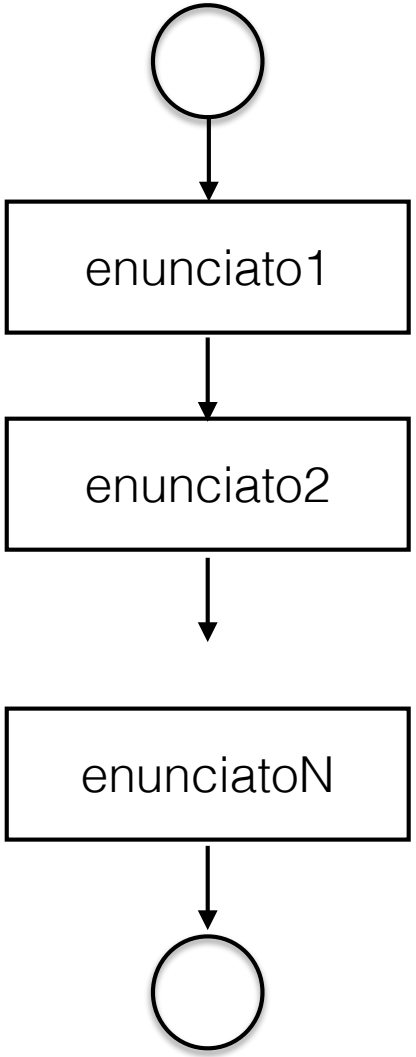
# Strutture di controllo

introduzione alla programmazione

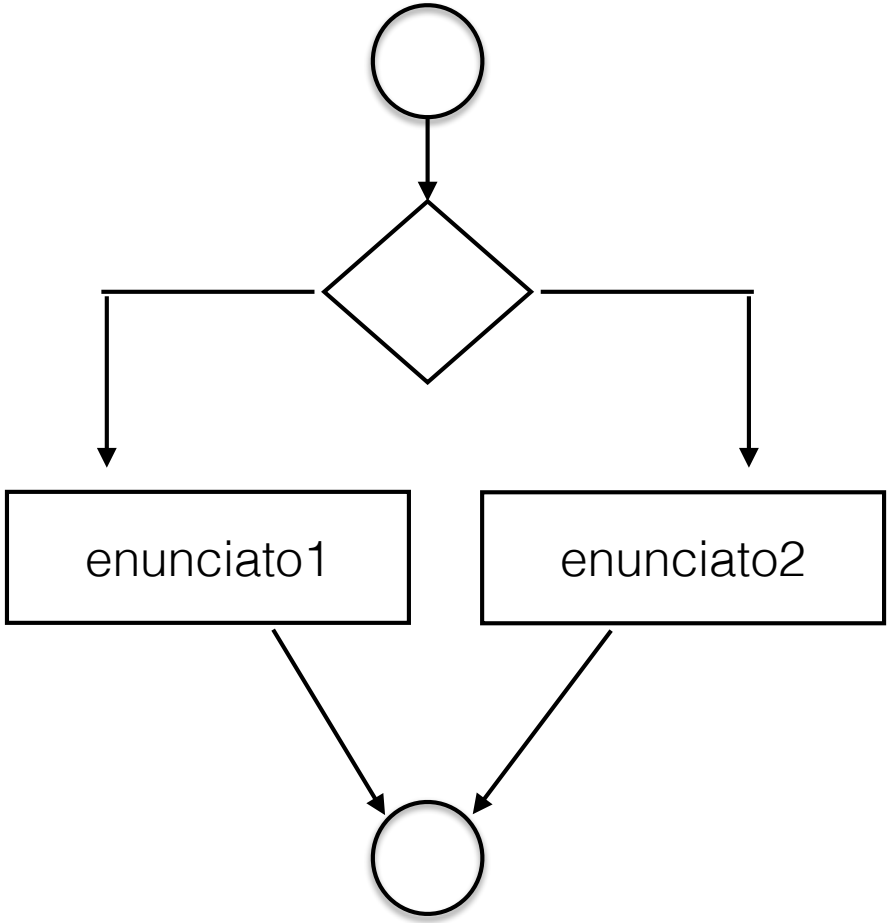
# Introduzione

- finora abbiamo analizzato programmi dal flusso lineare
- vedremo oggi costrutti che permettono di eseguire istruzioni in un ordine diverso o solo se determinate condizioni si verificano
- vedremo anche come ripetere le stesse istruzioni più volte, se necessario
- tutto questo viene formalizzato nel concetto *struttura di controllo*

