

Informatica

Algoritmi e programmazione

Thomas Barbera
thomas.barbera@unimib.it

Riadattamento delle slides di Marco Viviani e Simone Zini

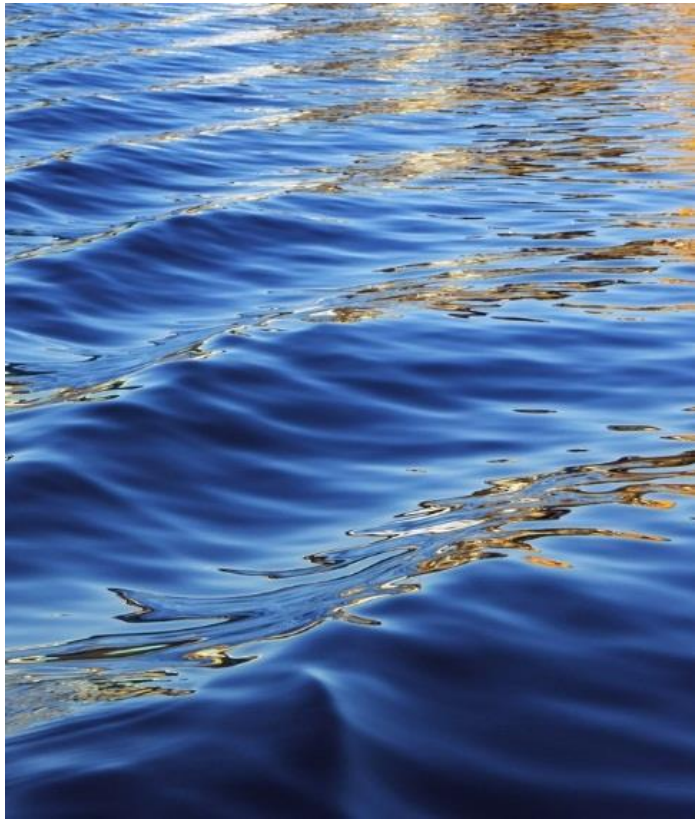






Seconda parte:
I linguaggi di programmazione

(I linguaggi di programmazione strutturati)



Linguaggio di programmazione

- **Un linguaggio di programmazione è un linguaggio formale** che comprende un insieme di istruzioni che producono vari tipi di output.

- I linguaggi di programmazione sono usati nella programmazione dei computer per **implementare algoritmi**.

- La **sintassi** di un linguaggio di programmazione è l'insieme di regole che definisce le combinazioni di simboli che sono considerate come affermazioni o espressioni correttamente strutturate in quel linguaggio.

Paradigma di programmazione

- Un **paradigma di programmazione** è uno stile fondamentale di programmazione, ovvero un insieme di strumenti concettuali forniti da un linguaggio di programmazione per la stesura del codice sorgente di un programma, definendo dunque il modo in cui il programmatore concepisce e percepisce il programma stesso.

- **Programmazione strutturata**: paradigma di programmazione che mira a migliorare la chiarezza, la qualità e il tempo di sviluppo di un programma per computer facendo un uso estensivo dei costrutti di **flusso di controllo strutturato** di selezione (if/then/else) e ripetizione (while e for), strutture a blocchi e sottoprogrammi.

Paradigmi di programmazione

- **Programmazione Procedurale** (o **imperativa**)
  - Un programma è composto da istruzioni che realizzano trasformazioni di stato. Uno stato è identificato da tutti i valori di un certo insieme di variabili ad un certo stadio dell'esecuzione
- **Programmazione Funzionale**
  - Un programma è un'espressione che viene valutata per ottenere un risultato
- **Programmazione Logica**
  - Un programma è un insieme di fatti e regole, la sua esecuzione equivale alla realizzazione di una dimostrazione
- **Programmazione orientata agli Oggetti**
  - Un programma è un insieme di oggetti (astrazioni della realtà) che comunicano fra loro tramite scambio di messaggi

Linguaggi strutturati

- Il nostro studio e l'approccio che adotteremo è focalizzato sull'uso di un linguaggio strutturato (MATLAB)

- Questi linguaggi sono basati su sei istruzioni fondamentali:
  1. **Leggi**: legge in input dall'esterno un valore e lo memorizza in una variabile (locazione di memoria)
  2. **Assegna**: assegna ad una variabile (ad una locazione di memoria) il valore di una espressione
  3. **Scrivi**: scrive in output il valore di una espressione o di una variabile (locazione di memoria)
  4. **Se … allora … altrimenti …**: modifica il 'flusso' del programma sulla base del valore di una espressione logica
  5. **Ripeti finchè …** modifica il 'flusso' del programma effettuando iterazioni (ripetizioni)
  6. **Vai al passo …**: modifica il 'flusso' del programma incondizionatamente
  7. **Fermati**: termina l'esecuzione del programma

Le costanti

- Una **costante** è un **dato non modificabile** situato in una porzione di memoria (una o più locazioni di memoria) destinata a contenere un **valore**, che **non può essere** appunto **modificato** nel corso dell'esecuzione di un programma

- Una costante può essere caratterizzata da un **nome** (inteso solitamente come una sequenza di caratteri e cifre)

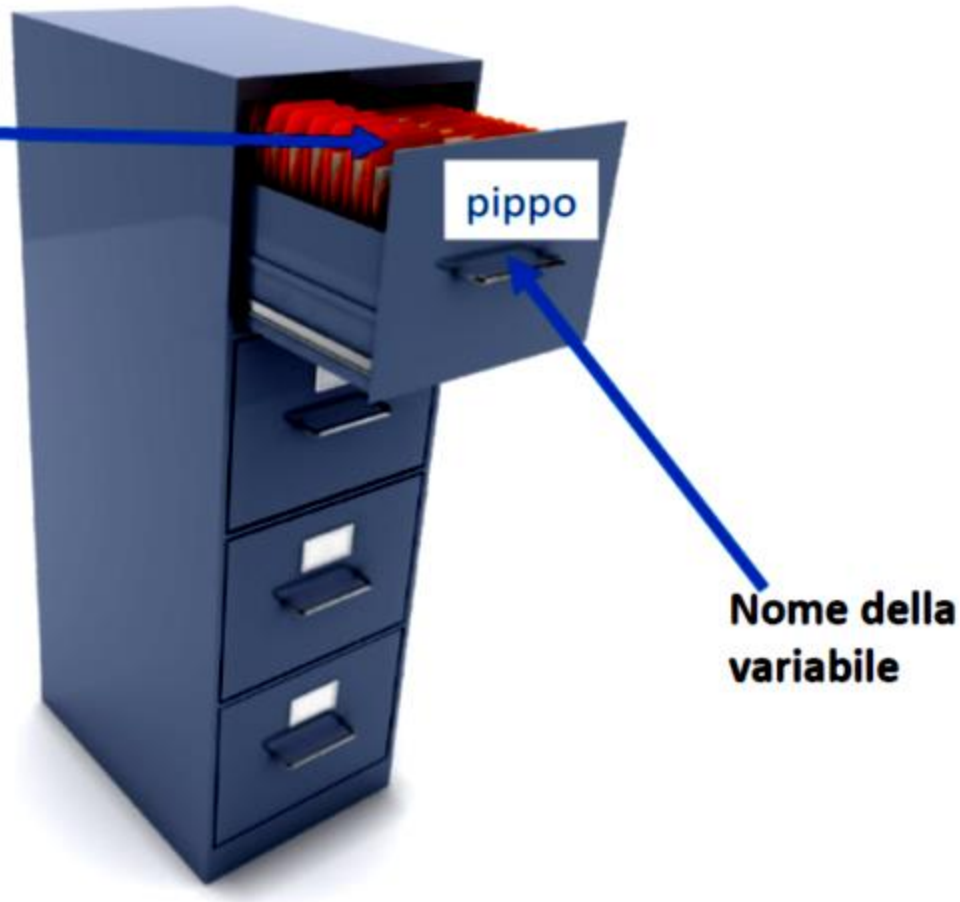
- Esempio:
  - 5 + 2
  - 5 e 2 sono costanti di tipo numerico
  - print(‘Hello world!’)
  - La stringa ‘Hello world!’ è una costante di tipo alfanumerico

