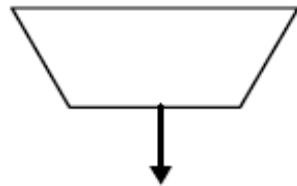


# **Programmazione strutturata**

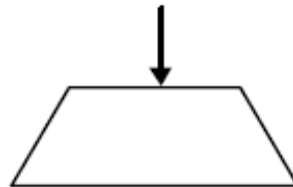
# Controllo del flusso

- **Flusso di esecuzione:** ordine in cui le istruzioni di un programma sono eseguite
- Salvo contrordini, è in **sequenza**
- Due possibili alterazioni:
  - **selezione:** sceglie un'azione da una lista di due o più azioni possibili
  - **ripetizione:** continua ad eseguire un'azione fino a quando non si verifica una condizione di termine

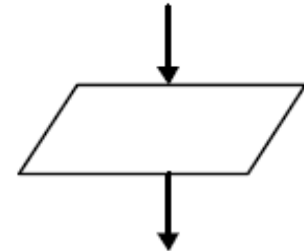
# Simbologia dei diagrammi di flusso



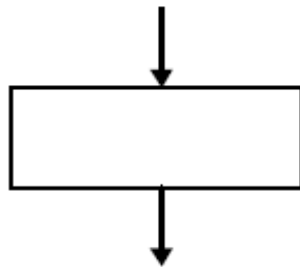
Inizio



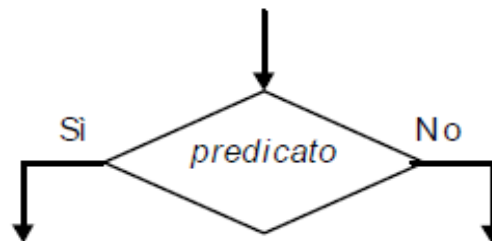
Fine



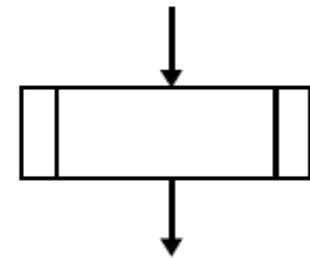
Operazioni  
di ingresso/uscita



Elaborazione



Selezione a due vie



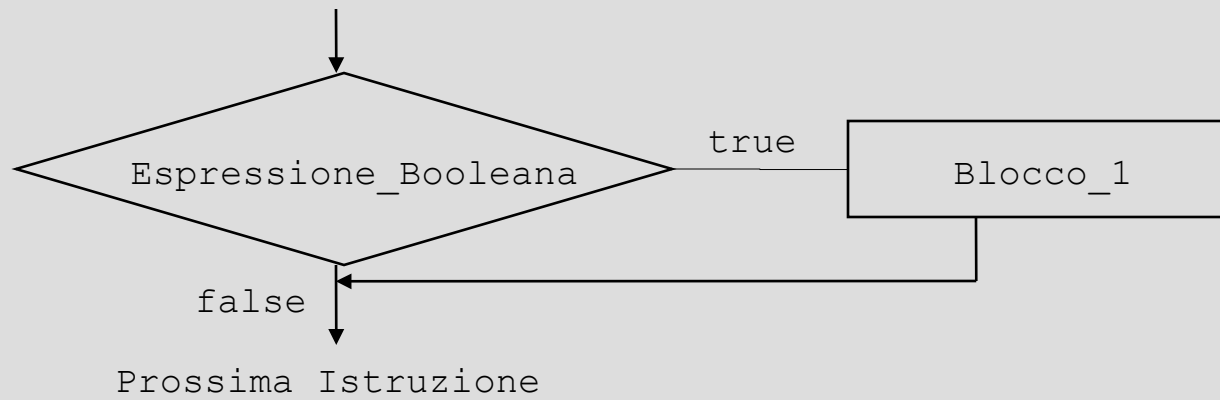
Sottoprogramma

- E' stato dimostrato che i programmi esprimibili tramite istruzioni di salto o diagrammi di flusso possono essere riscritti utilizzando le tre strutture di controllo (Teorema Boehm-Jacopini)