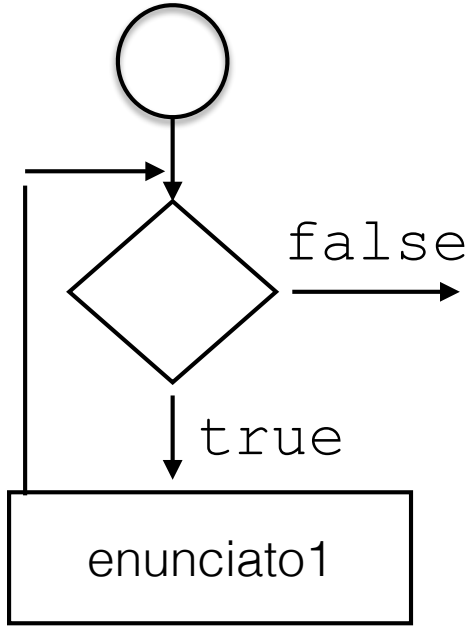
# flussi di esecuzione




sequenza




selezione




ripetizione

# espressioni logiche

- in C++ esiste un tipo base chiamato booleano che consiste di due soli valori `true` (vero) e `false` (falso)

```
bool controllo;    // dichiarazione di variabile di tipo bool
controllo = true; // esempio di assegnazione
```

- ***Le espressioni logiche o booleane***: espressioni che possono assumere valori vero o falso
  - una variabile o costante booleana
  - un'espressione seguita da un operatore relazionale seguito da un'espressione

# operatori relazionali

- == uguale a

notate bene!

- != diverso da
- < minore,  <= minore o uguale
- > maggiore, >= maggiore o uguale

**operatori relazionali**

- (2==7) è false
- (8<15) è true
- 3.1!=3.0 è true
- 6<=6 è true

# espressioni logiche

- Espressioni più complesse possono essere ottenute combinando espressioni logiche con gli operatori logici && (AND), || (OR) e ! (NOT)

- Esempio
  (6<6) && (1==1) è false
  (6<6)|| (1==1) è true
  !(6<6) è true

- Precedenze (l'operatore logico che arriva primo ha la precedenza)
  (5<3)&&(6<=6)||(5!=6) è true

# espressioni logiche e char

- le espressioni logiche dipendono dall'ordine dei caratteri nella tabella di riferimento utilizzata
  (noi facciamo riferimento alla tabella ASCII)

- Esempi
  `(' '<'a')` è true (nelle tabelle ASCII ' ' è 32 mentre 'a' è 97)
  `(8<'5')` è true!
  `('r'<'M')` è false! //le maiuscole sono prima

# espressioni logiche e string

- Esempio

```
string st1="Hello";
string st2="Hi";
```

ordine lessicografico

`st1<st2` è true

`st1=="hello"` è false

`st2 <= "Hii"` è true

# espressioni logiche e floating point

- i numeri floating point sono spesso soggetti ad errori di arrotondamento dovuti allo spazio finito in memoria ad essi dedicato

- vanno sempre trattati con cautela, ne vediamo un primo esempio nell'ambito delle espressioni logiche

- 3.0/7.0 + 2.0/7.0 + 2.0/7.0 == 1.0 può essere false

provate le seguenti varianti…

```
float f;
f=3.0/7.0 + 2.0/7.0 + 2.0/7.0;
cout << (f==1.0);
```

```
double d;
d=3.0/7.0 + 2.0/7.0 + 2.0/7.0;
cout << (d==1.0);
```

