI numeri devono essere letti (in questo testo abbiamo detto 10, ma in altri casi il numero può essere diverso e l'uso della costante può aiutare (evitando, nel caso si debba cambiare, correzioni estese nel corpo del programma,

Infine, scrivere un programma che legge numeri interi, stampando per ognuno il successivo. Quando viene fornito in input il valore 50, il programma termina, senza stampare 51.

Provare a realizzare il programma mediante l'uso di una istruzione di ripetizione **`while`**.

2.18. (Anticipazione della prossima EG) Dispari

Il quadrato di un numero naturale **n** puo' essere ottenuto facendo la somma dei primi **n** numeri naturali dispari.

Scrivere un programma che legge un numero intero positivo e ne calcola e stampa il quadrato secondo l'algoritmo enunciato sopra.

Un suggerimento: forse ci sarà

2.19. (Anticipazione della prossima EG) MCD, mcd2.c

Scrivere un programma che realizza l'algoritmo visto a lezione per il calcolo del massimo comun divisore: vengono letti da input due numeri interi positivi, e viene stampato il loro MCD. Provare ad ottenere il seguente output (immaginando di aver dato 36 e 14 come input:

il massimo comun divisore tra 36 e 16 e' 4

POI
Provare a scrivere un programma per il calcolo del MCD, che usi il seguente algoritmo:

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

Questo algoritmo si chiama Algoritmo di Euclide, basato sulle sottrazioni successive. Tra due esercizi ne viene programmata una variante che esegue di solito meno iterazioni, basata sulle divisioni successive.

2.20. Per sempre Dispari

Ripetere l'esercizio precedente, ma in modo che il programma chieda all'utente se vuole inserire un altro numero, di cui calcolare il quadrato.
Sì, e poi stampi anche il quadrato di quel numero, certo.

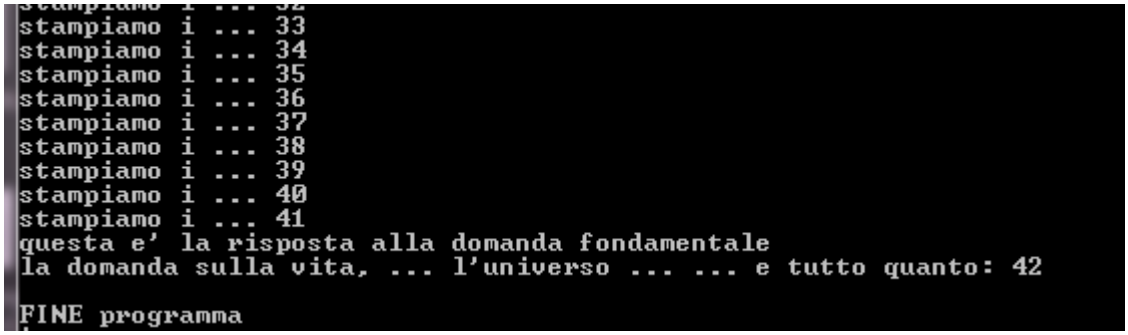
La risposta dell'utente potrebbe essere un intero (1/0), oppure un carattere (S/N).

Usare una variabile flag `ancora` per rappresentare il valore di verita' dell'evento "l'utente vuole inserire un altro dato". Quando questo valore di verita' e' 0 vuol dire che dobbiamo smettere di calcolare e andare verso la fine del programma.

Anticipazione dell'Esercitazione 3

3.1. 42, 100

Sulla scorta di quanto visto a lezione, scrivere un programma, che stampa i primi 42 numeri interi, producendo il seguente output



Poi realizzare un programma che stampa i primi cento numeri, facendo meno modifiche possibile al precedente …

3.2. Successivo, 10 volte e per un numero indeterminato di volte

Scrivere un programma che legge una sequenza di dieci numeri interi, stampando per ognuno il numero successivo.

Poi …

Poi scrivere il medesimo programma, in cui pero'

- Viene segnalato il numero d'ordine del numero che si sta leggendo (numero d'ordine nella sequenza di valori letti
- Viene usata una costante per rappresentare QUANTI numeri devono essere letti (in questo testo abbiamo detto 10, ma in altri casi il numero puo' essere diverso e l'uso della costante puo' aiutare (evitando, nel caso si debba cambiare, correzioni estese nel corpo del programma,

Infine, scrivere un programma che legge numeri interi, stampando per ognuno il successivo. Quando viene fornito in input il valore 50, il programma termina, senza stampare 51. Provare a realizzare il programma con una istruzione di ripetizione **`while`**.

3.3. Dispari

Il quadrato di un numero naturale **`n`** può essere ottenuto facendo la somma dei primi **`n`** numeri naturali dispari.

Scrivere un programma che legge un numero intero positivo e ne calcola e stampa il quadrato secondo l'algoritmo enunciato sopra.

Un suggerimento: usare una variabile **`sum`** per *accumulare* i valori degli **`n`** numeri dispari. Accumulare, in questo caso significa sommare in sum i vari numeri dispari che si succedono da **`1`** in poi, **`n`** volte. Al termine di questa "accumulazione" la *variabile accumulatore* contiene il quadrato ricercato.