# Istruzione if

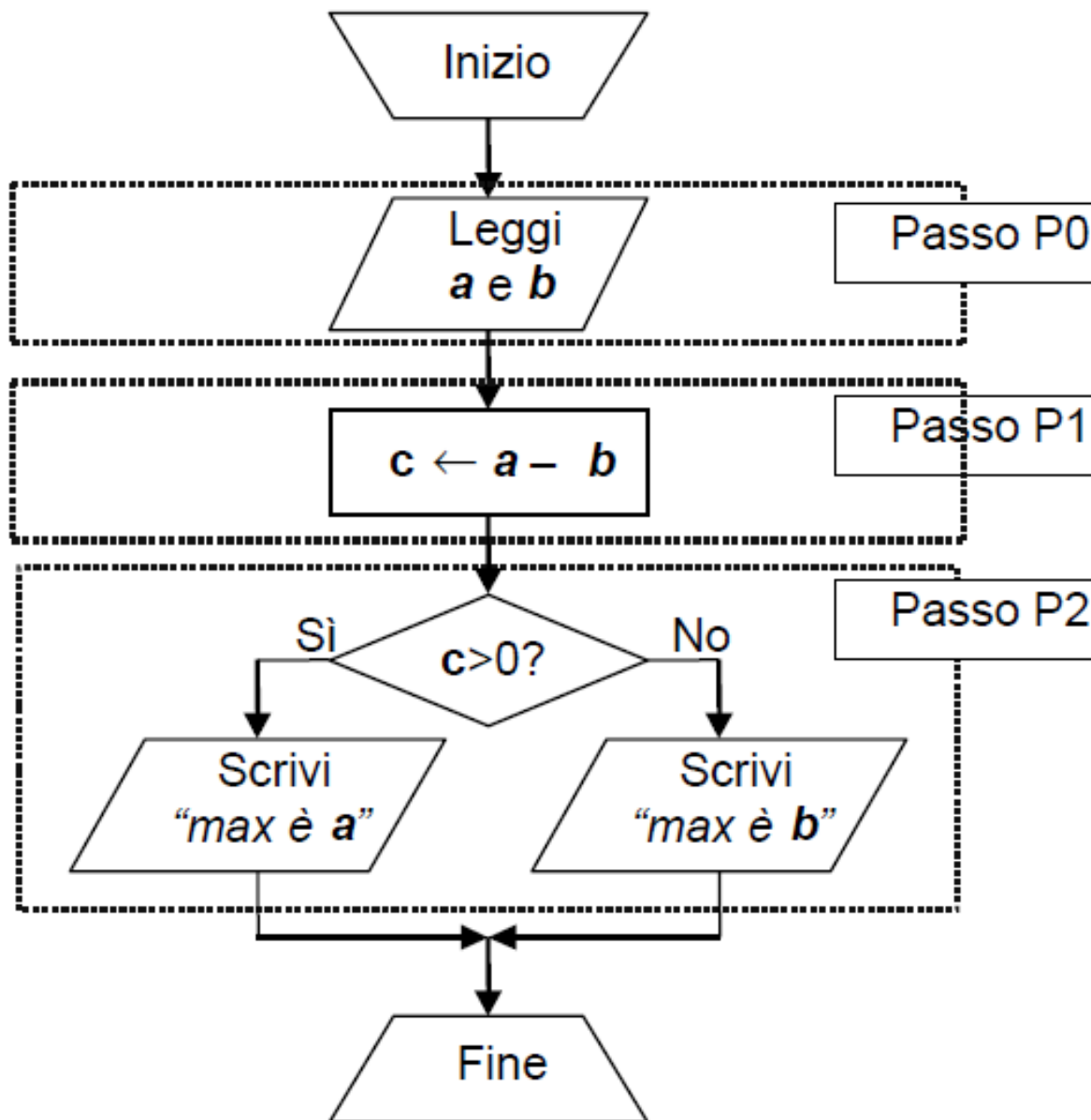
- Selezione semplice:
  - esegue un'azione se e solo se una certa condizione è verificata
- Sintassi:

```
if (Espressione_Booleana)  
    Blocco_1 //esegui solo se vera  
Prossima_Istruzione; //sempre eseguita
```



# Esempio

- Dati in ingresso due numeri A e B, si calcoli e stampi il maggiore.



# Istruzioni di *selezione* - L'istruzione **if**

La forma generale di **if** quando riferita a blocchi di istruzioni e' la seguente:

```
if(espressione)  
    Blocco_1 //esegui solo se vera  
else  
    Blocco_2 //esegui solo se falsa  
  
Prossima_Istruzione; //sempre eseguita
```

# Programma div.c

```
#include <stdio.h>
main(){
    int a,b;      /* variabili dividendo e divisore */
    printf("Inserisci il dividendo");
    scanf("%d",&a);
    printf("Inserisci il divisore");
    scanf("%d",&b);
    if(b) /* controlla se b e' zero */
    {
        printf("%d\n",a/b);
    }
    else
    {
        printf("Impossibile dividere per 0\n")
    }
}
```



# Programma div.c (2)

La stessa istruzione si potrebbe anche scrivere in questo modo:

```
if(b == 0)
{
    printf("Impossibile dividere per 0\n");
}
else
{
    printf("%d\n", a/b);
}
```

forma ridondante e potenzialmente inefficiente,  
sconsigliata

# Istruzioni `if` annidate

Un `if` *annidato* e' un'istruzione `if` controllata da un altro `if` o `else`. Gli `if` annidati sono molto comuni in programmazione. La cosa piu' importante da ricordare e' che un istruzione `else` si riferisce sempre all'istruzione `if` piu' vicina che sia all'interno del medesimo blocco dell'`else` e che non sia gia' associata a un `else`.

# Esempio in pseudo-C

```
if(i)
{
    if(j) istruzione1;
    if(k) istruzione2; /* questo if e' associato*/
    else                /* a questo else */
        istruzione3;
}
else istruzione 4;
/* questo if e' associato a if(i) */
```

# if-else-if

Un costrutto di programmazione a cui si ricorre frequentemente e che si basa sugli `if` annidati e' la sequenza `if-else-if`.

```
if(condizione)
    istruzione;
else if(condizione)
    istruzione;
else if(condizione)
    istruzione;
.
.
else
    istruzione;
```

Le espressioni condizionali vengono valutate dall'alto verso il basso. Alla prima condizione vera, viene eseguita l'istruzione ad essa associata ed il resto della scala `if-else-if` viene aggirato.

Se nessuna delle condizioni è vera allora viene eseguita l'istruzione `else` finale (se è presente).

# Istruzione (malsana) switch

- Istruzione switch:

```
switch (Espressione_Di_Controllo)
{
    case Etichetta_1:
        Sequenza_Istruzioni_1
    case Etichetta_2:
        Sequenza_Istruzioni_2
    ...
    case Etichetta_n:
        Sequenza_Istruzioni_n
    default:
        Sequenza_Istruzioni_Default
}
```

# Operatore condizionale

- È un operatore ternario:
- `espr1 ? espr2 : epr3;`

```
if (espr1)
    espr2;
else
    espr3;
```

Ad esempio:

```
z = (x>y) ? x : y;
```

# L'operatore ,

- Assegna il valore di b a x, poi incrementa a e poi incrementa b:
- **`x = (a++, b++);`**

