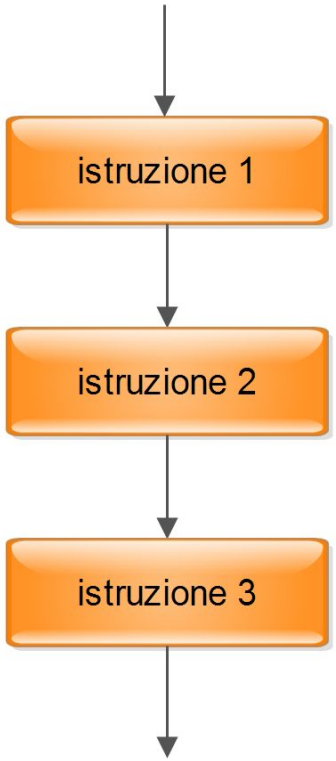
Strutture di Controllo del C

STRUTTURE FONDAMENTALI

Osservando molti flowchart ci si accorge che è sempre possibile individuare tre schemi tipici:

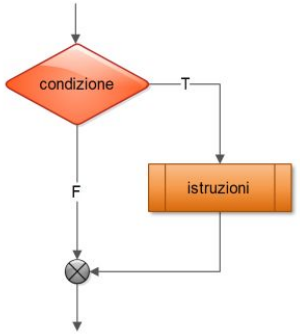
1. **sequenza: i blocchi si susseguono uno dopo l'altro nell'ordine in cui sono eseguiti.**



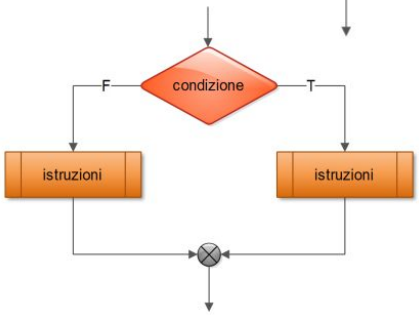
STRUTTURE FONDAMENTALI

2. selezione: il flusso arriva ad un blocco di selezione e da lì si possono avere più ramificazioni; a seconda dei casi si distinguono selezione:

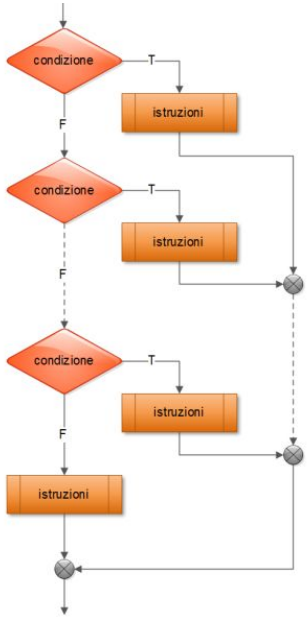
- **semplice**: quando il verificarsi della condizione porta all'esecuzione di una o più istruzioni, ma poi viene ripreso il flusso principale.



- **binaria**: quando il flusso prevede due percorsi logicamente distinti.

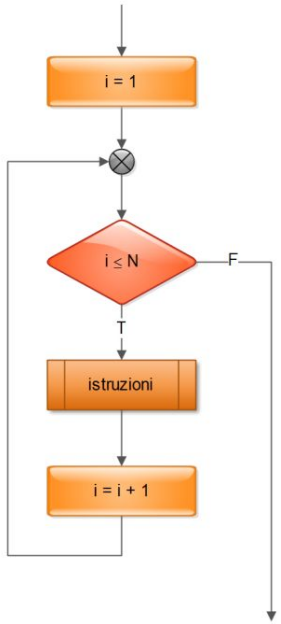


- **multipla**: quando si ha una serie di selezioni successive, che portano ad una rapida ramificazione dei possibili flussi.



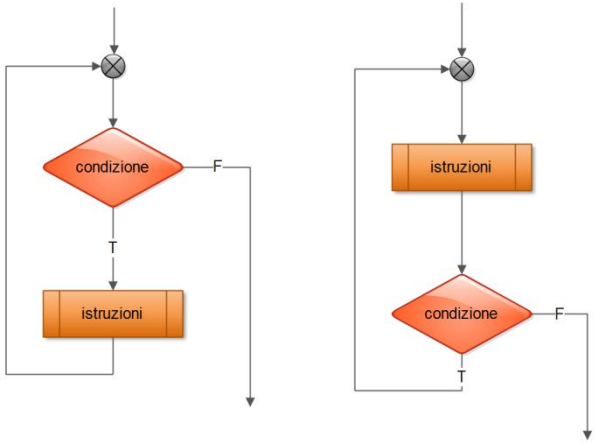
STRUTTURE FONDAMENTALI (CONT.)