Le variabili

- Una **variabile**, in informatica, è un «contenitore» di dati **situato in una porzione di memoria** (una o più locazioni di memoria) destinata a contenere **valori**, suscettibili di **modifica** nel corso dell'esecuzione di un programma

- Una variabile è caratterizzata da un **nome** (inteso solitamente come una sequenza di caratteri e cifre)

Le variabili



Esempio:
pippo + 2
Sommo la costante 2 al valore che assume la variabile a in quel momento

Istruzioni di assegnamento

- L'**assegnamento** è l'operazione che permette di impostare o reimpostare il valore di una variabile

- A sinistra dell'operatore di assegnamento deve esserci il **nome di una e una sola variabile**

- A destra dell'operatore di assegnamento **può esserci una costante**
  - E.g.,

$$x = 6$$

Istruzioni di assegnamento

- A destra dell'operatore di assegnamento **può esserci una variabile**
  - E.g.,

$$x = a$$

- o anche un'espressione complessa
  - E.g.,

$$x = 6 + 1$$

Tipi di dato

- Le variabili possono contenere un **tipo di dato**

- Le operazioni che svogliamo devono essere **compatibili** con il tipo di dato delle variabili coinvolte
  - Esempio: non avrebbe senso applicare l'operazione di divisione $\frac{a}{b}$ se $a =$ 'ciao' e $b =$ 'mamma'

Tipi di dato

Le variabili devono avere dunque un **tipo** che rappresenta:

- I **valori** ammissibili
- Le **operazioni** che possono essere compiute sui suoi valori
- Esempio:
  - una variabile di tipo numerico può assumere solo valori numerici
  - può apparire come operando in una espressione matematica

Tipi di dato

- I tipi di dati possono essere classificati secondo la struttura in **tipi atomici** (o **primitivi**) e **tipi derivati**.

- I tipi primitivi sono i tipi semplici che non possono essere decomposti. Esempi:
  - Boolean, char, int, float, double,

- I tipi derivati si ottengono dai tipi atomici mediante opportuni **operatori** forniti dal linguaggio
  - essi includono i **tipi strutturati** che contengono più informazioni in un unico elemento: **vettori**, **matrici**, **stringhe**

Tipi di dato elementari

- **Boolean**: Il tipo booleano ha due soli valori: *true* ("vero") e *false* ("falso"). Vengono utilizzati in modo speciale nelle espressioni condizionali per controllare il flusso di esecuzione e possono essere manipolati AND, OR, NOT

- **Caratteri**:
  - ***char***, es. ‘a’, ‘@’, ‘3’, …

- **Numeri**:
  - **Numeri interi**
    - ***int***
  - **Numeri in virgola mobile**
    - ***float*** (numeri con precisione singola)
    - ***double*** (numeri con precisione doppia)

Parentesi su precisione singola e doppia

- Come vengono rappresentati in macchina i dati in virgola mobile?
- La codifica più diffusa è la IEEE754 tale codifica prevede di rappresentare i dati utilizzando la **notazione scentifica**
- singola precisione (***float***):

  1: bit per il segno
  8: per l'esponente
  23: per la mantissa



- doppia precisione (***double***):

  1: bit per il segno
  11: per l'esponente
  52: per la mantissa



Tipi di dato derivati

- **Vettori**:
  - *Array:* sequenza finita di elementi appartenenti a un determinato tipo, indicizzata mediante un numero intero

```
int v[10];
v[0] = 0;
v[1] = 1;
v[2] = 1;
v[3] = 2;
v[4] = 3;
…
v[9] = …;
```



Tipi di dato derivati

- **Matrici**: Le matrici sono insiemi di variabili, in cui ogni variabile è identificata all'interno della matrice da indici interi

- Si tratta di una **estensione dei vettori**: mentre un vettore v è un gruppo di variabili $v[0], v[1], v[2], \dots$, una matrice è sempre composta da più variabili, ma questa volta le variabili sono $v[0][0], v[0][1], v[0][2], \dots$, $v[1][0], v[1][1], \dots$

- In altre parole, mentre nei vettori un solo valore intero identifica una variabile, nelle matrici **le variabili sono identificate da due valori interi**

Tipi di dato derivati

- Il modo più comodo di visualizzare una matrice m è quello di disporre le variabili che la compongono in una griglia, come segue:

$m[0][0]$	$m[0][1]$	$m[0][2]$	…
$m[1][0]$	$m[1][1]$	$m[1][2]$	…
$m[2][0]$	$m[2][1]$	$m[2][2]$	…
…	…	…	…

- Quindi, data una matrice m e due numeri interi i e j, questi identificano la variabile $m[i][j]$ nella matrice

Tipi di dato derivati

```
char matrice[10][10];

matrice[0][0] = 'a';
matrice[0][1] = 'ç';
…
matrice[2][1] = '*';
matrice[2][2] = 'f';
…
matrice[9][9] = '§';
```

