Laboratorio di Programmazione

08:30 – 12:30 lunedì – aula Omega e Sigma

08:30 – 10:30 martedì – aula Omega e Delta

Lezione 2– Cicli e concetti introduttivi

Marco Anisetti (teoria)
Dipartimento di Informatica
marco.anisetti@unimi.it

Matteo Luperto (lab. turno A)
Dipartimento di Informatica
matteo.luperto@unimi.it

Nicola Bena (lab. turno B)
Dipartimento di Informatica
nicola.bena@unimi.it

Recap di Lezione 1

- Compilazione
  gcc -o eseguibile sorgente.c
- Variabili – tipizzate – nome con lettera minuscola
  `int somma = 0;`
  - Dichiarate a inizio programma
  - Da inizializzare prima del primo uso
- Tipi primitivi
  `int char float double`
- Tabella ASCII, un `char` è in realtà un `int`
- Costanti – tipizzate, immutabili – lettere MAIUSCOLE
  ```
  #define N 10
  #define LETTERA 'A'
  ```
- Dare a variabili e COSTANTI nomi significative
- Placeholder tipizzati: `%d` → `int` `%c` → `char`
- Caratteri di escape `\n` → new line

Decimal	Hexadecimal	Binary	Octal	Char
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_

Lezione	Data	Ore	
1	6/10	4	Parte 1: Introduzione alla programmazione – in C
2	13/10	4	
3	14/10	2	
4	20/10	4	
5	21/10	2	
6	10/11	4	Parte 2: Programmazione imperativa – in C
7	11/11	2	
8	17/11	4	
9	18/11	2	
10	24/11	4	
11	25/11	2	
12	1/12	4	Parte 3: Introduzione alla programmazione ad oggetti – in Java
13	2/12	2	
14	9/12	2	
15	15/12	4	
16	16/12	2	

Input/output: `scanf` e `printf`

- Sono definite in `stdio.h` e vengono usate per leggere gli input e scrivere gli output del programma
- Basate su una *stringa di formato* e una serie di variabili il cui valore viene scritto su `stdout` (`printf`) o letto da `stdin` e assegnato alle variabili (`scanf`)
- La *stringa di formato* serve per *tipizzare* la lettura/scrittura delle variabili mediante dei placeholder
- `%d` → `int`
- `%c` → `char`
- `%f` → `float`
- `%lf` → `double`

Stampare a schermo con `printf`

- `printf(<format string>, var1, var2, …, varn)`
  - `Format string` è la stringa di formato con i placeholder che definiscono i tipi delle variabili da mostrare (racchiusa tra apici)
  - `var1, …, varn` sono le variabili (o costanti) il cui valore viene mostrato su `stdout` secondo il placeholder che abbiamo definito
  - Funzione particolare con numero di parametri variabili

```
#define PI 3.14
int num=0;
char c = 'a';
printf("PI: %lf, num: %d, c: %c\n", PI, num, c);
```

Format string

Variabili/costanti

Stampare a schermo con `printf`

- `printf(<format string>, var1, var2, …, varn)`
  - `format string` è la *stringa di formato* con i placeholder che definiscono i tipi delle variabili da mostrare (racchiusa tra apici)
  - `var1, …, varn` sono le variabili/costanti il cui valore viene mostrato su `stdout` secondo il placeholder che abbiamo definito
  - Funzione particolare con numero di parametri variabili

```
#define PI 3.14 // è double (3.14f per float)
int num=0;
char c = 'a';
printf("PI: %lf, num: %d, c: %c\n", PI, num, c);
```

Format string

Variabili/costanti

Funzione `printf`

- `printf(<format string>, var1, var2, …, varn)`
  - `Format string` è la stringa di formato con i placeholder che definiscono i tipi delle variabili da mostrare (racchiusa tra apici)
  - `var1, …, varn` sono le **espressioni** il cui valore viene mostrato su `stdout` secondo il placeholder che abbiamo definito
  - Funzione particolare con numero di parametri variabili

```
#define PI 3.14
printf("Il doppio di PI: %lf\n", PI*2);
```

Format string

Espressioni

Funzione `printf`

- `printf(<format string>, var1, var2, …, varn)`
  - `Format string` è la stringa di formato con i placeholder che definiscono i tipi delle variabili da mostrare (va racchiusa tra apici)
  - `var1, …, varn` sono le **espressioni** il cui valore viene mostrato su `stdout` secondo il placeholder che abbiamo definito
  - ***Undefined behavior* quando il tipo del placeholder e dell'espressione non corrispondono** (lo vedremo in dettaglio)

```
int num=3;
printf("as int %d, as double: %lf\n", num, num);
```

```
> gcc -o es03_printf_ub es03_printf_ub.c
es03_printf_ub.c:5:48: warning: format specifies type 'double' but the argument has type 'int' [-Wformat]
    5 |     printf("as int %d, as double: %lf\n", num, num);
      |                                    ~~~          ^~~
      |                                    %d
1 warning generated.
```

← Il compilatore vi aiuta

```
> ./es03_printf_ub
as int 3, as double: 0.000000
```

Funzione `getchar`

- È definita in `stdio.h` e viene usata per leggere *un singolo carattere* da `stdin`
- Restituisce una variabile di tipo `char`
  - Formalmente restituisce una variabile di tipo `int`
  - L'assegnamento a una variabile di tipo char applica una conversion implicita (errori possibili!)
- Possiamo usarla per leggere l'input dell'utente

Funzione `getchar`

```
#include <stdio.h>


/* Scrivere un programma che chiede all'utente 2 caratteri in input e li stampa */

int main(){
	char c1, c2;
	printf("Inserisci il primo carattere: ");
	c1 = getchar();
	printf("Inserisci il secondo carattere: ");
	c2 = getchar();
	printf("Hai inserito \'%c\' e \'%c\'\n", c1, c2);
}
```

Funzione `getchar`

```
#include <stdio.h>


/* Scrivere un programma che chiede all'utente 2 caratteri in input e li
stampa */

int main(){
        char c1, c2;
        printf("Inserisci il primo carattere: ");
        c1 = getchar();
        printf("Inserisci il secondo carattere: ");
        c2 = getchar();
        printf("Hai inserito \'%c\' e \'%c\'\n", c1, c2);
}
```

Funzione `getchar`

```
#include <stdio.h>


/* Scrivere un programma che chiede all'utente 2 caratteri in input e li
stampa */

int main(){
        char c1, c2;
        printf("Inserisci il primo carattere: ");
        c1 = getchar();
        printf("Inserisci il secondo carattere: ");
        c2 = getchar();
        printf("Hai inserito \'%c\' e \'%c\'\n", c1, c2);
}
```

```
> ./es01_getchar
Inserisci il primo carattere: c
Inserisci il secondo carattere: Hai inserito 'c' e '
'
```

Funzione `getchar`

```c
#include <stdio.h>


/* Scrivere un programma che chiede all'utente 2 caratteri in input e li
stampa */

int main(){
        char c1, c2;
        printf("Inserisci il primo carattere: ");
        c1 = getchar();
        printf("Inserisci il secondo carattere: ");
        c2 = getchar();
        printf("Hai inserito \'%c\' e \'%c\'\n", c1, c2);
}
```

Legge `c` e lo assegna a `c1`

Legge il prossimo carattere da `stdin`, è il `newline` che abbiamo digitato dopo `c`, lo assegna a `c2`

```
❯ ./es01_getchar
Inserisci il primo carattere: c
Inserisci il secondo carattere: Hai inserito 'c' e '
'
```

Funzione `getchar`

```
#include <stdio.h>


/* Scrivere un programma che chiede all'utente 2 caratteri in input e li
stampa */

int main(){
        char c1, c2;
        printf("Inserisci il primo carattere: ");
        c1 = getchar(); getchar();
        printf("Inserisci il secondo carattere: ");
        c2 = getchar();
        printf("Hai inserito \'%c\' e \'%c\'\n", c1, c2);
}
```

Legge `newline` da `stdin` senza assegnarlo a nessuna variabile

Se avessimo altre chiamate a `getchar`, servirebbe anche qui

Funzione `scanf`

- È definita in `stdio.h` e viene usata per leggere da `stdin`
- Converte l'input secondo il formato che viene specificato
  - Possiamo leggere `int`, `char`, `float`, `double`…
- `scanf(<format string>, &var, &var2, …,&varn)`
  - `<format string>` è la *stringa di formato* con i placeholder che definiscono i tipi delle variabili che saranno assegnati (racchiusa tra apici)
  - Segue le regole della `printf`
  - `vari` è la variabile a cui viene assegnata la porzione dell'input letta secondo `format string`
  - Dobbiamo usare **&**`vari` (vedremo più avanti perché)
- Restituisce un `int` che indica il numero di elementi assegnati