3. **Iterazione (ciclo): quando una selezione controlla la ripetizione di un gruppo di operazioni; la si riconosce facilmente, poiché è l'unico caso in cui una freccia torna in un punto già visitato del diagramma. Anche in questo caso si distinguono due tipi di iterazioni:**



- **Definita** (Enumerativa): quando è noto a priori il numero di iterazioni da eseguire; tipicamente tale numero è confrontato con una variabile che tiene il conto delle iterazioni stesse (variabile contatore).

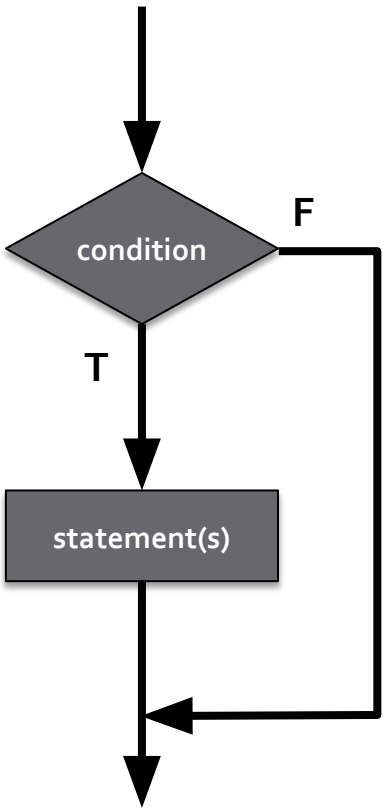
- **Indefinita**: quando non è noto a priori il numero delle iterazioni; tipicamente il verificarsi della condizione di uscita dal ciclo è legato alle operazioni eseguite nel corpo del ciclo.



Implementazione del C

if

Selezione Semplice

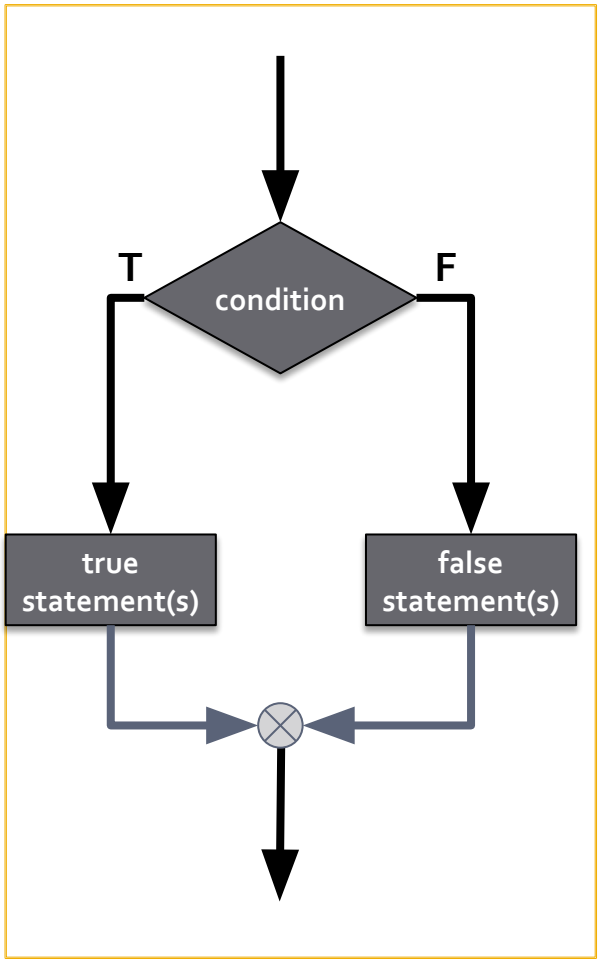



```
if(condition)
{
    statement(s)
}
```

