Controllo di flusso

Marco Alberti

Dipartimento di Matematica e Informatica

Università degli Studi di Ferrara

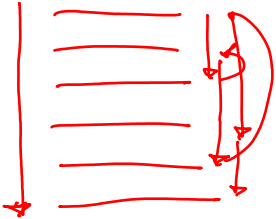
Programmazione e Laboratorio, A.A. 2020-2021

Ultima modifica: 7 ottobre 2020

Attenzione! Questo materiale didattico è per uso personale dello studente ed è coperto da copyright.
Ne sono vietati la riproduzione e il riutilizzo anche parziale, ai sensi e per gli effetti della legge sul diritto d'autore.

Istruzioni di controllo di flusso

In generale, il flusso predefinito (corrispondente all'ordine testuale delle istruzioni in un programma) non è sufficiente ad ottenere gli effetti desiderati.
In questa serie di diapositive vedremo le istruzioni di controllo di flusso, cioè quelle che consentono di alterare il flusso rispetto a quello predefinito.
Vedremo anche una tecnica (detta programmazione strutturata) per mantenere comprensibili i programmi che ne fanno uso.



Sommario

1. Istruzioni condizionali (o di selezione)
2. Salti
3. Blocchi
4. Iterazione
5. Programmazione strutturata
6. Selezione multipla

Sommario

1. Istruzioni condizionali (o di selezione)
2. Salti
3. Blocchi
4. Iterazione
5. Programmazione strutturata
6. Selezione multipla

Problema

pin 44122 → "Accesso consentito"
pin 44123 → ""

Scrivere un programma che simuli la verifica di un PIN inserito dall'utente. In particolare, il programma deve richiedere un numero all'utente e, se il numero è uguale a 44122, stampare il messaggio "Accesso consentito"

```
#include <stdio.h>

main() {
  int pin;
  scanf("%d", &pin);
  // Stampiamo il messaggio o no?
}
```

printf("Accesso consentito\n"),

Dobbiamo compiere o meno l'azione (stampa del messaggio) a seconda di una condizione (pin è uguale a 44122?) che non possiamo verificare mentre scriviamo il programma, perché dipende da cosa digita l'utente.
I costrutti visti finora non consentono di risolvere il problema.
C'è bisogno di poter eseguire o meno una istruzione a seconda del verificarsi, a tempo di esecuzione, di una condizione.
Questo è reso possibile dall'istruzione condizionale. SELEZIONE

Istruzione `if`

SE

0 → FALSO
1 → VERO
CAZZO → VERO

-1 → VERO

Sintassi

⟨*istruzioneCondizionale*⟩ ::= `if` (⟨*espressione*⟩) ⟨*istruzione*⟩

Semantica

Se ⟨*espressione*⟩ è vero, esegui ⟨*istruzione*⟩.

LOGICO
BOOLEANO

A differenza di altri linguaggi, il C non ha un tipo apposito per rappresentare i valori vero e falso; in C un'espressione di valore 0 è falsa, un'espressione di valore diverso da 0 è vera.
Quindi nell'istruzione condizionale

```
if (__condizione__) __istruzione__
```

`__istruzione__` viene eseguita se e solo se `__condizione__` ha valore diverso da 0.
La stessa espressione può quindi avere un significato logico o numerico a seconda del contesto.

Esercizio

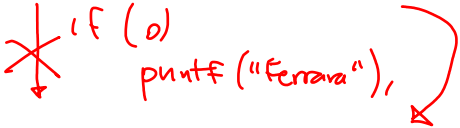
Che cosa fanno le seguenti istruzioni condizionali?

- `if (1) printf("a. Vero");` — a. Vero
- `if (4 - 2 * 2) printf("b. Vero");` — —
- `if (2 * 3) printf("c. Vero");` — c. Vero

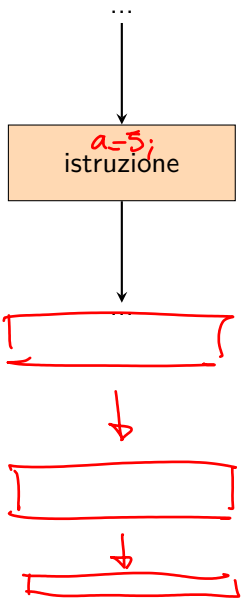
Scrivere un programma per verificare la correttezza delle risposte.