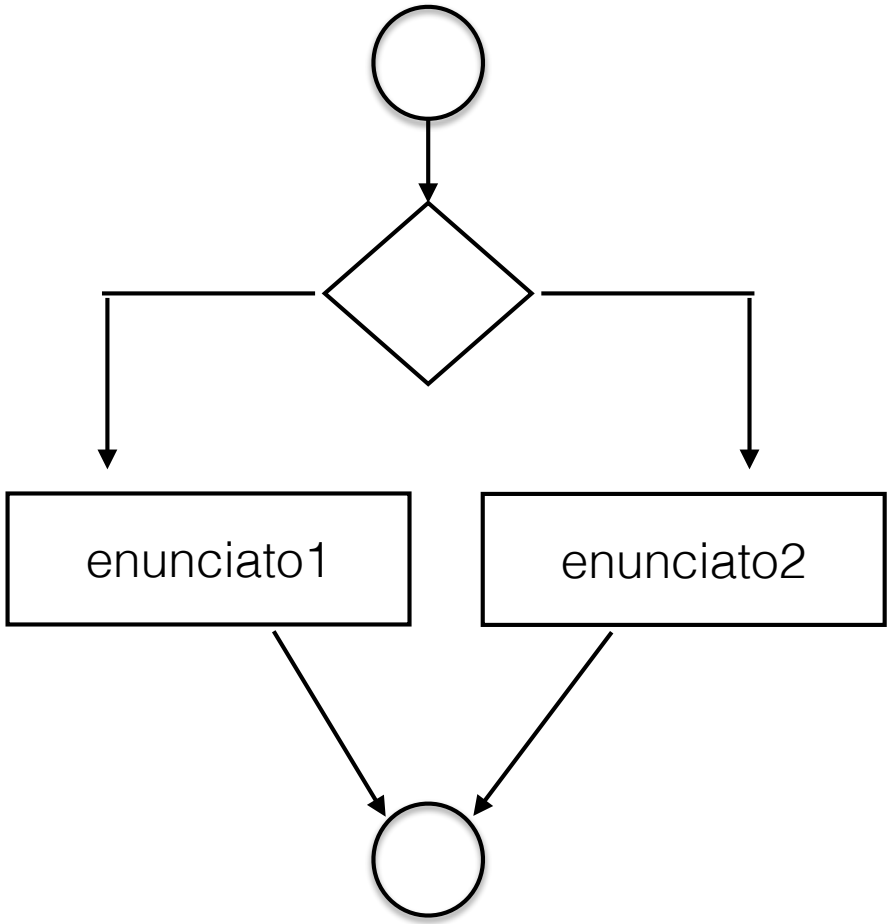
# espressioni logiche e floating point

- E' buona norma includere una *tolleranza* ossia accettare che il numero approssimato sia un po' diverso da quello atteso*:*

```
#include<cmath>
....
x= 3.0/7.0 + 2.0/7.0 + 2.0/7.0;
y=1.0;
fabs(x-y)<0.00001;
```