- Se la *condizione* è vera esegue le ***Istruzioni*** tra le parentesi
- Se la condizione è falsa riprende l'esecuzione dopo la chiusura della parentesi graffa (salta il blocco).

if else

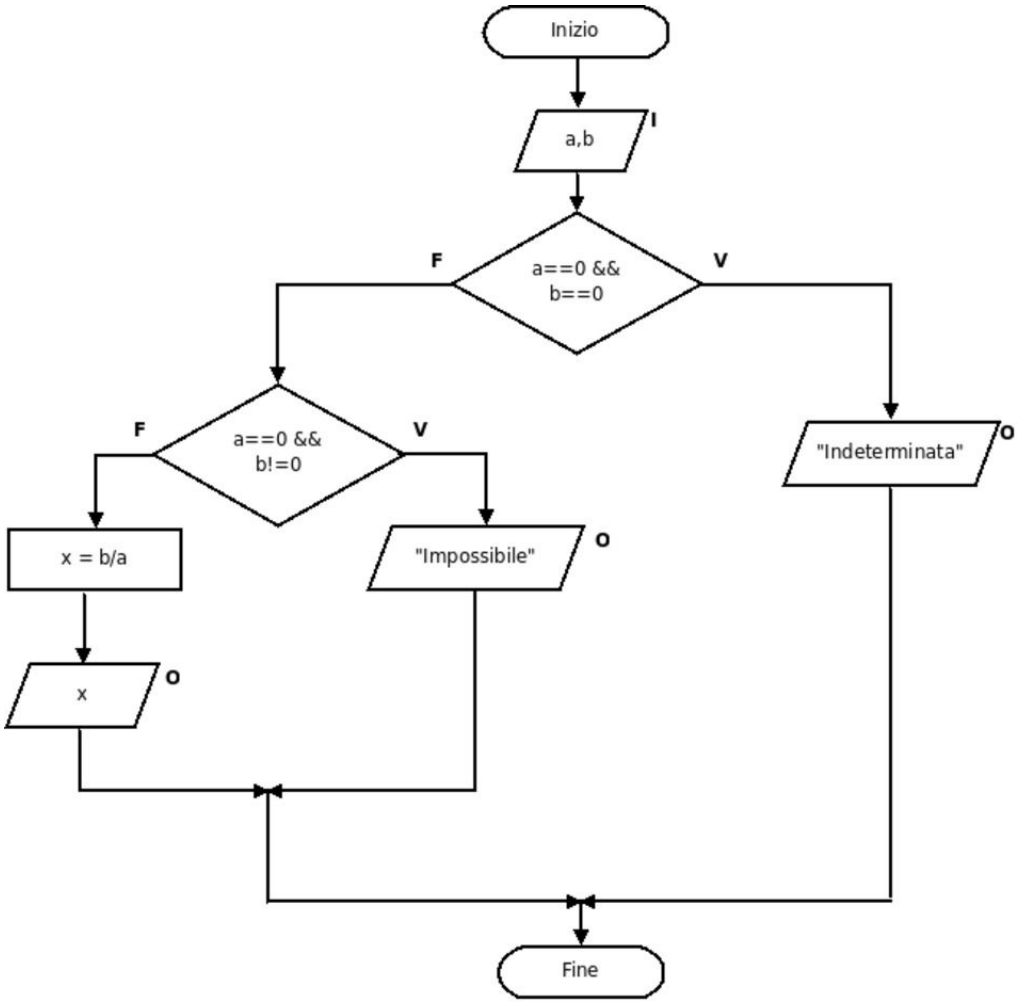
Selezione Doppia




```
if(condition)
{
     statement_true
}
else
{
    statement_false
}
```

- Se la *condizione* è vera esegue le *Istruzioni* del primo blocco (prima dell'**`else`**)
- Se la condizione è falsa esegue le *istruzioni* del secondo blocco (dopo **`else`**)