Funzione `scanf`

- Scriviamo un programma che legge due numeri, un intero e un float separate da spazio, usando `scanf`

Funzione `scanf`

- Scriviamo un programma che legge due numeri, un intero e un float separati da spazio usando solo `scanf`

```
int n1;
float n2;
scanf("%d %f", &n1, &n2);
```

Funzione `scanf`

- Scriviamo un programma che legge due numeri, un intero e un float separati da spazio usando solo `scanf`

```
int n1;
float n2;
scanf("%d %f", &n1, &n2);
```

Funzione `scanf`

- Scriviamo un programma che legge due numeri, un intero e un float separati da spazio usando solo `scanf`, verificando che la lettura sia ok

```
int n1;
float n2;
int ret_val = scanf("%d %f", &n1, &n2);
```

- La variable `ret_val` contiene il numero di letture/assegnamenti effettuati dalla `scanf`

Funzione `scanf`

- Scriviamo un programma che legge due numeri, un intero e un float separati da spazio usando solo `scanf`, verificando che la lettura sia ok

```
int n1;
float n2;
int ret_val = scanf("%d %f", &n1, &n2);
```

- La variable `ret_val` contiene il numero di letture/assegnamenti effettuati dalla `scanf` 2 in questo caso
- Dobbiamo controllare il contenuto di `ret_val`…

La selezione

Costrutto di selezione in C

```
if (<espressione>) <blocco then>

if (<espressione>) <blocco then> else <blocco else>
```

- Blocco di codice: se è costituito da più di una istruzione è definito da { }
  Se è di una sola istruzione, si possono omettere le { }
- Di <espressione> si valuta il valore logico (vero o falso)
- Per C, una espressione, è per convenzione lavora un numero
  - (il tipo boolean VERO/ FALSO è stato introdotto dopo nel C99)
- Per il C il numero 0 è falso e qualsiasi altro numero è vero (per convenzione si intende l'1)

Espressione: una prima approssimazione

- *E' una sequenza di simboli a cui è associato un valore*
- Per il momento ci accontentiamo di supporre che siano valide le regole della matematica per ottenere il valore dell'espressione
- Abbiamo operatori relazionali (<,>,<=, >=, ==, !=)
- Operatori aritmetici (+,-,*,/,%)
- Operatori logici: (!, &&, ||, …)
- Si possono usare le parentesi tonde per chiarire le precedenze anche se esiste un ordine di precedenza

Logica Booleana

Val1	Val2	Operator	Result
1	1	&&	1
1	0	&&	0
1	1	||	1
1	0	||	1

Val	Operator	Result
1	!	0
0	!	1

Logica Booleana

Priorità alta	! ++ -- - (unario)
	* / %
	+ - (binario)
	> >= < <=
	== !=
	&&
	||
Priorità bassa	=

```
int a=0, b=0, c;
c = !a || b;
```

Logica Booleana

Priorità alta	! ++ -- - (unario)
	* / %
	+ - (binario)
	> >= < <=
	== !=
	&&
	||
Priorità bassa	=

```
int a=0, b=0, c;
c = !a || b;
```

not 0 → espressione vale 1

Logica Booleana

Priorità alta	! ++ -- - (unario)
	* / %
	+ - (binario)
	> >= < <=
	== !=
	&&
	||
Priorità bassa	=