3.4. MCD, mcd2.c

Scrivere un programma che realizza l'algoritmo visto a lezione per il calcolo del massimo comun divisore: vengono letti da input due numeri interi positivi, e viene stampato il loro MCD. Provare ad ottenere il seguente output (immaginando di aver dato 36 e 14 come input:

il massimo comun divisore tra 36 e 16 è 4

POI

Provare a scrivere un programma per il calcolo del MCD, che usi il seguente algoritmo:

Algoritmo:

```
1) leggere n, m
4) mentre n diverso da m
    2.1) se m>n allora diminuisci m di n,
         altrimenti diminuisci n di m
5) ora in n (ed m!)c'e' il MCD tra n e m.
```

Questo algoritmo si chiama **Algoritmo di Euclide**, basato sulle sottrazioni successive. Tra due esercizi ne viene programmata una variante che esegue di solito meno iterazioni, basata sulle divisioni successive.

3.5. Per sempre Dispari

Ripetere l'esercizio precedente, ma in modo che il programma chieda all'utente se vuole inserire un altro numero, di cui calcolare il quadrato.
Sì, e poi stampi anche il quadrato di quel numero, certo.

La risposta dell'utente potrebbe essere un intero (1/0), oppure un carattere (S/N).

Usare una variabile flag `ancora` per rappresentare il valore di verita' dell'evento "l'utente vuole inserire un altro dato". Quando questo valore di verità è 0 vuol dire che dobbiamo smettere di calcolare e andare verso la fine del programma.

3.6. La base dell'economia: lo scambio

Scrivere un programma che legge due numeri interi, in due variabili **`n`** ed **`m`**, stampa le variabili così come le ha lette, scambia tra loro i contenuti delle variabili medesime e poi le stampa di nuovo, per verificare che lo scambio sia effettivamente avvenuto.
È più lungo a dirsi che a farsi … Se serve un suggerimento è in fondo alla pagina.

SUGGERIMENTO

Per scambiare due variabili serve una terza variabile, chiamiamola aux. Mettiamo in **aux** il contenuto di **n**, cosi' rimane disponibile per dopo. Poi mettiamo in **n** il contenuto di **m**.
E poi mettiamo in **m** il contenuto che prima era di **n** (ma ora non più … per fortuna ora quel valore è conservato in **aux**).

3.7. MCD, mcd3.c

Scrivere un programma che realizzi il seguente algoritmo per il calcolo del MCD (algoritmo di Euclide per divisioni successive).

1) Usiamo quattro variabili: **`n`**, **`m`** per i numeri dei quali calcolare il MCD; **`r`** per conservare il resto di divisioni; **`mcd`** (non indispensabile: contiene il MCD calcolato, in sostanza solo per stamparlo).
2) Leggi **`n m`**
3) Ora fare in modo che in **`n`** ci sia il piu' grande dei due: se occorre scambiare **`n`** con **`m`**;
4) **`r =`** resto della divisione tra **`n`** ed **`m`**;
5) ripetere, mentre **`r != 0`**
    a. mettere in **`n`** il contenuto di **`m`**
    b. mettere in **`m`** in valore di **`r`**
    c. **`r =`** resto della divisione tra **`n`** ed **`m`**;
6) assegna ad **`mcd`** il valore di **`m`**
7) stampa **`mcd`**

3.8. MINIME, …(MINIMO.C, …)

Scrivere un programma che legge una sequenza di 8 numeri e stampa il minimo. Ricordare la tecnica del massimo parziale ... solo che ora va applicata per cercare il minimo, e quindi si usa una variabile minParz, che, durante la scansione della sequenza, contiene il minimo valore "visto finora".

3.9. MCD, consapevolezza

In due esercizi precedenti sono state implementate due diverse soluzioni (Euclide) al calcolo del MCD. Ora è il momento di confrontare le due soluzioni.

Confrontare poi il comportamento dei due algoritmi che abbiamo messo sotto il nome di Euclide.

Quante iterazioni vengono eseguite se si calcola il MCD tra due numeri molto diversi? (cioè uno molto più grande dell'altro)?

Per eseguire questa verifica, vanno modificati un po' i programmi:

- si può mettere una stampa in ogni iterazione del ciclo di calcolo, in modo che ad ogni iterazione si possa vedere come evolvono alcune variabili significative (come ris);
- un altro modo consiste nell'aggiungere variabili ed istruzioni al programma (ai programmi), in modo che venga conteggiato il numero di iterazioni eseguite, con stampe che mostrino in output l'evoluzione di questo numero ("iterazione 1, iterazione 2, …). Questa soluzione può essere unita alla precedente, facendo sì che l'utente che assiste all'esecuzione del programma abbia parecchie informazioni interessanti;
- Una modifica (o un arricchimento) della scelta precedente potrebbe essere quello di stampare dei dati consuntivi, come

```
... "per fare questo calcolo abbiamo
    eseguito %d sottrazioni successive …"
```

oppure

```
... "per fare questo calcolo abbiamo
eseguito %d divisioni successive …"
```