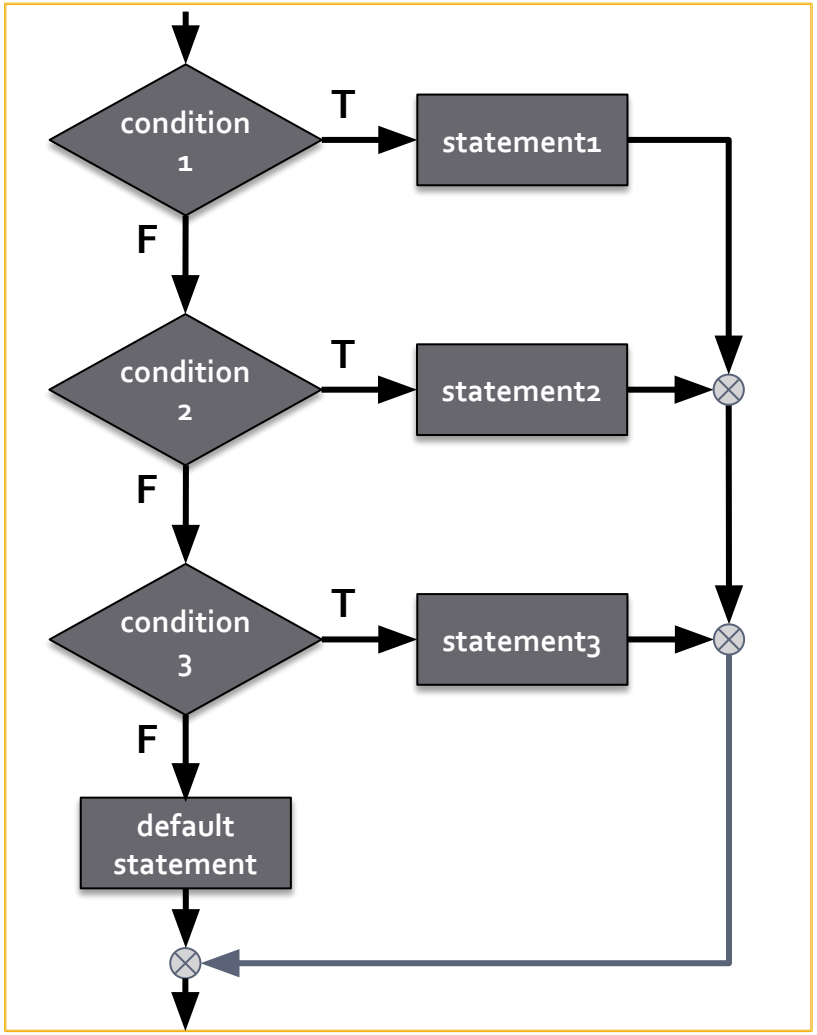
Selezione Nidificate



Implementare il flowchart a fianco: a cosa serve?

if
else if

Selezione Multipla

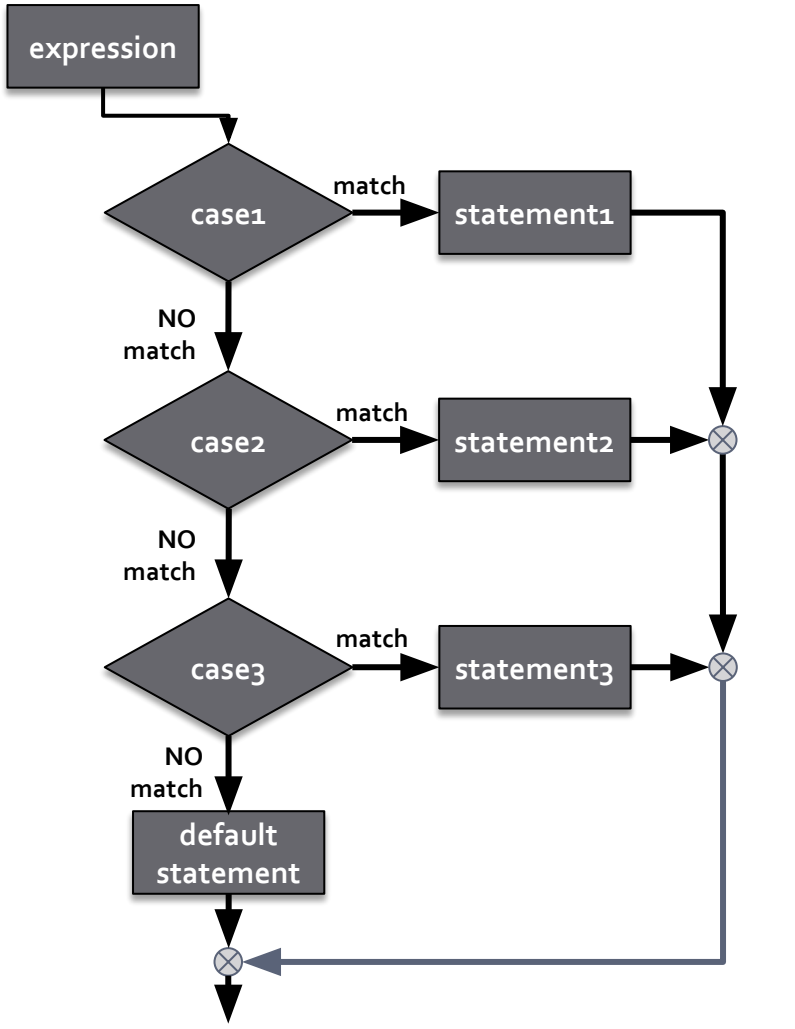



```
if(condition1)
{
      statement1
}
else if(condition2)
{
     statement2
}
else if(condition3)
{
     statement3
}
else
{
     default_statement
}
```