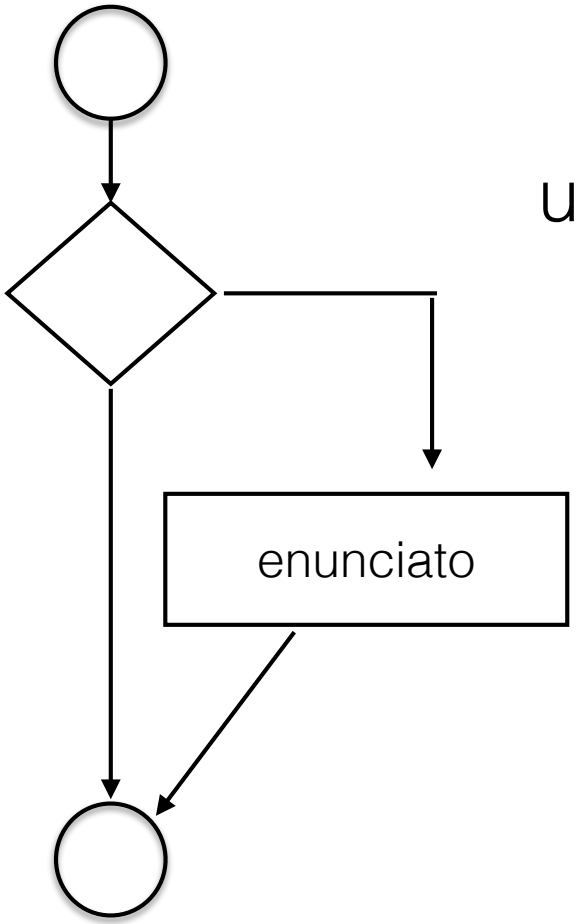
# Strutture di controllo: diramazioni

- diramazioni o
  frasi condizionali




selezione

# Strutture di controllo: diramazioni




un caso semplice: deviazione…

```
if(espressione)
   enunciato;
```

# Strutture di controllo: diramazioni

```
if(espressione)
    enunciato1;
else
    enunciato2;
```

```
int main(){

    int a, b;

    cout << "Inserisci due interi (stando ben attento a non sbagliare!)" << endl;
    cin >> a >> b;

    if (a==b)
        cout << "Gli interi a e b sono uguali" << endl;
    else
        cout << "Gli interi a e b sono diversi" << endl;

}
```

# i blocchi

```
int main(){

    int a, b;

    cout << "Inserisci due interi (stando ben attento a non sbagliare!)" << endl;
    cin >> a >> b;

    if (a==b)
        cout << "Gli interi a e b sono uguali" << endl;
    else
        {
          int c=b-a;
          cout << "Gli interi a e b sono diversi" << endl;
          cout << “e la loro differenza è “ << c << endl;
        }

}
```

le parentesi graffe sono importanti perche’ in questo modo le istruzioni contenute al loro interno vengono tutte eseguite in modo condizionale!

# i blocchi

```
int main(){

    int a, b;

    cout << "Inserisci due interi (stando ben attento a non sbagliare!)" << endl;
    cin >> a >> b;

    if (a==b)
        cout << "Gli interi a e b sono uguali" << endl;
    else

          int c=b-a;
          cout << "Gli interi a e b sono diversi" << endl;
          cout << “e la loro differenza è “ << c << endl;


}
```

cosa cambia in questo modo?

# if annidati

- All'interno di un blocco possono risiedere altri if; questo dà luogo a if annidati

- Esempio

```
if (temperature >=25)
   if (temperature >=30)
       cout << "Good day for swimming" << endl;
   else
      cout << "Good day for golfing" << endl;
else
   cout<<"Good day to play tennis" << endl;
```

# if annidati

```
if (month==1)
    cout << "January";
if (month==2)
    cout << "February";
if (month==3)
    cout << "March";
...
if (month==12)
    cout << "December";
cout << endl;
```

```
if (month==1)
    cout << "January";
else if (month==2)
         cout << "February";
     else if (month==3)
              cout << "March";
...
              else if (month==12)
                 cout << "December";
cout << endl;
```

più efficiente,
fa meno confronti

… non troppo leggibile
(molti nesting, troppe indent)

# if annidati

```
if (month==1)
    cout << "January";
else if (month==2)
    cout << “February”;
else if (month==3)
    cout << "March";
…
else if (month==12)
     cout << "December";
cout << endl;
```

stile molto più chiaro!

