Tecniche della Programmazione, lez.7

Continuiamo sulle

- Istruzioni ripetitive, istruzioni iterative, istruzioni di ciclo…

Istruzione ripetitiva: while (1/5)

Cosa fa?

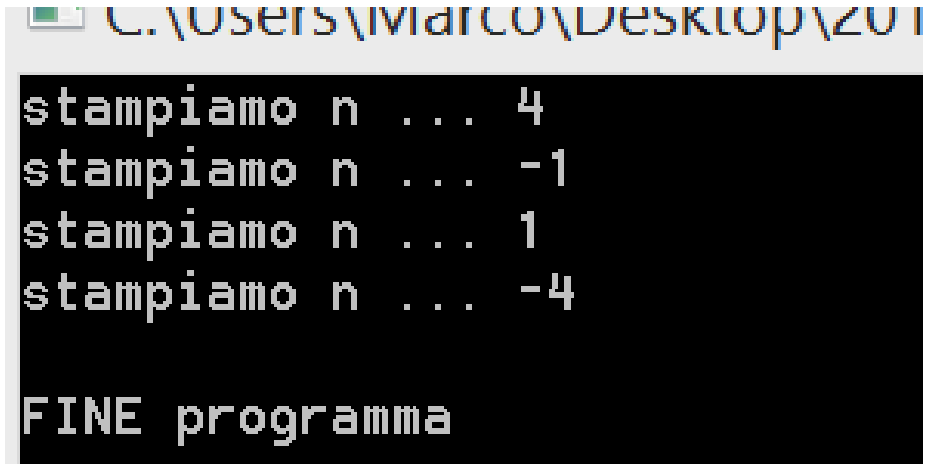
```
#include <stdio.h>

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?

```
#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;
}
```

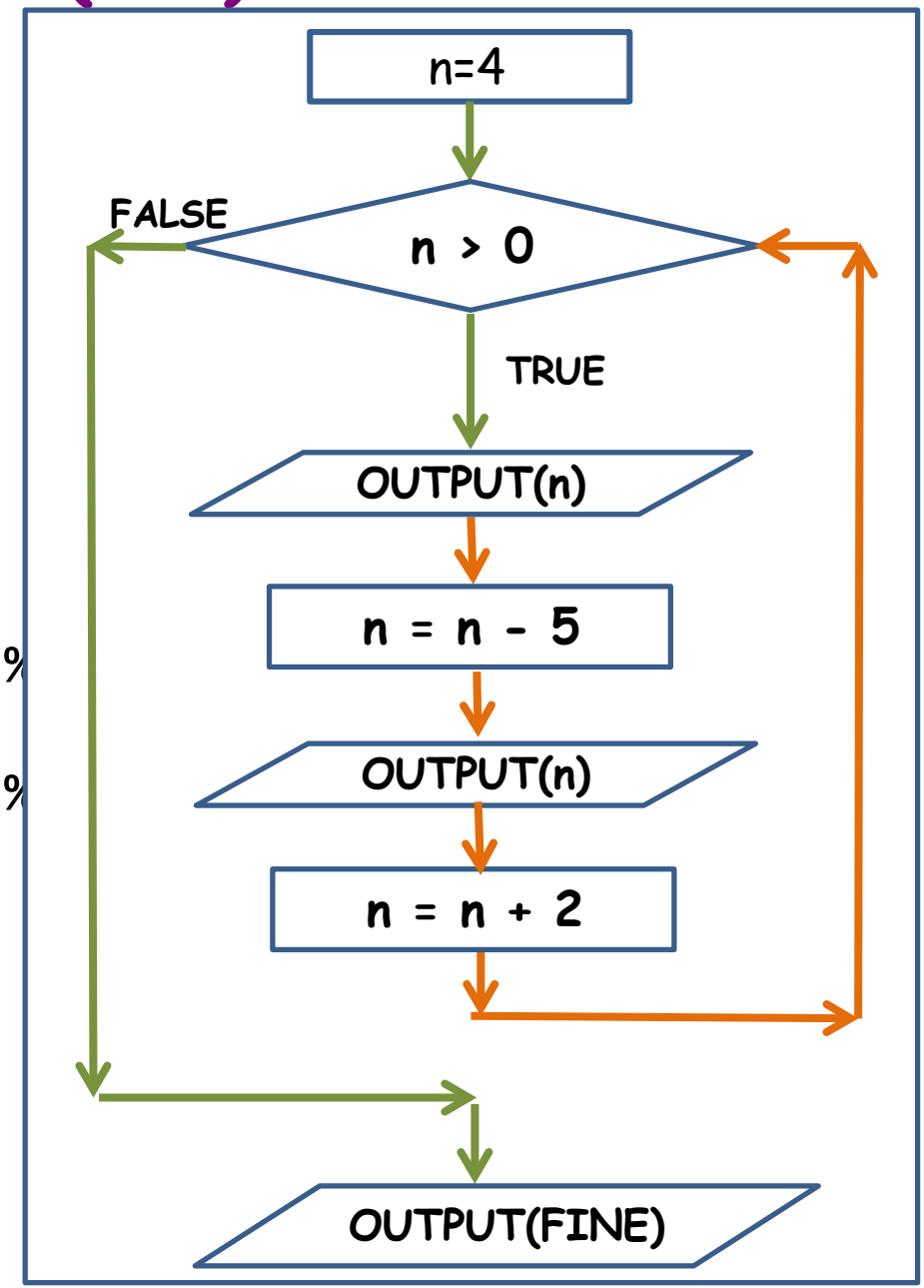


Algoritmo? Fai il Diagramma di Flusso, e l'algoritmo per passi

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

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

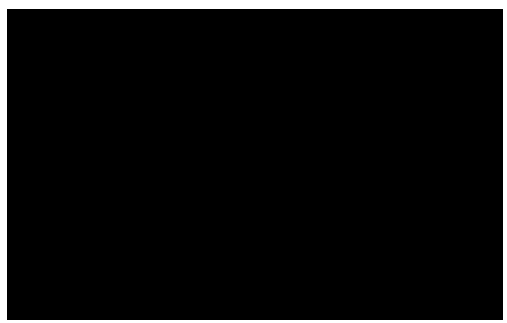
2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

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

init

4	n

n>0: iterazione 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	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

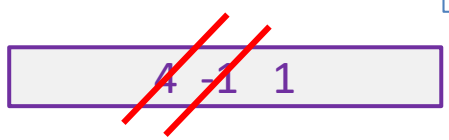
1) Inizializzazione n=4

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
```