Tipi di dato derivati

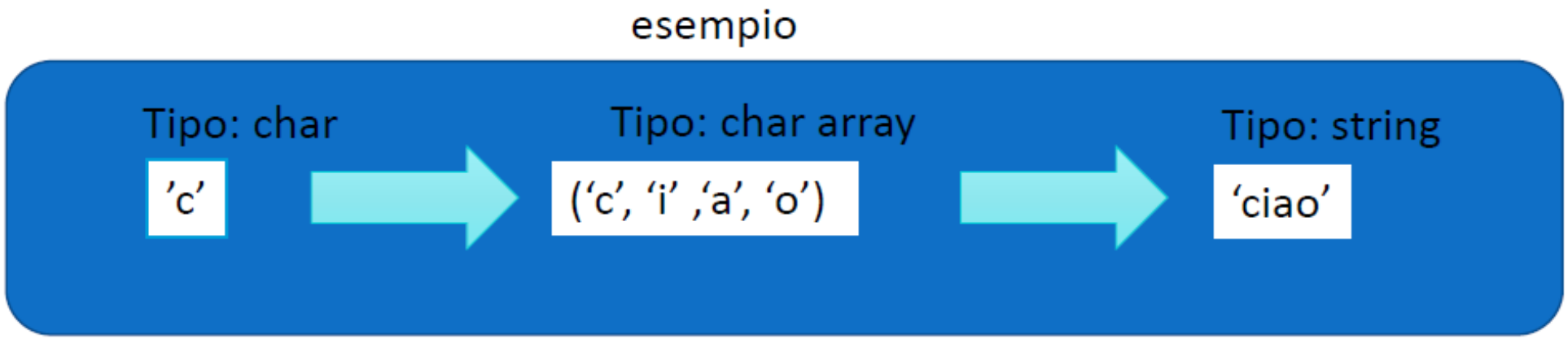
- Una **stringa** in informatica è una sequenza di caratteri. Sequenze di altri tipi di dato vengono chiamati vettori o, in inglese, *array*.

- La gran parte delle implementazioni delle stringhe somigliano ad *array* i cui elementi contengono i codici corrispondenti ai caratteri nel corrispondente set di caratteri

F	R	A	N	K	NUL	k	e	f	w
46	52	41	4E	4B	00	6B	65	66	77

Tipi di dato derivati

- Differenza tra vettore di caratteri e stringa



Dichiarazione e assegnamento

- Le istruzioni di dichiarazione descrivono le **costanti** e le **variabili** utilizzati nell'algoritmo, definendone
  - **tipo**
  - **struttura**

- Esempio:

```
int x, y;          // Dichiarazione

x = 1; y = 2;      // Assegnamento
```

Dichiarazione e assegnamento

- Sono previste **dalla maggior parte** dei linguaggi di programmazione

- Linguaggi ad altissimo livello come MATLAB e Python non richiedono all'utente di dichiarare il tipo di dato di una variabile perché sono tipicamente in grado di distinguerle da solo
  - E.g., se assegno a x il valore 3 → Matlab capirà che si tratta di un **numero intero**
  - E.g., se assegno a x il valore 3.01 → Matlab capirà che si tratta di un **float**

Testo e numeri

Attenzione!

- Nemmeno MATLAB o Python sono in grado di capire quando vogliamo rappresentare testo invece che valori numerici

- Perciò è necessario l'utilizzo degli **apici** (o doppi apici) per i caratteri testuali

```
x = 3;
```
MATLAB interpreterà il 3 come un numero

```
x = '3';
```
MATLAB capirà che si tratta di testo

Inizializzazione

- Il primo assegnamento della variabile è detto **inizializzazione**
- Molti linguaggi richiedono o preferiscono che la variabile una volta dichiarata venga anche inizializzata **a un valore arbitrario**

Istruzioni I/O

- Permettono lo **scambio** di dati tra l'utente e l'esecutore.
- Le istruzioni di **INPUT** permettono all'utente di comunicare i dati iniziali su cui l'esecutore deve effettuare l'elaborazione (variabili di input)
- Le istruzioni di **OUTPUT** permettono all'esecutore di comunicare i risultati dell'elaborazione all'utente (variabili di output)

Istruzioni I/O

- La **sintassi** delle istruzioni dipende dalle caratteristiche del linguaggio di programmazione

- LEGGI n acquisisci un valore dall'esterno (il dato fornito dall'utente) e assegnalo alla variabile n
  - Noi useremo `read n;`

- SCRIVI n significa: «prendi il valore della variabile n e comunicalo all'esterno (all'utente)»
  - Noi useremo `write n;`

Operazioni

- Operatori aritmetici: +, -, /, ...
  - agiscono sugli operandi (variabili o costanti)
  - producono un valore numerico

- Operatori di confronto: >, =, ...
  - agiscono sugli operandi (variabili o costanti)
  - producono un **valore logico** (**vero** o **falso**)

- Espressioni: 5+cos(y), ...
  - composizione di operatori, funzioni, variabili e costanti
  - ad essa è associato un valore

Semplice esempio

```
// Dichiarazione e inizializzazione variabili
int x = 2;
int y = 3;
int z = 0;

// Corpo del codice
z = x + y;
write z;

// Output = ?
```

Le funzioni

- I linguaggi di alto livello permettono di definire istruzioni non primitive per risolvere parti specifiche di un problema: i sottoprogrammi (**funzioni** e **procedure**).

- Esempi
  - `mean(x,y,z)`
  - Istruzioni I/O

L’ordine delle istruzioni

1. Prendi una padella
2. Metti la padella sul fuoco
3. Accendi il fuoco sotto la padella
4. Attendi temperatura corretta
5. Prendi uovo
6. Rompi uovo
7. Metti uovo in padella → se l’uovo si rompe?
8. Aggiungi sale
9. Attendi 5’ prendi un piatto
10. Spegni il fuoco
11. Versa il contenuto della padella nel piatto



Torna
all’istruzione 5

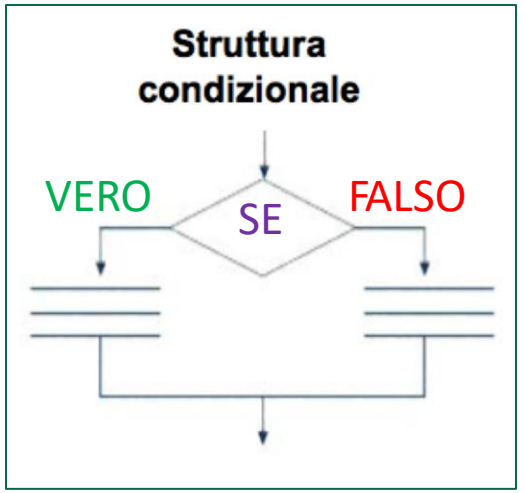
Strutture di controllo

- **Sequenza**: le operazioni sono eseguite una in seguito l'una all'altra

- **Selezione**: blocchi di operazioni sono eseguite in alternativa
  - Esempio: **istruzioni condizionali** «SE»
  - Se la condizione A è vera fai queste operazioni, se è falsa fai altre operazioni

