Tecniche della Programmazione, lez.7

Continuiamo sulle

- Istruzioni ripetitive, istruzioni iterative, istruzioni di ciclo…

Istruzione ripetitiva: while (1/5)

Cosa fa?

```
#include <stdio.h>

int main () {
  int n=4;

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

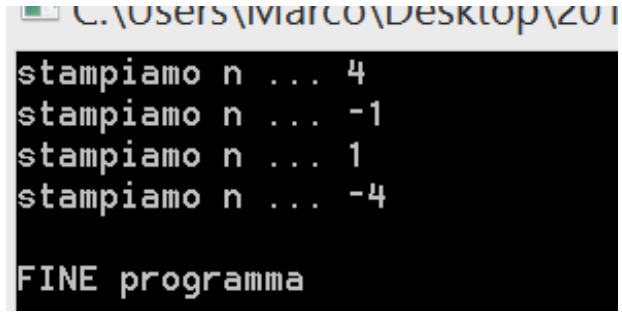
cosa stampa?

Istruzione iterativa: while (2/5)

Cosa fa?

Bisogna fare un'esecuzione passo-passo:
Falla e Vedi Approfondimenti prima di continuare

```
#include <stdio.h>

int main () {
  int n=4;

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

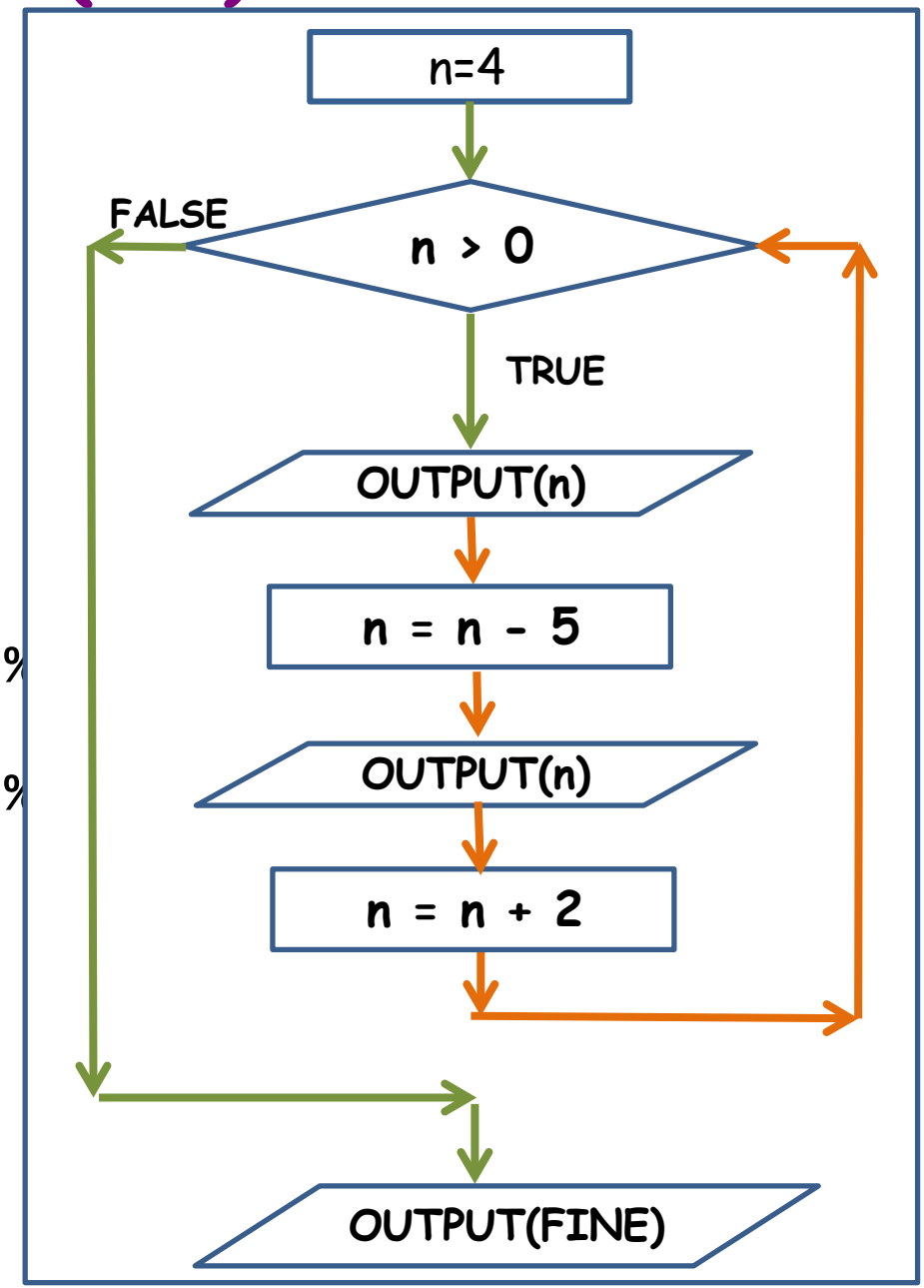


Algoritmo? Fai il Diagramma di Flusso, e l'algoritmo per passi

Istruzione ripetitiva: while (3/5)

diagramma a blocchi corrispondente

```
#include <stdio.h>

int main () {
  int n;

  n=4;
  while (n>0) {
     printf ("stampiamo n ... %
     n = n-5;
     printf ("stampiamo n ... %
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```



Istruzione iterativa: while (4/5)

Algoritmo corrispondente

```
#include <stdio.h>

int main () {
  int n;

  n=4;

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

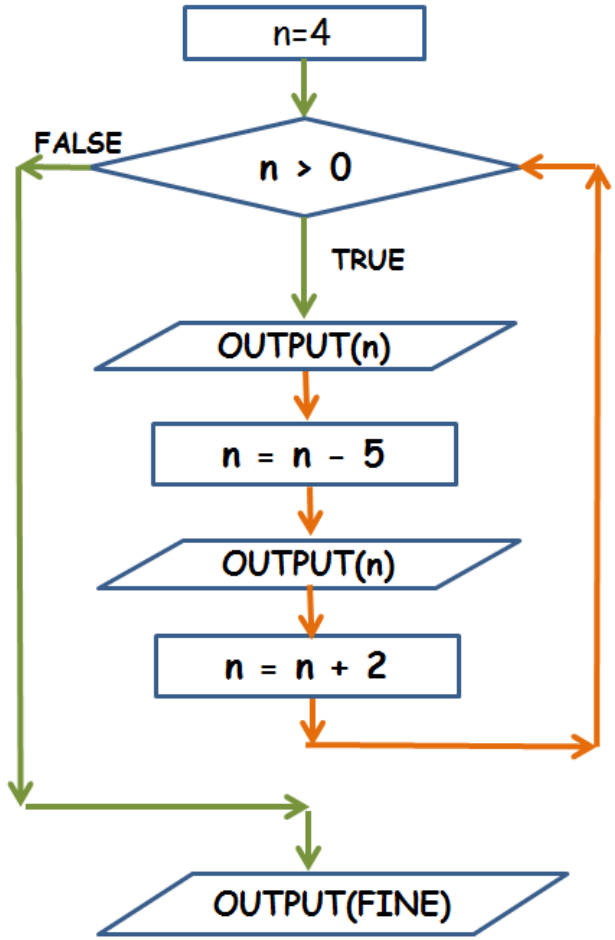
2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbè, abbiamo finito (stampa FINE PROGRAMMA e poi end)

Istruzione iterativa: COMPONENTI FONDAMENTALI

Quel che non deve mai mancare, per non creare mostri

```
n=4;
  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
```



- condizione

- body

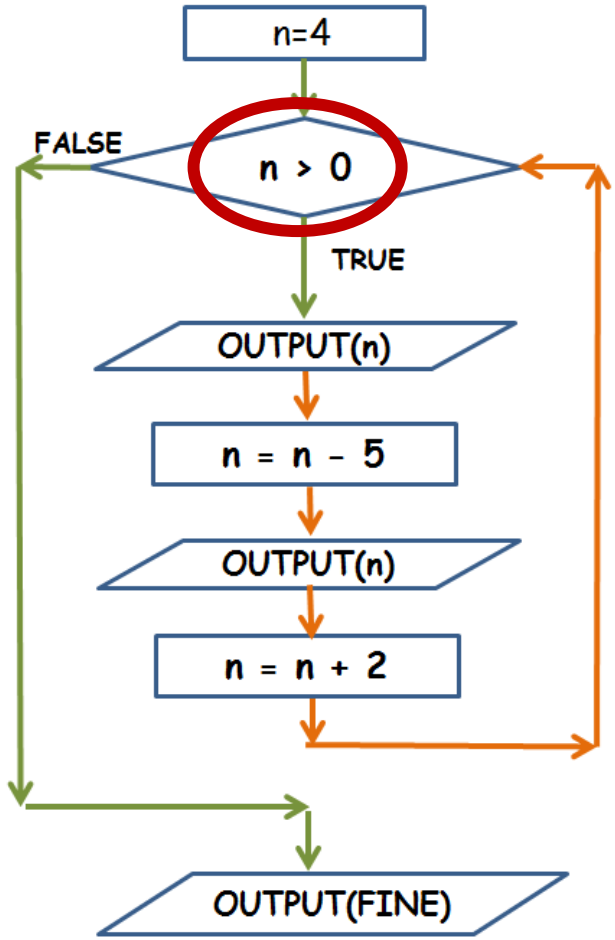
terminazione del ciclo

- inizializzazione

Istruzione ripetitiva: COMPONENTI FONDAMENTALI

Quel che non deve mai mancare, per non creare mostri

```
n=4
  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
```



- **condizione** (qui "di ripetizione")
  è un'espressione, fatta, di solito, usando una o più "variabili di test"

- body

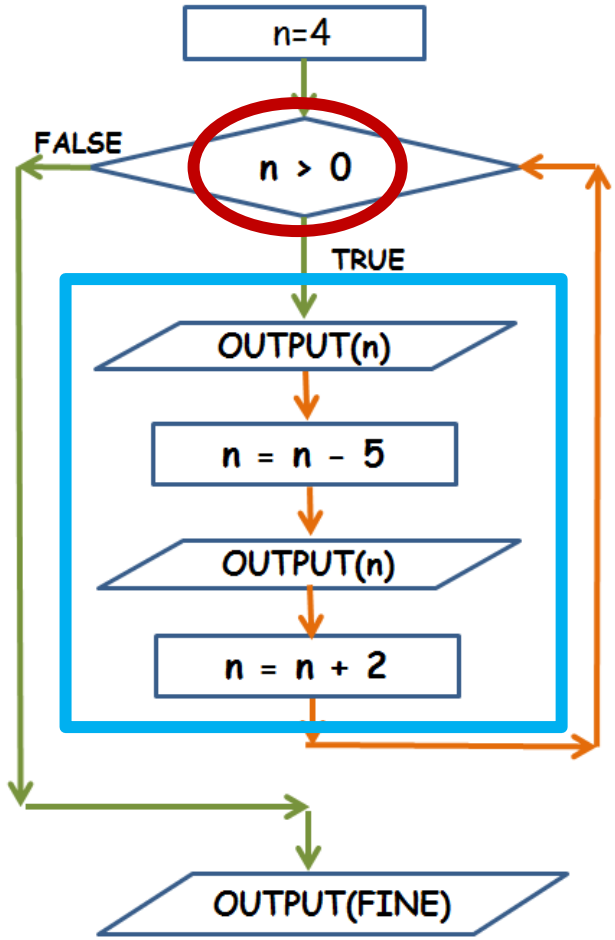
terminazione del ciclo

- inizializzazione

Istruzione iterativa: COMPONENTI FONDAMENTALI

Quel che non deve mai mancare, per non creare mostri

```
n=4
  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
```



- **condizione** (tipicamente "di ripetizione")
  è un'espressione, fatta, di solito, usando una o piu' "variabili di test"