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

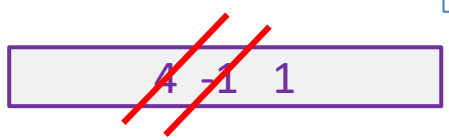
2) MENTRE n>0
   - 2.1) stampa n
   - 2.2) cambia n: n=n-5
   - 2.3) stampa n
   - 2.4) cambia n: n+=2

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~~ **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
```

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)

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
```

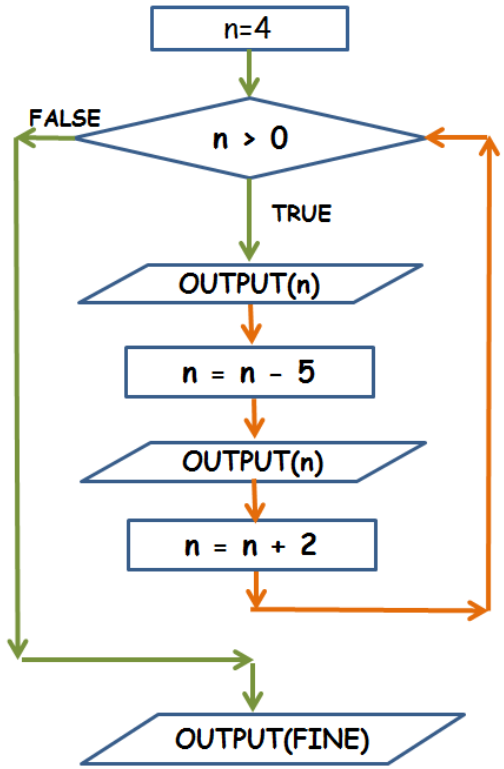
challenge

Per quali valori di n il programma esegue esattamente 8 iterazioni

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

prima scrivere questi valori (e' uno solo?)

e poi far girare il programma per verificare

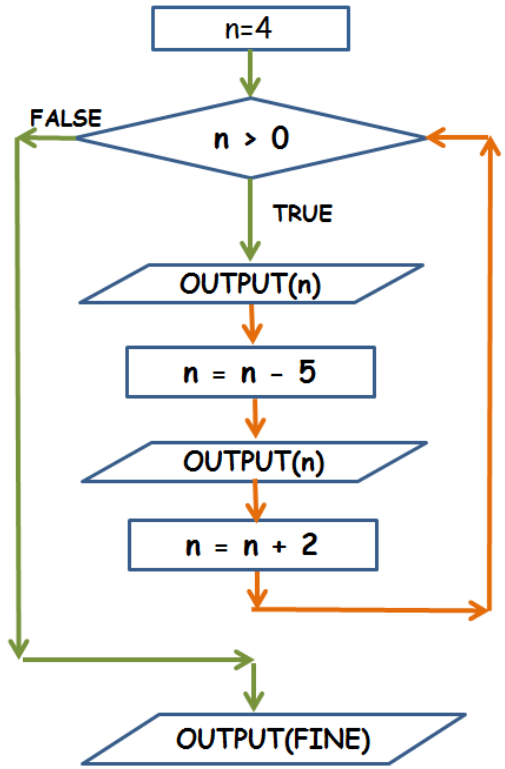


challenge – DOPO LA LEZIONE

Per quali valori di n il programma esegue esattamente 12 iterazioni

modificare il programma in modo che produca il seguente output

```
--- iterazione 1
stampiamo n ... 16
stampiamo n ... 11

--- iterazione 2
stampiamo n ... 13
stampiamo n ... 8

--- iterazione 3
stampiamo n ... 10
stampiamo n ... 5

--- iterazione 4
stampiamo n ... 7
stampiamo n ... 2

--- iterazione 5
stampiamo n ... 4
stampiamo n ... -1

--- iterazione 6
stampiamo n ... 1
stampiamo n ... -4


FINE programma
```



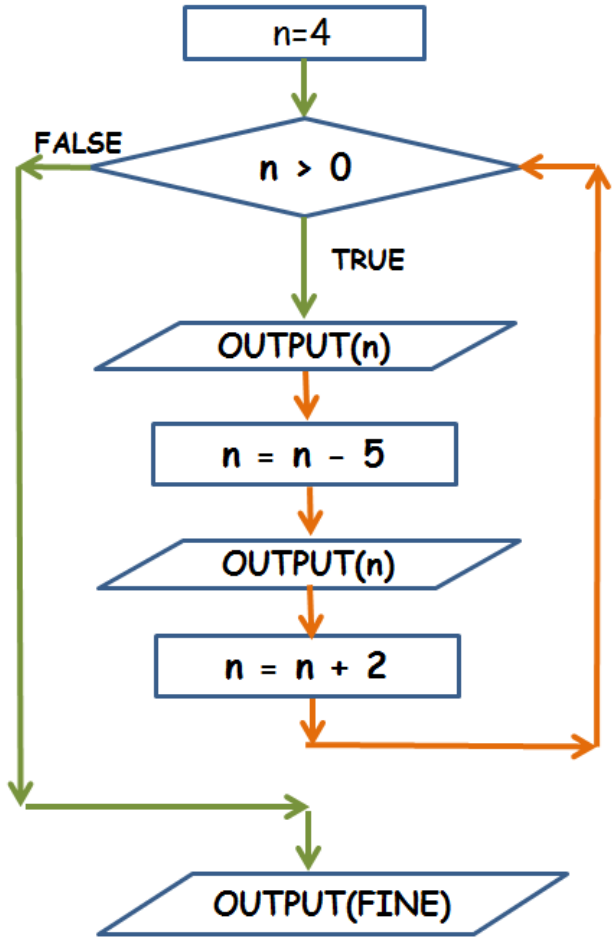
inoltre stabilire per quale valore di n si ottiene il seguente output