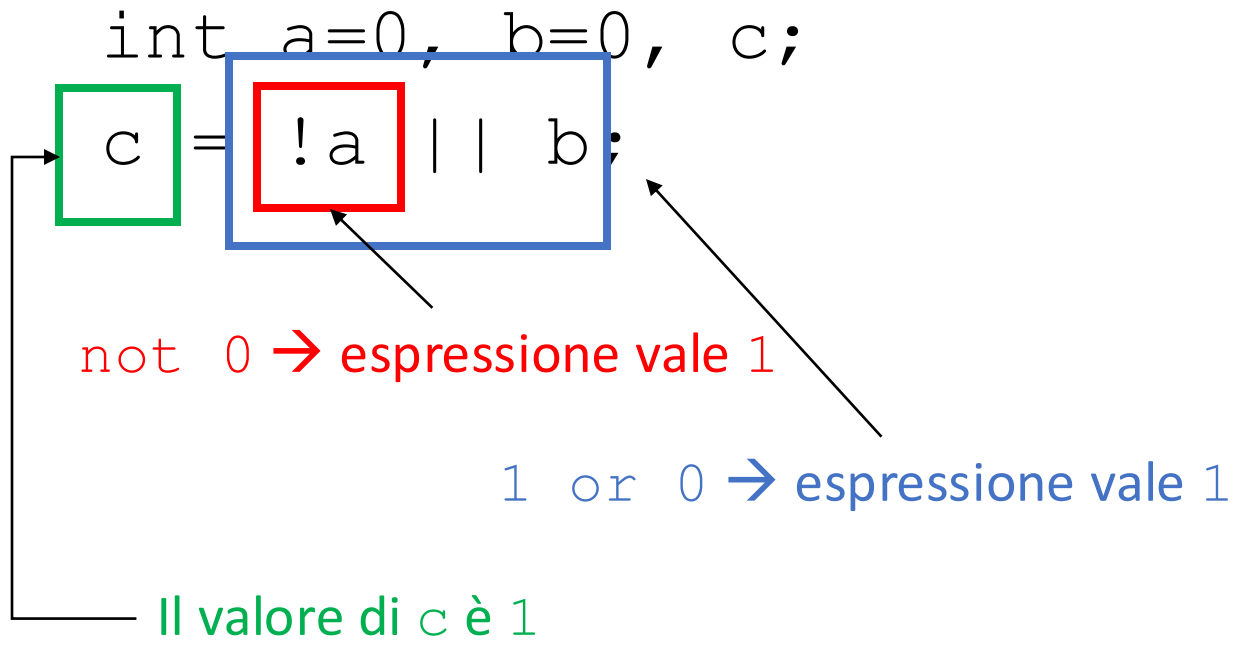
```
int a=0, b=0, c;
c = !a || b;
```

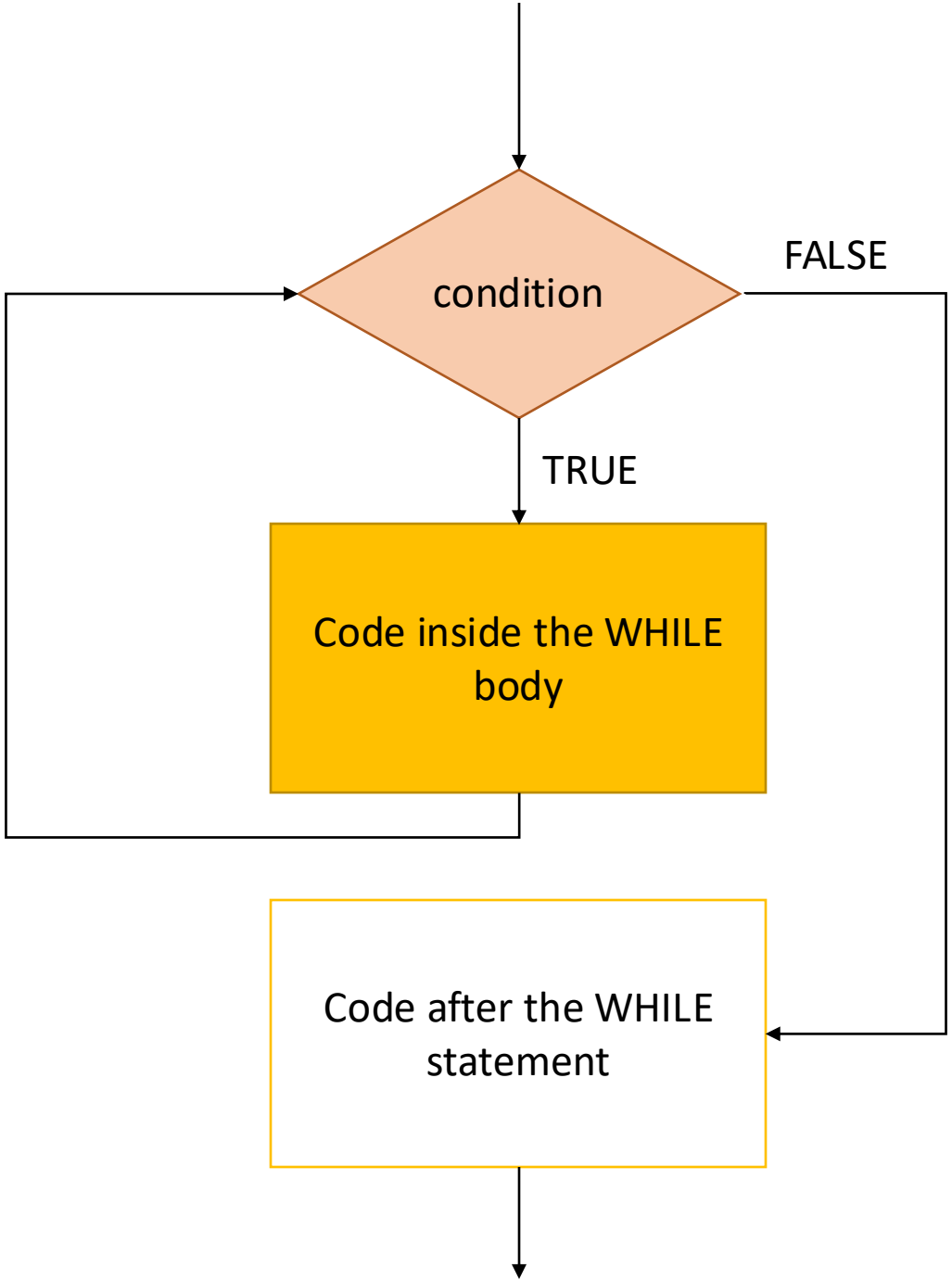
not 0 → espressione vale 1

1 or 0 → espressione vale 1

Logica Booleana

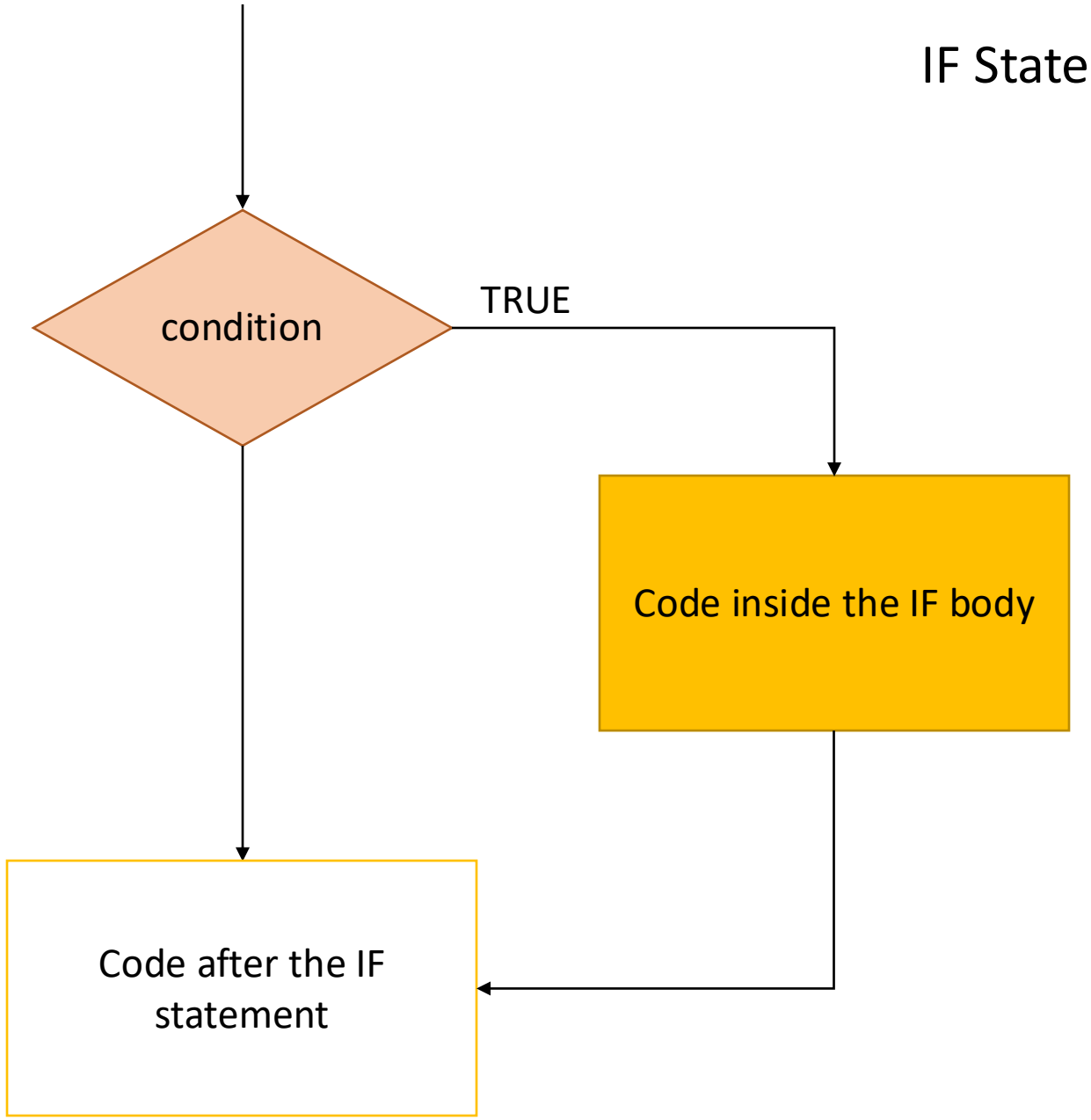
Priorità alta	! ++ -- - (unario)
	* / %
	+ - (binario)
	> >= < <=
	== !=
	&&
	||
Priorità bassa	=