- **body** di istruzioni da iterare

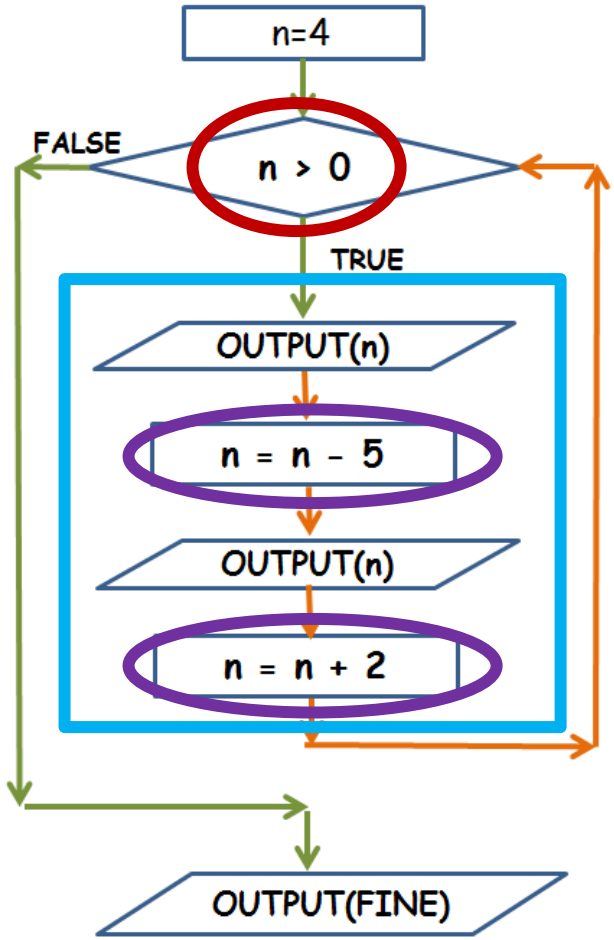
  terminazione del ciclo

- inizializzazione

Istruzione ripetitiva: COMPONENTI FONDAMENTALI

Quel che non deve mai mancare, per non creare mostri

```
n=4
  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
```



- **condizione** (tipicamente "di ripetizione") e' un'espressione, fatta, di solito, usando una o piu' "variabili di test"

- body di istruzioni da iterare

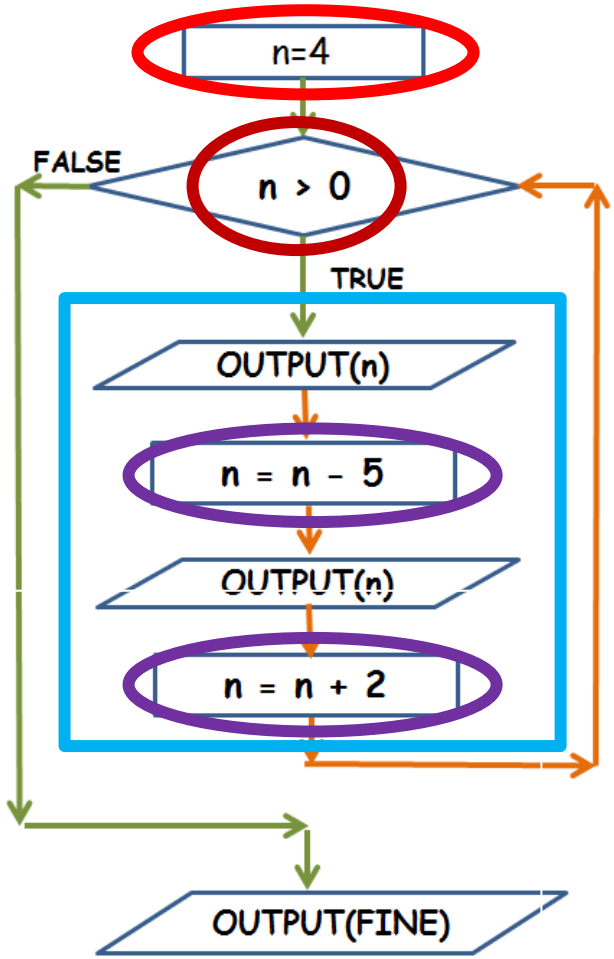
- almeno una **istruzione**, nel body, che modifichi variabili di test (in modo che la condizione possa cambiare). Inoltre questi cambiamenti devono permettere di *convergere* verso la **terminazione del ciclo**

- inizializzazione

Istruzione iterativa: COMPONENTI FONDAMENTALI

Quel che non deve mai mancare, per non creare mostri

```
n=4
  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
```



- **condizione** (tipicamente "di ripetizione") e' un'espressione, fatta, di solito, usando una o piu' "variabili di test"

- body di istruzioni da iterare

- almeno una **istruzione**, nel body, che modifichi variabili di test (in modo che la condizione possa cambiare). Inoltre questi cambiamenti devono permettere di *convergere* verso la **terminazione del ciclo**

- inizializzazione delle variabili di test

Istruzione ripetitiva: while

Ciclo solo parzialmente visibile:

supponendo che termini, cosa stampa la **printf()**?

```
#include <stdio.h>

int main () {
  int n=21;

  while (n!=7) {
     n=n*12;
     pri                    o n ... %d\n", n);
     r

     n =
  }
printf ("stampiamo n ... %d\n", n);
printf ("\nFINE programma\n");
return 0;
}
```

Vedi Approfondimenti prima di continuare

Esercizio

Programma che stampa i primi 42 numeri positivi
(usando una costante per il 42)

```
stampiamo i ... 37
stampiamo i ... 38
stampiamo i ... 39
stampiamo i ... 40
stampiamo i ... 41
stampiamo i ... 42

FINE programma
```

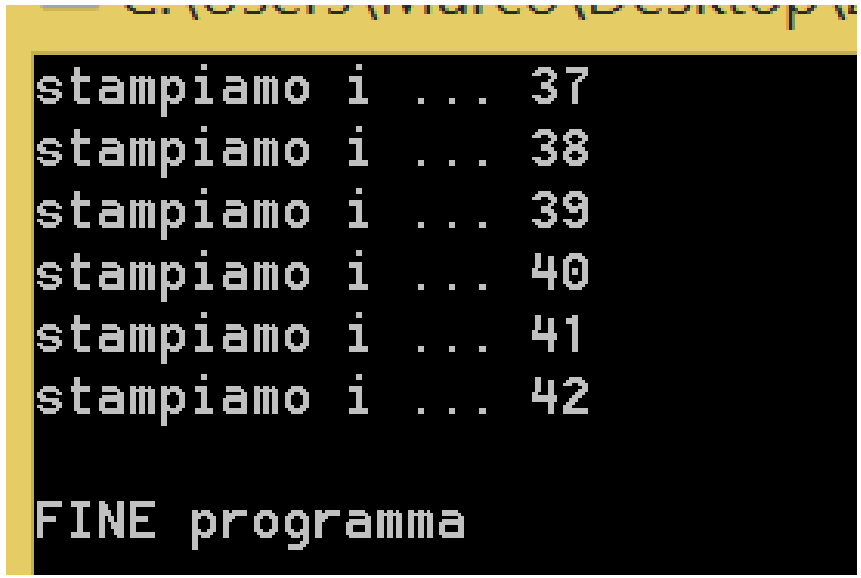
```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;
```

☺ **Algoritmo?**

```
printf ("\nFINE programma\n");
return 0;
}
```

Esercizio

Programma che stampa i primi 42 numeri positivi
(usando una costante per il 42)



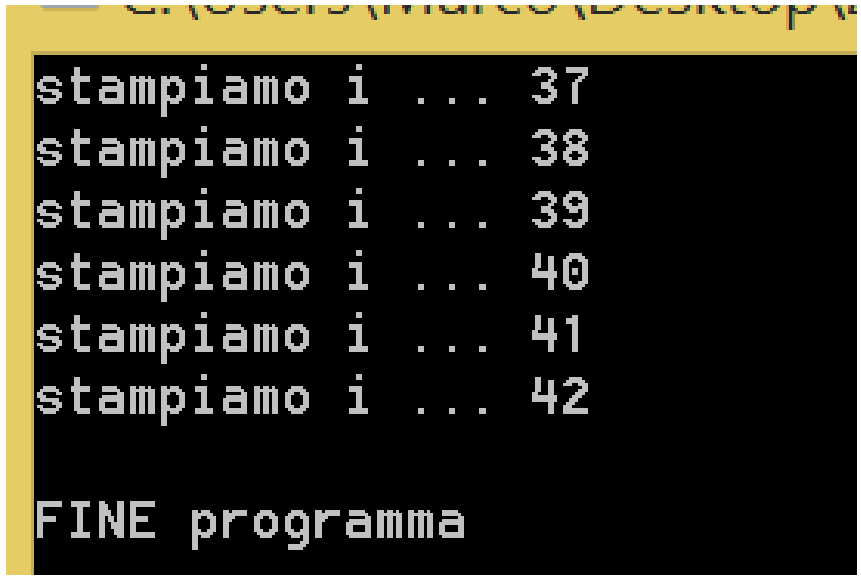

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;
```

1)
2) mentre i<42
   2.1) stampa i ☹
3) FINE programma

```
printf ("\nFINE programma\n");
return 0;
}
```

Esercizio

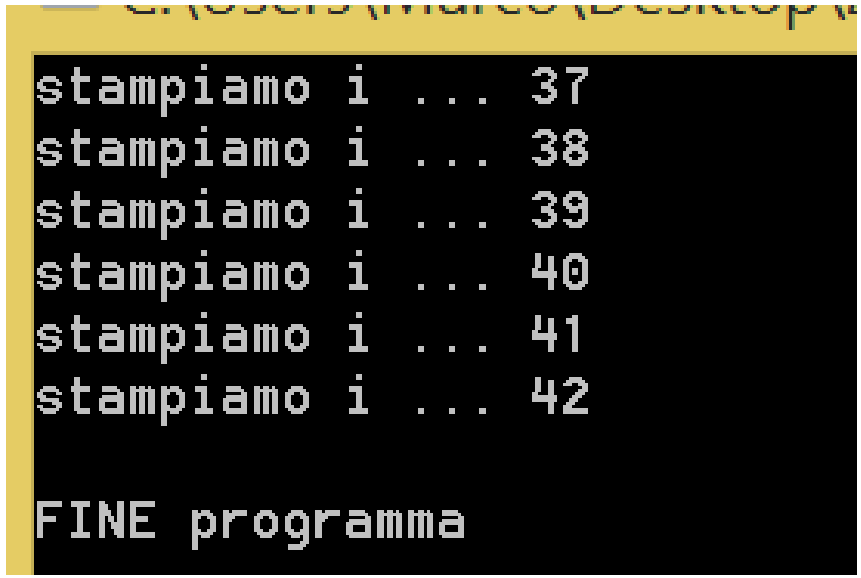
Programma che stampa i primi 42 numeri positivi
(usando una costante per il 42)




```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

1) inizializzazione di i: i=1;
2) mentre i< …
      2.1) stampa i
      2.2) modifica di i, verso 42 … i=i+1;
3) FINE

printf ("\nFINE programma\n");
return 0;
}
```

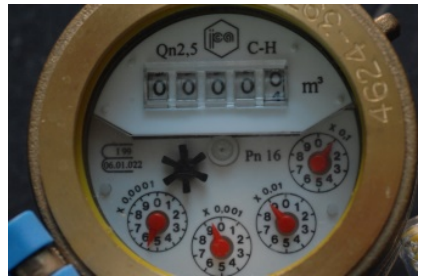
Esercizio

Programma che stampa i primi 42 numeri positivi
(usando una costante per il 42)




```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  i=1;                        /* init */
  while (i<=QUANTINUM) {
     printf ("stampiamo i ... %d\n", i);
     i++;                     /* modifica var di test */
  }
printf ("\nFINE programma\n");
return 0;
}
```

Variabile contatore



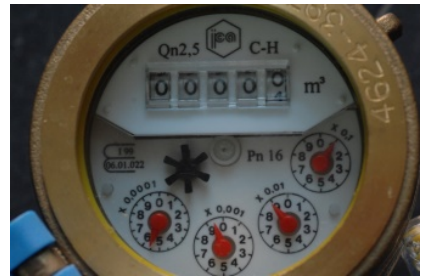
nel programma precedente, i è una variabile *contatore*,
cioè una variabile che tiene traccia, **in generale**
del numero di volte in cui si verifica un certo evento durante le iterazioni.

Una variabile contatore serve, in sostanza a contare qualcosa …

ad esempio quante volte abbiamo iterato (ripetuto) il body di un ciclo …
in questo caso il test serve a decidere
"se abbiamo o non abbiamo eseguito un numero
giusto di volte l'iterazione del body del ciclo".

MA NON SOLO QUESTO

Variabile contatore



nel programma precedente, i è una variabile *contatore*,
cioè una variabile che tiene traccia, **in generale**
del numero di volte in cui si verifica un certo evento durante le iterazioni.

Esempi di uso di una variabile contatore

- quante volte abbiamo ripetuto il body di un ciclo

- quante volte, in una sequenza di numeri in input, un numero verifica una certa proprietà
  (***positivo, negativo, divisibile per 3, uguale a 61***)

- quanti caratteri ci sono in una parola data in input

- quanti caratteri di una parola sono uguali a 'q'

(Avremo un'altra *variabile_che_decide_sul_valore_della_condizione*, più tardi …)

Contatore del numero di iterazioni:

Problema del Controllo del segno di un numero ... serve tra poco

Preliminari:
Come si fa a risolvere il problema di
leggere un numero intero e decidere cosa stampare tra "positivo!", "negativo!", e "ma e' zero!"