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

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

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

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

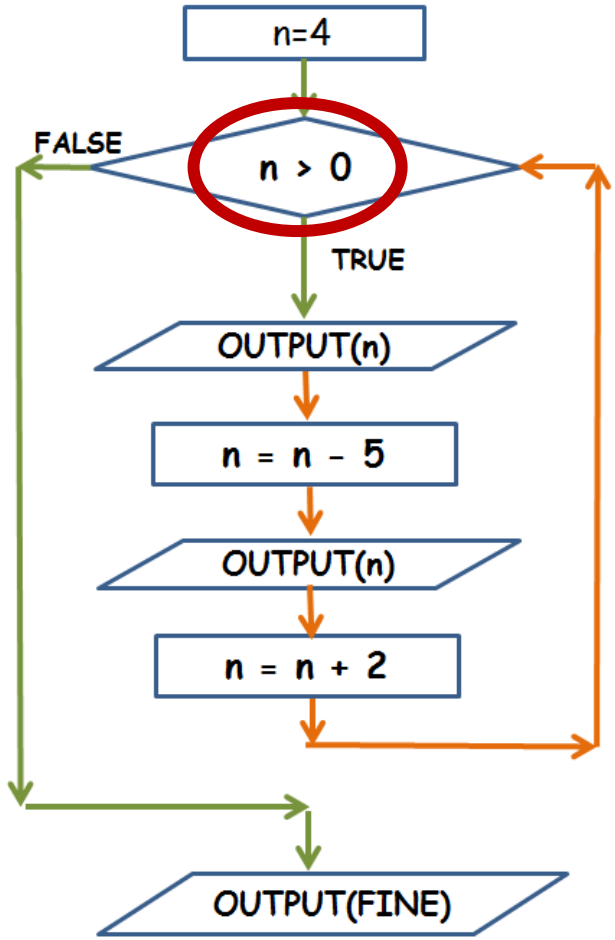
terminazione del ciclo

- 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")
  e' un'espressione, fatta, di solito, usando una o piu' "variabili di test"

- body

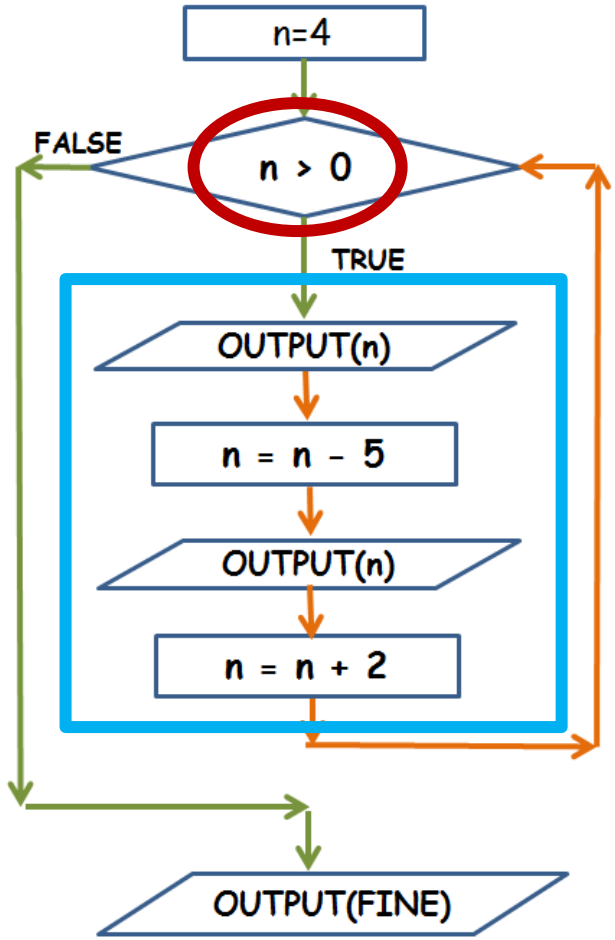
terminazione del ciclo

- 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") e' un'espressione, fatta, di solito, usando una o piu' "variabili di test"