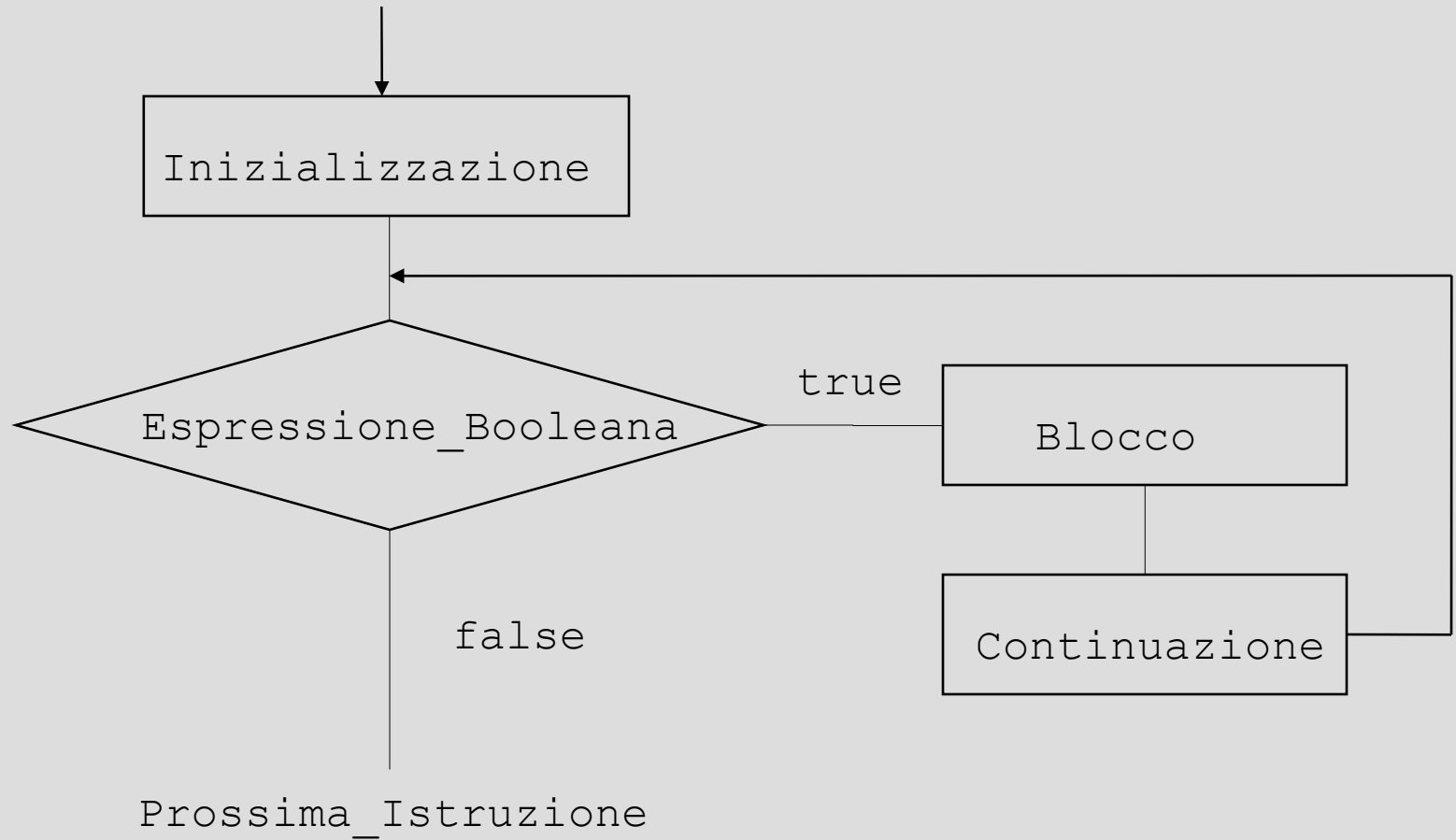
# Istruzioni di Iterazione

Fanno parte di questa classe di istruzioni i cicli:

- `for`
- `while`
- `do-while`



# Il ciclo for



# Il ciclo **for** (1)

Viene utilizzato per ripetere un'istruzione un numero specificato di volte.

```
for(inizializzazione;espressione;incremento) istruzione;
```

Per ripetere un blocco di istruzioni la forma generale è:

```
for(inizializzazione;espressione;incremento)
{
    istruzione1;
    .
    .
    istruzionen;
}
```

# Il ciclo **for** (2)

- *L'inizializzazione*: di solito un'istruzione di assegnamento che imposta il valore iniziale della *variabile di controllo del ciclo*, che ha la funzione di contatore.
- Per *espressione* si intende un'espressione condizionale che determina se il ciclo continuerà (se vera) oppure no (se falsa).
- *L'incremento* definisce la quantità di cui la variabile di controllo cambierà ogni volta che il ciclo viene iterato.
- devono essere separate da punti e virgola.

# Il ciclo for (3)

```
/* programma che stampa i caratteri ASCII dal 33 al 127 */  
#include <stdio.h>  
  
main()  
{  
    unsigned char i; /* definisco la variabile i di  
iterazione del ciclo */  
    for(i=33;i<128;i++) /* ciclo for con indice i*/  
    {  
        printf("[ %c ]\n",i);/* stampa un  
carattere racchiuso tra parentesi quadre, ad ogni ciclo  
*/  
    }  
}
```

# Il ciclo `while` (1)

La sintassi è:

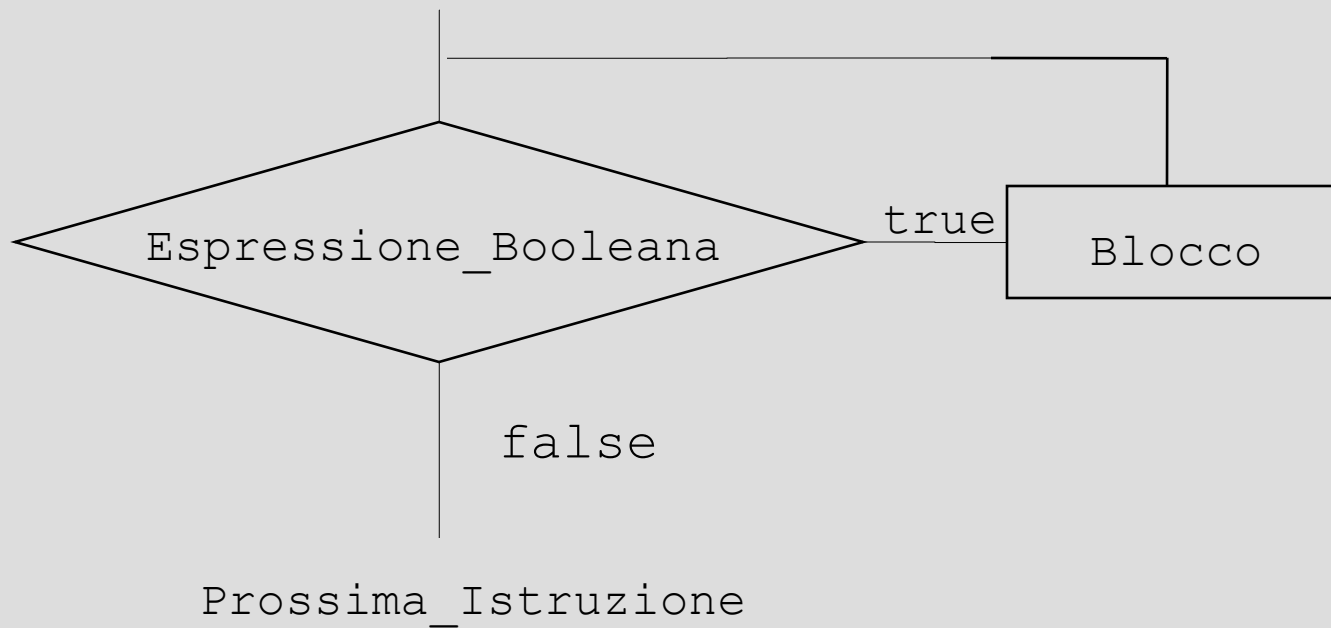
```
while(espressione) istruzione;
```

oppure,

```
while(espressione)
{
    istruzione_1;
    .
    .
    istruzione_n;
}
```

- L'*espressione* definisce la condizione che controlla il ciclo
- Viene valutata **prima** di eseguire il blocco di istruzioni

# Istruzione while



## Il ciclo **while** (2)

- È essenzialmente equivalente ad un ciclo **for** senza inizializzazione e l'incremento:
- **for ( ; condizione ; )**
- è equivalente a
- **while (condizione)**

# Il ciclo `while` (3)

```
/* programma che stampa i caratteri ASCII dal 33 al 127 */
#include <stdio.h>

main()
{
    unsigned char ch=33; /* inizializzo la variabile al
carattere codice ASCII 33*/

    while(ch<128)
    { /* ciclo while controllato dall'espressione ch<128
*/
        printf("[ %c ]\n",ch); /* stampa il singolo
carattere ch */
        ch++; /* incrementa il valore del carattere */
    }
}
```



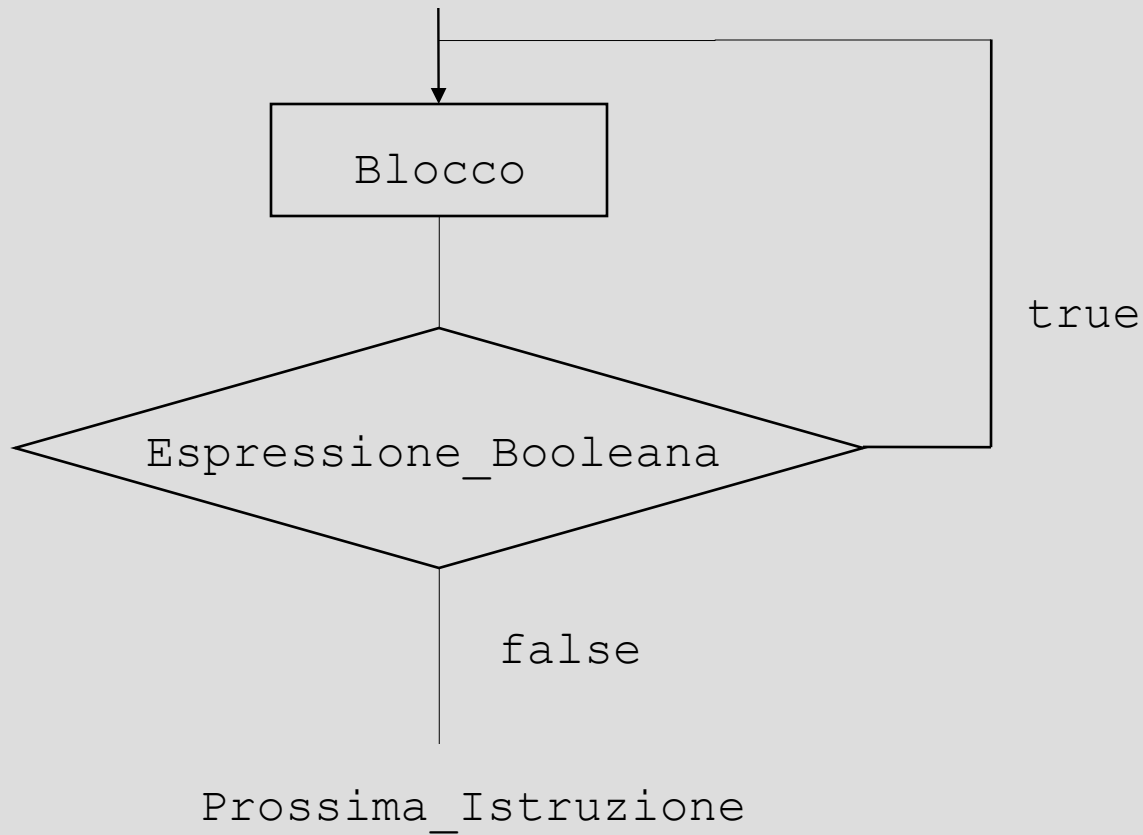
# Il ciclo **do-while** (1)