```
Algoritmo

0) dato … n

1)   chiedere e leggere n

2)   se n>0
            1.2.1) stampare "positivo"
     altrimenti
            1.2.2) se n<0
                        stampare "negativo"
                   altrimenti
                        stampare "ma …"

3) fine programma
```

però questo funziona per un solo numero:
se vogliamo controllare 5 numeri bisogna
eseguire il programma 5 volte (puah!)

ok. E se volessimo controllare 5 numeri?

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualità (essere negativo, positivo o zero)

Algoritmo

0) ??

1) ??
2) ??

?) fine programma

controllo di un numero dato in input … 5 volte

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualità (essere negativo, positivo o zero)

Algoritmo

0) n1, n2, n3, n4, n5 i dati
1) lettura dei 5 numeri
2) ??

?) fine programma

☹

provare a scrivere l'algoritmo completo e poi proseguire

controllo di un numero dato in input … 5 volte

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualità (essere negativo, positivo o zero)

Algoritmo

0) n1, n2, n3, n4, n5 i dati
1) lettura dei 5 numeri
2) controllo e stampa per n1
3) controllo e stampa per n2
4) …
5) …
6) controllo e stampa per n5

7 !?) fine programma

☹

provare a scrivere l'algoritmo completo e poi proseguire (e smettere quando si realizza che si sta facendo una cosa un po' troppo lunga …

controllo di un numero dato in input … 5 volte

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualita' (essere negativo, positivo o zero)

Algoritmo

0) n1, n2, n3, n4, n5 i dati
1) lettura di n1 … n5
2) se n1>0
        2.1) stampare "positivo"
   altrimenti
        2.2) se n1<0
                stampare "negativo"
             altrimenti
                stampare "ma …"

e ora duplichiamo il passo 1 altre 4 volte

3) … per n2
4) …
5) … per n4
6) … per n5

7) fine programma

controllo di un numero dato in input … 5 volte

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualità (essere negativo, positivo o zero)

```
Algoritmo
0)  n1, n2, n3, n4, n5  i dati
1)   lettura di n1 … n5
2)  se n1>0
            2.1) stampare "positivo"
      altrimenti
            2.2) se n1<0
                     stampare "negativo"
                  altrimenti
                     stampare "ma …"
3)  se n3>0
            2.1) stampare "positivo"
      altrimenti
            2.2) se n1<0
                     stampare "negativo"
                  altrimenti
                     stampare "ma …"
4)  se n3>0
            2.1) stampare "positivo"
      altrimenti
            2.2) se n1<0
                     stampare "negativo"
                  altrimenti
                     stampare "ma …"
...


7) fine programma
```

no
no
il programma non può diventare tanto più lungo quanti più sono i numeri che dobbiamo controllare.
Non va bene.
Non è serio.

I passi 1 e 2 devono diventare qualcosa come
- leggere n
- e poi controllare n e stampare pos o neg

e questa coppia di operazioni va iterata, in questo caso, 5 volte;

in generale va iterata per il numero di volte necessario …

cosi` i programma e` in pratica sempre della stessa lunghezza ed e` cosi` che si fa …

controllo di un numero dato in input … 5 volte

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualità (essere negativo, positivo o zero)

Algoritmo

0) n1, n2, n3, n4, n5 i dati
1) lettura di n1 … n5
2) se n1>0
   2.1) stampare "positivo"
   altrimenti
   2.2) se n1<0
      stampare "negativo"
      altrimenti
      stampare "ma …"

i passi 1 (lettura) e 2 (controllo e stampa) vanno iterati: una volta per il primo numero, una per il secondo … etc…

- lo scriviamo una volta sola, ma riferito ad una variabile generica n
- non usiamo le 5 variabili n1, …, n5, ma solo n
- n viene letto all'inizio del body … e quel nuovo numero viene processato … il tutto, 5 volte

7) fine programma

controllo di un numero dato in input … 5 volte

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualità (essere negativo, positivo o zero).

Il contatore tiene traccia di quante volte abbiamo letto (e processato) un numero.

```
Algoritmo

0)  n per i dati, i come contatore, costante 5
1)  init i          i=1

2)  mentre (i <= 5)
    2.1) chiedere e leggere n
    2.2) se n>0
              2.2.1) stampare "positivo"
         altrimenti
              2.2.2) se n<0
                        stampare "negativo"
                     altrimenti
                        stampare "ma …"

3) fine programma
```

(le cinque iterazioni corrispondono ad i = 1,2,3,4,5)

☺
e ora scrivere il programma

controllo di un numero dato in input … 5 volte

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualità (essere negativo, positivo o zero)

```
0)  n per i dati, i come contatore, costante 5

1)  init i          i=1

2)  mentre i <= 5
    2.1) chiedere e leggere n
    2.2) se n>0
              2.2.1) stampare "positivo"
         altrimenti
              2.2.2) se n<0
                        stampare "negativo"
                   altrimenti
                        stampare "ma ..."
```

```
#include <stdio.h>
#define QUANTI 5
int main () {
  int n, i=1;

 while(i<=QUANTI) {
   printf ("..., dammi un numero ... \n");
   scanf("%d", &n);

   if (n>0)
      printf ("positivo!\n");
   else
      if (n<0)
         printf ("negativo!\n");
      else printf ("... ma, e' zero!\n");

}
printf ("\nFINE programma\n");
return 0;
}
```

controllo di un numero dato in input … 5 volte

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualità (essere negativo, positivo o zero)

```
Algoritmo

0)  n per i dati, i come contatore, costante 5

1)  init i          i=1

2)  mentre i <= 5
    2.1) chiedere e leggere n
    2.2) se n>0
              2.2.1) stampare "positivo"
         altrimenti
              2.2.2) se n<0
                          stampare "negativo"
                     altrimenti
                          stampare "ma …"
    2.3) i = i+1;

3) fine programma
```

controllo di un numero dato in input … 5 volte

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualità (essere negativo, positivo o zero)

```
Algoritmo                                          NB

0)  n per i dati, i come contatore,
    costante QUANTI uguale a 5

1)  init i                  i=1

2)  mentre i <= QUANTI
    2.1) chiedere e leggere n
    2.2) se n>0
              2.2.1) stampare "positivo"
         altrimenti
              2.2.2) se n<0
                        stampare "negativo"
                   altrimenti
                        stampare "ma …"
    2.3) i = i+1;
3) fine programma
```

iterazioni con i = 1, 2, 3, 4, 5

che succede se l'inizializzazione e`
i=0?

Cosa deve cambiare per ottenere il medesimo risultato?

Vedi Approfondimenti

scrivere il programma corrispondente e sperimentarlo con almeno 5 casi di input

in questi casi ci sono ridondanze? (test che sono inutili perché in sostanza un test precedente era equivalente …)
Se ci sono dubbi, parliamone.

Poi risolvere i problema di controllare 11 numeri, invece di 5.
Come cambia il programma?
Poco, vero?
(Se non cambia poco … parliamone

controllo di un numero dato in input … quante volte vuoi

Algoritmo per un programma che
- chiede quante volte deve ricevere un numero e stampare la sua qualità,
- e lo fa

```
Algoritmo

0)  n per i dati, i come contatore, ???
1)  ???
2)  init i          i=0
3)  mentre  ???
    3.1) chiedere e leggere n
    3.2) se n>0
             3.2.1) stampare "positivo"
         altrimenti
             3.2.2) se n<0
                        stampare "negativo"
                    altrimenti
                        stampare "ma …"
…
```

cosa cambia rispetto alle 5 volte?

se e` l'utente che specifica quante volte bisogna leggere e controllare, "quanti" (cioe` quanti numeri vanno controllati, cioe` quante iterazioni vanno eseguite, non puo` essere una costante …

Vedi Esercizi

Controllo del numero di iterazioni in base ad un dato di input

Qui non è possibile basarsi sul conteggio delle iterazioni fatte, per decidere di uscire dal ciclo ...
E` uno dei vari dati di input che decide ...

Programma che legge una sequenza di numeri interi, interrotta da zero, dicendo per ciascuno se è positivo o negativo

```
int main () {
  int n=1;

 while (n!=0) {
   printf ("caro/a utente, dammi... (0 per finire)\n");
   scanf("%d", &n);

   if (n>0)
      printf ("... ma, ma, %d e` positivo!\n", n);
   else
      if (n<0)
         printf ("... ma, ma, %d e` negativo!\n", n);
      else printf ("... ok, torna quando vuoi!\n");
 }

printf ("\nFINE programma\n");
return 0;
}
```

Controllo del numero di iterazioni in base ad un dato di input

Programma che legge una sequenza di numeri interi, interrotta da zero, dicendo per ciascuno se e' positivo o negativo

```
int main () {
  int n=1;

 while (n!=0) {
   printf ("caro/a utente, dammi... (0 per finire)\n");
   scanf("%d", &n);

   if (n>0)
      printf ("... ma, ma, %d e' positivo!\n", n);
   else
      if (n<0)
         printf ("... ma, ma, %d e' negativo!\n", n);
      else printf ("... ok, torna quando vuoi!\n");
 }