- **body** di istruzioni da iterare

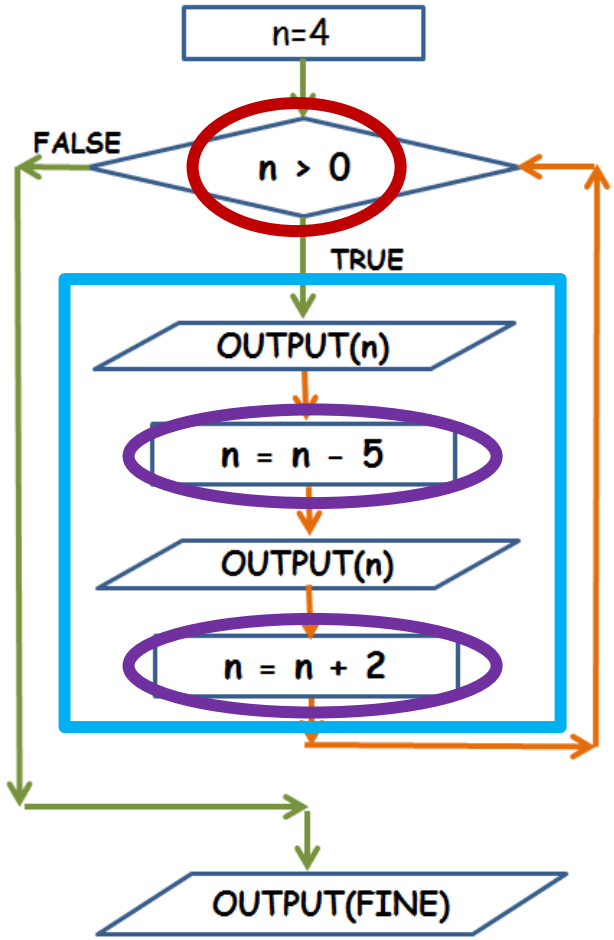
terminazione del ciclo

- 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

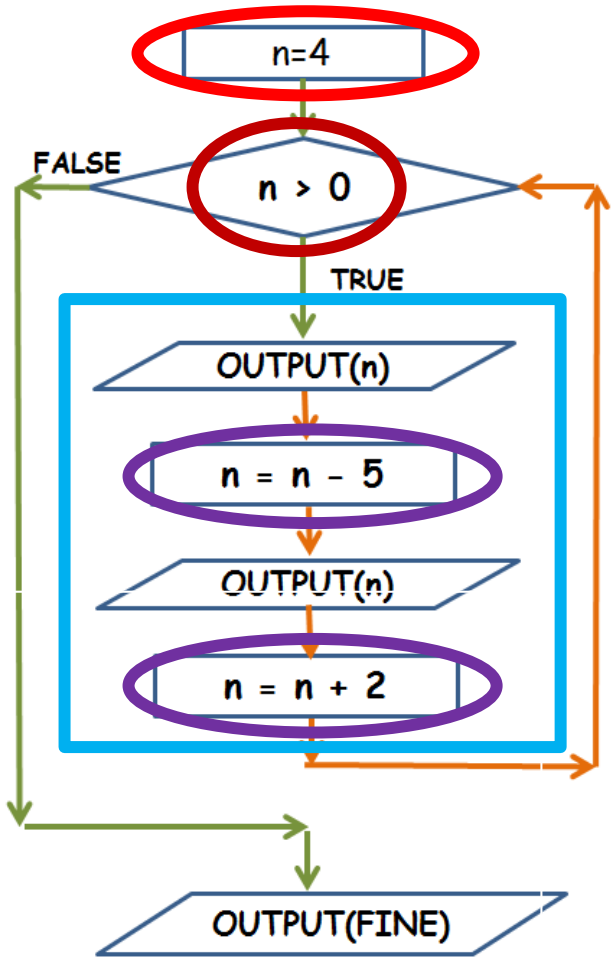
- almeno una **istruzione**, nel body, che modifichi variabili di test (in modo che la condizione possa cambiare). Inoltre questi cambiamenti devono permettere di *convergere* verso la **terminazione del ciclo**

- 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()**?

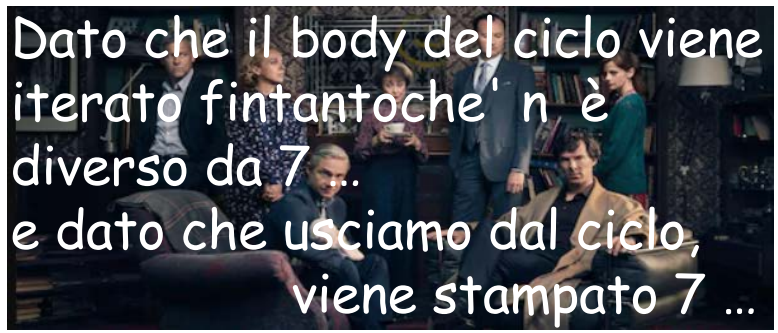
```
#include <stdio.h>

int main () {
  int n=21;

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

Istruzione iterativa: while

Ciclo solo parzialmente visibile:

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

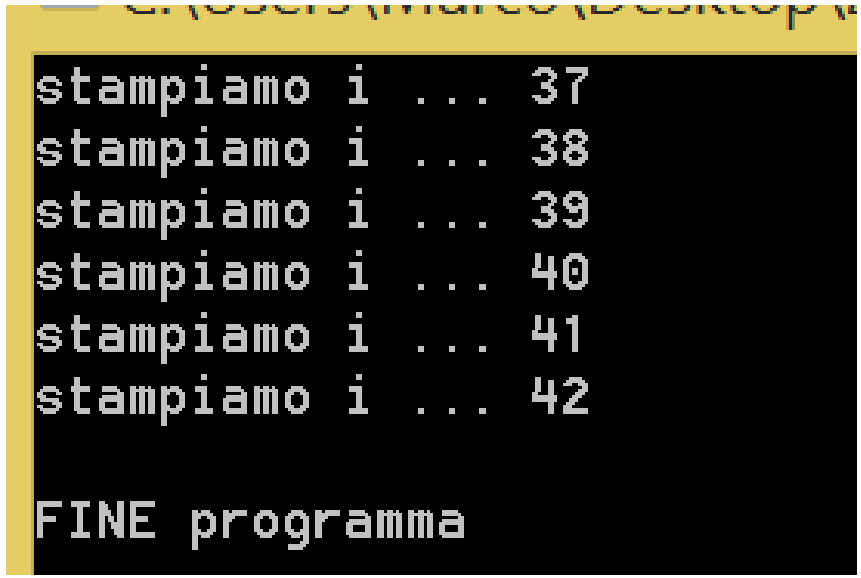
```
#include <stdio.h>