- **Iterazione**: blocchi di operazioni sono eseguite diverse volte
  - Esempio: **istruzioni iterative** o «cicli»
  - Ripeti queste operazioni finché la condizione B è vera

Strutture condizionali



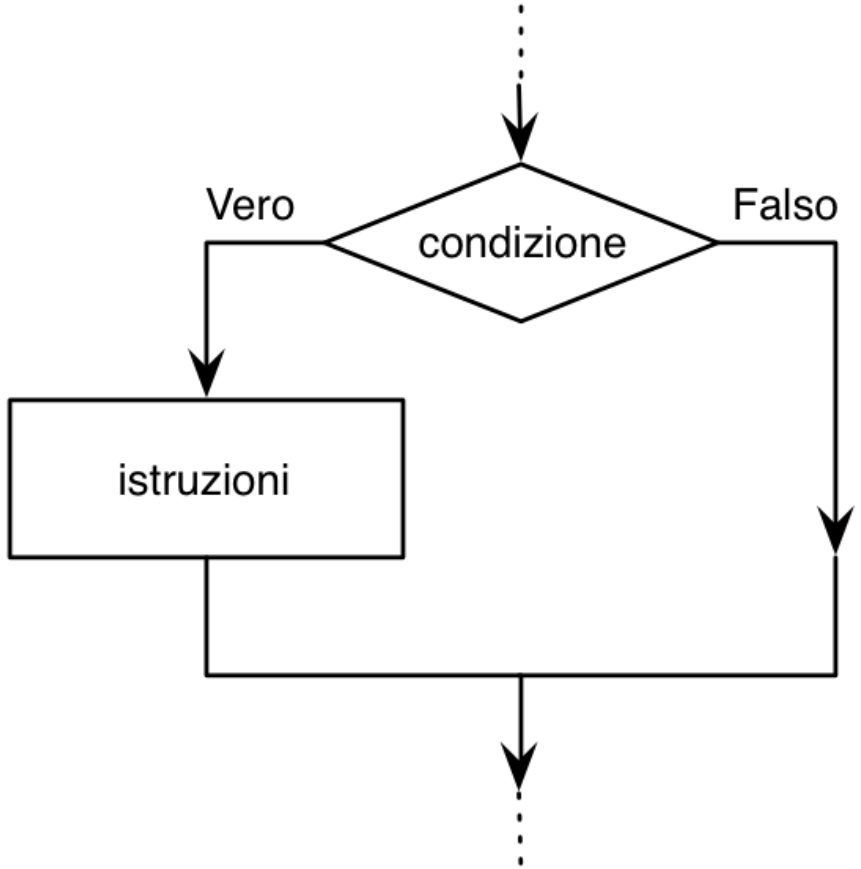
- Verificano il valore di verità in un'espressione:
  - Se è vera si compiono certe istruzioni
  - Se è falsa si compiono *altre* istruzioni
- Rappresentabile con la funzione **IF** (SE)
- Esempio: dato un numero introdotto dall'utente
  - Se il numero è maggiore o uguale a 0 (valore VERO) → Scrivere «Numero Positivo»
  - Altrimenti (valore FALSO) → Scrivere «Numero Negativo»

Struttura condizionale semplice «if-then»

- Rappresenta la situazione:

1. se condizione allora
2. istruzioni
3. fine-condizione

```
if (condizione) then
      lista di comandi;
endif
```

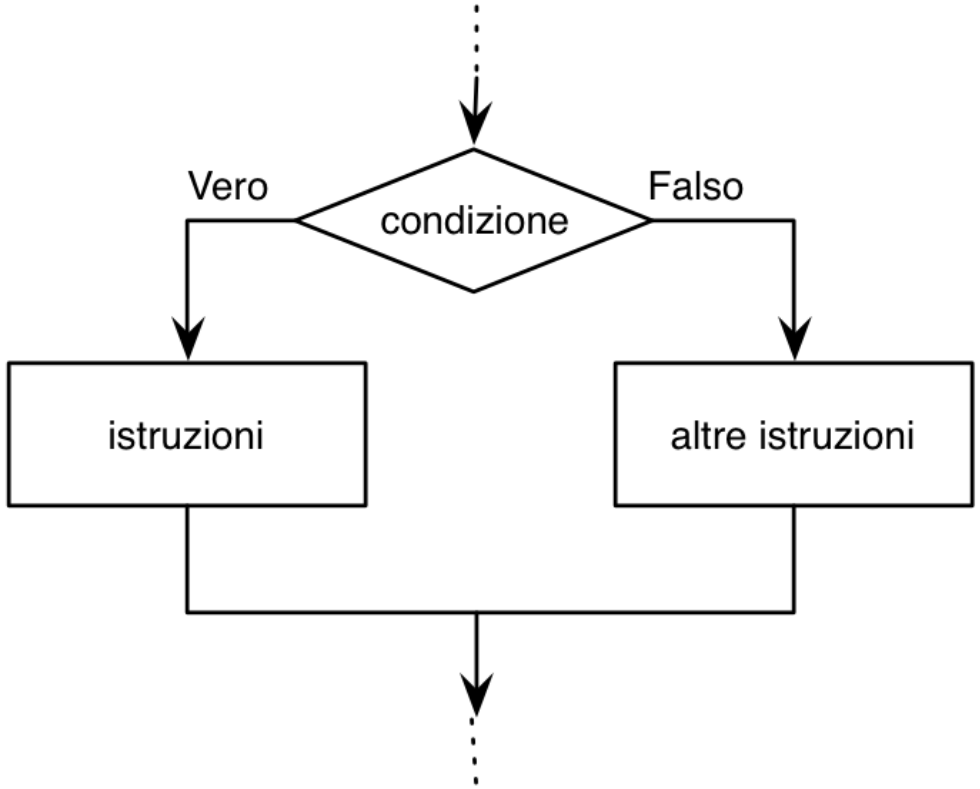


Struttura condizionale «if-then-else»

- Rappresenta la situazione:

1. se condizione allora
2. istruzioni
3. altrimenti
4. istruzioni
5. fine-condizione

```
if (condizione) then
        lista di comandi;
else
        lista di comandi;
endif
```