Diagrammi di flusso

- `if` è un primo esempio di istruzione di controllo di flusso, cioè di istruzione che modifica il flusso (l'ordine di esecuzione delle istruzioni) rispetto a quello predefinito (quello testuale). Infatti può causare la mancata esecuzione di un'istruzione (se la condizione è falsa).
- Un diagramma di flusso di un programma è una rappresentazione grafica del programma stesso, che aiuta a comprenderne il flusso.
- E' formato da blocchi il cui significato dipende dalla forma. Le frecce che uniscono i blocchi rappresentano il passaggio del flusso da un blocco a un'altro.
- Possiamo immaginare che la macchina astratta si muova nel diagramma di flusso agendo a seconda dei blocchi che trova.

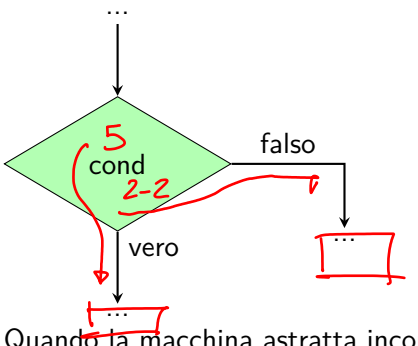


Blocco istruzione



Il blocco rettangolare rappresenta un'istruzione qualsiasi.
Ha zero o più frecce in ingresso e al massimo una in uscita.
Quando la macchina astratta incontra un blocco di questo tipo, esegue l'istruzione in esso contenuta.

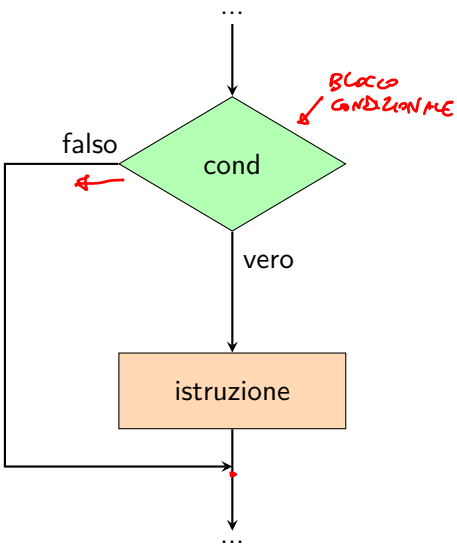
Blocco condizionale



Il rombo rappresenta la valutazione di una condizione, cioè di un'espressione il cui valore può essere vero o falso.
Ha due frecce in uscita: una seguita in caso valga vero, l'altra in caso valga falso.

Quando la macchina astratta incontra un blocco condizionale, valuta "cond": se è vera (cioè ha valore diverso da 0) percorre il ramo etichettato con "vero", altrimenti quello etichettato con "falso".

Diagramma di flusso `if`




```
if (__cond__)
```

```
__istruzione__
```

Soluzione del problema alla slide 2

Scrivere un programma che simuli la verifica di un PIN inserito dall'utente. In particolare, il programma deve richiedere un numero all'utente e, se il numero è uguale a 44122, stampare il messaggio "Accesso consentito".

050_controllo_di_flusso/pin.c

```c
#include <stdio.h>

main() {
  int pin;
  scanf("%d", &pin);

  if (pin == 44122)

    printf("Accesso consentito\n");
}
```

pin 44122

pin 44123

OPERATORE RELAZIONALE DI UGUAGLIANZA

2 == 2 → 1

2 == 3 → 0

Operatori relazionali

= ASSEGNAMENTO

L'operatore == verifica l'uguaglianza dei suoi operandi, cioè vale 1 se gli operandi sono uguali e 0 altrimenti. E' uno degli operatori relazionali offerti dal linguaggio C.

Espressione	Vale 1 se e solo se	Vale 0 se e solo se
a == b	a e b sono uguali	a e b sono diversi
DIVERSO a != b	a e b sono diversi	a e b sono uguali
a < b	a è minore di b	a è maggiore o uguale a b
a <= b	a è minore o uguale a b	a è maggiore di b
a > b	a è maggiore di b	a è minore o uguale a b
a >= b	a è maggiore o uguale a b	a è minore di b

Nota: la terza colonna si ricava dalla seconda, in quanto ne è la negazione.