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;
}
```



Esercizio

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

```
#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 (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   ☹


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

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

FINE programma
```

Esercizio

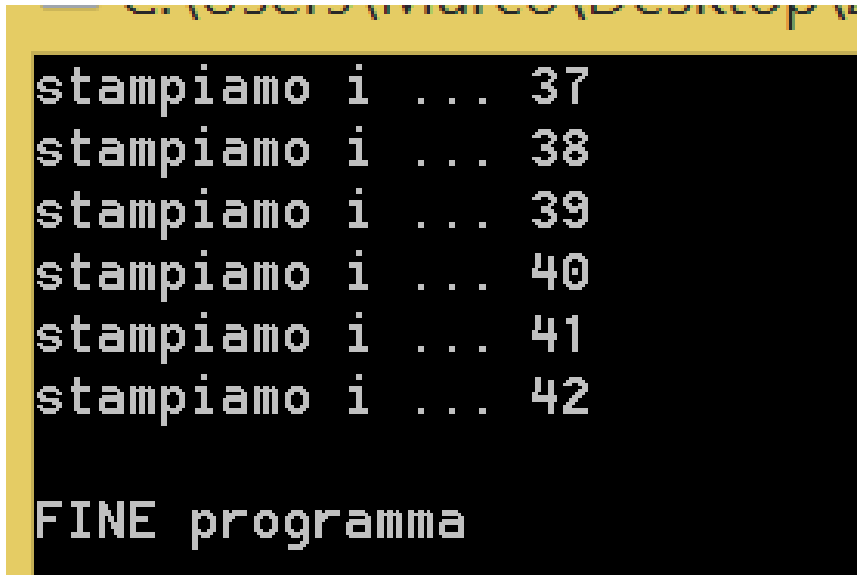
Programma che stampa i primi 42 numeri (usando una costante per il 42)

```
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;

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;
}
```

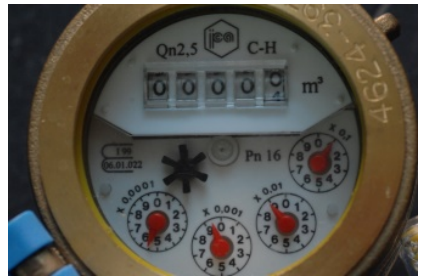
Esercizio

Programma che stampa i primi 42 numeri (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 e' una variabile *contatore*,
cioè una variabile che tiene traccia

del numero di iterazioni eseguite durante il ciclo

e/o, in generale

del numero di volte in cui si verifica un certo evento durante le iterazioni.

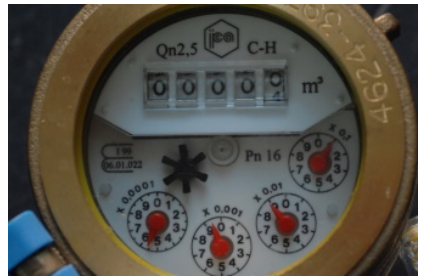
Una variabile contatore serve, in sostanza a contare qualcosa …

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 e' una variabile *contatore*,
cioè una variabile che tiene traccia

del numero di iterazioni eseguite durante il ciclo

e/o, 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 proprieta'

  (*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*, piu' tardi ...)

brevesercizio che serve dopo …

Algoritmo per un programma che legge un numero intero e stampa
"positivo!", oppure "negativo!", oppure "ma e' zero!"

brevesercizio che serve dopo …

Algoritmo per un programma che legge un numero intero e stampa
"positivo!", oppure "negativo!", oppure "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
```

brevesercizio che serve dopo … 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) ??

1) ??
2) ??

?) fine programma

brevesercizio … 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 dei 5 numeri
2) ??

?) fine programma

brevesercizio … 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 n4

7) fine programma

brevesercizio … 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 …"

☹ no … il passo 1 va iterato …

provare a scrivere un ciclo che iteri il passo 1 … cioe' lo ripeta … 5 volte in tutto

7) fine programma

brevesercizio … 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 …"

il passo 1 va iterato …

☹ ma come facciamo a scriverlo una volta sola (e poi ripeterlo) per n1, n2, …, n5

☹ sembra che il codice debba essere riscritto per ciascun numero !!

7) fine programma

brevesercizio … 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 …"

il passo 1 va iterato …
- lo scriviamo una volta sola, ma riferito ad una variabile generica n
- non usiamo le 5 variabili ma solo n
- n viene letto all'inizio del body … e quel nuovo numero viene processato … il tutto, 5 volte

7) fine programma

brevesercizio … 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).

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

brevesercizio … 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)

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;
}
```

e ora correggere il programma, che manca di una cosa essenziale

brevesercizio … 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)  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
```

brevesercizio … 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)

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
```

brevesercizio … 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)

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
```

(se si inizia da 0, le cinque iterazioni corrispondono ad i = 0,1,2,3,4)

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 ("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;
}
```

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 ("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;
}
```

- ☺ perche' 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?