A differenza dei cicli **for** e **while**, nei quali le condizioni sono verificate in testa al ciclo, il ciclo **do-while** verifica la condizione in fondo al ciclo. Ciò significa che un ciclo **do-while** verrà sempre eseguito almeno una volta.

```
do {  
    istruzioni;  
} while(espressione);
```

Il ciclo viene eseguito fino a che la condizione espressa dall'*espressione* è vera. (se è falsa esce)

# Istruzione do-while



# Il ciclo do-while (2)

```
/* stampa i caratteri ascii dal 33 al 128 */
```

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    unsigned char ch=33;
```

```
    /* il ciclo do-while viene sempre eseguito almeno una volta. Se ch=128 il ciclo verrebbe eseguito sempre e comunque una volta soltanto perche' la condizione ch<128 viene controllata a fine ciclo */
```

```
    do{
```

```
        printf("[ %c ]\n",ch);
```

```
        ch++;
```

```
    }while(ch<128);
```

```
}
```

# Istruzione di salto **break** (1)

Questa istruzione consente di uscire forzatamente da un ciclo aggirando la verifica condizionale.

Quando in un ciclo si incontra l'istruzione **break**, esso termina immediatamente ed il controllo del programma riprende dall'istruzione successiva al ciclo

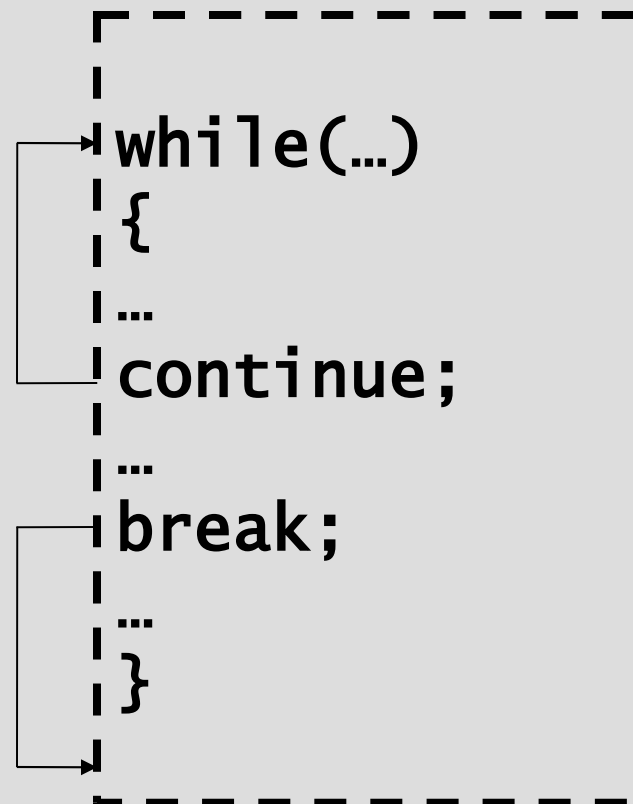
# Istruzione di salto **continue**

## (1)

Serve per ottenere forzatamente l'iterazione anticipata di un ciclo aggirando la sua normale struttura di controllo. L'uso di **continue** all'interno di un ciclo provoca forzatamente il salto alla successiva iterazione del ciclo stesso saltando il codice che si trova tra il **continue** e l'espressione condizionale che controlla il ciclo.

# Istruzione di salto **continue** (2)

```
/* stampa I numeri pari da 0 a 20 */  
#include <stdio.h>  
  
main()  
{  
    unsigned int i;  
  
    for(i=0;i<=20;i++) /* iteriamo da 0 a 20 compresi */  
    {  
        if(i%2) continue; /* se il numero e' dispari  
salta alla successiva iterazione del ciclo */  
        printf(" %d \n",i); /* stampa i numeri */  
    }  
}
```



```
while(...)
{
...
continue;
...
break;
...
}
```

The diagram illustrates a while loop structure. A dashed rectangular box encloses the code. On the left side, two vertical lines with arrows at the top and bottom point to the start and end of the loop body, respectively, indicating the loop's repetition. The code inside the box is as follows:

- `while(...)`
- `{`
- `...`
- `continue;`
- `...`
- `break;`
- `...`
- `}`

# Cicli infiniti

- Cause principali:
  - errata espressione Booleana
  - errata (o assente) alterazione delle variabili coinvolte nell'espressione Booleana
- Esempio:

```
int total = 0;
int count = 1;
while (count != 10)
{
    total = total + count;
    count += 2;
}
```



# Esempi ed esercizi

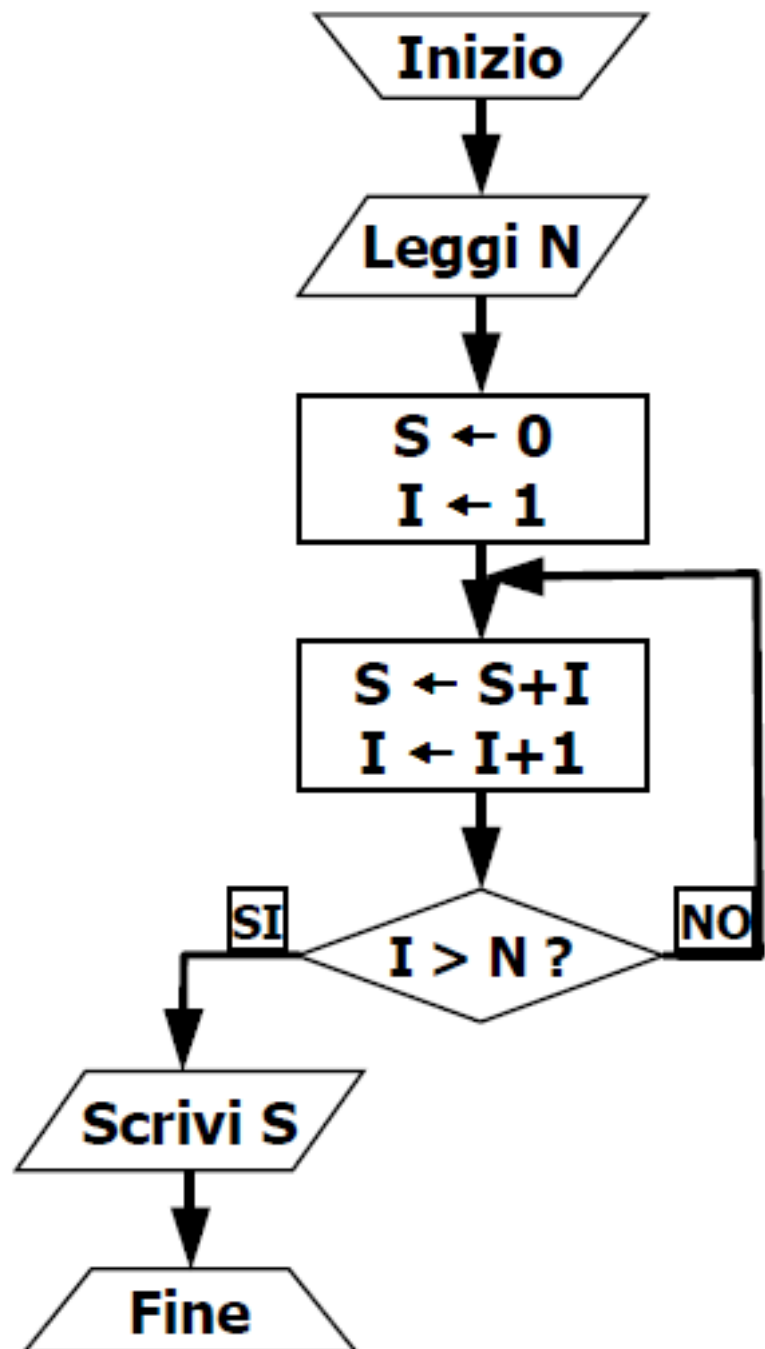
Somma dei primi  $n$  numeri

Calcolo del fattoriale

Ricerca di numeri primi

# Esempio

- Calcolare e poi stampare la somma dei primi  $N$  numeri naturali

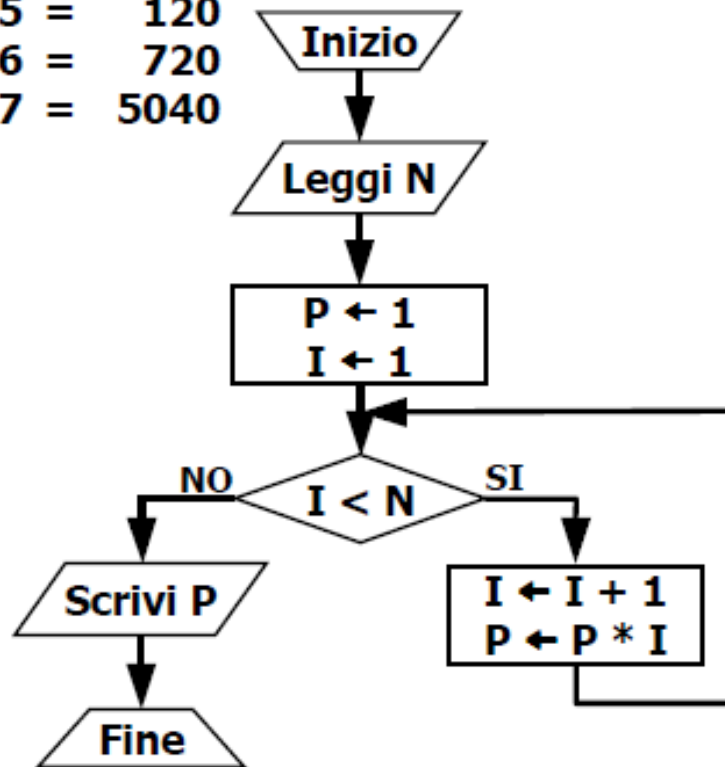


# Esercizio svolto

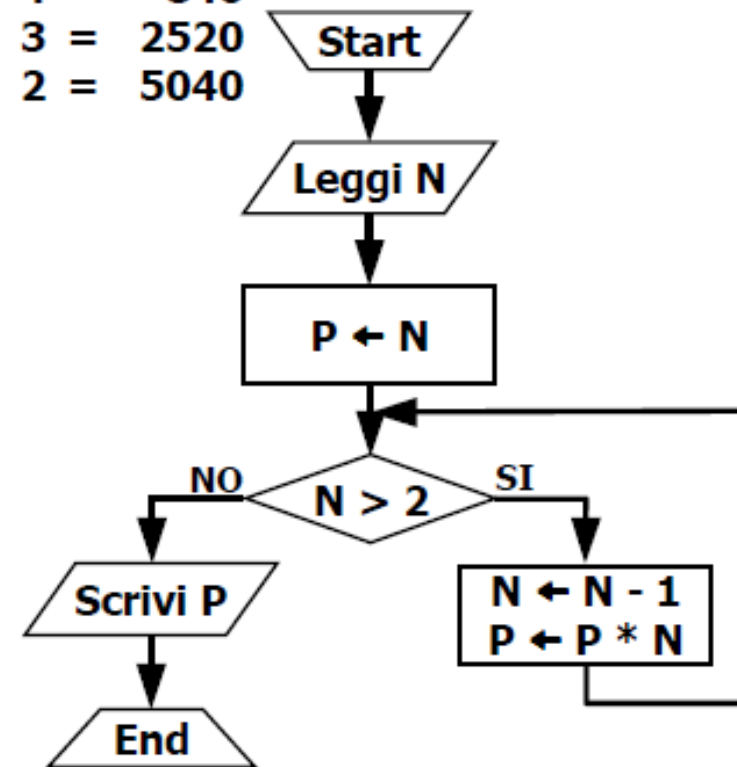
- L'esecutore deve leggere un intero  $N$  e restituire il fattoriale di questo numero, cioè il valore ottenuto da  $N \times (N-1) \times (N-2) \times \dots \times 1$ .
- Scrivere l'algoritmo immaginando che i dati di ingresso siano sempre corretti (cioè sempre maggiori di zero).