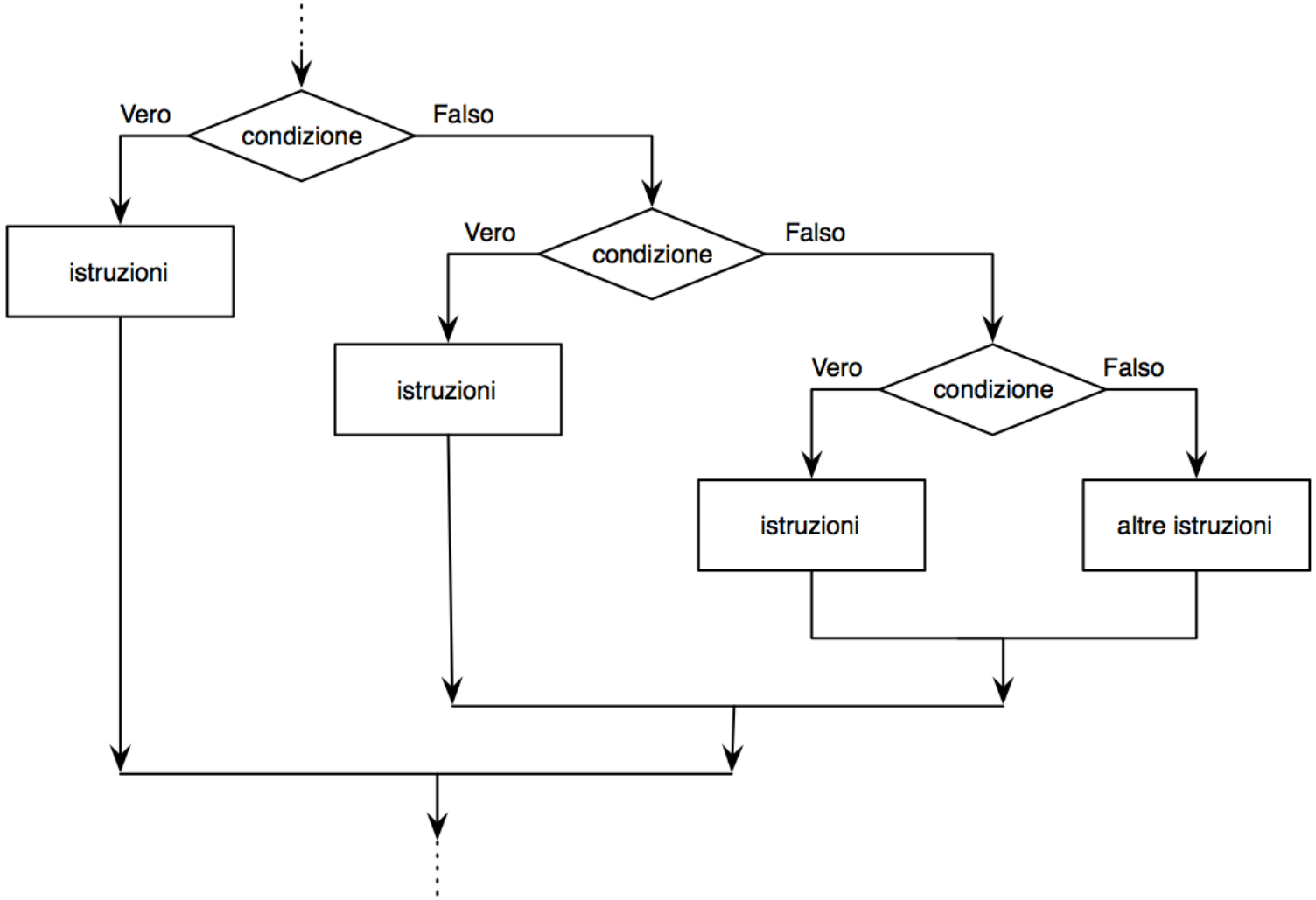
Struttura condizionale «a cascata»

- È possibile codificare una struttura condizionale più articolata, con la verifica di condizioni “a cascata”, secondo il seguente schema:

1. se *condizione*$_1$ allora
2. *istruzioni*$_1$
3. altrimenti se *condizione*$_2$ allora
4. ...
5. altrimenti se *condizione*$_n$ allora
6. *istruzioni*$_n$
7. altrimenti
8. *istruzioni*$_{n+1}$
9. fine-condizione

```
if condizione 1 then
      lista di comandi;
else if condizione 2 then
…
else if condizione n then
      lista di comandi;
else
      lista di comandi;
endif
```

Struttura condizionale «a cascata»