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 e' intelligente …

brevesercizio … 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 …"
…
```

brevesercizio che serve dopo … quante volte vuoi

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

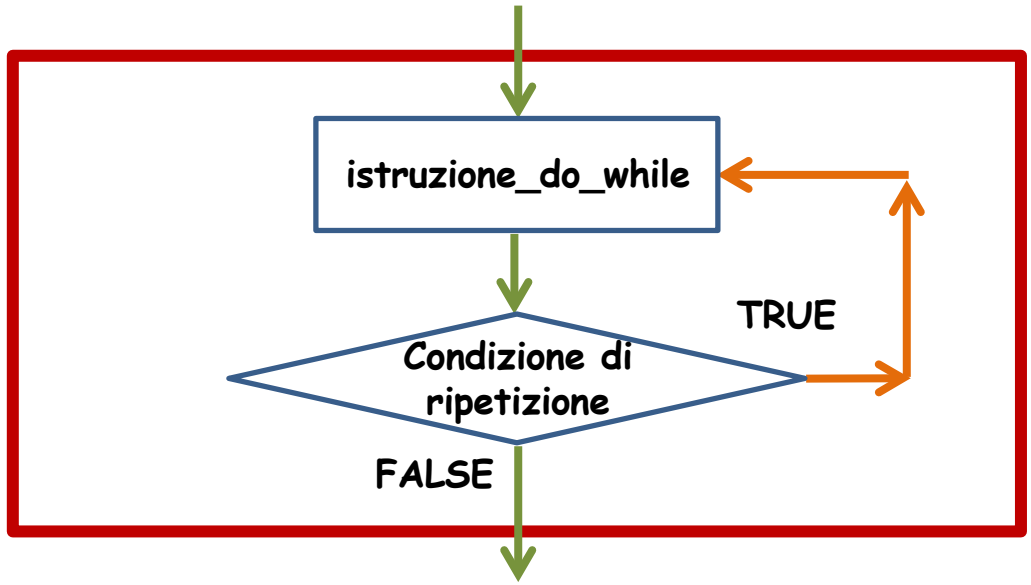
```
Algoritmo

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
```

Altri tipi di ciclo in C: do_while

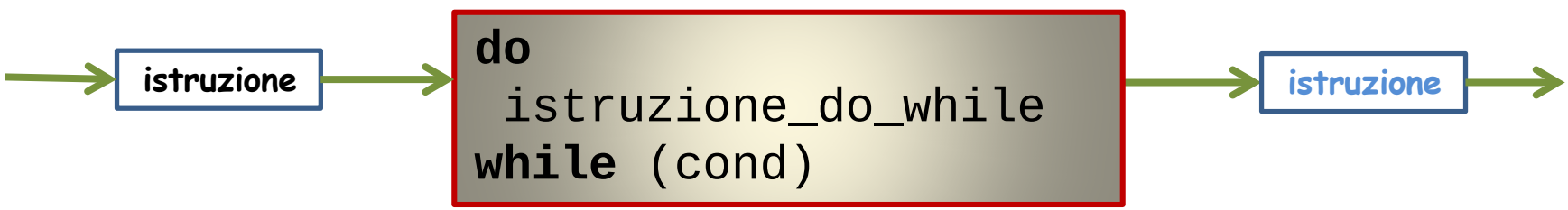
In C esistono due altri costrutti per ottenere un'istruzione iterativa.
Ci sono poche differenza ma ognuno ha aspetti interessanti …

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 fintantoche' 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 e' falsa, si esce dall'istruzione



ciclo do_while - esempio

Programma che legge una sequenza di numeri interi, interrotta da zero, dicendo per ciascuno se e' 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 - esen

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 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
```

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
   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 e' solo dopo la prima iterazione,
  - cioe' 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, perche' 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 e' una lotta. Sono costrutti equivalenti, cioe' si puo' fare tutto sia con l'uno che con l'altro.

```
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

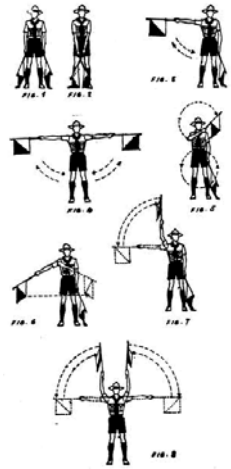
Non e' una lotta. Sono costrutti equivalenti, cioe' si puo' fare tutto sia con l'uno che con l'altro.

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

L'unica differenza tra do_while e while e' che il body del primo verra' 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 rpima itarzione.

```
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 e' accaduto (e la condizione di ripetizione assume un valore di verita' conseguente).

```
/* si leggono numeri e viene segnalato quando viene
inserito 61 */
…
 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 e' stato stampato 61, in quest'ultimo caso aggiungendo un commento appropriato

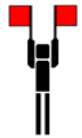
inserito_61 ==0 significa non abbiamo ancora visto 61

inserito_61 ==1 significa abbiamo visto 61

```
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)



2) dire se e' stato incontrato 61 e poi fine programm
```

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 e' stato stampato 61, in quest'ultimo caso aggiungendo un commento appropriato

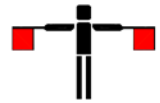
tsk tsk … dov'e'?

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 e' 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 e' 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

```
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 e' 61 assegnare flag ad 1
    while (n != 0      AND     inserito_61  != 1)