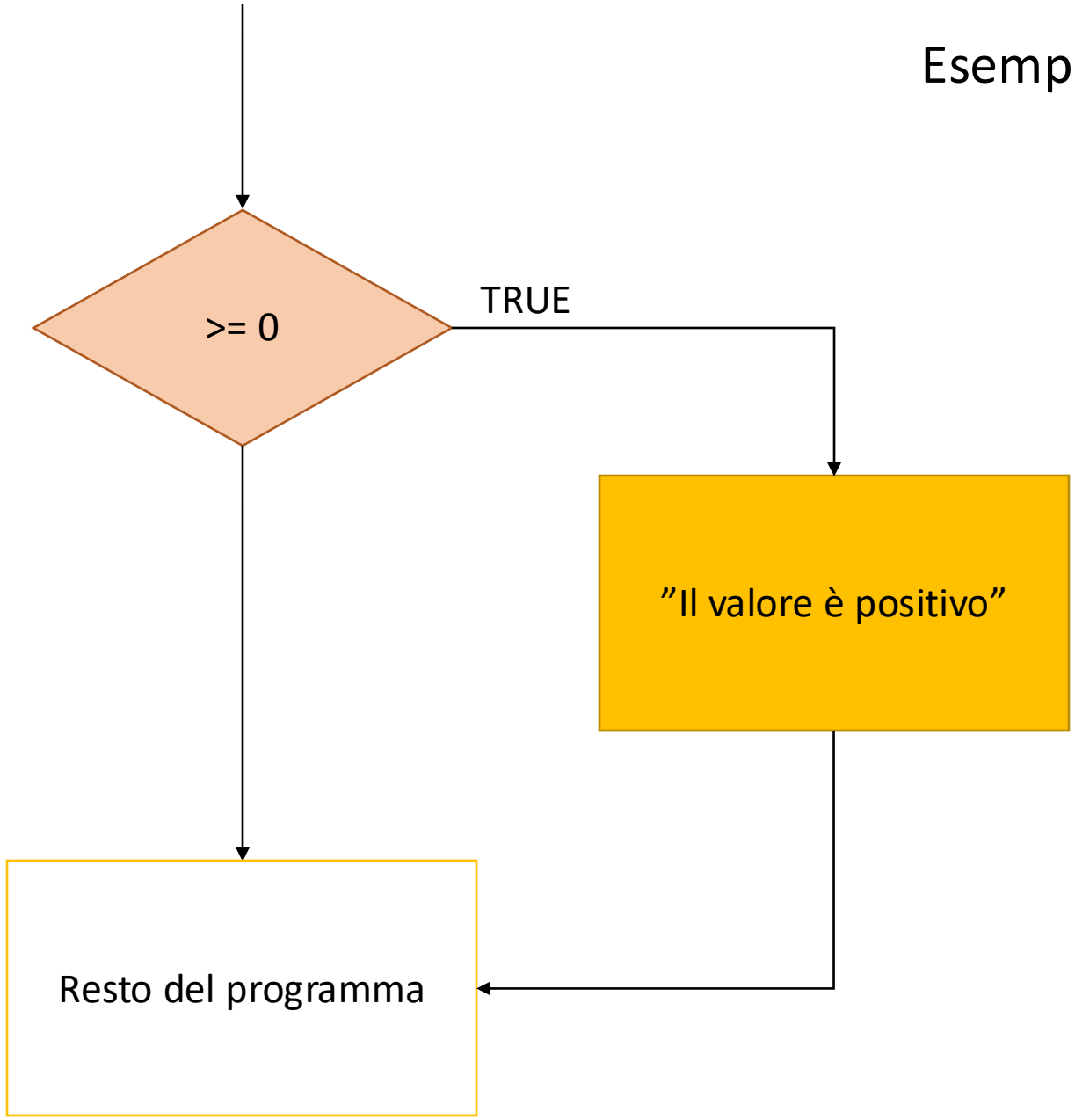


WHILE Statement

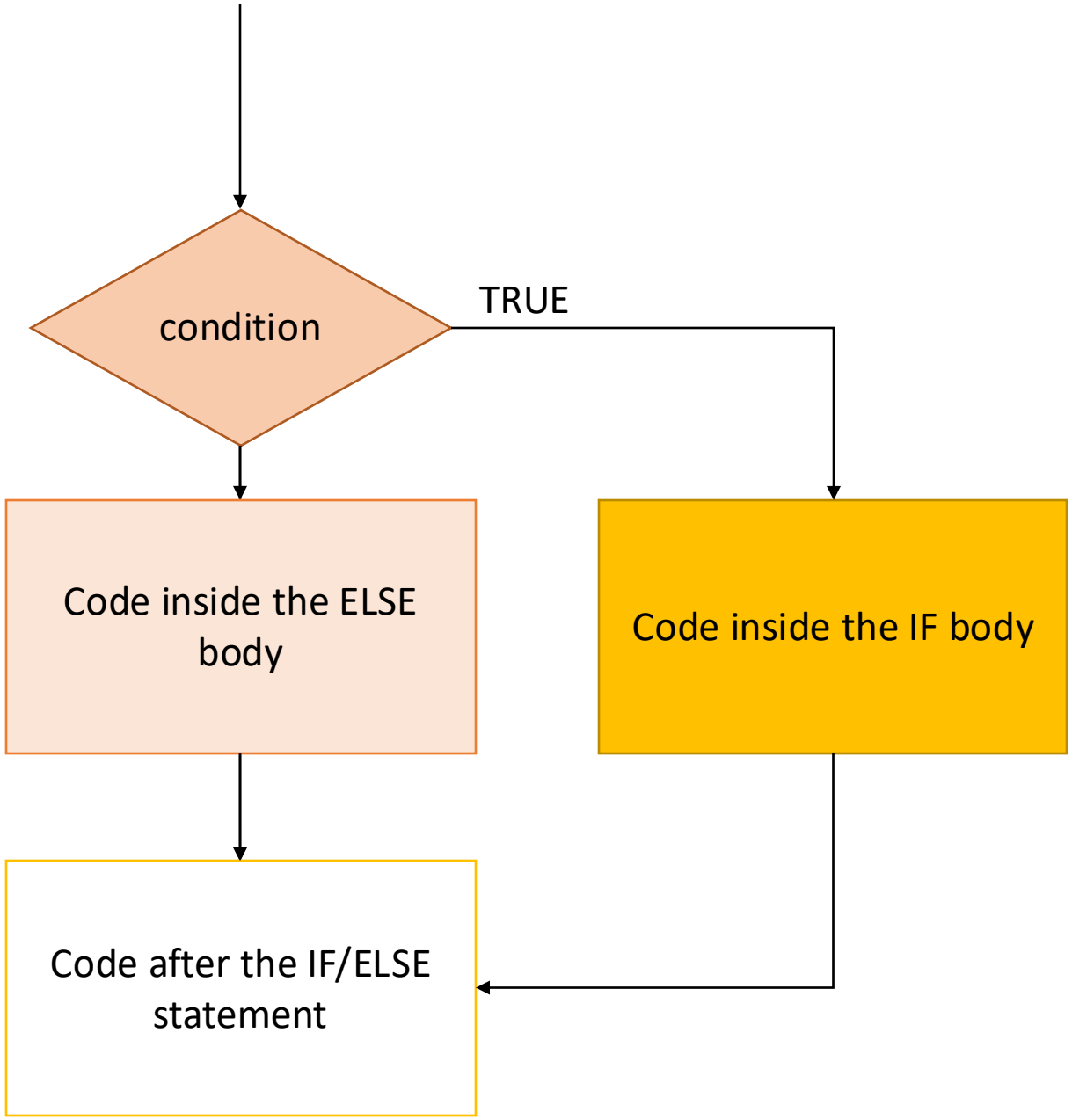
IF Statement



Esempio: check positivo

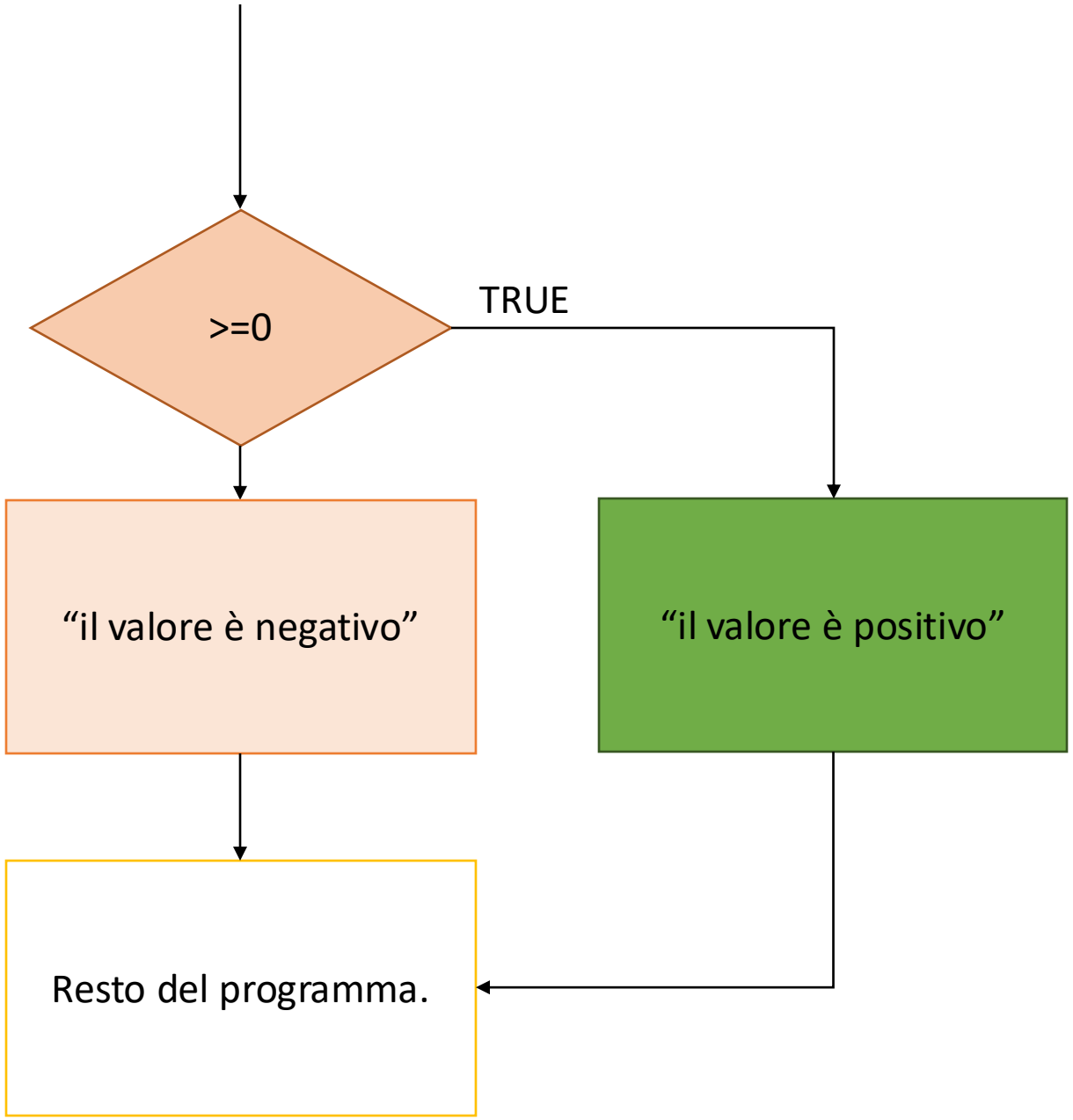


IF Statement




```
if (condition)
        // blocco di codice
        // condizione verificata
        // (Obbligatorio)
else
        // blocco di codice
        // condizione NON verificata
        // (Opzionale)
return 0;
}
```