# errori comuni

- if votoIP >= 18
  voto = “sufficiente”;

errore di sintassi

- if (votoIP>=18);
  voto =“sufficiente”;

errore di semantica

# else “pendenti”

- nel caso di if annidati può succedere che diventi difficile capire a quale if fa riferimento un dato else
  - la chiarezza e la pulizia del sorgente sono fondamentali
- in alcuni casi il codice può diventare indiscutibilmente ambiguo:

il compilatore associa l’else all’if senza else più vicino

```
if (media < 19)
 if (media < 18)
     cout << "Esame Fallito";
 else cout << "Passato per il rotto della cuffia"
```

in questo caso l’interpretazione sarebbe stata giusta

# else “pendenti”

- in alcuni casi il codice può diventare indiscutibilmente ambiguo:

in questo caso no!

```
if (media >= 18)
 if (media < 19)
     cout << "Passato per il rotto della cuffia”;
 else cout << "Esame Fallito";
```

- ecco la versione corretta:

```
if (media >= 18) {
 if (media < 19)
     cout << "Passato per il rotto della cuffia”;
 }
else cout << "Esame Fallito";
```

# operatore condizionale

- L'operatore condizionale ?: è un operatore **ternario** (necessita di 3 operandi)

- `espressione1 ? espressione2 : espressione3`

- equivalente a

```
if (espressione1)
        espressione2;
else
        espressione3;
```

- esempio:

```
max=(a>=b) ? a : b;
```

# struttura switch

- struttura di selezione del C++ che permette di scegliere tra molte alternative
- inoltre non richiede la valutazione di un'espressione logica

deve assumere un valore di uno dei tipi di dato semplici

```
switch (espressione)
{
case valore1:
     enunciati1
     break;
case valore2:
     enunciati2
     break;
…
case valoreN:
     enunciatiN
     break;

default:
     enunciati
}
```

# torniamo all'algoritmo dell'anno bisestile - codifica (ver1)

**verifica se l'anno N è bisestile:**

**Versione 5 (un po' più schematica):**

1. Divido N per 4
2. **IF** il resto non è 0
   **RETURN** false
3. Divido N per 100
4. **IF** il resto non è 0
   **RETURN** true
5. Divido N per 400
6. **IF** il resto non è 0
   **RETURN** false
7. **RETURN** true

# torniamo all'algoritmo dell'anno bisestile - codifica (ver1)

**verifica se l'anno N è bisestile:**

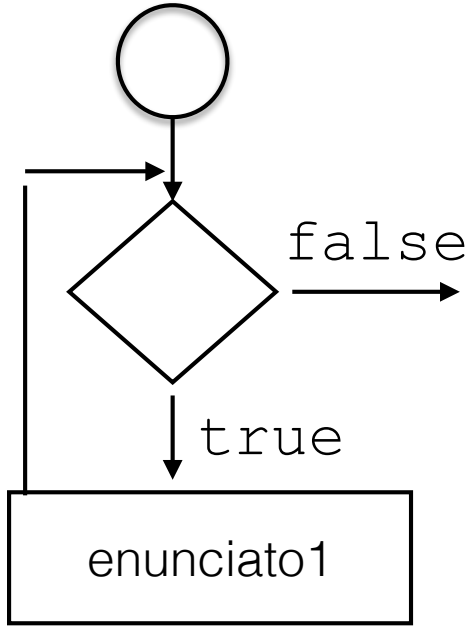
**Versione 5 (un po' più schematica):**

1. Divido N per 4
2. **IF** il resto non è 0
   **RETURN** false
3. Divido N per 100
4. **IF** il resto non è 0
   **RETURN** true
5. Divido N per 400
6. **IF** il resto non è 0
   **RETURN** false
7. **RETURN** true

```
#include <iostream>

using namespace std;


int main()
{
    int anno;
    cout << "Inserisci da tastiera un anno di quattro cifre" << e
    cin >> anno;

    if (anno % 4 != 0) {
        cout << anno << " non e` bisestile\n";
        return 0;
    }

    if (anno % 100 != 0 ) {
        cout << anno << "  e` bisestile\n";
        return 0;
    }


    if (anno % 400 !=0 ) {
        cout << anno << " non e` bisestile\n";
        return 0;
    }

    cout << anno << "  e` bisestile\n";
    return 0;
}
```