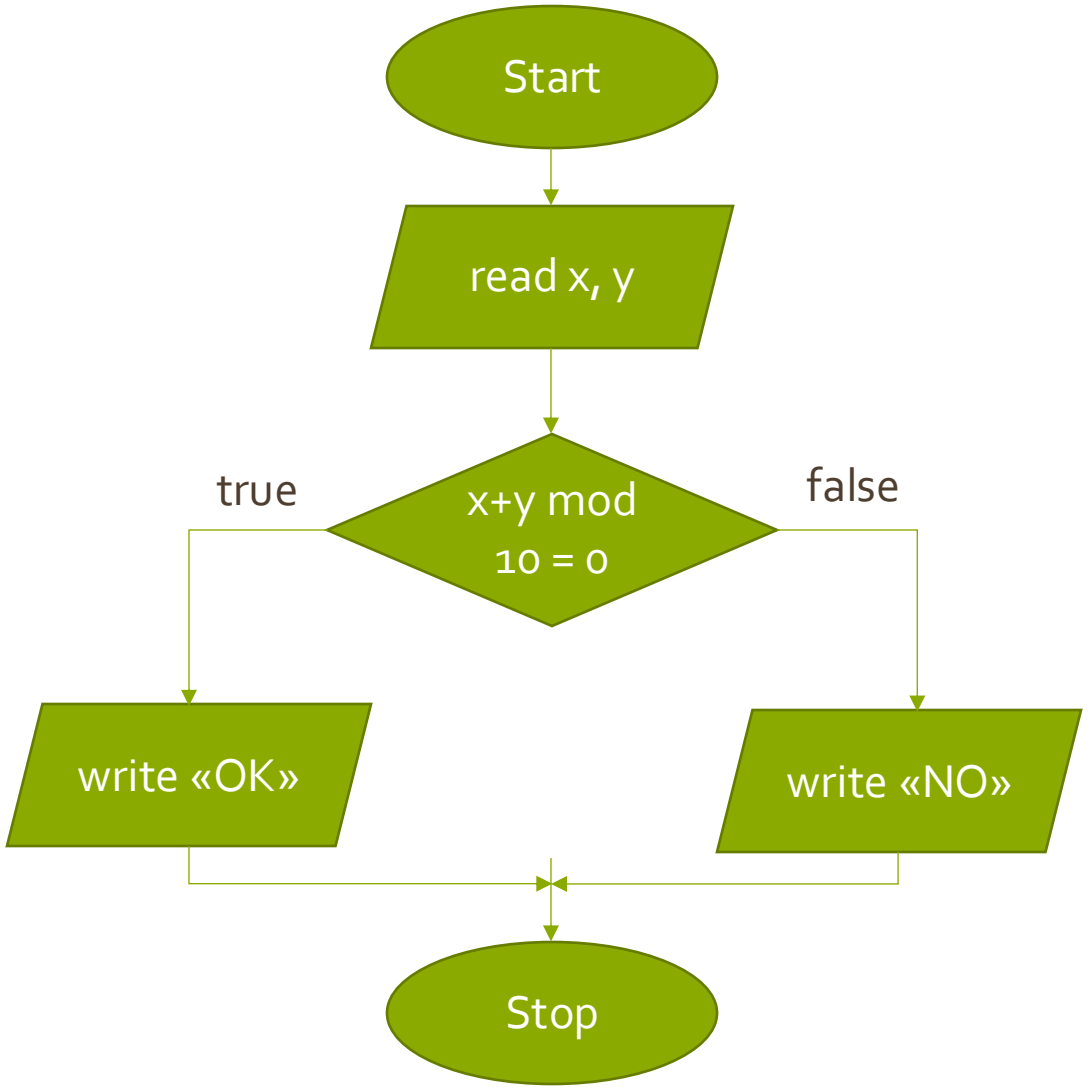
Esercizio

- Descrivere il diagramma di flusso e tramite la struttura if-then-else la seguente situazione.

- Leggere due numeri
- Sommarli
- Se la somma è multiplo di 10, stampare «OK»
- Altrimenti stampare «NO»

Esercizio

```
read x;
read y;
if (x+y mod 10 == 0) then
      write 'OK';
else
      write 'NO';
endif
```



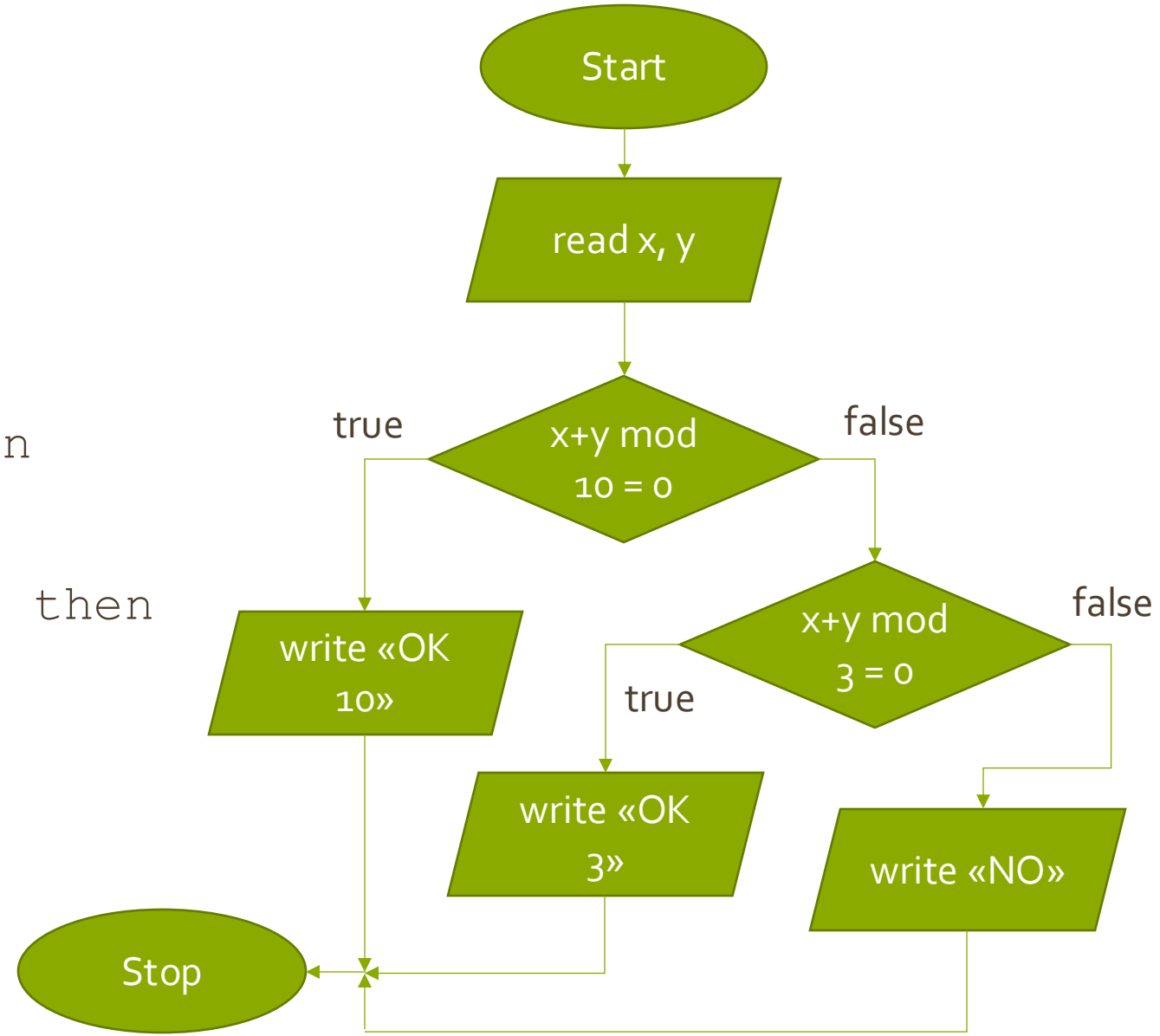
Esercizio

- Descrivere il diagramma di flusso e tramite la struttura if-then-else la seguente situazione.

- Leggere due numeri
- Sommarli
- Se la somma è multiplo di 10, stampare «OK 10»
- Altrimenti
- Se la somma è multiplo di 3 stampare «OK 3»
- Altrimenti stampare «NO»

Esercizio

```
read x;
read y;
if (x+y mod 10 == 0) then
      write 'OK 10';
else if (x+y mod 3 == 0) then
      write 'OK 3';
else
      write 'NO';
endif
```