3) se stampato 61 (cioe' se inserito_61 e' 1), dirlo
4) fine programma
```

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=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, cioe' hanno sempre il medesimo valore di verita'

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

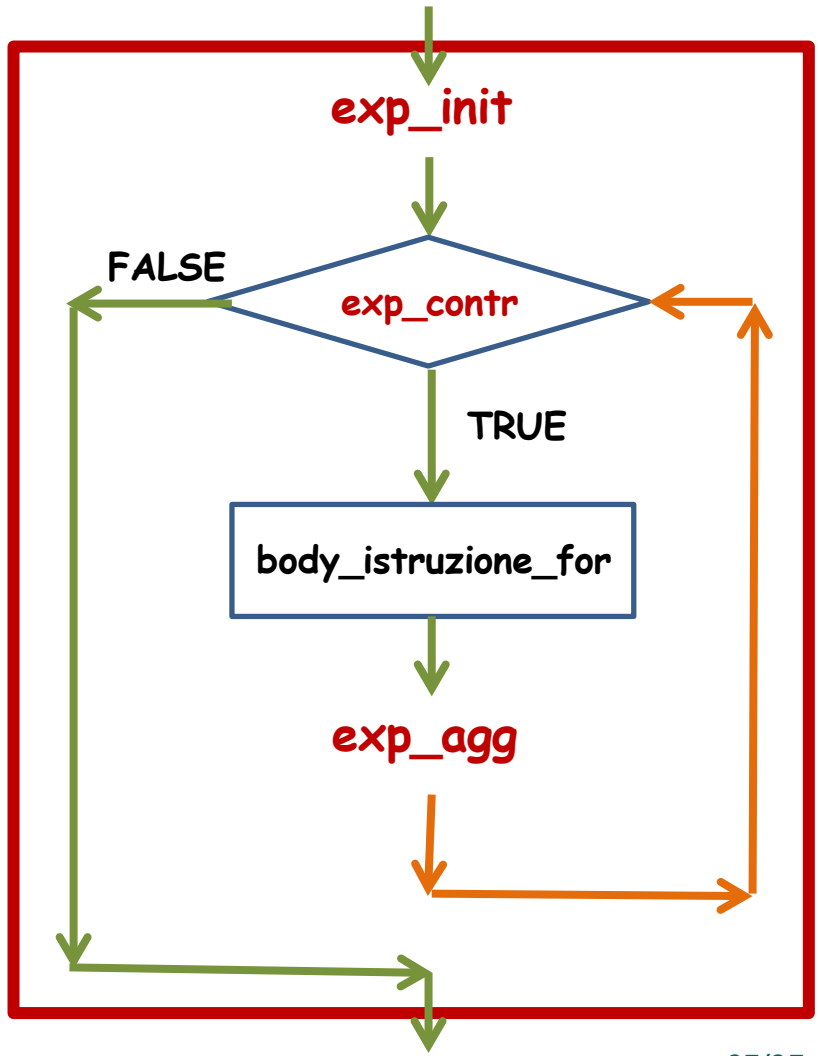
☺ risolvi e poi vedi in fondo

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



Esercizio

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;
}
```

Esercizio

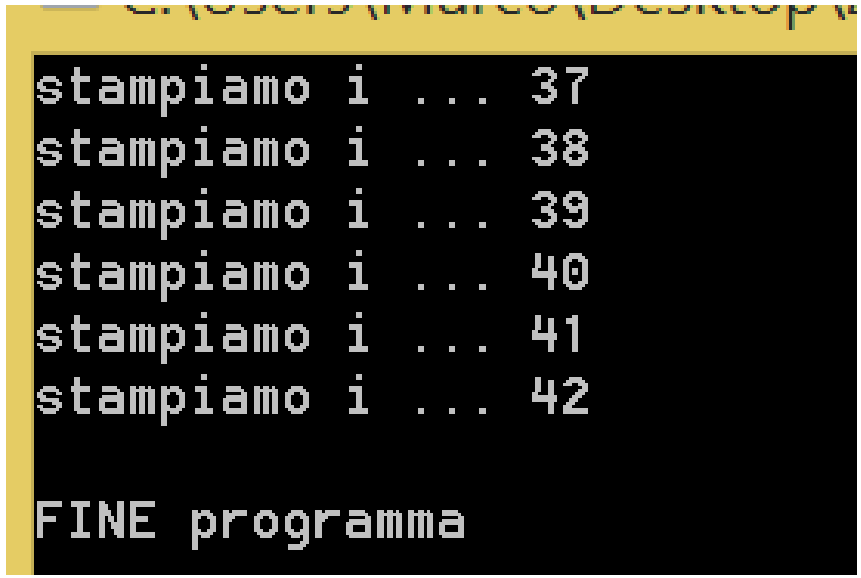
Programma che stampa i primi 42 numeri (usando una costante per il 42)

```
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;
}
```

Esercizio

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




```
#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;
}
```

☺

Esercizio

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;
}
```

☺

Esercizio

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 2

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;
}
```

Esercizio 2

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
```

```c
#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

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

☺ in fondo

Esercizio 3

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 3

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

☺ in fondo

Esercizio 4

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 4

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 … crescente

Esercizio 4

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 verra' stampato … se lo verra')**
  - **incrementa i di 1**

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

Esercizio

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?
modificando la condizione di ripetizione in modo che sia i<= QUANTINUM,
e' tutto come prima ...