printf ("\nFINE programma\n");
return 0;
}
```

- ☺ perché mettere questa istruzione qui?
- ☺ quali altri valori potrei assegnare ad n, ottenendo lo stesso effetto?
- ☺ quali valori non dovrei assegnare ad n per evitare di rendere il programma malfunzionante?

Vedi Approfondimenti

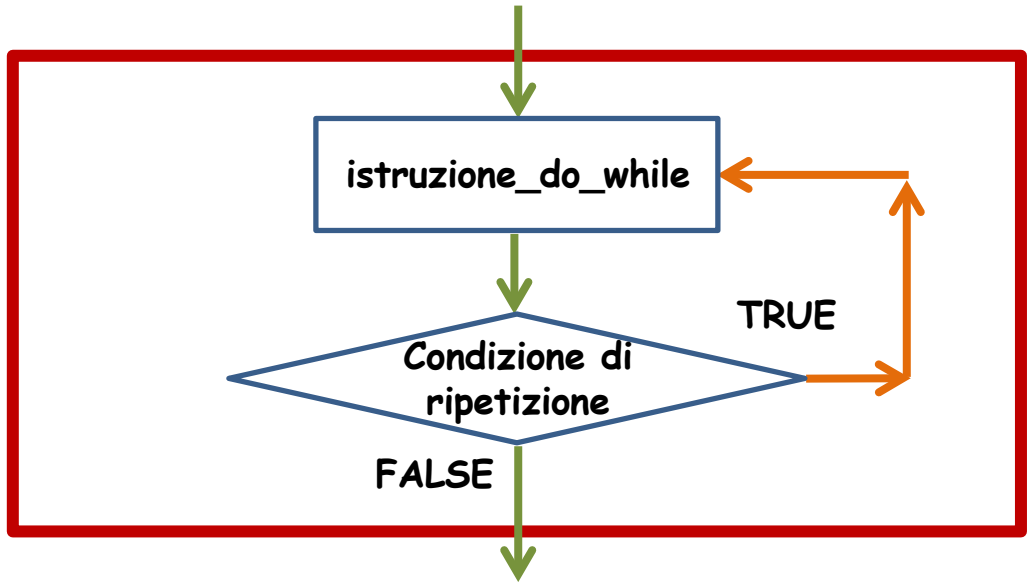
Altri tipi di ciclo in C: do_while

Oltre al "while", in C esistono due altri costrutti per ottenere un'istruzione iterativa.

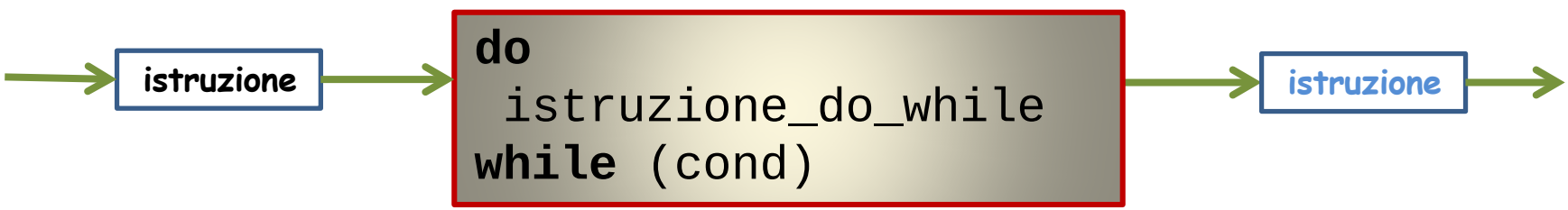
Ci sono poche differenza ma ognuno ha aspetti interessanti …

Istruzione do_while

E' un'istruzione di ripetizione; ha le medesime componenti del WHILE, ma il body viene eseguito almeno una volta, e poi ripetuto fintantoché' vale la *condizione di ripetizione*.



- si "entra nel ciclo"
- si esegue l'istruzione (strutturata) istruzione_do_while
- si verifica la condizione di ripetizione
  - se la condizione vale, si ripete l'istruzione_do_while
  - se la condizione è falsa, si esce dall'istruzione



ciclo do_while – esempio sul problema precedente

Programma che legge una sequenza di numeri interi, interrotta da zero, dicendo per ciascuno se è positivo o negativo

```
Algoritmo

0) dati … n
1)   eseguire
        1.1)  chiedere e leggere n
        1.2) se n>0
                1.2.1)     ?
                altrimenti
                1.2.2)  ?
                              stampare "negativo"
                          ?
                              stampare "ciao"
     MENTRE  ?

2) fine programma
```

```
caro/a utente, dammi un intero n ... (0 per finire
6
... ma, ma, 6 e' positivo!
caro/a utente, dammi un intero n ... (0 per finire
8
... ma, ma, 8 e' positivo!
caro/a utente, dammi un intero n ... (0 per finire
-99
... ma, ma, -99 e' negativo!
caro/a utente, dammi un intero n ... (0 per finire
0
... ok, torna quando vuoi!

FINE programma
```

☺ completare l'algoritmo

ciclo do_while – esempio sul problema precedente

Programma che legge una sequenza di numeri interi, interrotta da zero, dicendo per ciascuno se e' positivo o negativo

```
caro/a utente, dammi un intero n ... (0 per f
6
... ma, ma, 6 e' positivo!
caro/a utente, dammi un intero n ... (0 per f
8
... ma, ma, 8 e' positivo!
caro/a utente, dammi un intero n ... (0 per f
-99
... ma, ma, -99 e' negativo!
caro/a utente, dammi un intero n ... (0 per f
0
... ok, torna quando vuoi!

FINE programma
```

Algoritmo

0) dati … n
1) do
    1.1) chiedere e leggere n
    1.2) se n>0
        1.2.1) stampare "positivo"
        altrimenti
        1.2.2) se n<0
            stampare "negativo"
        altrimenti
            stampare "ciao"
  while (n != 0)

2) fine programma

ciclo do_while – esempio ...

Programma che legge una sequenza di numeri interi, interrotta da zero, dicendo per ciascuno se e' positivo o negativo

```
int main () {
  int n;

 do {
   printf ("caro/a utente, dammi... (0 per finire)\n");
   scanf("%d", &n);

   if (n>0)
      printf ("... ma, ma, %d e' positivo!\n", n);
   else
      if (n<0)
         printf ("... ma, ma, %d e' negativo!\n", n);
      else printf ("... ok, torna quando vuoi!\n");
 } while (n!=0);

printf ("\nFINE programma\n");
return 0;
}
```

ciclo do_while - esempio

Programma che legge una sequenza di numeri interi, interrotta da zero, dicendo per ciascuno se e' positivo o negativo

```
int main () {
  int n;

 do {
   printf ("caro/a utente
   scanf("%d", &n);

   if (n>0)
      printf ("... ma, ma, %d e' positivo!\n", n);
   else
      if (n<0)
         printf ("... ma, ma, %d e' negativo!\n", n);
      else printf ("... ok, torna quando vuoi!\n");
 } while (n!=0);

printf ("\nFINE programma\n");
return 0;
}
```

rispetto all'esempio while precedente, qui non serve assegnare n prima del ciclo, infatti

- una prima assegnazione di n avviene proprio all'inizio dell'iterazione,
- la prima volta che n viene controllato è solo dopo che la prima iterazione è finita,
  - cioè solo dopo che n ha avuto almeno un valore significativo

ciclo do_while: componenti fondamentali … le stesse!!!

Programma che legge una sequenza di numeri interi, interrotta da zero, dicendo per ciascuno se e' positivo o negativo

```
int main () {
  int n;

 do {
   printf ("caro/a ute
   scanf("%d", &n);

   if (n>0)
      printf ("... ma,
   else
      if (n<0)
         printf ("...
      else printf ("..
 } while (n!=0);

printf ("\nFINE programma\n");
return 0;
}
```

- **condizione**
- **body**
- almeno una **istruzione**, nel body, che modifichi variabili di test
- **inizializzazione** delle variabili di test

Inizializzazione non necessariamente prima del body. Potrebbe stare nel body, perché questo viene eseguito almeno la prima volta.

Comunque l'inizializzazione ci deve essere sempre (magari *da input*, come in questo caso).

do_while VS while

Non è una lotta. Sono costrutti equivalenti, cioè si può fare tutto sia con l'uno che con l'altro: vedi prossima slide.

```
n=1;
 while (n!=0) {
    printf ("caro/a utente, dammi... (0 per
finire)\n");
    scanf("%d", &n);

    if (n>0)
       printf ("... ma, ma, %d e' positivo!\n", n);
    else
       if (n<0)
          printf ("... ma, ma, %d e' negativo!\n", n);
       else printf ("... ok, torna quando vuoi!\n");
 }        /* fine while */

printf ("\nFINE programma\n");
return 0;
}
```

do_while VS while

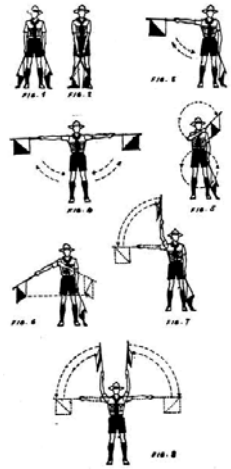
Non e' una lotta. Sono costrutti equivalenti, cioè si può fare tutto sia con l'uno che con l'altro.

Ad esempio, ecco come ottenere lo stesso programma di prima con un while invece che con un do_while (questa è una riscrittura del do_while di prima, in forma di un while).

L'unica differenza tra do_while e while è che il body del primo verrà eseguito almeno una volta, mentre il body del while potrebbe non essere mai eseguito. E' per questo che nella riscrittura del do_while in forma di while dobbiamo usare il trucchetto di dare ad n un valore iniziale "fittizio", che permetta di "entrare nel while" ed eseguire almeno) la prima iterazione.

```
n=1;
 while (n!=0) {
    printf ("caro/a utente, dammi... (0 per finire)\n");
    scanf("%d", &n);

    if (n>0)
       printf ("... ma, ma, %d e' positivo!\n", n);
    else
       if (n<0)
          printf ("... ma, ma, %d e' negativo!\n", n);
       else printf ("... ok, torna quando vuoi!\n");
 }        /* fine while */

printf ("\nFINE programma\n");
return 0;
}
```

Variabile *flag* nelle istruzioni ripetitive

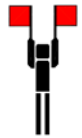


Si tratta, in genere, di una variabile che segnala quando un certo evento è accaduto (e la condizione di ripetizione assume un valore di verità conseguente).

```
/* si leggono numeri, fermandosi quando viene inserito 61
che poi viene segnalato da un commento appropriato */
…
 n=1;
 while (n!=61) {
    printf ("car* utente, dammi... (61 per finire)\n");
    scanf("%d", &n);
 }        /* fine while */
 printf (" brav*! Hai inserito 61!\n");

printf ("\nFINE programma\n");
return 0;
}
```

nel programma precedente sui numeri positivi/negativi, **n** era una variabile flag (anche detta *sentinella*, bah) … solo che la bandiera si alzava per lo zero.

Variabile *flag* nelle istruzioni ripetitive - 2



Programma che legge numeri interi e li stampa; termina quando viene inserito 0, oppure quando è stato stampato 61, in quest'ultimo caso aggiungendo un commento appropriato

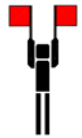
inserito_61 ==0 significa non abbiamo ancora visto 61

inserito_61 ==1 significa abbiamo visto 61 (e stampato)

```
Algoritmo

0) dati … n; inserito_61 variabile flag (a valori 0/1)
1)   do
        1.1)  chiedere e leggere n
        1.2) se n==0
                1.2.1) stampare "ok, basta"
               altrimenti
                1.2.2) stampare n
                1.2.3) se n è 61 assegnare flag ad 1
     while (n != 0      AND      inserito_61  != 1)


2) dire se è stato incontrato 61 e poi fine programma
```

tsk tsk …. cosa manca?

Variabile *flag* nelle istruzioni ripetitive - 2



Programma che legge numeri interi e li stampa; termina quando viene inserito 0, oppure quando è stato stampato 61, in quest'ultimo caso aggiungendo un commento appropriato

tsk tsk … dov'e'?

inserito_61 ==0 significa non abbiamo ancora visto 61

inserito_61 ==1 significa abbiamo visto 61

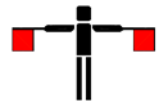
Algoritmo

0) dati … n; inserito_61 variabile flag (a valori 0/1)
1) do
    1.1) chiedere e leggere n
    1.2) se n==0
        1.2.1) stampare "ok, basta"
        altrimenti
        1.2.2) stampare n
        1.2.3) se n e' 61 assegnare flag ad 1
while (n != 0 AND inserito_61 != 1)

tsk tsk …. quanto vale la prima volta?

2) dire se è stato incontrato 61 e poi fine programma

Variabile *flag* nelle istruzioni ripetitive - 2



Programma che legge numeri interi e li stampa; termina quando viene inserito 0, oppure quando è stato stampato 61, in quest'ultimo caso aggiungendo un commento appropriato

```
Algoritmo

0) dati … n; inserito_61 variabile flag (a valori 0/1)
1)  inizializzare inserito_61
2)  do
        2.1)  chiedere e leggere n
        2.2) se n==0
                2.2.1) stampare "ok, basta"
                altrimenti
                2.2.2) stampare n
                2.2.3) se n è 61 assegnare flag ad 1
    while (n != 0      AND     inserito_61  != 1)

3) se stampato 61 (cioè se inserito_61 è 1), dirlo
4) fine programma
```

scrivere il programma corrispondente

Variabile *flag* nelle istruzioni ripetitive - 3

Programma che legge numeri interi e li stampa; termina quando viene inserito 0, oppure quando è stato stampato 61, in quest'ultimo caso aggiungendo un commento appropriato