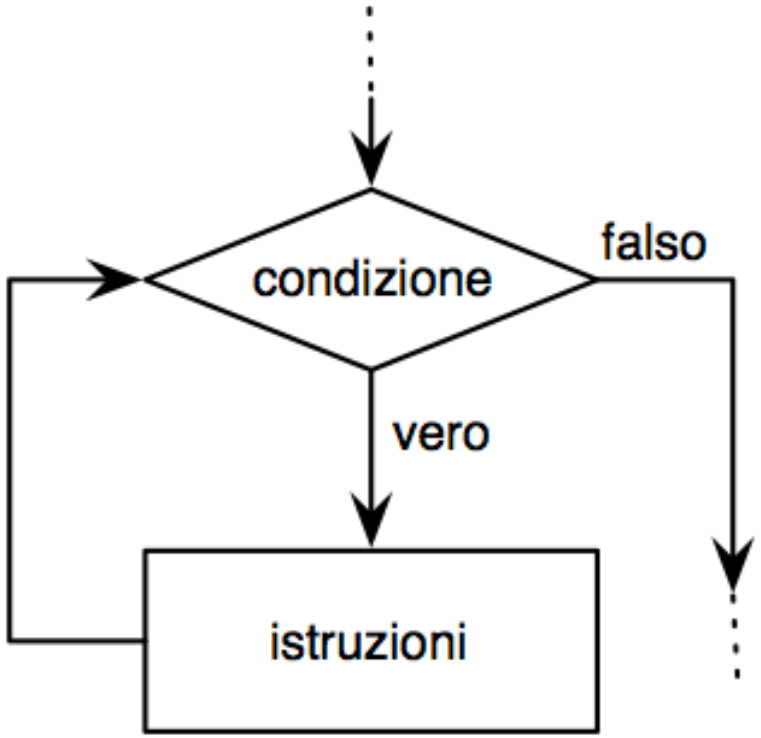
Struttura iterativa: i cicli

- Il **ciclo** o ***loop*** è uno schema di flusso che descrive problemi in cui un gruppo di operazioni deve essere **ripetuto più volte**
  - Esempio: Stampare 10 volte il contenuto di una variabile

- La posizione del corpo del ciclo **rispetto alla condizione di terminazione** caratterizza vari tipi di strutture iterative
  - l'iterazione con controllo **in testa**
  - l'iterazione con controllo **in coda**

- Esistono **tre istruzioni principali** per la realizzazione di iterazioni (cicli):
  - ***while***
  - ***repeat-until***
  - ***for***

Controllo in testa



Il ciclo termina quando la valutazione della condizione restituisce il valore *falso*

Controllo in testa – L'istruzione «while»

1. fintanto che *l'istruzione di controllo restituisce il valore «vero»* ripeti:
2. *istruzioni*
3. fine-ciclo

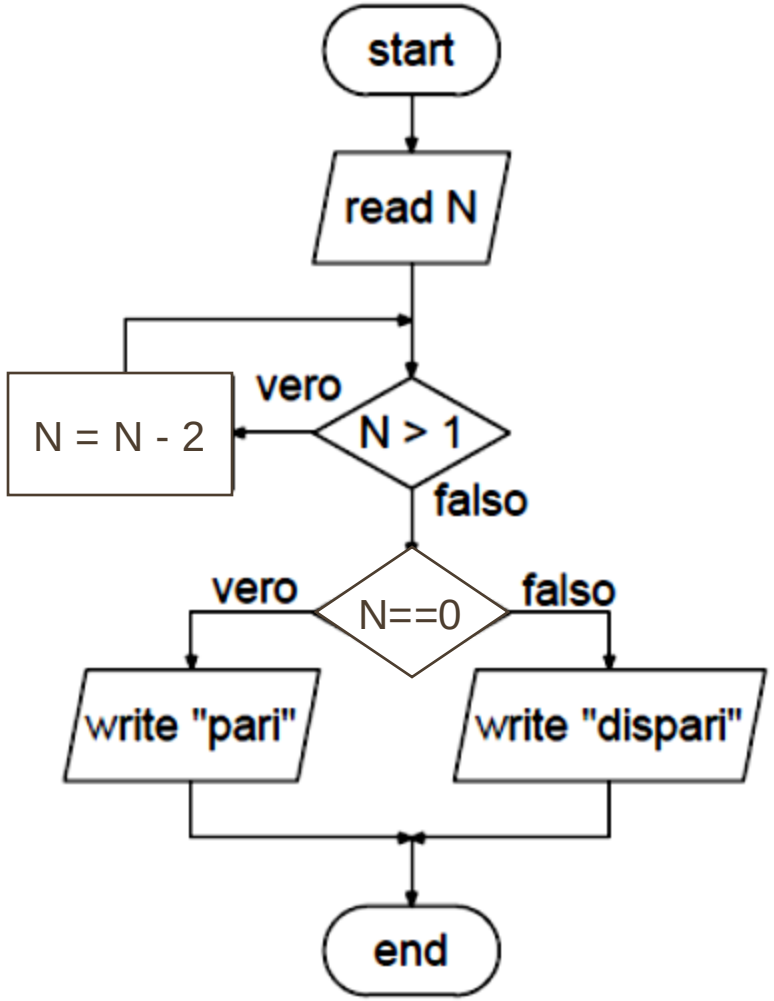
```
// Pseudocodice:
while istruzione di controllo
do
      istruzioni;
endwhile
```

Esercizio

- Dato un numero, e le istruzioni **while** e **if-then-else**, verificare se è pari o dispari e stampare il relativo messaggio. Fare anche il diagramma di flusso

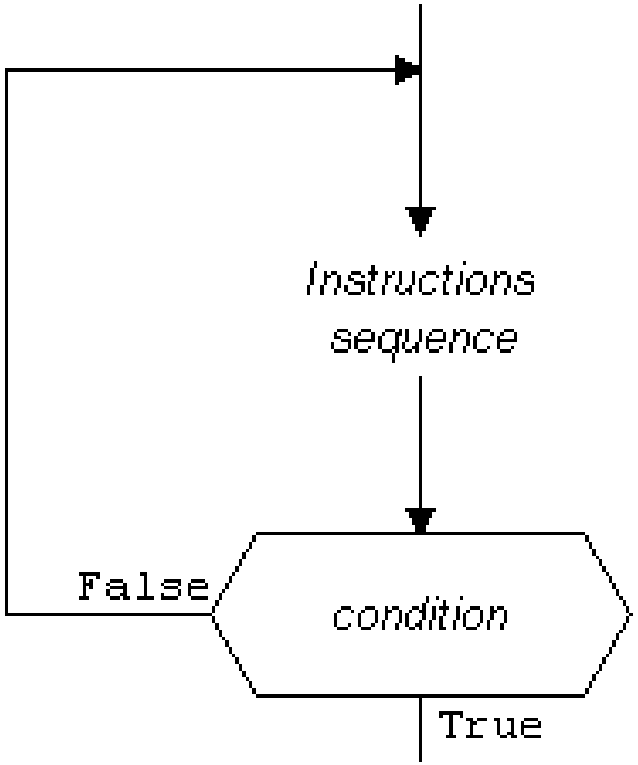
- Suggerimenti:
  - Per il ciclo while → decrementare il numero N in ingresso di 2 finchè…
  - Per l'if-then-else → se il valore di N uscito dal ciclo while ha un certo valore, allora…, altrimenti…

Esercizio




```
start
    read N
    while (N > 1) do
        N = N - 2 ;
    end while
    if (N == 0) then
        write "pari" ;
    else
        write "dispari" ;
    end if
end
```