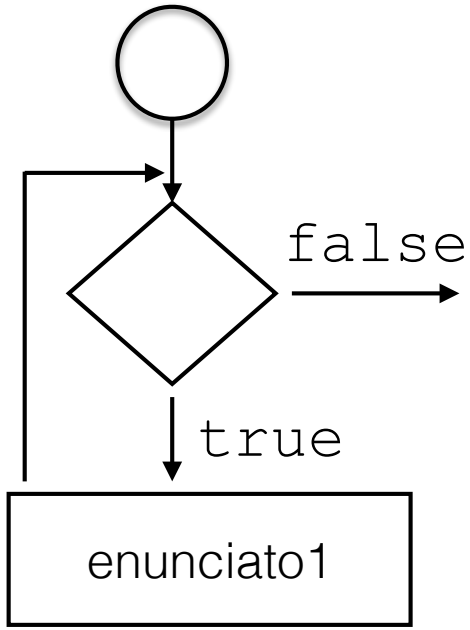
# Stutture di controllo: ripetizioni o loop

- loop: una struttura di controllo che fa sì che un enunciato (o un insieme di enunciati) venga ripetuto più volte




ripetizione

# Il while




```
while (espressione)
    enunciato;
```

```
while (espressione)
{
    enunciato1;
    …
    enunciatoN;
}
```

# Il while

Esempi

- “continua a chiedere un numero finche’ l’utente non ti fornisce un numero positivo”
- “ripeti la stampa ‘BIANCO NERO BIANCO’ 10 volte”

# while controllato da un contatore

- “ripeti la stampa ‘BIANCO NERO BIANCO’ 10 volte”

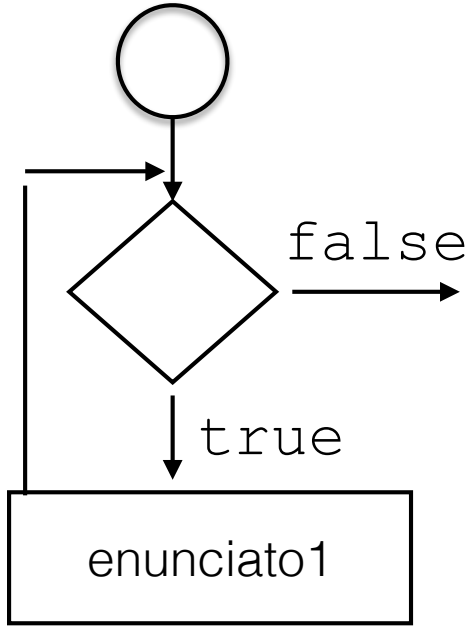
```
cout << "BIANCO NERO BIANCO " << endl;
cout << "BIANCO NERO BIANCO " << endl;
cout << "BIANCO NERO BIANCO " << endl;
cout << "BIANCO NERO BIANCO " << endl;
cout << "BIANCO NERO BIANCO " << endl;
cout << "BIANCO NERO BIANCO " << endl;
cout << "BIANCO NERO BIANCO " << endl;
cout << "BIANCO NERO BIANCO " << endl;
cout << "BIANCO NERO BIANCO " << endl;
cout << "BIANCO NERO BIANCO " << endl;
```

- un’alternativa migliore:

```
contatore=1;
while (contatore <=10)
{
    cout << "BIANCO NERO BIANCO " << endl;
    contatore ++;
}
```

# Il while

- “continua a chiedere un numero finche’ l’utente non ti fornisce un numero positivo”




while (espressione che verifica se il numero è negativo)
    se sono entrato nel while (l’espressione è vera,
    ossia il numero è negativo)
    chiedo un  altro numero

```
while (numero <= 0)
      cin >> numero;
```

occorre inizializzare la variabile numero

siamo sicuri di uscire dal loop?