```
#include <stdio.h>
int main () {          int n;          int inserito_61;

  inserito_61=0;     /* variabile flag: … */

 do {
   LETTURA n

   if (n==0) printf ("... ok, torna quando vuoi!\n", n);
   else {
       printf ("... bene, ho letto %d \n", n);
       if (n==61)
          inserito_61 = 1;
   }
 } while ( (n!=0) && (inserito_61 == 0) );

 if (inserito_61==1)
       printf ("bene bene, hai inserito 61!\n");

printf ("\nFINE programma\n");
return 0;
}
```

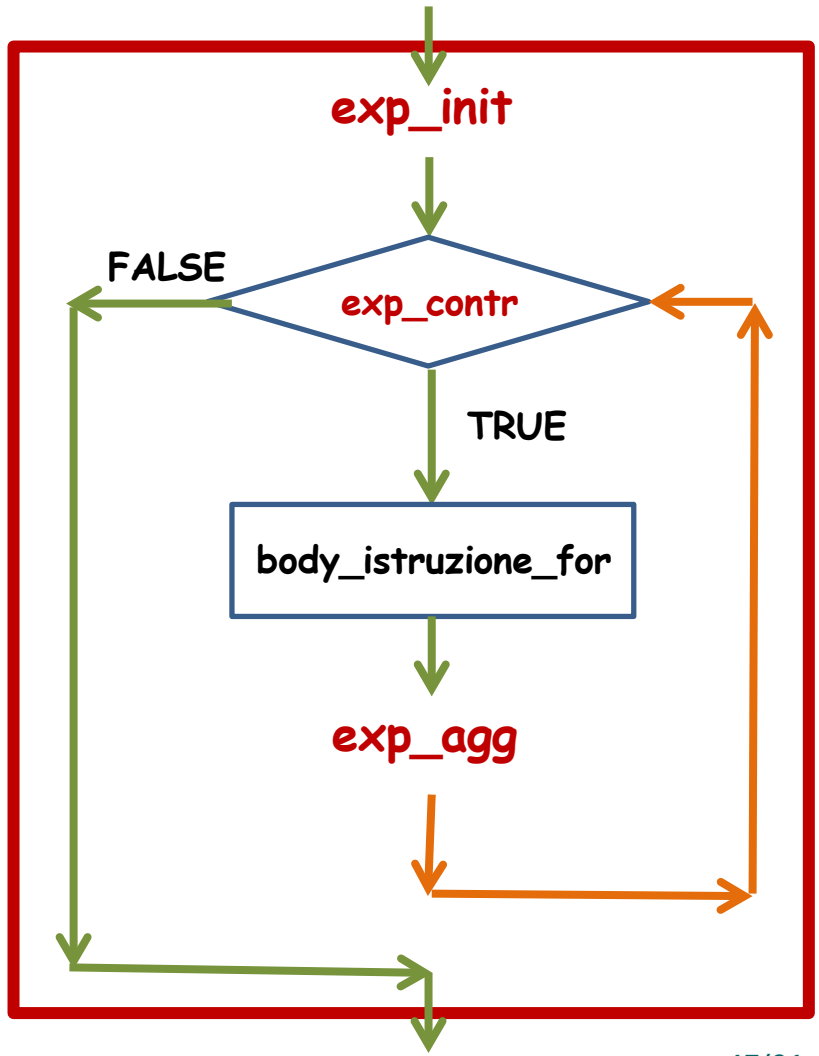
Vedi Approfondimenti

Istruzione iterativa: for

Istruzione for
Esegue il body mentre vale la condizione espressa dalla **exp_contr**

La **exp_init** viene valutata una sola volta, all'inizio, cioe' prima della prima esecuzione eventuale del body

La **exp_agg** viene valutata dopo ogni iterazione del body



I 42 con il for

Programma che stampa i primi 42 numeri (usando una costante per il 42)

```
stampiamo i ... 37
stampiamo i ... 38
stampiamo i ... 39
stampiamo i ... 40
stampiamo i ... 41
stampiamo i ... 42

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (INIT i; CHECK i; ADVANCE i)
     USE i

printf ("\nFINE programma\n");
return 0;
}
```

☺

Vedi Esercizi

Esercizio A su for

Programma che stampa i primi numeri pari non negativi minori/uguali 42 (usando una costante per il 42)

```
stampiamo i ... 36
stampiamo i ... 38
stampiamo i ... 40
stampiamo i ... 42

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (INIT i; CHECK i; ADVANCE i)
     USE i

printf ("\nFINE programma\n");
return 0;
}
```

Vedi Esercizi

Esercizio B su for

Programma che stampa i primi numeri dispari non negativi minori di 42 (usando una costante per il 42)

```
stampiamo i ... 33
stampiamo i ... 35
stampiamo i ... 37
stampiamo i ... 39
stampiamo i ... 41

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (INIT i; CHECK i; ADVANCE i)
     USE i

printf ("\nFINE programma\n");
return 0;
}
```

Esercizio B su for

Programma che stampa i primi numeri dispari non negativi minori/uguali 42 (usando una costante per il 42)

```
stampiamo i ... 33
stampiamo i ... 35
stampiamo i ... 37
stampiamo i ... 39
stampiamo i ... 41

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (i=1; i<=QUANTINUM; i += 2)
     printf ("stampiamo i ... %d\n", i);

printf ("\nFINE programma\n");
return 0;
}
```

ALGORITMO

☺ poi vedi Esercizi

Esercizio C su for

Programma che stampa i primi 42 numeri dispari non negativi (usando una costante per il 42)

```
stampiamo j ... 73
stampiamo j ... 75
stampiamo j ... 77
stampiamo j ... 79
stampiamo j ... 81
stampiamo j ... 83

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i,j;

  for (INIT i; CHECK i; ADVANCE i)
     USE i and j

printf ("\nFINE programma\n");
return 0;
}
```

☺ (suggerimento nella prossima slide)

i usato come contatore, per eseguire 42 ripetizioni esatte della stampa

Ad ogni iterazione viene stampato un numero dispari … crescente (j)

Esercizio C su for

Programma che stampa i primi 42 numeri dispari non negativi (usando una costante per il 42)

```
stampiamo j ... 73
stampiamo j ... 75
stampiamo j ... 77
stampiamo j ... 79
stampiamo j ... 81
stampiamo j ... 83

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i,j;

  for (INIT i; CHECK i; ADVANCE i)
     USE i and j

printf ("\nFINE programma\n");
return 0;
}
```

i 0 1 2 3 4 … … 40 41

j 1 3 5 7 9 … … 81 83

i usato come contatore, per eseguire 42 ripetizioni esatte della stampa

Ad ogni iterazione viene stampato un numero dispari, j, … crescente

Esercizio C su for

Programma che stampa i primi 42 numeri dispari non negativi (usando una costante per il 42)

```
stampiamo j ... 73
stampiamo j ... 75
stampiamo j ... 77
stampiamo j ... 79
stampiamo j ... 81
stampiamo j ... 83

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i,j;
```

ALGORITMO

- **inizializza i (ad esempio a 0 per indicare zero iterazioni fatte)**
- **inizializza j con il primo dispari … j=1**
- **itera 42 volte usando i come contatore** (si dice *per i che va da 0 a 41*)
  - **stampa j (attuale dispari)**
  - **incrementa j di 2 (prossimo dispari che verrà stampato … se lo verrà)**
  - **incrementa i di 1**

```
printf ("\nFINE programma\n");
return 0;
}
```

Esercizio C su for

Programma che stampa i primi 42 numeri dispari non negativi (usando una costante per il 42)

```
stampiamo j ... 73
stampiamo j ... 75
stampiamo j ... 77
stampiamo j ... 79
stampiamo j ... 81
stampiamo j ... 83

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i,j;

  j=1;    /* primo numero dispari che ci interessa */
  for (i=0; i<QUANTINUM; i+=1) {
     printf ("stampiamo j ... %d\n", j);
     j = j+2;
  }

printf ("\nFINE programma\n");
return 0;
}
```

che succede se si init i con 1?

Beh, se si modifica la condizione di ripetizione in
i<= QUANTINUM,
è tutto come prima …

Istruzione iterativa: for VS while

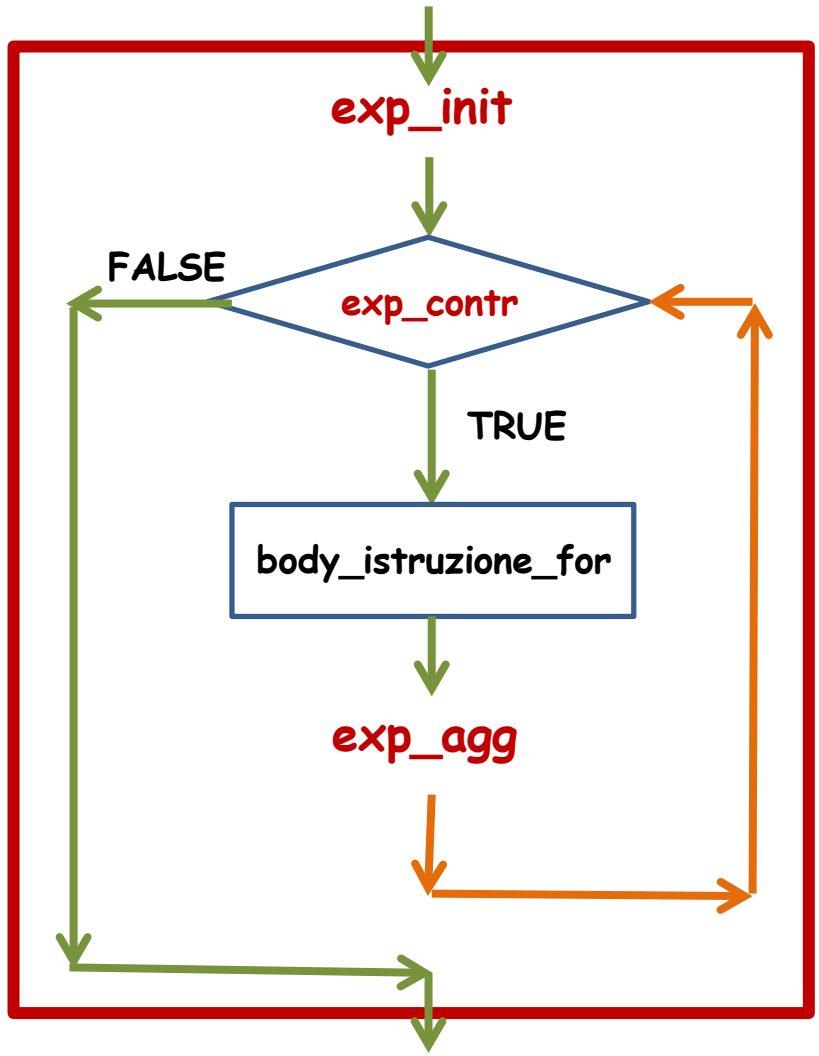
Istruzione for

Esegue il body mentre vale la condizione espressa dalla **exp_contr**

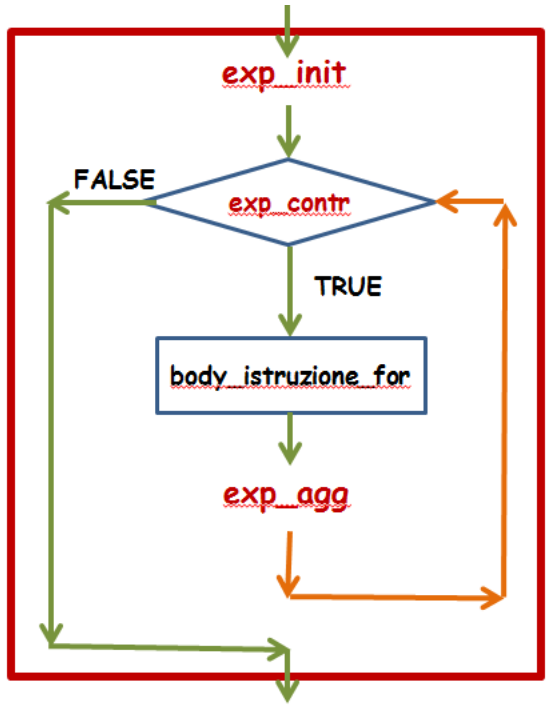
La **exp_init** viene valutata una sola volta, all'inizio, cioe' prima della prima esecuzione eventuale del body

La **exp_agg** viene valutata dopo ogni iterazione del body

Evidentemente anche in questo caso le due while e for, sono equivalenti:
si può riscrivere un for come un while e viceversa



for VS while



Evidentemente sono equivalenti anche loro: si può riscrivere un for come un while e viceversa

Poi vedi Esercizi