Controllo in coda



Il blocco di istruzioni viene eseguito fino a che l'istruzione di controllo che segue il blocco di istruzioni restituisce il valore *vero*

Controllo in coda – L'istruzione «repeat-until»

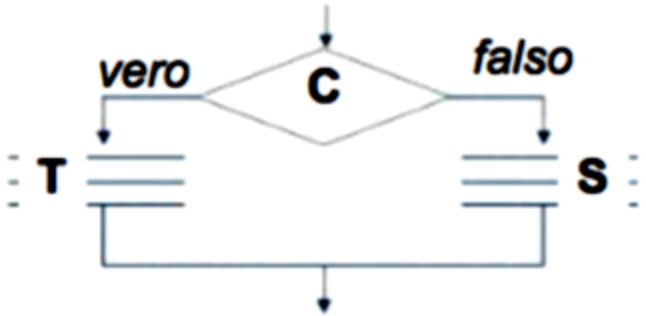
1. ripeti
2. blocco di istruzioni
3. finchè la condizione non restituisce il valore vero
4. fine ciclo

```
// Pseudocodice:
repeat
      istruzioni;
until istruzione di controllo
endrepeat
```

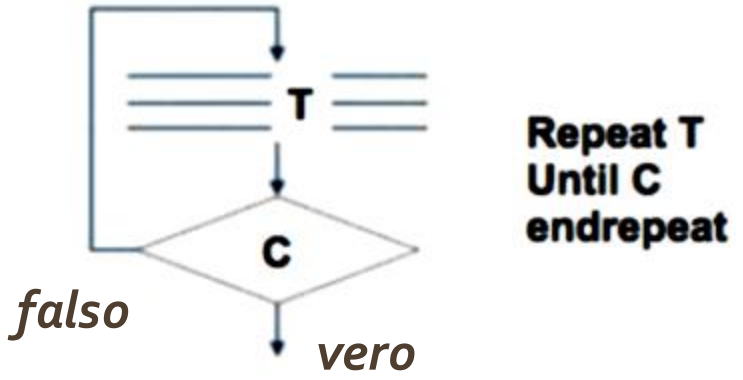
Ricapitolando: Corrispondenza diagrammi di flusso e pseudocodice

Condizionale

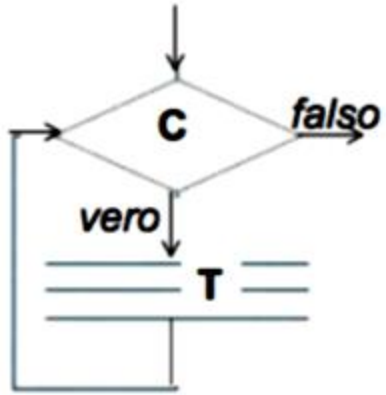
If C
Then T
Else S
endif



Iterativa



Repeat T
Until C
endrepeat



While C
Do T
endwhile

Ciclo con contatore

- Posso inserire un **contatore** del ciclo (per esempio una variabile i), che tiene memoria di quante iterazioni sono state effettuate. In genere il contatore è un numero intero.

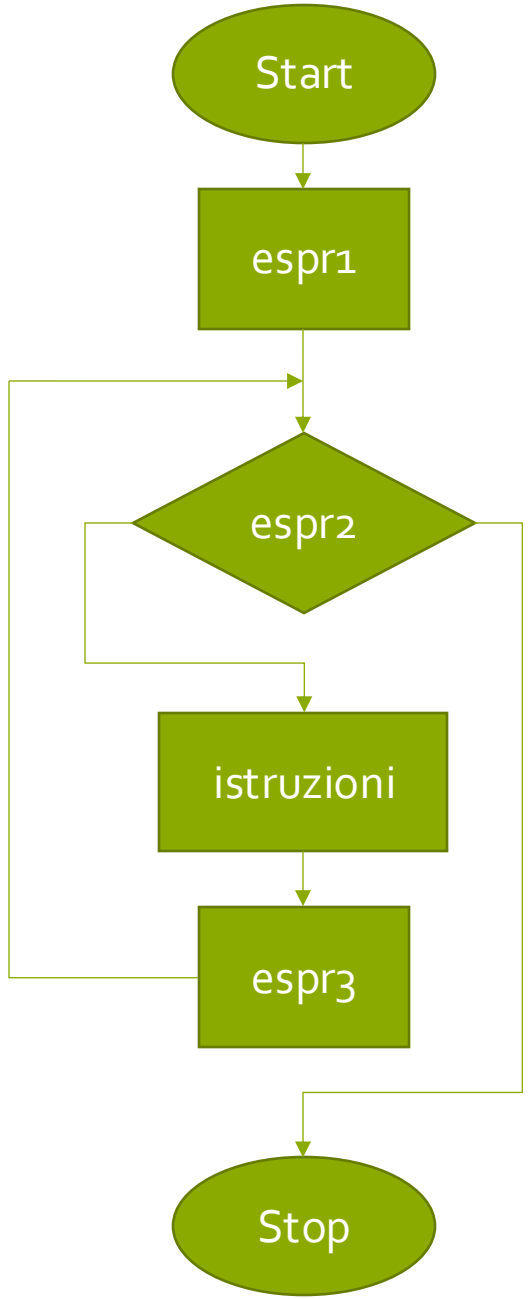
- **Può essere incrementato**: viene inizializzato ad un valore minimo (ad es. 0 o 1) e incrementato ad ogni esecuzione del ciclo
  - si esce dal ciclo quando il valore del contatore raggiunge il numero di iterazioni desiderate

- **Può essere diminuito**: viene inizializzato al numero di iterazioni desiderate e diminuito di uno ad ogni iterazione
  - si esce dal ciclo quando il valore del contatore raggiunge 0 (o 1)

L'istruzione «for»

- L'istruzione **for** consente in modo molto diretto di implementare una struttura di controllo iterativa basata su un **contatore**

```
\\ Forma generale
for (( espr1 ; espr2 ; espr3 )) do
      istruzioni;
endfor
```



L’istruzione «for»

- La prima espressione (espr1) generalmente è l'istruzione con cui viene **inizializzata una variabile contatore**

- La seconda espressione (espr2) è **la condizione con cui si verifica se il contatore ha raggiunto o superato la soglia**: se la condizione restituisce il valore «vero» viene eseguito il blocco di istruzioni delimitate dalle parole chiave “do” e “endfor”, altrimenti il ciclo termina e lo script prosegue con l'istruzione immediatamente successiva a “endfor”

- La terza espressione (espr3), infine, è l'**istruzione con cui viene incrementato (o decrementato) il valore del contatore**

L'istruzione «for»

```
// Pseudocodice
int i;

for (i=1; i<=10; i=i+1) do
      write ("Numero = ", i);
endfor
```