- Utile per condizioni complesse
- Verificata una condizione le altre non vengono testate

switch case

Selezione Multipla

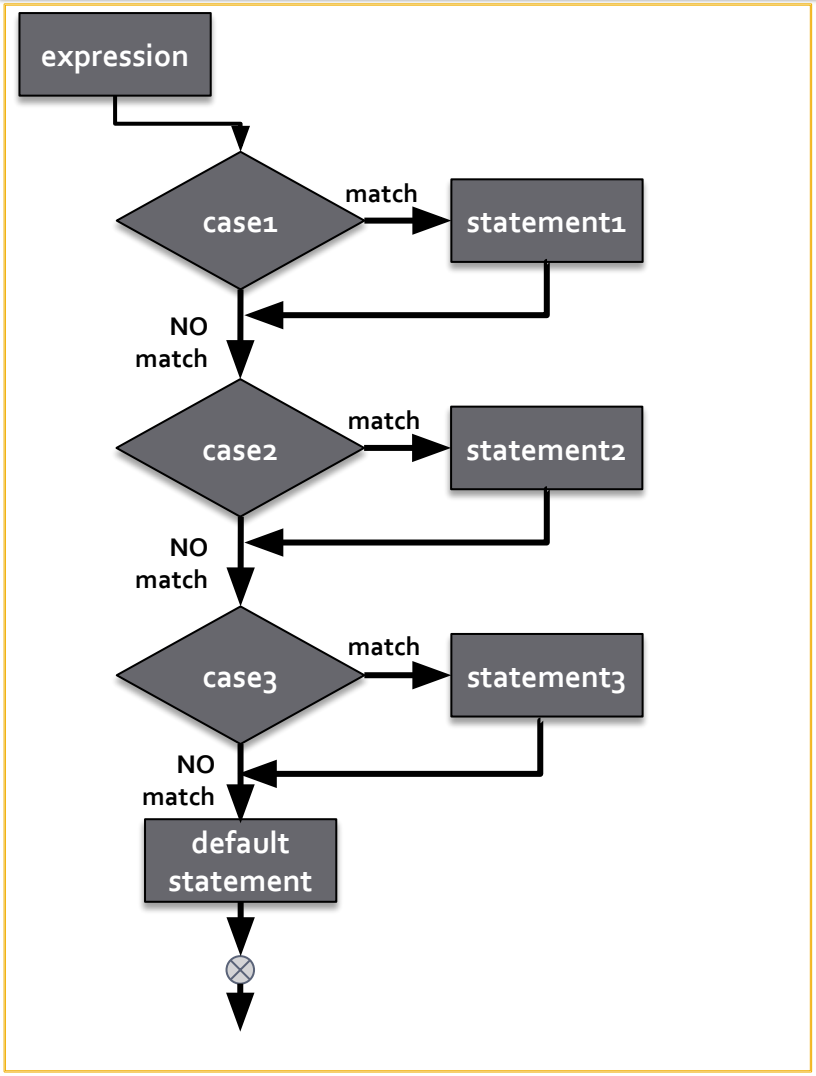



```
switch(expression)
{
    case 1:
        statement1
        break
    case 2:
        statement2
        break
    case 3:
        statement3
        break
    default:
        default_statement

}
```