Esempio: check positivo




```
if (valore >= 0)
        printf(“Il valore è positivo\n”);
else
        printf(“Il valore è negativo\n”);
```

Come impostare un programma

Quando scrivete un nuovo programma (di brevi dimensioni, come quelli oggetto di questo corso), cercate di dividere logicamente le parti principali del programma (dove possibile)

1. Dichiarazione delle variabili
2. Ottenimento degli input (es, `scanf`)
3. Fase principale – calcolo
4. Restituzione degli output (es, `printf`)
5. Conclusione: `return`

Inoltre

1. Cercate di scomporre il programma in più funzioni/sottoprogrammi
2. Evitate la duplicazione del codice
   (se dovete duplicare il codice in più parti, è un buon indizio che potete usare una funzione al posto del codice duplicato)

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
	/* Questo è un commento, può essere su più linee */
	// Questo è un commento che sta su una linea sola
	int numero;

	printf("Inserisci un valore intero\n");
	scanf("%d",&numero);

	if (numero < 0)
		numero = -numero;

	printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
	return 0;
}
```

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Commentare il codice ne aumenta la *leggibilità*, e vi aiuta a trovare eventuali errori

Permette di specificare assunzioni fatte

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Fase iniziale/1:
dichiaro le variabili

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Fase iniziale/2:
inizializzo le variabili

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Quando utilizzate la scanf, anteponete al nome della variabile una “&”.
Questo perché alla scanf dovete fornire un *puntatore* alla variabile (concetto che vedremo più avanti)

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Parte principale del programma

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Indentare il codice ci fa capire che l'istruzione è parte di questo `if`

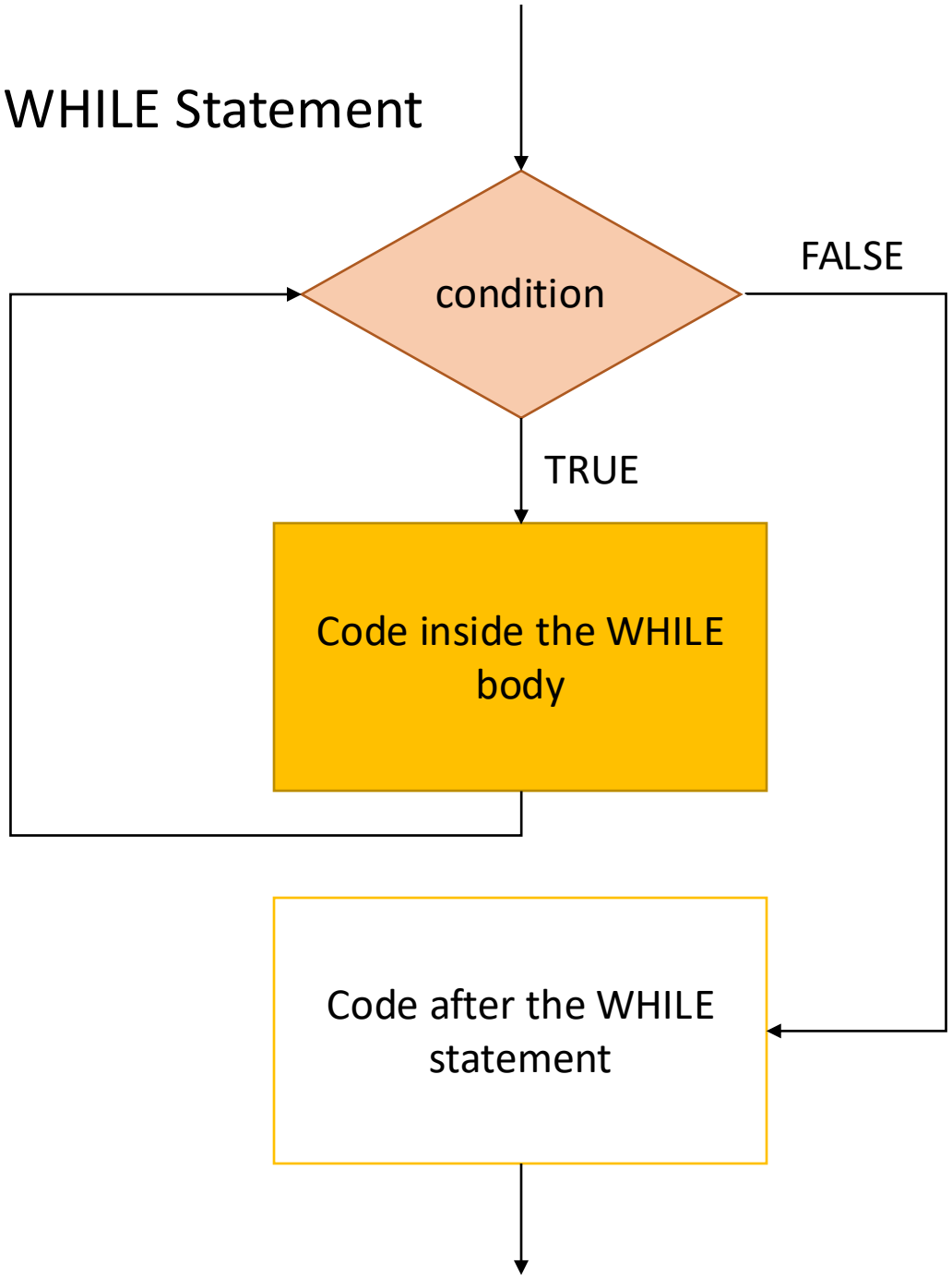
```c
#include <stdio.h>
#include <stdlib.h>

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Parte conclusiva:
restituisco il risultato

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Il programma termina segnalando che è andato tutto bene

WHILE Statement



Il ciclo While

Costrutto di loop while in C

```
while (<espressione>) <blocco>
```

- Blocco di codice: se è costituito da più di una istruzione è definito da { }
  Se è di una sola istruzione, si possono omettere le { }
- Di `<espressione>` si valuta il valore logico (vero o falso)
- Il ciclo verrà eseguito fino a che `<espressione>` è vera
- Attenzione! **«;»** è una singola istruzione (vuota)

Il ciclo While

```
#include <stdio.h>

void main(){
        //infinite loop
        while(true){
                //do some stuff
                printf("I am in a loop\n");
        }
        printf("This will never be printed.\n");
}
```

Il ciclo While

```
int x = 0;

while (x < 10) {
        printf ("x = %d\n", x);
        x++;
}

printf("ho finito!\n");
```

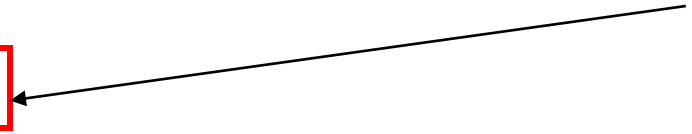
Inizializzo la variabile che utilizzerò in <espressione>

Il ciclo While

```
int x = 0;

while (x < 10) {
        printf ("x = %d\n", x);
        x++;
}

printf("ho finito!\n");
```

Il ciclo While

```
int x = 0;

while (x < 10) {
        printf ("x = %d\n", x);
        x++;
}
printf("ho finito!\n");
```



Quando voglio uscire dal ciclo?
Quando X >= 10
Quindi devo controllare che la **negazione** della condizione che voglio sia rispettata sia vera.

