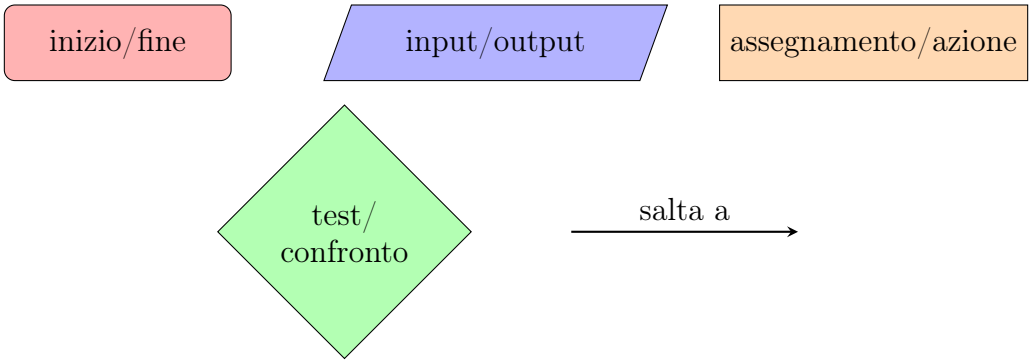
Flowchart

I flowchart o diagrammi di flusso sono uno dei modi più intuitivi di rappresentare formalmente l'andamento logico di un algoritmo. Sono utilizzati per "metter su carta" un nostro ragionamento prima di passare al codice (soprattutto se non si è ancora familiari con esso).



Solitamente un ragionamento o la soluzione ad un problema è posto in forma scritta ed un modo efficace per impostare un algoritmo adatto alla traduzione in codice è quello di dividere in piccoli passi le azioni ed isolare le possibili variabili, eventualmente aggiungendone altre.

Un esempio chiarirà subito:
implementare un programma (completamente inutile) che legge due numeri, ne fa la media e ci dice se questa media è un numero pari o dispari.

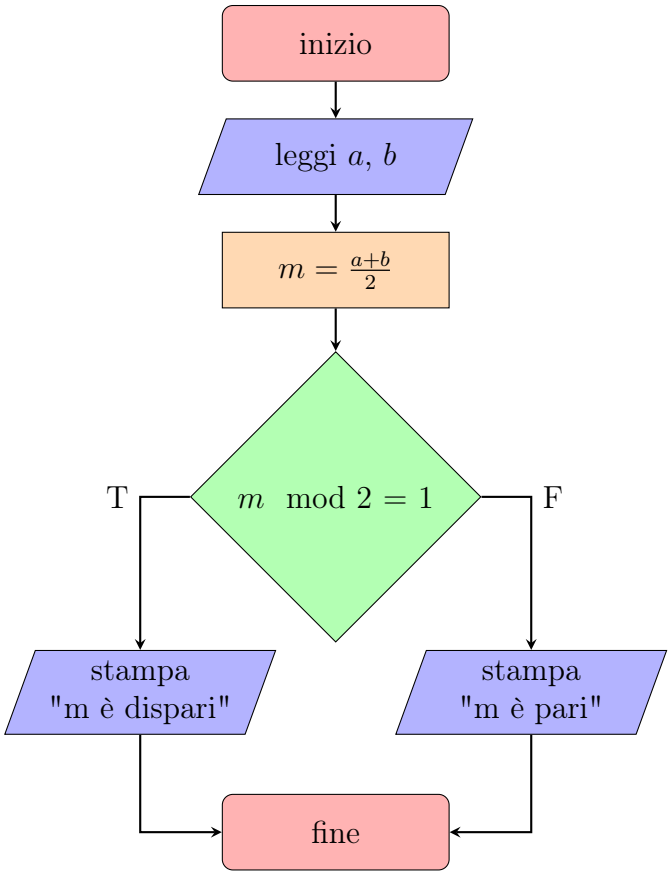
Isoliamo le possibili variabili:

definiamo a e b come variabili che andranno a contenere i numeri inseriti da tastiera

definiamo anche una variabile m che conterrà il risultato

e sulla quale verificheremo la parità
[l'operazione modulo fornisce il resto di una divisione, quindi se divido per 2 ed ottengo il resto (di 1), vuol dire che m è dispari]

