Laurea In Ingegneria dell'Informazione

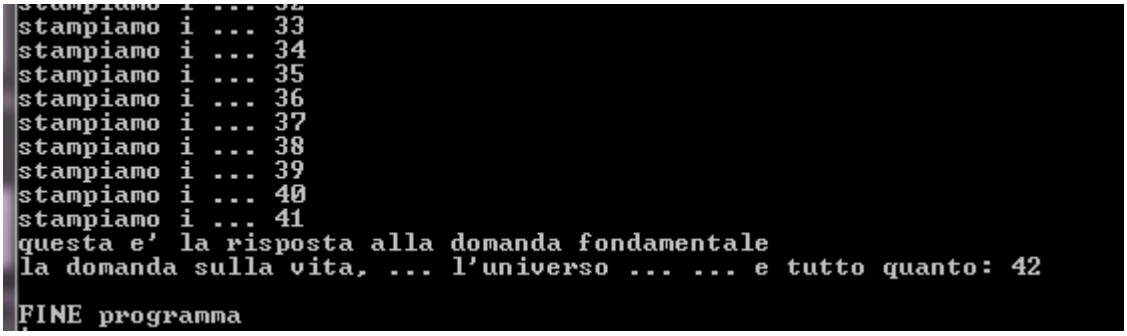
Esercitazioni Guidate di Tecniche della Programmazione

Note introduttive: Seguire le raccomandazioni date nelle precedenti EG …

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** 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: 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 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.

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.
Si', 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.

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 cosi' 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.
E' piu' lungo a dirsi che a farsi … Se serve un suggerimento e' 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 piu' … per fortuna ora quel valore e' 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? (cioe' uno molto piu' grande dell'altro)?

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

- si puo' 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 puo' essere unita alla precedente, facendo si' 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 …"
```

3.10. … , MEDIE E MASSIME

Scrivere un programma che legge una sequenza di 5 numeri e ne calcola e stampa la media. Ricordare la tecnica di accumulazione vista poco fa: si sommano i numeri e si divide ... stando attenti al fatto che la media tipicamente deve essere un numero float o double.

3.11. … , MEDIE E MASSIME (…, MAX999.C)

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

3.12. A1 e A2

Scrivere un programma che stampa i numeri da 1 a 15, uno per riga, sul video.

Il consiglio è di usare un contatore … dopo leggete il resto …

…Avete usato il for? Allora adesso rifate il programma con un while.
…Avete usato il while? Allora adesso rifate il programma con un for.

Tanto per essere chiari, alla fine dovreste aver fatto l'esercizio usando una volta il for e una volta il while:

…due volte in tutto …

Soluzioni proposte in `A1.C` e `A2.C`

3.13. 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`)

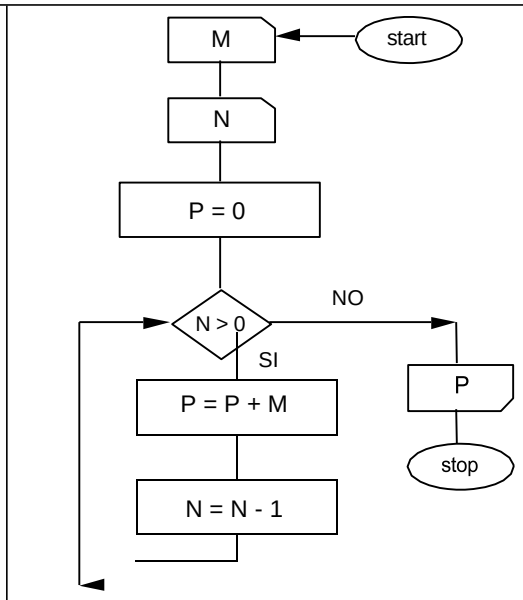
```
#include<stdio.h>

int main() {
  int n, m, p;

  scanf("%d", &m);
  scanf("...", ...);

  for (...; n>0; n...)
    p=p+m;

  printf("... ... ... e' %d\n", p);
printf("FINE\n");
return 0;
}
```



Dopo (cioè, insomma, nella prossima pagina) facciamo una versione diversa …

Si`, ora è dopo …

Ora bisogna provare a realizzare un programma analogo a quello precedente, 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
```

3.14. 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	0<=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`.

Troppo facile?

Vabbè … 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 (cioe' 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.
Prima pero' bisogna scrivere l'algoritmo … (poi il codice C).