```
for (i=0; i<=QUANTINUM; i++)
   printf ("stampiamo i ... %d\n", i);
```

Esercizio – dieci per dieci

Programma che stampa i primi 100 numeri interi positivi, su 10 righe di 10 numeri ciascuna

```
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99

FINE programma
```

```
#include <stdio.h>

int main () {
  int i,j;

  for (INIT i; CHECK i; ADVANCE
     USE i and j

printf ("\nFINE programma\n");
return 0;
}
```

Esercizio – dieci per dieci – annidamento di cicli

Programma che stampa i primi 100 numeri interi positivi, su 10 righe di10 numeri ciascuna

```
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99

FINE programma
```

```
#include <stdio.h>



int main () {
  int i,j;
```

ALGORITMO

- **ciclo con i che va da 0 a 9**
  - **ad ogni iterazione** (`i==0, i==1, …, i==9`)
    - **viene stampata una riga di numeri che sono quelli che vanno da i*10 a i*10 + 9**
    - **fine riga** (insomma: si va a capo)

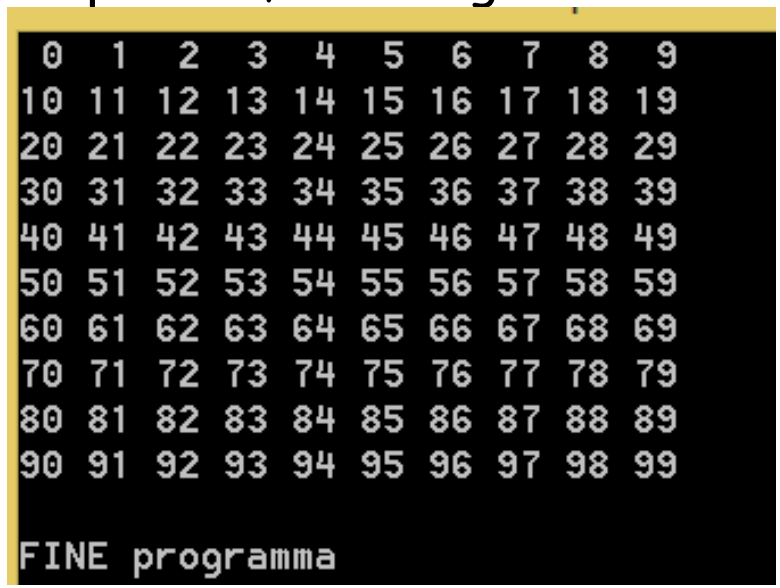
```
printf ("\nFINE programma\n");
return 0;
}
```

☺ prova a spiegare cosa avviene ad ogni iterazione del ciclo su i

☺ prova a spiegare cosa avviene ad ogni iterazione del ciclo su j
☺ (cioe` , cosa succede per j=0? Cosa per j=1? 2 3 … 9?

Esercizio – dieci per dieci – annidamento

Programma che stampa i primi 100 numeri interi positivi, su 10 righe di10 numeri ciascuna




```
#include <stdio.h>



int main () {
  int i,j;
```

ALGORITMO

1) ciclo con i che va da 0 a 9

- ad ogni iterazione viene stampata una riga di numeri che sono quelli che vanno da **i*10** a **i*10 + 9**

  CIOE`

  1.1) ciclo con j che va da 0 a 9

  - **ad ogni iterazione viene stampato il numero i*10 + j**

  1.2) andare a capo per la prossima riga di numeri

- **fine …**

```
printf ("\nFINE programma\n");
return 0;
}
```

Esercizio – dieci per dieci

Programma che stampa i primi 100 numeri interi positivi, su 10 righe di10 numeri ciascuna

```
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99

FINE programma
```

```
#include <stdio.h>



int main () {
  int i,j;

 for (i=0; i<10; i++) {
       /* stampa la riga i-esima di 10 numeri */
       for (j=0; j<10; j++)              /* 1) */
          printf ("%2d ", i*10 + j);
       printf ("\n");                    /* 2) fine riga */
 }
printf ("\nFINE programma\n");
return 0;
}
```

break

E' la keyword per interrompere l'esecuzione di un ciclo.
(ovvero per uscire dal blocco in esecuzione}.

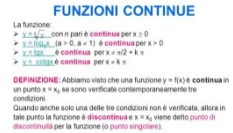
Di solito viene inserita in un'istruzione ripetitiva,
in modo che venga eseguita al verificarsi di una condizione che significhi "impossibile continuare le iterazioni".

Esempio:
Programma che riceve una sequenza di QUANTE_VOLTE numeri interi e ne stampa i reciproci; se viene introdotto zero si arrabbia

```
Algoritmo

0)  str. dati … n, i;      costante QUANTE_VOLTE
1)  per i che va da 0 a QUANTE_VOLTE -1
      1.1) chiedere e leggere n
      1.2) se n==0
                1.2.1)  arrabbiarsi e interrompere il ciclo
           altrimenti
                1.2.2) stampare 1/n
2) fine programma
```

Vedi Approfondimenti

continue



E' la keyword per l'istruzione capace di interrompere l'esecuzione della attuale iterazione e far proseguire con la prossima.
A volte può essere utile, ma in linea di principio se ne può fare a meno: basta usare bene le condizioni nelle istruzioni di controllo. Comunque …
Esempio:
Programma che riceve una sequenza di QUANTE_VOLTE numeri interi e ne stampa i reciproci; se viene introdotto zero, non si arrabbia ma prosegue con il prossimo numero

```
Algoritmo

0)  dati … n, i;              costante QUANTE_VOLTE
1)  per i che va da 0 a QUANTE_VOLTE-1
      1.1) chiedere e leggere n
      1.2) se n==0
                1.2.1) interrompere l'iterazione e passare alla successiva
           altrimenti
                1.2.2) stampare 1/n
2) fine programma
```

Vedi Approfondimenti

Tecniche della Programmazione, lez. 7

- Approfondimenti

Istruzione iterativa: while (5/5)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4;

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

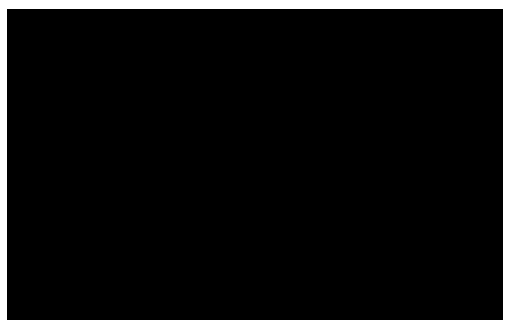
2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

4	n



Istruzione iterativa: while (5/5)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

4	n

```
..... 4
```

Istruzione iterativa: while (5/5)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

4 ~~-1~~ **n**

```
..... 4
```

Istruzione iterativa: while (5/5)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

4 ~~-1~~ **n**

```
..... 4
..... -1
```

Istruzione iterativa: while (5/5)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

~~4~~ ~~-1~~ 1 **n**

```
...... 4
...... -1
```

Istruzione iterativa: while (5/5)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

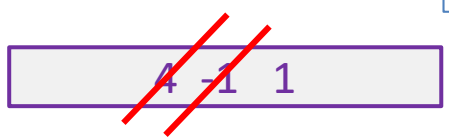
2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

n>0: iterazione 2




```
...... 4
...... -1
```

Istruzione iterativa: while (5/5)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4;

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

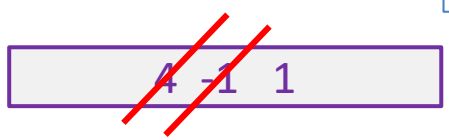
2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

n>0: iterazione 2




```
...... 4
...... -1
...... 1
```

Istruzione iterativa: while (5/5)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

n>0: iterazione 2

4 ~~-1~~ ~~1~~ ~~-4~~ **n**

```
...... 4
...... -1
...... 1
```

Istruzione iterativa: while (5/5)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

```
init
n>0: iterazione 1
n>0: iterazione 2
```

4 ~~-1~~ ~~1~~ ~~-4~~ **n**

```
...... 4
...... -1
...... 1
...... -4
```

Istruzione iterativa: while (5/5)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

n>0: iterazione 2

4 ~~-1~~ ~~1~~ ~~-4~~ -2 **n**

```
...... 4
...... -1
...... 1
...... -4
```

Istruzione iterativa: while (5/5)

(anche se sono 11 pagine ...)

esecuzione simulata

```
#include <stdio.h>

int main () {
  int n;

  n=4

  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbe', abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

n>0: iterazione 2

n>0: FALSE

4 ~~-1~~ ~~1~~ ~~-4~~ -2 **n**

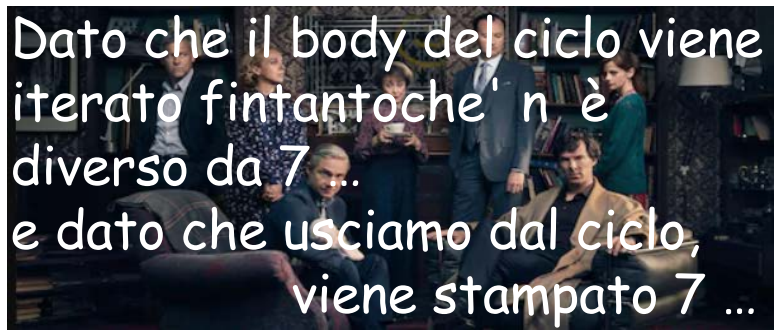
```
...... 4
...... -1
...... 1
...... -4
FINE
```

Istruzione iterativa: while

Ciclo solo parzialmente visibile:

supponendo che termini, cosa stampa la **printf()**?

```
#include <stdio.h>

int main () {
  int n=21;

  while (n!=7) {
     n=n*12;
     pr                o n ... %d\n", n);
     n =
  }
printf ("stampiamo n ... %d\n", n);
printf ("\nFINE programma\n");
return 0;
}
```



controllo di un numero dato in input … 5 volte

Algoritmo per un programma che esegue 5 volte la lettura di un numero e la stampa della sua qualità (essere negativo, positivo o zero)

NB

```
Algoritmo

0)  n per i dati, i come contatore,
    costante QUANTI uguale a 5

1)  init i              i=1

2)  mentre i <= QUANTI
    2.1) chiedere e leggere n
    2.2) se n>0
              2.2.1) stampare "positivo"
         altrimenti
              2.2.2) se n<0
                        stampare "negativo"
                     altrimenti
                        stampare "ma …"
    2.3) i = i+1;
3) fine programma
```

iterazioni con i = 1, 2, 3, 4, 5

iterazioni con i = 0, 1, 2, 3, 4

NBB

```
Algoritmo

0)  n per i dati, i come contatore,
    costante QUANTI uguale a 5

1)  init i              i=0

2)  mentre i < QUANTI
    2.1) chiedere e leggere n
    2.2) se n>0
              2.2.1) stampare "positivo"
         altrimenti
              2.2.2) se n<0
                        stampare "negativo"
                     altrimenti
                        stampare "ma …"
    2.3) i = i+1;
3) fine programma
```

Controllo del numero di iterazioni in base ad un dato di input

Programma che legge una sequenza di numeri interi, interrotta da zero, dicendo per ciascuno se e' positivo o negativo

```
int main () {
  int n=1;

 while (n!=0) {
   printf ("caro/a utente, dammi... (0 per finire)\n");
   scanf("%d", &n);

   if (n>0)
      printf ("... ma, ma, %d e' positivo!\n", n);
   else
      if (n<0)
         printf ("... ma, ma, %d e' negativo!\n", n);
      else printf ("... ok, torna quando vuoi!\n");
 }

printf ("\nFINE programma\n");
return 0;
}
```

- ☺ perché mettere questa istruzione qui?
- ☺ quali altri valori potrei assegnare ad n, ottenendo lo stesso effetto?
- ☺ quali valori non dovrei assegnare ad n per evitare di rendere il programma malfunzionante?