infine, stampiamo l'output a seconda di come è stata valutata l'espressione

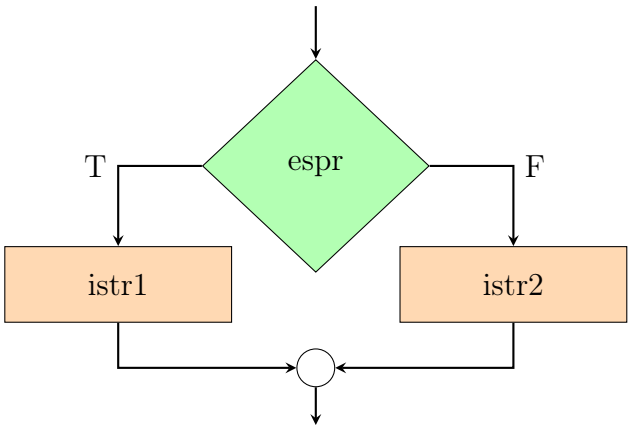


Strutture di controllo

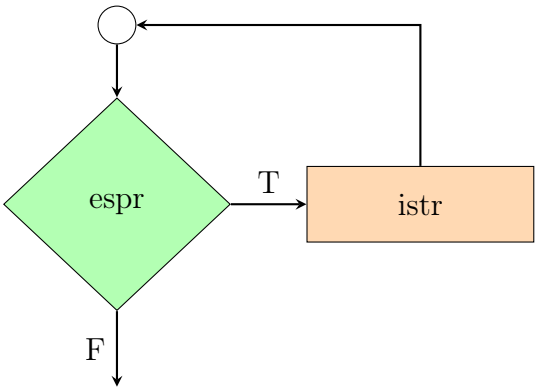
Abbiamo già visto che le istruzioni sono delle espressioni delimitate da punto-e-virgola, i costrutti di controllo invece facco eccezione a questa regola.
Queste struttre fanno intrinsecamente parte della sintattica del linguaggio C, ma ricorrono anche in altri linguaggi con delle varianti. Le strutture di controllo che descriviamo sono:

- `if ( espr ) istr [ else istr ]`
- `while ( espr ) istr`
- `do istr while ( espr ) ;`
- `for ( espr ; espr ; espr ) istr`
- `switch ( espr-intera )  case:  ....`
- `break ; continue ; return [ espr ] ;`

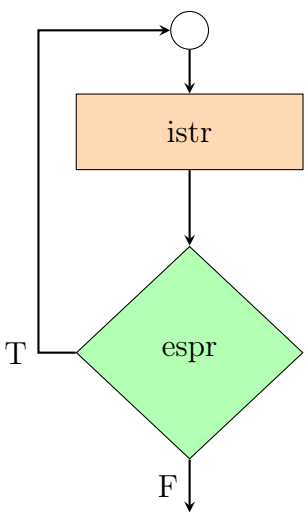
`if ( espr ) istr1 else istr2`



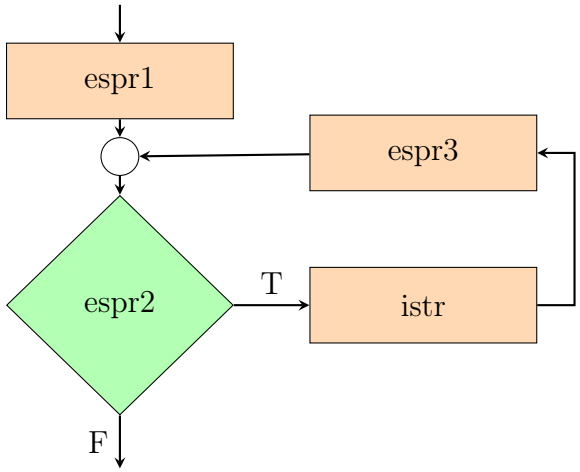
`while ( espr ) istr`



`do istr while ( espr ) ;`



`for ( espr1 ; espr2 ; espr3 ) istr`



`switch ( espr-intera )  case:  ....`

```
switch ( espressione-intera ) {
  case espressione-costante :
    [ istr ]
    [ ... ]
    [ break ; ]
  case espressione-costante :
    [ istr ]
    [ ... ]
    [ break ; ]
  [ default: ]
    [ istr ]
    [ ... ]
    [ break ; ]
}
```

`break;`

L'espressione `break` termina il ciclo più interno nel quale è contenuta; serve per "romprere" (od uscire dai) cicli for e while.

`continue;`

`continue` serve per "saltare" una iterazione dentro un ciclo for o while; quando eseguita fa saltare l'iterazione corrente alla valutazione della condizione del ciclo.

`return [ espr ] ;`

L'ultima espressione termina la funzione nella quale è contenuta restituendo alla funzione chiamante ("funzione padre") il valore della `espr` (ad es. 0 oppure una variabile). Se una funzione è dichiarata `void` non bisogna metterci `return`.