- Utile per uguaglianze esatte (scelta voce menu)
- Terminato il case prescelto, esce dallo switch (per effetto del break)

switch case

Selezione Multipla

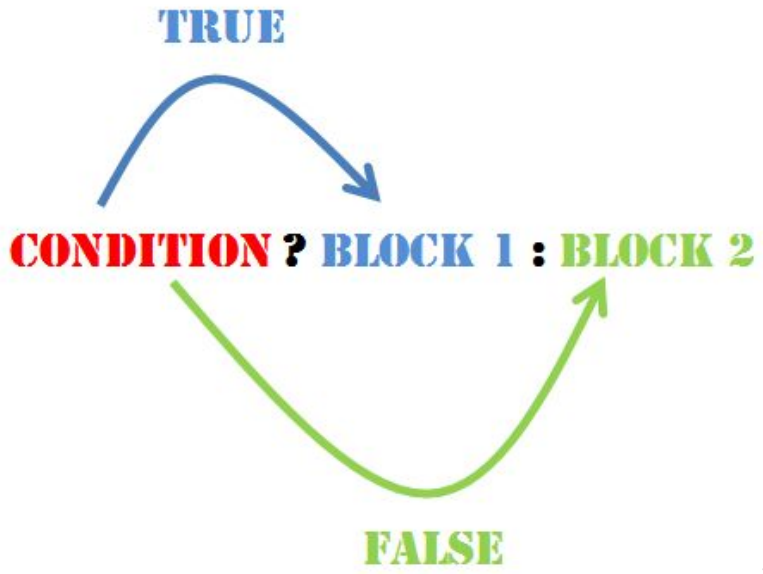



```
switch(expression)
{
    case 1:
        statement1

    case 2:
        statement2

    case 3:
        statement3

    default:
        default_statement

}
```

- Senza il **break**, tutti i casi vengono testati anche se uno ha già riscontrato il match
- Poco efficiente, ma può tornare utile

cond

Operatore Ternario



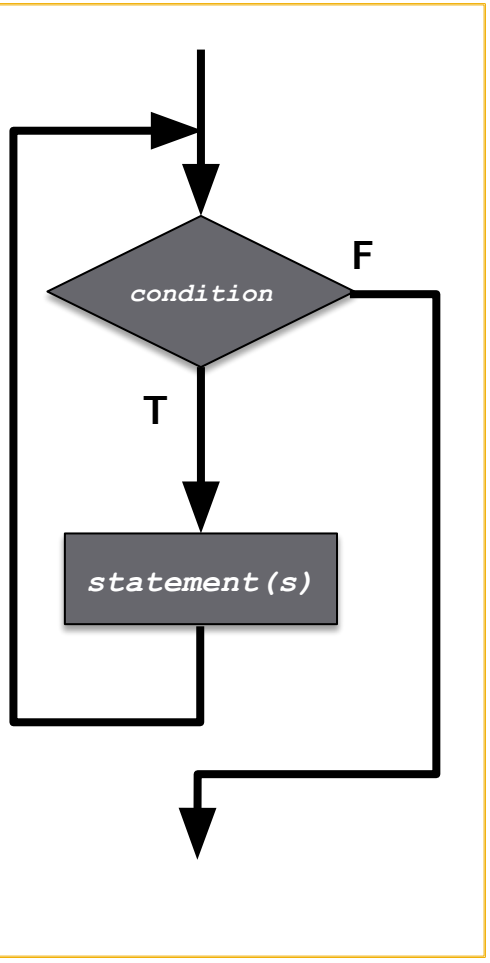
expr ? stat_true : stat_false

- velocizza implementazione
- Poco leggibile
- Difficile debug

```
#include <stdio.h>

int main()
{
    int a=10, b;

    printf( "b is %d\n", (a == 1) ? 20: 30 );

    printf( "b is %d\n", (a == 10) ? 20: 30 );
    return 0;
}
```

while()