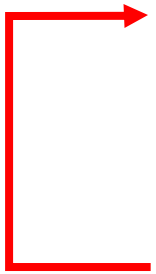
Il ciclo While

```
int x = 0;

while (x < 10) {
        printf ("x = %d\n", x);
        x++;
}

printf("ho finito!\n");
```

Se è verificata, eseguo il blocco di codice del `while`, ovvero stampo a schermo e incremento il valore di `x`

Il ciclo While

```
int x = 0;

while (x < 10) {
        printf ("x = %d\n", x);
        x++;
}

printf("ho finito!\n");
```



A questo punto effettuo un salto e ritorno al controllo della condizione

Il ciclo While

```
int x = 0;

while (x < 10) {
        printf ("x = %d\n", x);
        x++;
}

printf("ho finito!\n");
```

Il ciclo While

```
int x = 0;

while (x < 10) {
        printf ("x = %d\n", x);
        x++;
}

printf("ho finito!\n");
```

Se è verificata, eseguo il blocco di codice del `while` ovvero stampo a schermo e incremento il valore di `x`

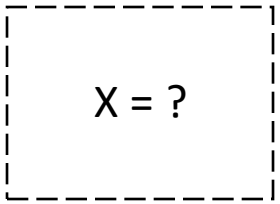
Il ciclo While

```
int x = 0;

while (x < 10) {
        printf ("x = %d\n", x);
        x++;
}
printf("ho finito!\n");
```

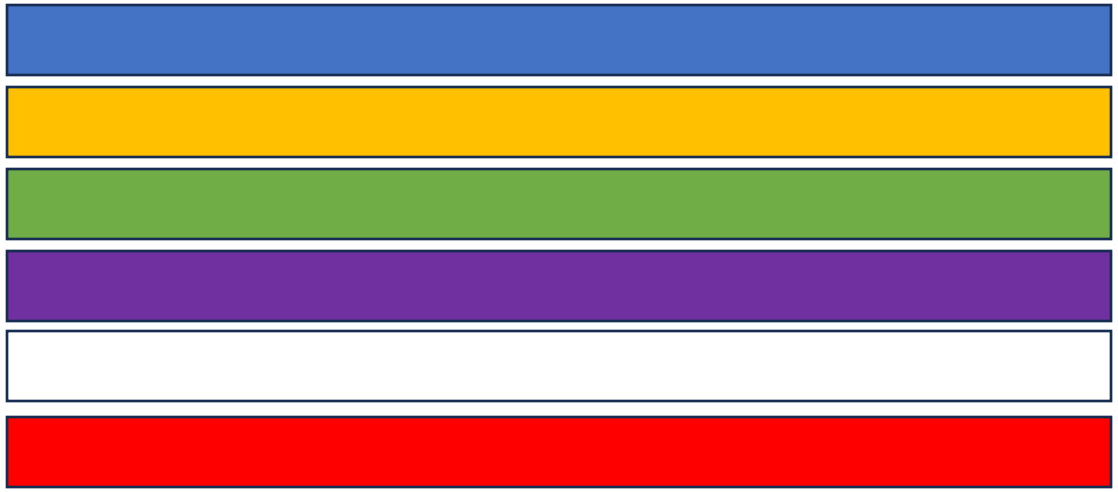
Se non è verificata, salto all'istruzione successiva

Il ciclo While

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```

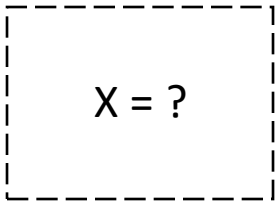


→ Valore di x

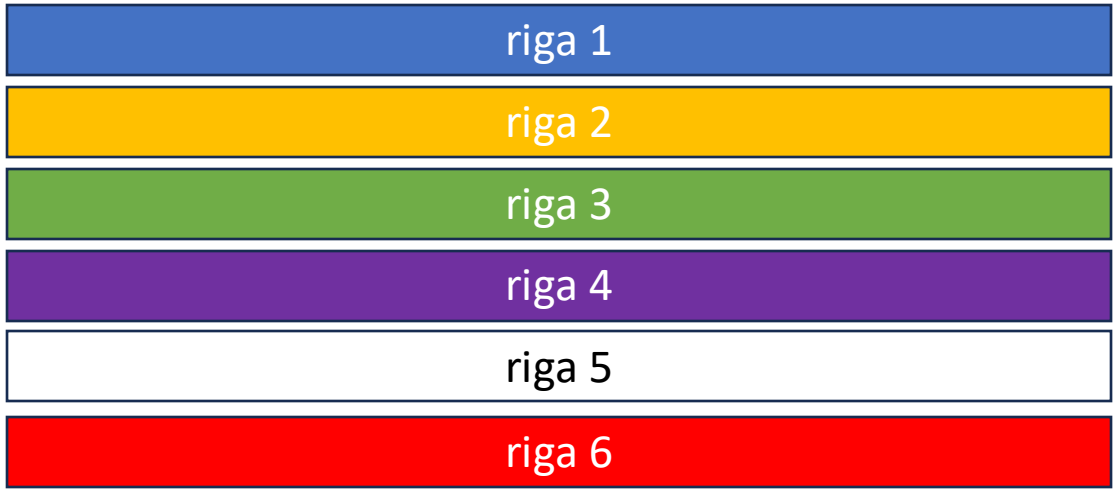


Il ciclo While

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```



→ Valore di x



Il ciclo While

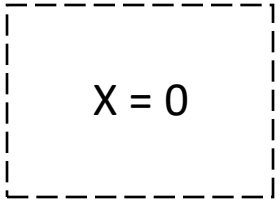
```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```

X = 0

→ Valore di x

riga 1

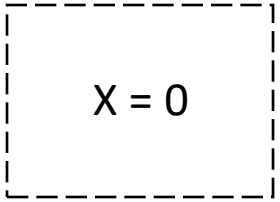
Il ciclo While



→ Valore di x

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```

Il ciclo While



→ Valore di x

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```

Il ciclo While

X = ~~0~~ 1 → Valore di x

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```


```
x = 0
```

Il ciclo While

X = ~~0~~ 1 → Valore di x

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```


```
x = 0
```

Il ciclo While

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```

```
x = 0
```

X = ~~0~~ 1

→ Valore di x

riga 1

riga 2

riga 3

riga 4

riga 5

riga 2

Il ciclo While

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```

```
x = 0
x = 1
```

X = ~~0~~ 1 → Valore di x

riga 1
riga 2
riga 3
riga 4
riga 5
riga 2
riga 3

Il ciclo While

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```

```
x = 0
x = 1
```

X = ~~0 1~~ 2 → Valore di x

riga 1
riga 2
riga 3
riga 4
riga 5
riga 2
riga 3
riga 4

Il ciclo While

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```

```
x = 0
x = 1
```

X = ~~0 1~~ 2 → Valore di x

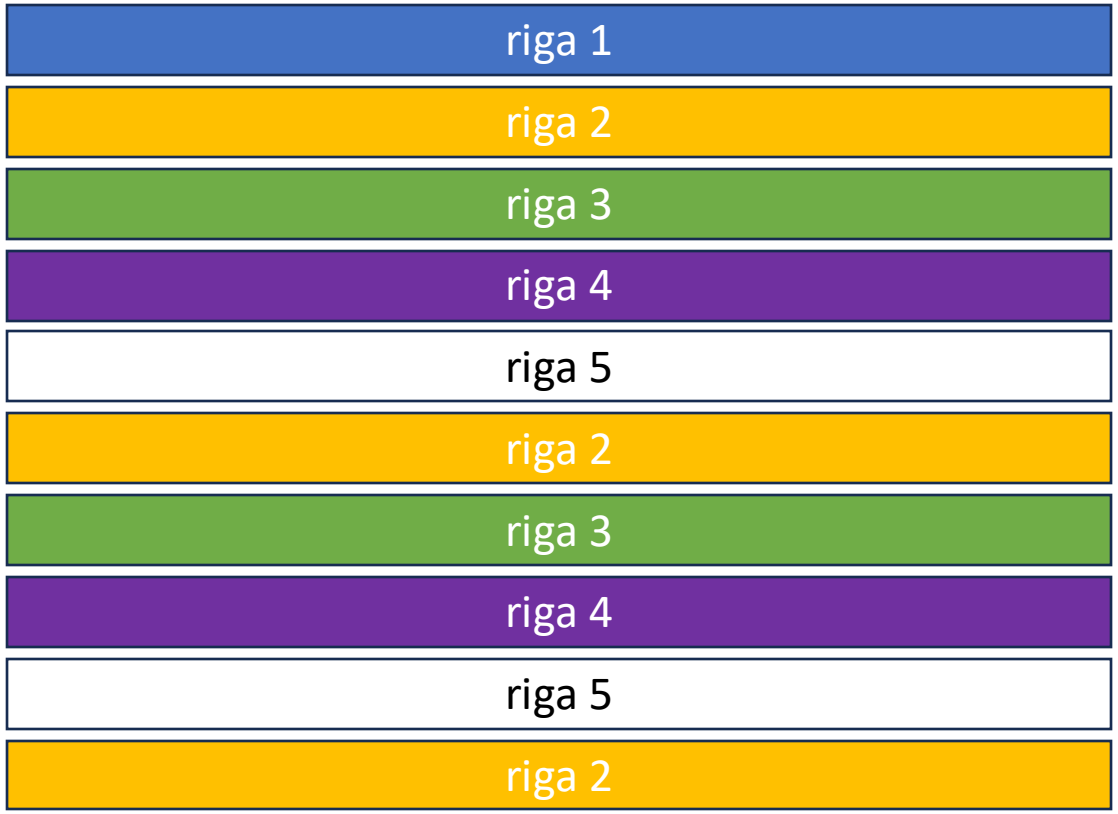
riga 1
riga 2
riga 3
riga 4
riga 5
riga 2
riga 3
riga 4
riga 5

Il ciclo While

X = ~~0~~ ~~1~~ 2 → Valore di x

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```