- Modificare l'algoritmo in modo da considerare anche la possibilità che siano inseriti valori inferiori a 1.

# Diverse alternative

1	*	2	=	2
2	*	3	=	6
6	*	4	=	24
24	*	5	=	120
120	*	6	=	720
720	*	7	=	5040

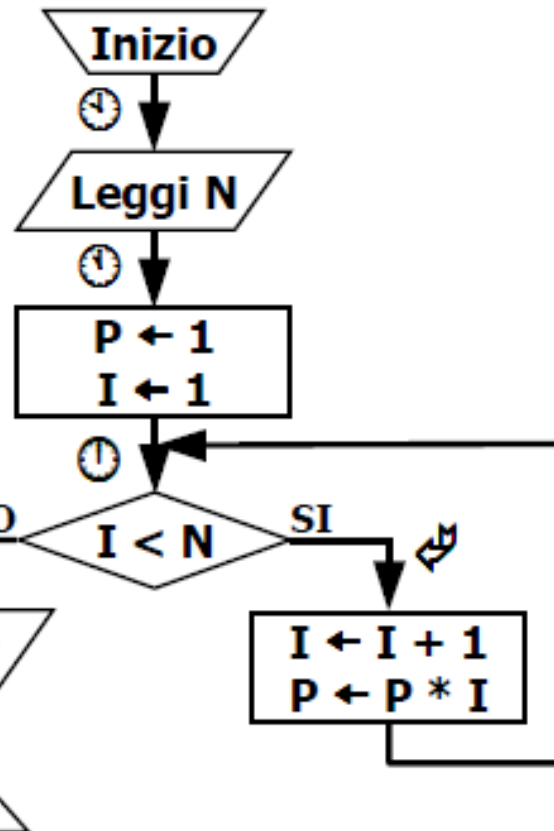


7	*	6	=	42
42	*	5	=	210
210	*	4	=	840
840	*	3	=	2520
2520	*	2	=	5040



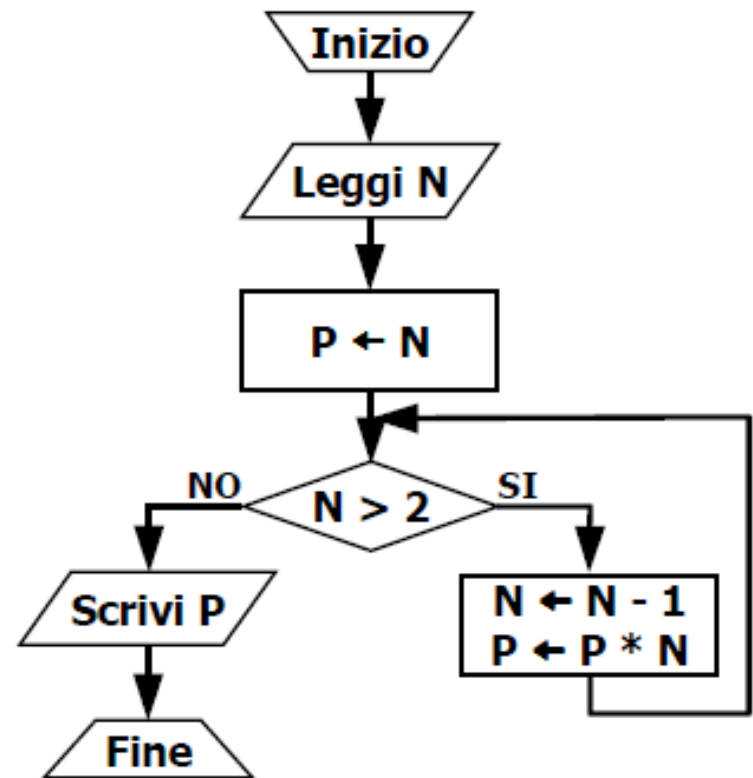
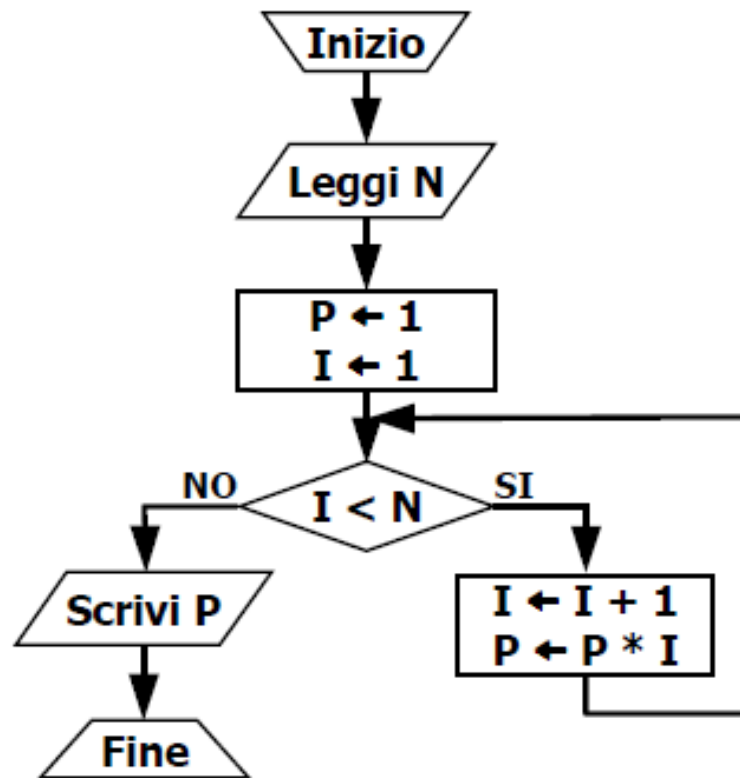
# "Tracciato" dell'esecuzione

Ipotizziamo di calcolare 4!



T	pos	N	I	P	note
t <sub>1</sub>	⌚	??	??	??	
t <sub>2</sub>	⌚	4	??	??	
t <sub>3</sub>	⌚	4	1	1	I < N
t <sub>4</sub>	↩	4	1	1	
t <sub>5</sub>	⌚	4	2	2	I < N
t <sub>6</sub>	↩	4	2	2	