Ciclo pre-Condizionato

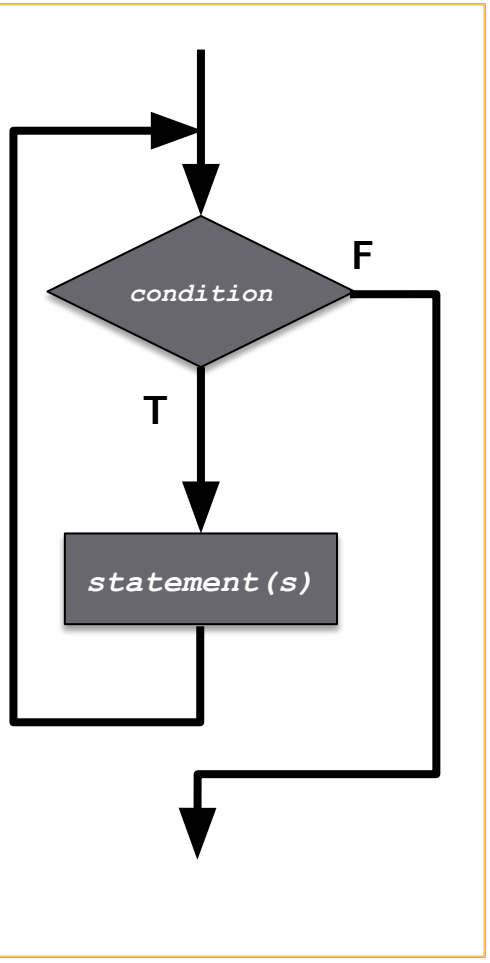



```
while(condition)
{
    statement(s)
}
```

- Finché la *condizione* è vera ripete l'esecuzione del blocco di *Istruzioni* (***statement***)
- Ad ogni iterazione la *condizione* viene controllata

while()

Ciclo pre-Condizionato

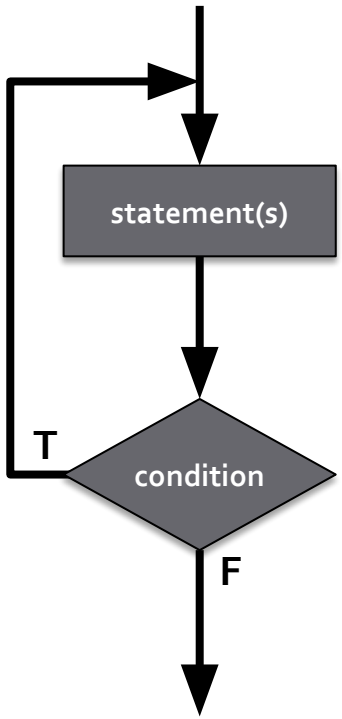



```
while(condition)
{
    statement(s)
}
```

```
#include <stdio.h>

int main (void)
{
   int a = 10;
   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;
   }
   return 0;
}
```

do while

Ciclo Post-condizionato

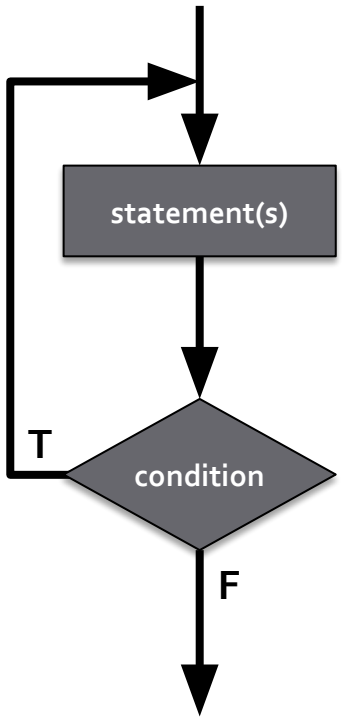



```
do {
    statement(s)
} while(condition)
```

- Esegue il blocco di *Istruzioni* tra le parentesi una prima volta e continua fintanto che la *condizione* è vera
- Il blocco di *Istruzioni* è eseguito **almeno una** volta