Vedi Approfondimenti

ciclo while - esempio

Programma che legge una sequenza di numeri interi, interrotta da zero, dicendo per ciascuno se e' positivo o negativo

```
int main () {
  int n=1;

 while (n!=0) {
   printf ("carc
   scanf("%d", &

   if (n>0)
      printf ("... ma, ma, %d e' positivo!\n", n);
   else
      if (n<0)
         printf ("... ma, ma, %d e' negativo!\n", n);
      else printf ("... ok, torna quando vuoi!\n");
 }

printf ("\nFINE programma\n");
return 0;
}
```

☺ questa inizializzazione serve a dare ad n un valore, in modo che
1) possa essere valutata la condizione di ripetizione, e
2) questa permetta di "entrare nel ciclo" (fare almeno una iterazione)
... altrimenti si passerebbe alla fine senza mai leggere numeri ...

☺ qualunque valore diverso da zero fa ottenere l'effetto descritto qui sopra

☺ eh, beh ... se inizializziamo n con 0 ... passiamo direttamente alla fine ... non viene eseguita nessuna iterazione ...

Variabile *flag* nelle istruzioni ripetitive - 3

Programma che legge numeri interi e li stampa; termina quando viene inserito 0, oppure quando è stato stampato 61, in quest'ultimo caso aggiungendo un commento appropriato

```
#include <stdio.h>
int main () {          int n;          int inserito_61;

  inserito_61=0;     /* variabile flag: … */

 do {
   printf ("caro/a utente, dammi un intero n ... (0 per finire)\n");
   scanf("%d", &n);

   if (n==0) printf ("... ok, torna quando vuoi!\n", n);
   else {
       printf ("... bene, ho letto %d \n", n);
       if (n==61)
          inserito_61 = 1;
   }
 } while ( (n!=0) && (inserito_61 == 0) );

 if (inserito_61==1)
       printf ("bene bene, hai inserito 61!\n");

printf ("\nFINE programma\n");
return 0;
}
```

Variabile *flag* nelle istruzioni ripetitive - 3

Programma che legge numeri interi e li stampa; termina quando viene inserito 0, oppure quando e' stato stampato 61, in quest'ultimo caso aggiungendo un commento appropriato

```
#include <stdio.h>
int main () {          in

  inserito_61=0;     /* v

 do {
   printf ("caro/a utent
   scanf("%d", &n);

   if (n==0) printf ("..
   else {
       printf ("... bene
       if (n==61)
          inserito_61 = 1
   }
 } while ( (n!=0) && (!inserito_61) );

 if (inserito_61)
       printf ("bene bene, hai inserito 61!\n");

printf ("\nFINE programma\n");
return 0;
}
```

NB (inserito_61)
e
(inserito_61 != 0)
sono espressioni booleane equivalenti, cioè hanno sempre il medesimo valore di verità

idem per (inserito_61 == 0)
e
(!inserito_61)

Variabile *flag* nelle istruzioni ripetitive - 3

Programma che legge numeri interi e li stampa; termina quando viene inserito 0, oppure quando e' stato stampato 61, in quest'ultimo caso aggiungendo un commento appropriato

```
#include <stdio.h>
int main () {          int n

  inserito_61=0;     /* var

 do {
   printf ("caro/a utente,
   scanf("%d", &n);

   if (n==0) printf ("... c
   else {
       printf ("... bene, ho letto %d \n", n);
       if (n==61)
          inserito_61 = 1;
   }
 } while ( (n!=0) && (!inse

 if (inserito_61)
       printf ("bene bene,

printf ("\nFINE programma\n
return 0;
}
```

inserito_61	inserito_61!=0
TRUE	TRUE
FALSE	FALSE

!inserito_61	inserito_61==0
☺ risolvi e poi vedi successiva	

Variabile *flag* nelle istruzioni ripetitive - 3

Programma che legge numeri interi e li stampa; termina quando viene inserito 0, oppure quando e' stato stampato 61, in quest'ultimo caso aggiungendo un commento appropriato

```
#include <stdio.h>
int main () {          int n;          int inserito_61;
```

inserito_61	inserito_61!=0	!inserito_61	inserito_61==0
TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	TRUE

```
        printf ("... bene, ho letto %d \n", n);
        if (n==61)
           inserito_61 = 1;
    }
 } while ( (n!=0) && (!inserito_61) );

 if (inserito_61)
        printf ("bene bene, hai inserito 61!\n");

printf ("\nFINE programma\n");
return 0;
}
```

break

E' la keyword per interrompere l'esecuzione di un ciclo.
(ovvero per uscire dal blocco in esecuzione}.

Di solito viene inserita in un'istruzione ripetitiva,
in modo che venga eseguita al verificarsi di una condizione che significhi "impossibile continuare le iterazioni".

Esempio:
Programma che riceve una sequenza di QUANTE_VOLTE numeri interi e ne stampa i reciproci; se viene introdotto zero si arrabbia

```
Algoritmo

0)  str. dati … n, i;      costante QUANTE_VOLTE
1)  per i che va da 0 a QUANTE_VOLTE -1
    1.1) chiedere e leggere n
    1.2) se n==0
                1.2.1)  arrabbiarsi e interrompere il ciclo
           altrimenti
                1.2.2) stampare 1/n
2) fine programma
```

break

E' la k

```
--- reciproconumero 0 ---
caro/a utente, dammi un intero diverso da zero: 2
va bene, caro/a, il reciproco di 2 e' 0.5
--- reciproconumero 1 ---
caro/a utente, dammi un intero diverso da zero: 0
NO!!!! ARRGHHH! BASTA!

FINE programma
```

ecuzione}.

Di soli

in moc

nifichi

terazioni".

Esemp

Progr

nteri e ne

stampa i reciproci; se viene introdotto zero si arrabbia

Algoritmo

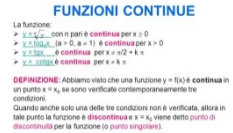
0) str. dati … n, i; costante QUANTE_VOLTE
1) per i che va da 0 a QUANTE_VOLTE -1
   1.1) chiedere e leggere n
   1.2) se n==0
        1.2.1) arrabbiarsi e interrompere il ciclo
   altrimenti
        1.2.2) stampare 1/n
2) fine programma

break --- esempio

Programma che riceve una sequenza di QUANTE_VOLTE numeri interi e ne stampa i reciproci; se viene introdotto zero si arrabbia

```
#include <stdio.h>
#define QUANTE_VOLTE 5
int main () {
  int i, n;

 for (i=0; i<QUANTE_VOLTE; i++) {
     printf ("--- reciproconumero %d ---\n", i);
     printf ("caro/a utente, dammi … diverso da zero: ");
     scanf("%d", &n);

     if (n==0) {
        printf ("NO!!!! ARRGHHH! BASTA!\n");
        break;
     }
     else
        printf ("… reciproco di %d e' %g\n", n, 1.0/n);
 }
printf ("\nFINE programma\n");
return 0;   }
```

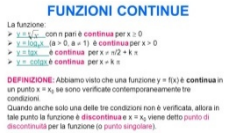
continue



E' la keyword per l'istruzione capace di interrompere l'esecuzione della attuale iterazione e far proseguire con la prossima.
A volte può essere utile, ma in linea di principio se ne può fare a meno: basta usare bene le condizioni nelle istruzioni di controllo. Comunque …
Esempio:
Programma che riceve una sequenza di QUANTE_VOLTE numeri interi e ne stampa i reciproci; se viene introdotto zero, non si arrabbia ma prosegue con il prossimo numero

```
Algoritmo

0)  dati … n, i;              costante QUANTE_VOLTE
1)  per i che va da 0 a QUANTE_VOLTE-1
      1.1) chiedere e leggere n
      1.2) se n==0
                1.2.1) interrompere l'iterazione e passare alla successiva
           altrimenti
                1.2.2) stampare 1/n
2) fine programma
```

continue



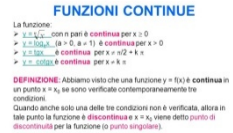
E' la ke…one
della at…
A volte…a meno:
basta u…
Esempi…
Prograr…i interi e ne
stampa…prosegue con
il pross…

```
--- reciproconumero 0 ---
caro/a utente, dammi un intero diverso da zero: 3
va bene, caro/a, il reciproco di 3 e' 0.333333
--- reciproconumero 1 ---
caro/a utente, dammi un intero diverso da zero: 0
continuiamo; non faccio commenti che e' meglio...
--- reciproconumero 2 ---
caro/a utente, dammi un intero diverso da zero: 4
va bene, caro/a, il reciproco di 4 e' 0.25
--- reciproconumero 3 ---
```

Algoritmo

0) dati … n, i; costante QUANTIE_VOLTE
1) per i che va da 0 a QUANTIE_VOLTE -1
   1.1) chiedere e leggere n
   1.2) se n==0
      1.2.1) interrompere l'terazione e passare alla successiva
   altrimenti
      1.2.2) stampare 1/n
2) fine programma

continue



Programma che riceve una sequenza di QUANTE_VOLTE numeri interi e ne stampa i reciproci; se viene introdotto zero, non si arrabbia ma prosegue con il prossimo numero

```
#include <stdio.h>
#define QUANTE_VOLTE 5
int main () {
  int i, n;

 for (i=0; i<QUANTE_VOLTE; i++) {
     printf ("--- reciproconumero %d ---\n", i);
     printf ("caro/a utente, …da zero: ");
     scanf("%d", &n);

     if (n==0) {
        printf ("continuiamo; … e' meglio...\n");
        continue;
     }
     else
          printf ("va bene, … %d e' %g\n", n, 1.0/n);
 }
printf ("\nFINE programma\n");     return 0;        }
```

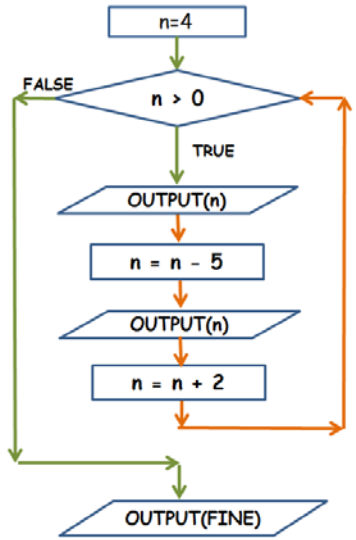
Tecniche della Programmazione, lez. 7

- Esercizi

challenge 1

Per quali valori iniziali di n il programma esegue esattamente 8 iterazioni?

```
n=?;
  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
```



sono più valori possibili, o è uno solo?

Qui c'è il diagramma di flusso (flow-chart); nella slide successiva c'è l'algoritmo scritto verbalmente.

Successivamente ancora c'è un suggerimento su come venire a capo di questo problema.

challenge 1

Per quali valori iniziali di n il programma esegue esattamente 8 iterazioni?

```
n=?;
  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0
   2.1) stampa n
   2.2) cambia n: n=n-5
   2.3) stampa n
   2.4) cambia n: n+=2

3) Vabbè, abbiamo finito (stampa FINE PROGRAMMA e poi end)

sono più valori possibili, o è uno solo?

nella prossima c'e` un suggerimento su come venire a capo di questo problema. Ma e` bene provarci da soli prima ... suvvia ... giriamo un ` queste rotelle coperte dalla sabbia dei social ...

challenge 1

Per quali valori iniziali di n il programma esegue esattamente 8 iterazioni?