# ancora while

- non sempre si conosce in anticipo il numero di volte in cui eseguire il ciclo.
  - while controllato da una sentinella - rimango nel ciclo fino a che la variabile di controllo non raggiunge un valore speciale detto “sentinella”
  - while controllato da un *flag* - il flag è una variabile di tipo bool

# while controllato da una sentinella

```
cin >> variabile;

while (variabile!=sentinella)
     {
     cin >> variabile;
      ….
      }
```

- ES: calcolo la media di una sequenza di numeri interi positivi letti da tastiera. Utilizzo un intero non positivo come sentinella per la chiusura del loop

# while controllato da un flag

- utilizza una variabile booleana per realizzare il controllo del ciclo

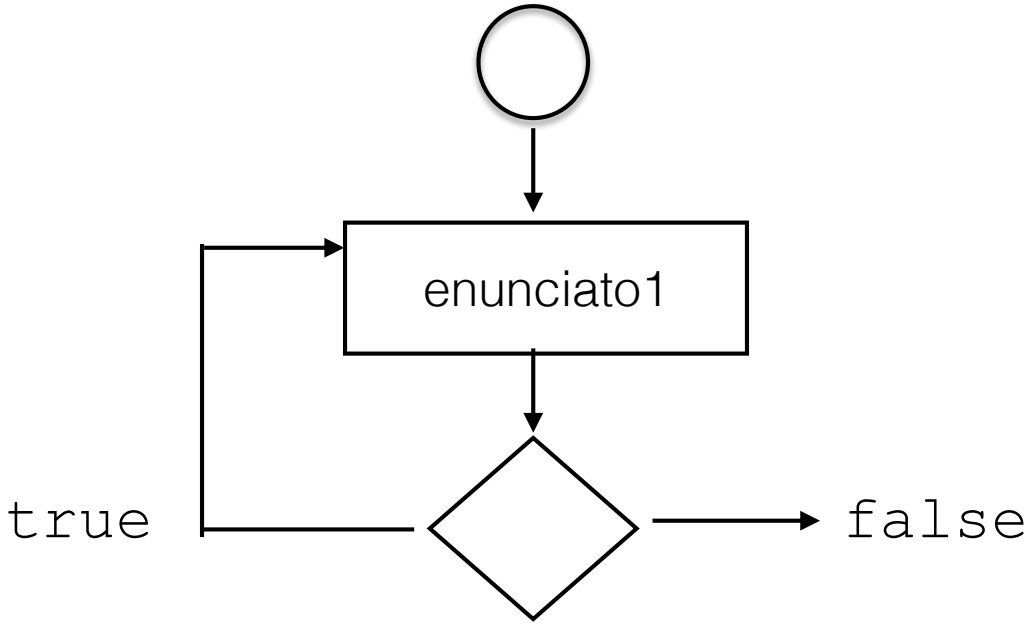
```
found=false;

while (!found)
    {
     ….
     if (espressione)
         found=true;
     }
```

# Esempio

- Sommare una sequenza di dieci numeri interi dispari letti da tastiera.

- Condizione di uscita dal loop: ho sommato 10 numeri del tipo giusto
  [Hint: all'interno del ciclo occorre una selezione]

# do-while




```
do
   enunciato
while (espressione);
```

```
do {
     enunciato1;
     …
     enunciatoN;
}
while (espressione);
```