do while

Ciclo Post-condizionato

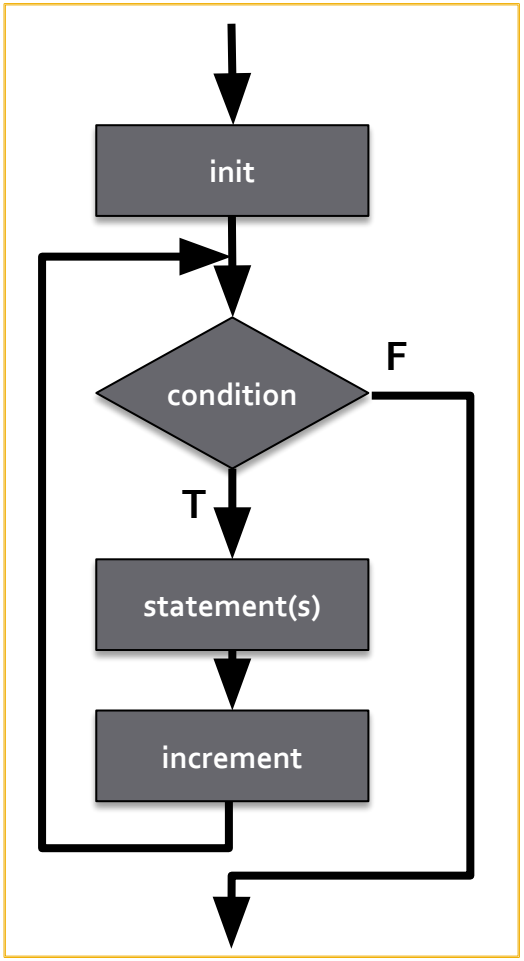



```
do {
    statement(s)
} while(condition)
```

```
#include <stdio.h>

int main (){
   int a = 10;

   do
   {
        printf("value of a: %d\n", a);
        a = a + 1;
   }while( a < 20 );

   return 0;
}
```

for

Ciclo Enumerativo

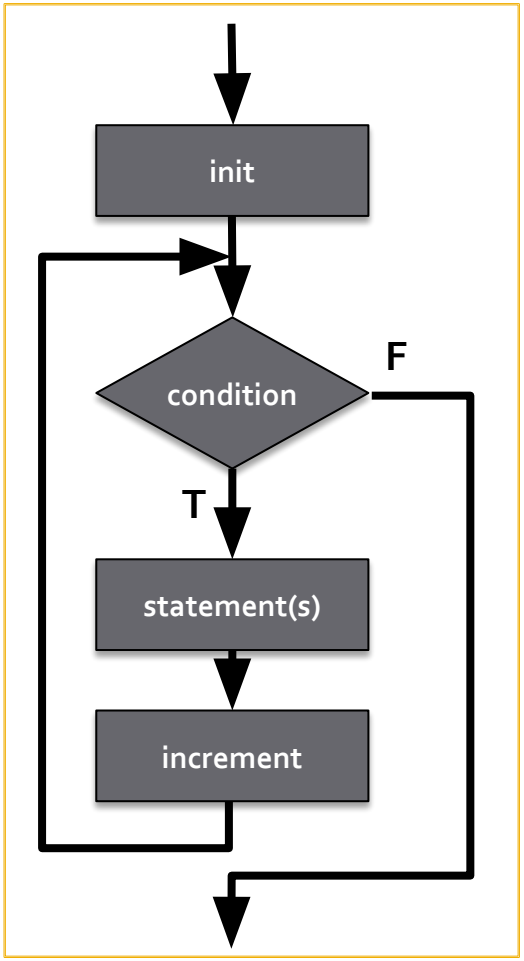



```
for(init, condition, increment)
{
    statement(s)
}
```

- Il blocco di *Inizializzazione* è eseguito per primo e solo una volta (non è obbligatorio)
- Dopo l'esecuzione del blocco di *Istruzioni* viene eseguita l'istruzione di *Incremento*

for

Ciclo Enumerativo




```
for(init, condition, increment)
{
    statement(s)
}
```

```
#include <stdio.h>

int main (){
   int a;
   for(a = 10; a < 20; a = a + 1 )
   {
      printf("value of a: %d\n", a);
   }

   return 0;
}
```

Ciclo Infiniti

To do..

Break vs. COntinue

```
for(i = 0; i < 5; i++){
        scanf("%d", %d);
        if(d < 0) {
                break;
        }
        if(d > 10){
                continue;
        }
        printf("d は 10 以下¥n");
}
prinf("i が 5 より小さいなら d は負¥n");
```

```
while (expr)
    statement
    statement
    break
    statement
    statement
    continue
    statement
    statement
statement
```

- **break**, permette di terminare anticipatamente il ciclo
- **continue**, consente di terminare l'attuale iterazione e procedere con la valutazione della condizione

Sitografia

- Tutorialspoint
- Html.it