Istruzione iterativa: for VS while

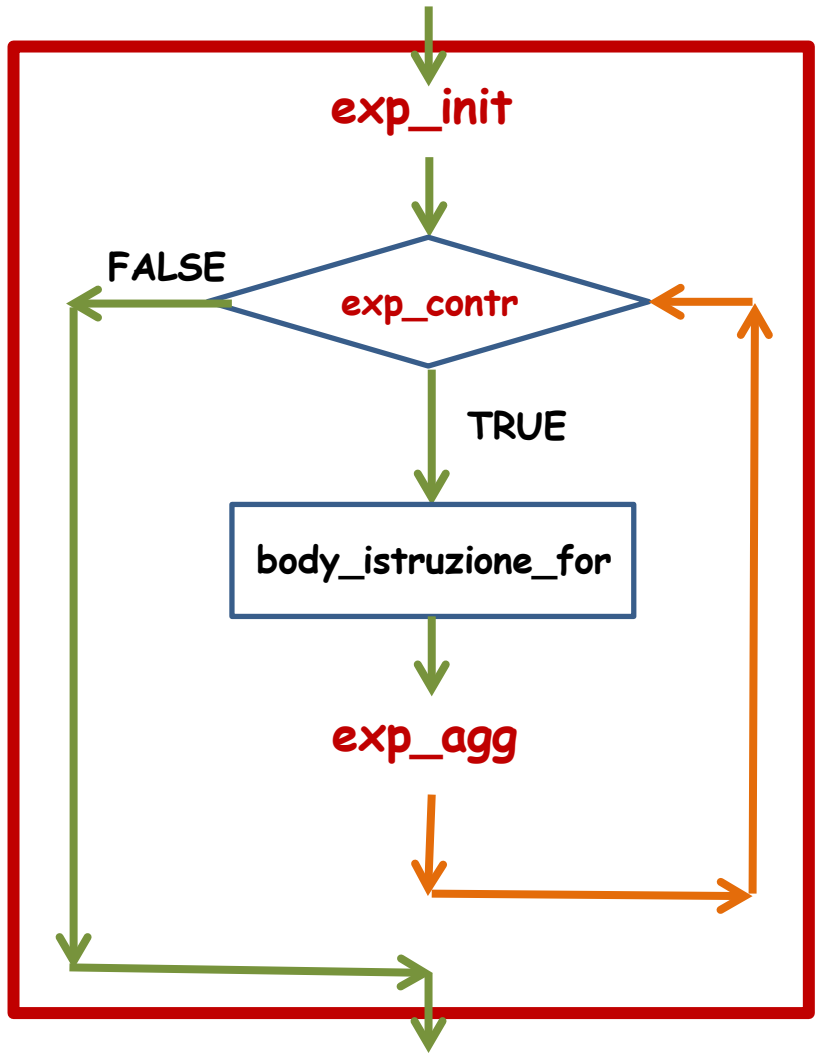
Istruzione for

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

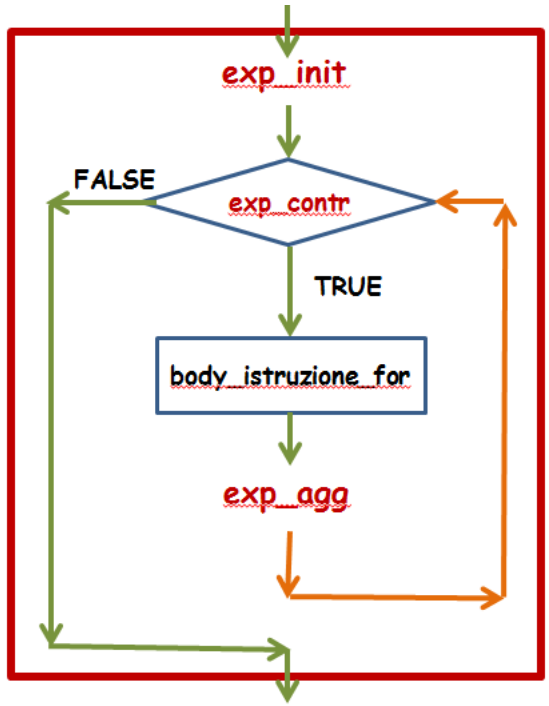
La **exp_init** viene valutata una sola volta, all'inizio, cioe' prima della prima esecuzione eventuale del body

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

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



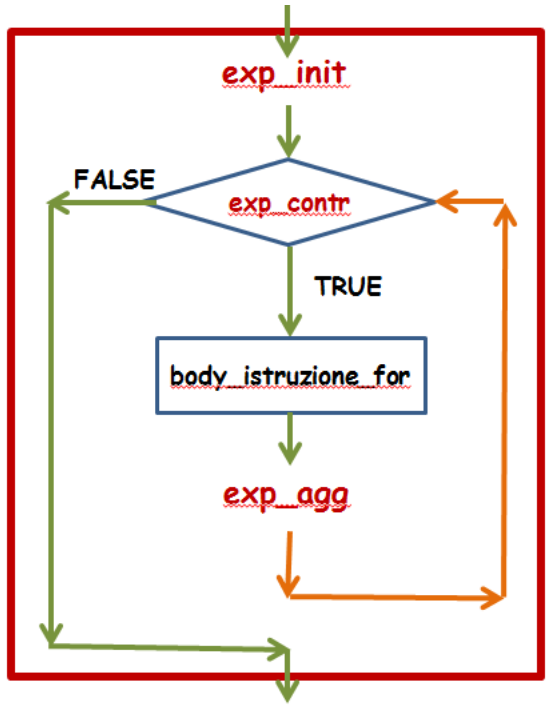
for VS while



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

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

for VS while



Evidentemente sono equivalenti anche loro: si puo' 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);
```

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

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

☺ prova a sviluppare meglio questo algoritmo … spiega meglio cosa avvine ad ogni iterazione in termini di come vengono usati i e j

comunque, nelle prossime slide comunque andiamo per gradi ….

Esercizio – dieci per dieci – annidamento

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

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;
}
```

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 …

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

Progrc

nteri e ne

stampa i reciproci; se viene introdotto zero si arrabbia

Algoritmo

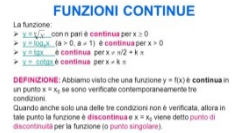
0) str. dati … n, i; costante QUANTE_VOLTE
1) per i che va da 0 a QUANTE_VOLTE -1
   1.1) chiedere e leggere n
   1.2) se n==0
        1.2.1) arrabbiarsi e interrompere il ciclo
   altrimenti
        1.2.2) stampare 1/n
2) fine programma

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;   }
```

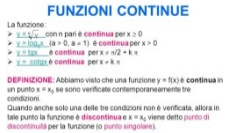
continue



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



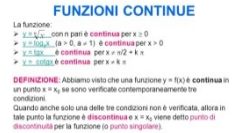
E' la ke…one
della at…
A volte…a meno:
basta u…
Esempi…
Prograr…i interi e ne
stampa…prosegue con
il pross…

```
--- 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;        }
```

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;
}
```

Esercizio 2

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 3

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;
}
```