# Esercizi

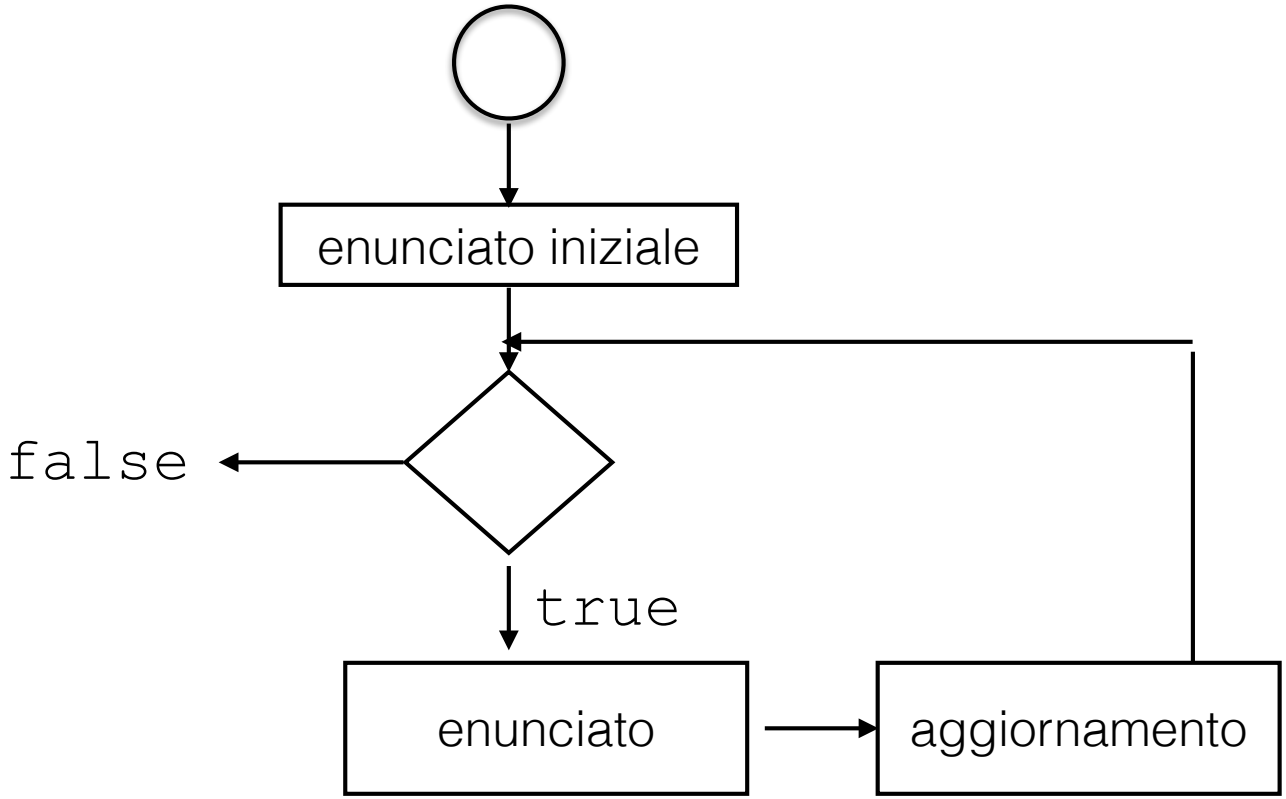
- scrivere un programma che calcola l'età media (o massima) degli studenti di IP presenti in classe
  [Hint: usiamo un controllo con sentinella come elemento di verifica di fine loop]

- scrivere un programma che calcoli il fattoriale di un numero intero *n* inserito in input dall'utente
  [Hint: riusciamo a prevedere la lunghezza del ciclo?]

# il ciclo for

- e' una forma particolare di ciclo che semplifica il  caso di ciclo con contatore

```
for(enunciato iniziale; condizione; aggiornamento)
        enunciato
```



# while o for?

```
int contatore=1;
while (contatore <=10)
{
    cout << "BIANCO NERO BIANCO " << endl;
    ++contatore;
}

int contatore;
for (contatore=1; contatore <=10; ++contatore)
{
    cout << "BIANCO NERO BIANCO " << endl;
}
```

O ANCHE:

```
for (int contatore=1; contatore <=10; ++contatore)
{
    cout << "BIANCO NERO BIANCO " << endl;
}
```

# Un altro esercizio

- Progettare e implementare un algoritmo che genera un numero casuale tra 0 e 100 e chiede all'utente di indovinarlo
- se l'utente indovina stampa un messaggio di complimenti
- altrimenti fornisce un aiuto ("il numero che proponi è maggiore al numero che devi indovinare. Riprova")
- Il ciclo si chiude quando l'utente indovina