t <sub>7</sub>	⌚	4	3	6	I < N
t <sub>8</sub>	↩	4	3	6	
t <sub>9</sub>	⌚	4	4	24	I = N
t <sub>10</sub>	↩	4	4	24	
t <sub>11</sub>	↩	4	4	24	

# Le alternative sono "diverse"?



**Cosa succede se il dato in ingresso non rispetta le specifiche ( $N > 0$ )?  
Per esempio, che risultato restituisce l'esecutore per  $N = 0$  e per  $N = -4$ ?**

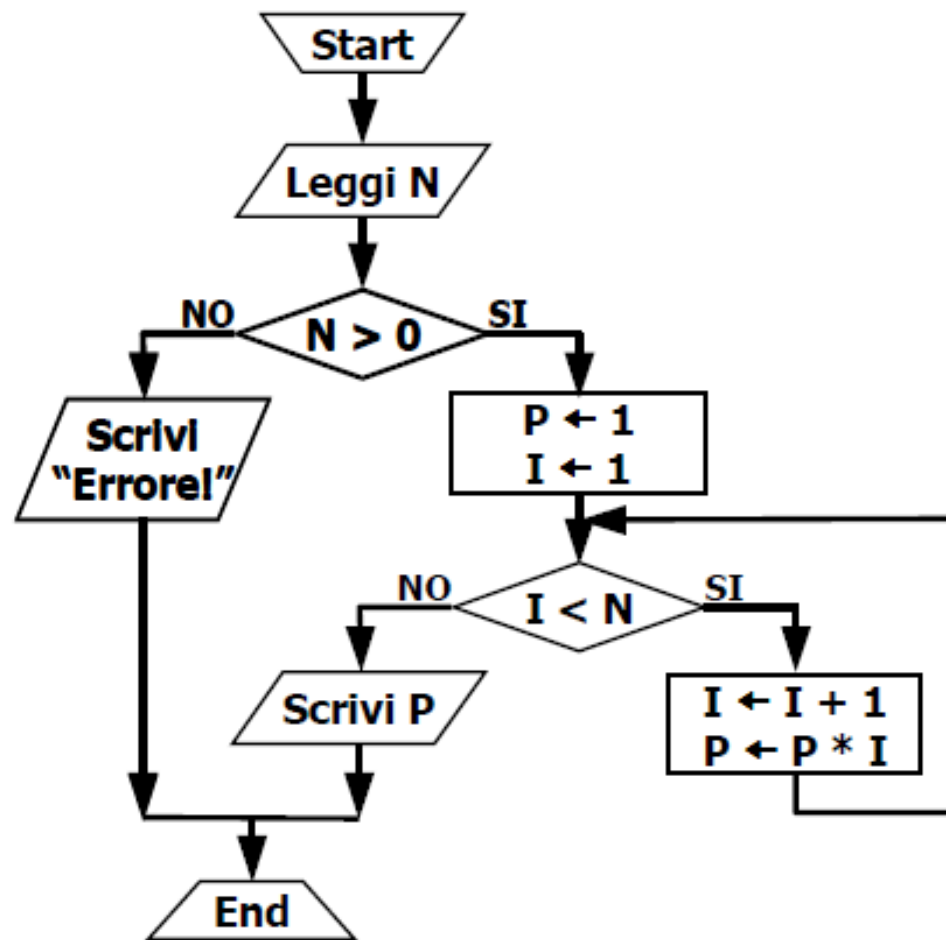
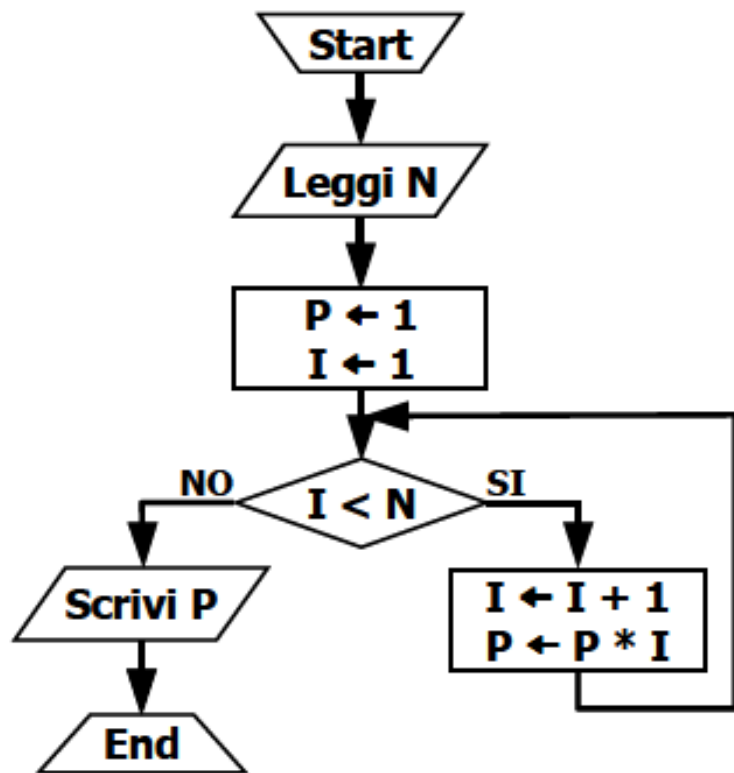
$$N = 0 \rightarrow P = 1$$

$$N = -4 \rightarrow P = 1$$

$$N = 0 \rightarrow P = 0$$

$$N = -4 \rightarrow P = -4$$

# Come gestire le "eccezioni"



Algoritmo per il caso "normale".

Come lo modifico per gestire anche i casi che non erano stati previsti?