```
n=?;
  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
```

sono più valori possibili, o è uno solo?

suggerimento: aggiungere nel corpo del ciclo una istruzione di stampa che visualizzi il numero d'ordine dell'iterazione e il valore di n quando quell'iterazione ha inizio...

```
--- iterazione 1: valore di n all'inizio dell'iterazione: 1
stampiamo n ... 18
stampiamo n ... 13
--- iterazione 2: valore di n all'inizio dell'iterazione: 1
stampiamo n ... 15
stampiamo n ... 10
--- iterazione 3: valore di n all'inizio dell'iterazione: 1
stampiamo n ... 12
stampiamo n ... 7
--- iterazione 4: valore di n all'inizio dell'iterazione: 9
```

e poi osservare l'output

suggerimento finale segue ... ma prima rifletterci ...

challenge 1

Per quali valori iniziali di n il programma esegue esattamente 8 iterazioni?

```
n=?;
  while (n>0) {
     printf ("stampiamo n ... %d\n", n);
     n = n-5;
     printf ("stampiamo n ... %d\n", n);
     n = n+2;
  }
```

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

3) Vabbè, abbiamo finito (stampa FINE PROGRAMMA e poi end)

sono più valori possibili, o è uno solo?

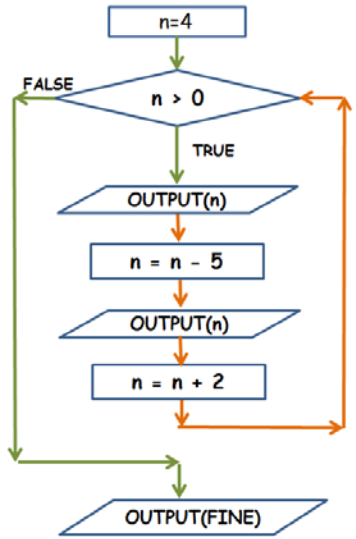
le stampe aggiunte prima dovrebbero aiutare.

Ma anche solo guardando il codice si capisce che alla fine di ogni iterazione n e` diminuito di 3 rispetto al valore che aveva quando l'iterazione e` iniziata ...

Quindi, ad esempio, se n all'inizio del programma e` inizializzato a 13 diventera` negativo al termine della quinta iterazione (non prima): una sesta iterazione non potrebbe esserci ...

challenge 2

Per quali valori iniziali di n il programma esegue esattamente 12 iterazioni



inoltre, stabilire per quali valori iniziali di n si ottengono 12 iterazioni

e poi

stabilire per quale valore di n si ottiene proprio il seguente output in corrispondenza delle ultime iterazioni

```
--- iterazione 10
stampiamo n ... 8
stampiamo n ... 3

--- iterazione 11
stampiamo n ... 5
stampiamo n ... 0

--- iterazione 12
stampiamo n ... 2
stampiamo n ... -3
```

controllo di un numero dato in input … quante volte vuoi

Algoritmo per un programma che
- chiede quante volte deve ricevere un numero e stampare la sua qualità,
- e lo fa

```
Algoritmo

0)  n per i dati, i come contatore, ???
1)  ???
2)  init i          i=0
3)  mentre  ???
    3.1) chiedere e leggere n
    3.2) se n>0
             3.2.1) stampare "positivo"
         altrimenti
             3.2.2) se n<0
                        stampare "negativo"
                    altrimenti
                        stampare "ma …"
…
```

cosa cambia rispetto alle 5 volte?

se e` l'utente che specifica quante volte bisogna leggere e controllare, "quanti" (cioe` quanti numeri vanno controllati, cioe` quante iterazioni vanno eseguite, non puo` essere una costante …

Vedi prossima slide

controllo di un numero dato in input … quante volte vuoi

Algoritmo per un programma che chiede quante volte deve ricevere un numero e stamapare la sua qualità, e lo fa…

```
Algoritmo

0)  n per i dati, i come contatore, quanti (dato dall'utente)
1)  chiedere e leggere quanti
2)  init i          i=0
3)  mentre   i < quanti
    3.1) chiedere e leggere n
    3.2) se n>0
              3.2.1) stampare "positivo"
         altrimenti
              3.12) se n<0
                        stampare "negativo"
                    altrimenti
                        stampare "ma …"
    3.2)  i+=1;

4) fine programma
```

Scrivere il programma corrispondente

I 42 con il for

Programma che stampa i primi 42 numeri (usando una costante per il 42)

```
stampiamo i ... 37
stampiamo i ... 38
stampiamo i ... 39
stampiamo i ... 40
stampiamo i ... 41
stampiamo i ... 42

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (INIT i; CHECK i; ADVANCE i)
     USE i

printf ("\nFINE programma\n");
return 0;
}
```

scrivi INIT e Vedi il resto

I 42 con il for

Programma che stampa i primi 42 numeri (usando una costante per il 42)

```
stampiamo i ... 37
stampiamo i ... 38
stampiamo i ... 39
stampiamo i ... 40
stampiamo i ... 41
stampiamo i ... 42

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (i=1; CHECK i; ADVANCE i)
     USE i

printf ("\nFINE programma\n");
return 0;
}
```

scrivi CHECK e Vedi il resto

I 42 con il for

Programma che stampa i primi 42 numeri (usando una costante per il 42)

```
stampiamo i ... 37
stampiamo i ... 38
stampiamo i ... 39
stampiamo i ... 40
stampiamo i ... 41
stampiamo i ... 42

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (i=1; i<=QUANTINUM; ADVANCE i)
     USE i

printf ("\nFINE programma\n");
return 0;
}
```

scrivi ADVANCE e Vedi il resto

I 42 con il for

Programma che stampa i primi 42 numeri (usando una costante per il 42)

```
stampiamo i ... 37
stampiamo i ... 38
stampiamo i ... 39
stampiamo i ... 40
stampiamo i ... 41
stampiamo i ... 42

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (i=1; i<=QUANTINUM; i++ )
     USE i

printf ("\nFINE programma\n");
return 0;
}
```

scrivi USE e Vedi successiva

I 42 con il for

Programma che stampa i primi 42 numeri (usando una costante per il 42)

```
stampiamo i ... 37
stampiamo i ... 38
stampiamo i ... 39
stampiamo i ... 40
stampiamo i ... 41
stampiamo i ... 42

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (i=1; i<=QUANTINUM; i++ )
     printf ("stampiamo i ... %d\n", i);

printf ("\nFINE programma\n");
return 0;
}
```

Esercizio A su for

Programma che stampa i primi numeri pari non negativi minori/uguali 42 (usando una costante per il 42)

```
stampiamo i ... 36
stampiamo i ... 38
stampiamo i ... 40
stampiamo i ... 42

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (i=2; i<=QUANTINUM; i += 2)
     printf ("stampiamo i ... %d\n", i);

printf ("\nFINE programma\n");
return 0;
}
```

?e se invece scrivessimo

```
for (i=2; ((i<=QUANTINUM) && (i%2==0)) ; i += 1)
     printf ("stampiamo i ... %d\n", i);
```

☺ poi vedi successiva

Esercizio A su for

Programma che stampa i primi numeri pari non negativi minori/uguali 42 (usando una costante per il 42)

```
stampiamo i ... 36
stampiamo i ... 38
stampiamo i ... 40
stampiamo i ... 42

FINE programma
```

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (i=2; i<=QUANTINUM; i += 2)
     printf ("stampiamo i ... %d\n", i);

printf ("\nFINE programma\n");
return 0;
}
```

?e se invece scrivessimo

```
for (i=2; ((i<=QUANTINUM) && (i%2==0)) ; i += 1)
     printf ("stampiamo i ... %d\n", i);
```

esecuzione simulata:
quando i vale 2 … … … … … … … … … … … stampa 2
quando i vale 3 ? … … … … … …fine ciclo
quindi non fa quel che volevamo

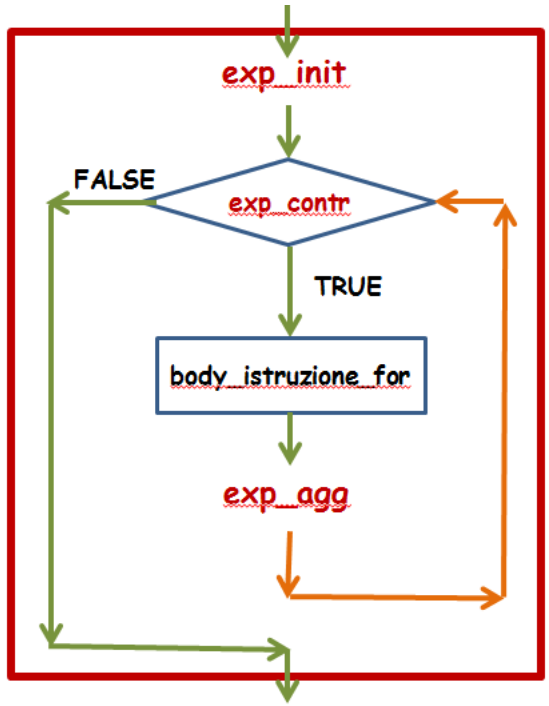
Esercizio B su for

Programma che stampa i primi numeri dispari non negativi minori di 42 (usando una costante per il 42)

ALGORITMO
- **inizializza i con il primo dispari: i=1**
- **PER i che va da 1 a 42 (o lo supera)**
  a passi di due incrementi per volta
  - Stampa i

```
#include <stdio.h>

#define QUANTINUM 42

int main () {
  int i;

  for (i=1; i<=QUANTINUM; i += 2)
     printf ("stampiamo i ... %d\n", i);

printf ("\nFINE programma\n");
return 0;
}
```

for VS while

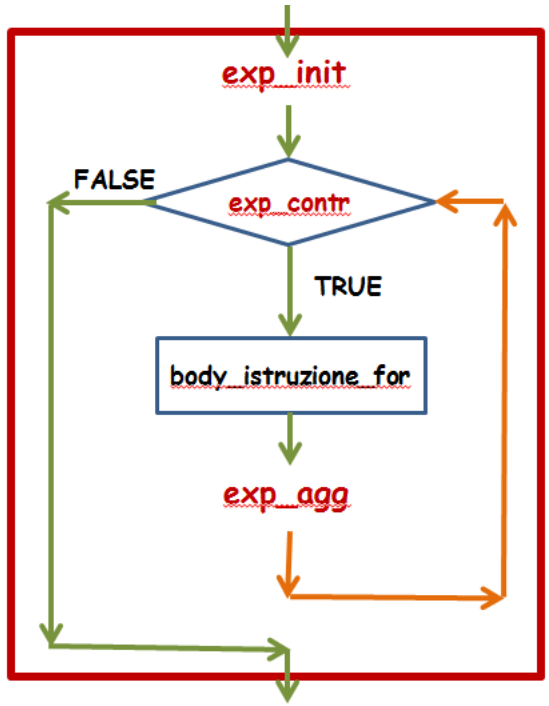


Evidentemente sono equivalenti anche loro: si può riscrivere un for come un while e viceversa

Poi vedi dopo

```
for (i=0; i<=QUANTINUM; i++)
   printf ("stampiamo i ... %d\n", i);
```

for VS while



Evidentemente sono equivalenti anche loro: si può riscrivere un for come unwhile e viceversa

```
i=0;                                /* init */
while (i<=QUANTINUM) {
   printf ("stampiamo i ... %d\n", i);
   i++;
}
```

```
for (i=0; i<=QUANTINUM; i++)
   printf ("stampiamo i ... %d\n", i);
```

Tipo base char: codici interi (-128??)

Con istruzione iterativa FOR

```
#include <stdio.h>
int main () {
  char c;
  int cod, c1;

 for (cod=-128; cod<=127; cod++)
    printf ("il carattere corrispondente al codice %4d
         (%3d) e' %c\n", cod, (cod<0? cod+256:cod), cod);
......
                    /* e con un while */
  cod = -128;              /* inizializzazione */
  while (cod <= 127) {
      printf ("il carattere corrispondente al codice %4d
(%3d) e' %c\n", cod, (cod<0? cod+256:cod), cod);
      cod+=1;
  }
printf("FINE\n");
getchar();
return 0;
}
```

☺ metterlo alla prova …