```
x = 0
x = 1
```

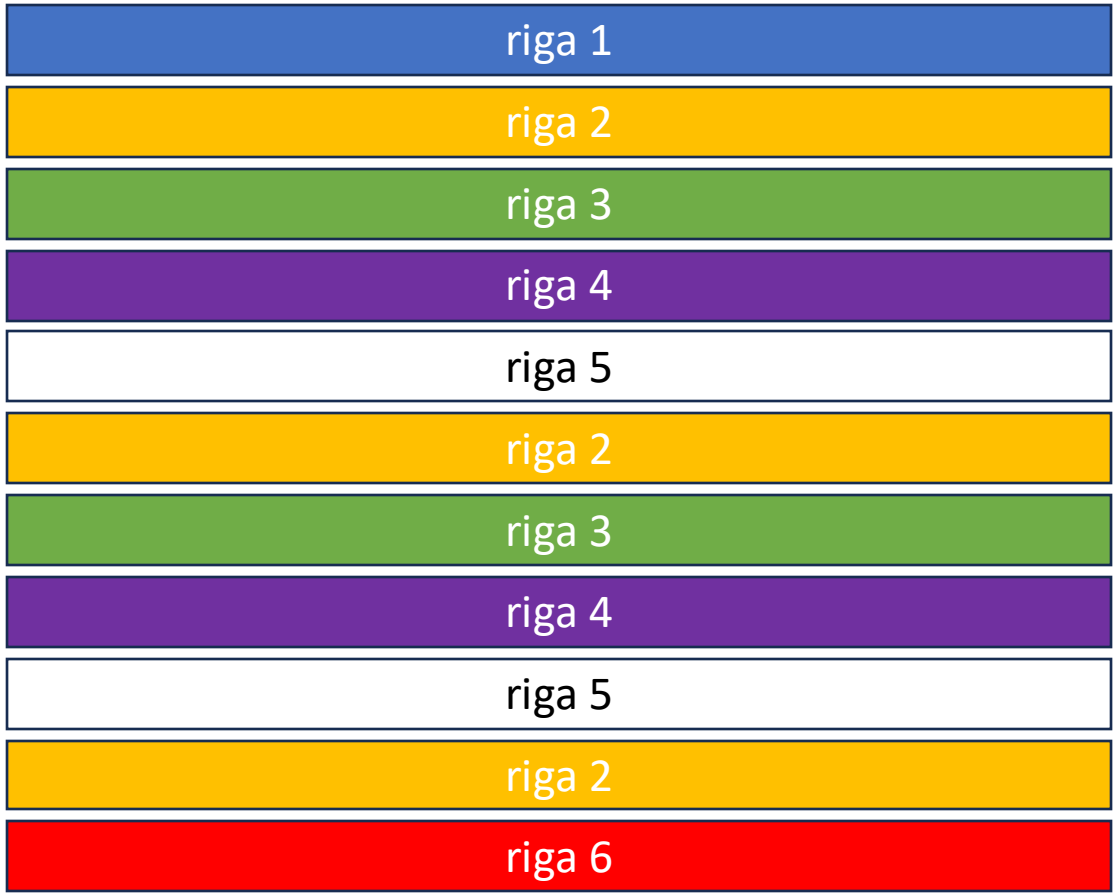


Il ciclo While

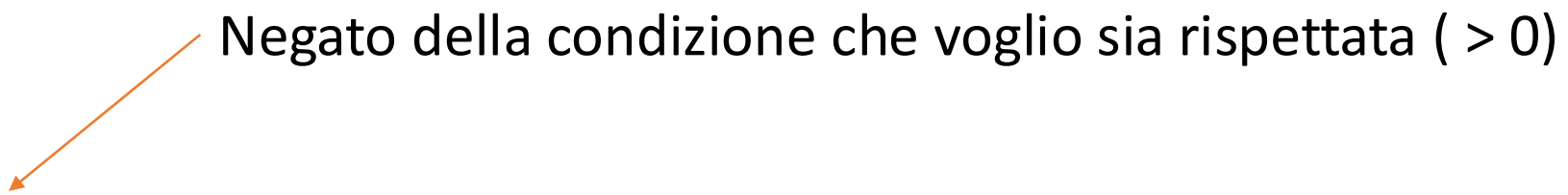
X = ~~0~~ ~~1~~ 2 → Valore di x

```
1: int x = 0;
2: while (x < 2) {
3: printf ("x = %d\n", x);
4: x++;
5:}
6: printf("ho finito!\n");
```

```
x = 0
x = 1
ho finito!
```



Il ciclo While




```
int x;
while (x <= 0) {
printf("Inserisci un valore > 0\n");
scanf("%d",&x);
printf("Hai inserito %d ",x);
if (x<=0)
printf(" e non va bene\n");
}
printf("\nHai inserito %d e va bene.",x);
```

Il ciclo While

```
int x;
while (x <= 0) {
        printf("Inserisci un valore > 0\n");
        scanf("%d",&x);
        printf("Hai inserito %d ",x);
        if (x<=0)
                printf(" e non va bene\n");
}
printf("\nHai inserito %d e va bene.",x);
```

Il ciclo While

```
int x; // errore! manca inizializzazione!
while (x <= 0) {
        printf("Inserisci un valore > 0\n");
        scanf("%d",&x);
        printf("Hai inserito %d ",x);
        if (x<=0)
                printf(" e non va bene\n");
}
printf("\nHai inserito %d e va bene.",x);
```

Il ciclo While