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 e' il seguente:

```
0)…
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
```
(se viene inserito 0 (cioe' se scelta ora contiene 0) vuol dire che non si vuole proseguire; se scelta viene assegnato ad 1, vuol dire che si vuole ripetere l'inserimento di un peso e tutto il resto …

```
ripetere 1-4 mentre scelta è uguale a 1
```

Scrivendo diversamente l'algoritmo si potrebbe ottenere

```
0) … variabile per peso, scelta …
1) Mentre (scelta==1)
   1.1) leggere un peso
   1.2) stampare categoria o notifica errore
   1.3) chiedere se si vuole continuare (1) o finire
   1.4) leggere la risposta in scelta
2) Messaggio di terminazione del programma
```

- L'algoritmo, nel suo stato attuale, va raffinato:
  chiediamoci … "cosa c'e' in `scelta` quando il test `scelta==1` viene eseguito la prima volta?"

- Yes, precisely, ci vuole una inizializzazione (da aggiungere in cima all'algoritmo scritto sopra come nuovo "passo 1)")

  `1) scelta = 1`

(i passi precedentemente numerati come 1 e 2 diventano, rispettivamente, 2 e 3 …)

Questo tipo di problemi si risolve molto naturalmente con l'uso di un costrutto `do-while`: eliminare il passo con l'inizializzazione si `scelta` (non serve piu' perche' il primo uso che facciamo di `scelta` e' in una lettura da input) e
ripetere il body dell'istruzione iterativa mentre `scelta!='S'` …

Se questo esercizio è stato già fatto … leggi dopo

Se questo esercizio è stato già fatto, allora scrivere il programma che codifica l'algoritmo e permette all'utente di inserire diversi pesi, specificando ogni volta se vuole o no inserire un altro peso, mediante una scelta in cui si scrive S per si' ed N per no.

Usare un carattere può dare qualche problema, come ad esempio scelte fatte scrivendo il carattere che sembra non vengano lette durante l'esecuzione.
Perché succede? … è un po' lunga da spiegare qui perché succede, anche se quando lo si è capito è una cosa molto semplice … e poi spiegarlo qui toglierebbe tutto il divertimento. Se non ne venite a capo ne parliamo in laboratorio (qualunque forma di laboratorio avremo).

3.15. 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).

3.16. DEBUGGING di un programma (somPSNO.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 risultati 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.

3.17. 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 esplode.

Suggerimento sull'algoritmo segue … rifletteteci un po' prima di guardarlo …

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.

3.18. Tabella Celsius/Fahreneit/Kelvin (tabella2.c)

Sapendo che, dato un valore in gradi Celsius, C, ile 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 ti, ps e limite e stampa la tabella contenente i valori Celsius da ti fino a non oltre limite, calcolati con passo ps, insieme con in corrispondenti 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
```

3.19. 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 Brooklyn 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 (in base all'interesse) del deposito, per ogni anno dal 1628 all'anno attuale (dato in input).

Poi …

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 al problema discusso sopra, 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.

Segue suggerimento

SUGGERIMENTO

Per realizzare quest’ultima soluzione, usare un’altra costante simbolica per l’intervallo 30. E contare il numero di iterazioni mentre le si effettua (ad esempio con una variabile conta). Quando conta e’ uguale all’intervallo stabilito, si stampa, … e solo in questo caso si stampa.

3.20. 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 equazione $f(x) = x^2 - a$ 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 è nella prossima pagina

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 ← START (l'approssimazione attualmente disponibile e' questa).
- Mentre $|X\_ATTUALE^2 - A| > EPSILON$ (bisogna calcolare nuove approssimazioni)
  - X_NUOVA ← calcolo della successiva approssimazione
  - X_ATTUALE ← 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_ATTUALE2 – A| >= EPSILON (se l'avesse realizzata, staremmo ancora dentro al ciclo, iteranndo …)
- Stampa dell'approssimazione ottenuta.

Una seconda possibilità è nella prossima pagina

Una seconda opportunità (`RADICEW2.C`)

Nella soluzione precedente abbiamo stabilito di interrompere il calcolo di nuove approssimazioni quando |X_ATTUALE^2 – A| <= 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 `eps` (essendo il procedimento convergente) qualunque ulteriore approssimazione sarà migliore di quanto abbiamo richiesto. In altre parole, il programma ha raggiunto lo scopo quando |APPR1 – APPR2| <= 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.