```
int x = -1; //OK! Uso valore fuori dal dominio
while (x <= 0) {
        printf("Inserisci un valore > 0\n");
        scanf("%d",&x);
        printf("Hai inserito %d ",x);
        if (x<=0)
                printf(" e non va bene\n");
}
printf("\nHai inserito %d e va bene.",x);
```

Il ciclo Do While

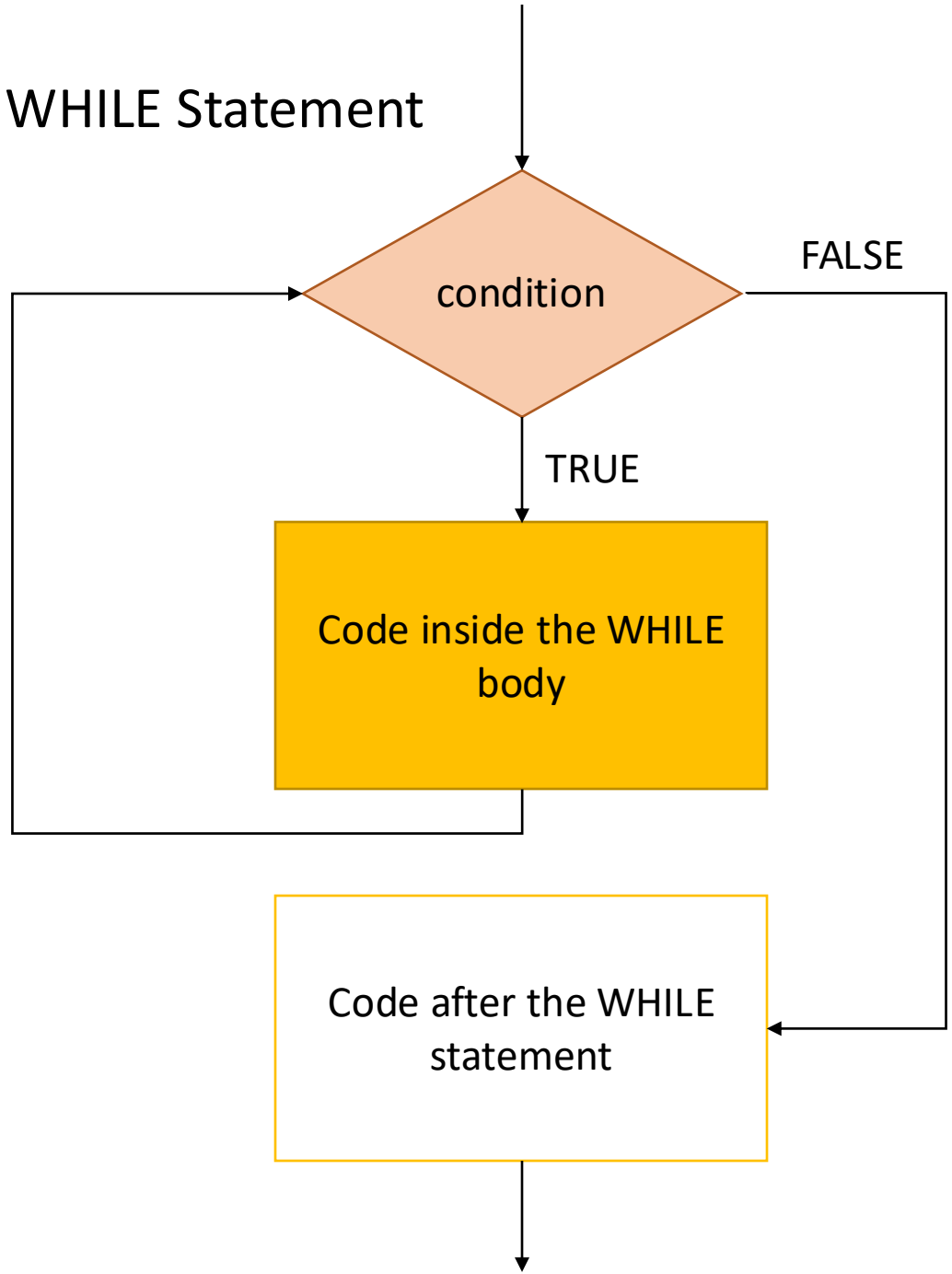
Costrutto di loop do while in C

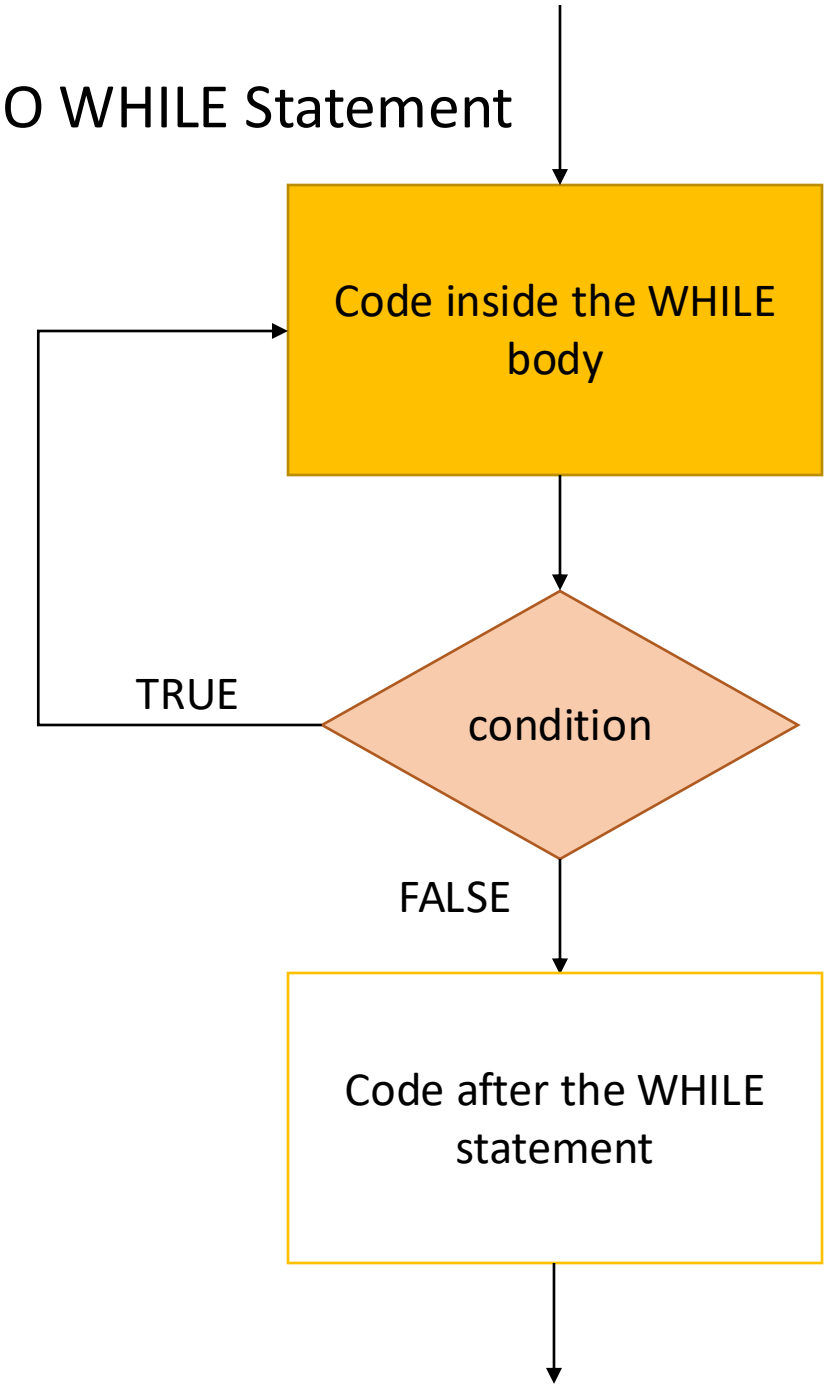
```
do
     <blocco>
while (<espressione>);
```

- Blocco di codice: se è costituito da più di una istruzione è definito da { }; se è di una sola istruzione, si possono omettere le { }
- Di `<espressione>` si valuta il valore logico (vero o falso)
- Il ciclo verrà eseguito fino a che `<espressione>` è vera
- Occhio al «;» dopo `<espressione>`

Il ciclo Do While

```
int x;  //Non serve inizializzare
do              //con il do-while viene eseguito almeno 1
{
        printf("Inserisci un valore > 0\n");
        scanf("%d",&x);
        printf("Hai inserito %d ",x);
        if (x<=0)
                printf(" e non va bene\n");
} while (x<=0);
printf("\nHai inserito %d e va bene.",x);
```


WHILE Statement
condition
FALSE
TRUE
Code inside the WHILE body
Code after the WHILE statement


DO WHILE Statement
Code inside the WHILE body
TRUE
condition
FALSE
Code after the WHILE statement

Cosa fa questo programma?

```
x=0;
while (x<10);
{
        x++;
        printf("%c ", x);
}
printf("\n");
```

Cosa fa questo programma? Corretto

```
x=0;
while (x<10); // «;» è una istruzione vuota
{
        x++;
        printf("%c ", x);
}
printf("\n");
```

Setup in lab:

- Linux
- Pluma (editor – o un altro editor di testo minimale)
- Compilatore su terminale

Setup da remoto:

- https://www.onlinegdb.com/ (online via web browser)
- Replica di setup del lab su vostra macchina o virtual machine

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
        int x = 0;
        do {
                scanf("%d",&x);
        }
while (x<0);
printf("Valore: %d\n",x);
return 0;
}
```

Comandi utili per il usare il terminale:

- **`cd nomeCartella`** si sposta in una cartella di nome **`nomeCartella`**
- **`mkdir nomeCartella`** crea una cartella di nome **`nomeCartella`**
- **`pwd`** nome della cartella corrente
- **`ls`** mostra contenuto cartella
- **`cd ..`** Torna alla cartella precedente

Comandi utili per il compilare ed eseguire il vostro programma:

- (scrivete il vostro programma su di un editor di testo, salvatelo)
- entrate nella cartella dove è salvato **`programma.c`**
- **`gcc programma.c -o eseguibile`** compila e crea eseguibile (leggere output di compilazione per errori)
- **`./eseguibile`** esegue il programma (stampa a terminale il risultato)