Esercizio

Se a e b sono variabili intere rispettivamente di valore 2 e −3, qual è il valore delle seguenti espressioni?

- a > b ~~VERO~~ 1
- a < 0 ~~FALSO~~ 0
- a + b >= 0 0
- a = b 0 a $\boxed{2-3}$ -3
- a == b 0
- a + b != b + a 0

a $\boxed{2}$

b $\boxed{-3}$

if (a=b) -3
printf(" .");

Verificare la correttezza delle risposte con un programma.

printf("%d\n", a>b);

Esercizio

Esprimere con espressioni in linguaggio C le seguenti proposizioni.

- *a* è minore di zero. $a < 0$

  $a \boxed{-1}$ $a < 0 \rightarrow 1$

  $a \boxed{5}$ $a < 0 \rightarrow 0$

- *a* è l'opposto di *b*. $a + b == 0$

  $a == -b$

- Il quadrato di *a* è minore di zero.

  $a * a < 0 \rightarrow 0$

- *a* è pari. $15 \% 2 \rightarrow 1$

  $a \% 2 == 0$ $16 \% 2 \rightarrow 0$

- *a* non è divisibile per *b*.

  $15 \% 3 \rightarrow 0$ $15 \% 4 \rightarrow 3$

  $a \% b \; != 0$

  ABBREVIATA PER CONDIZIONE

  $a \% b$

Esercizio

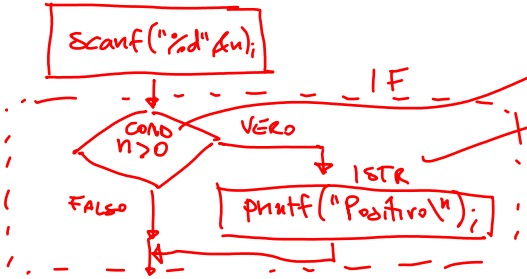
INPUT 3 OUTPUT "POSITIVO"
INPUT -5 OUTPUT ""

Positivo

Scrivere un programma che richieda all'utente un numero intero e stampi il messaggio "Positivo" se il numero è maggiore di 0.

Dispari

Scrivere un programma che richieda all'utente un numero intero e stampi il messaggio "Dispari" se il numero è dispari.




```
#include <stdio.h>
main() {
int n;
scanf("%d\n",
if (n>0)
printf("Positivo\n");
}
```

Problema

Scrivere un programma che simuli la verifica di un PIN inserito dall'utente.
In particolare, il programma deve richiedere un numero all'utente; se il numero è uguale a 44122, deve stampare il messaggio "Accesso consentito", mentre in caso contrario il messaggio "Accesso negato".

050_controllo_di_flusso/pin-e-non.c

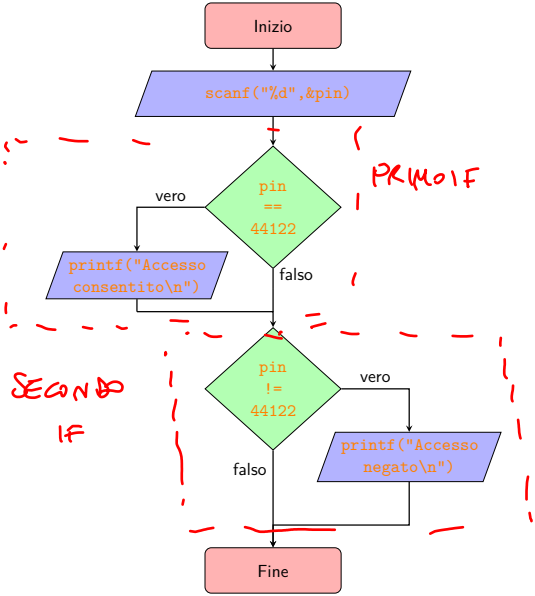
```c
#include <stdio.h>

main() {
  int pin;
  scanf("%d", &pin);
  if (pin == 44122)
    printf("Accesso consentito\n");
  if (pin != 44122)
    printf("Accesso negato\n");
}
```

PIN CORRETTO

DIVERSO

Diagramma di flusso




```
main() {
scanf("%d", &pin);

if (pin == 44122)
printf("Accesso consentito\n");

if (pin != 44122)
printf("Accesso negato\n");

}
```

Diagramma di flusso

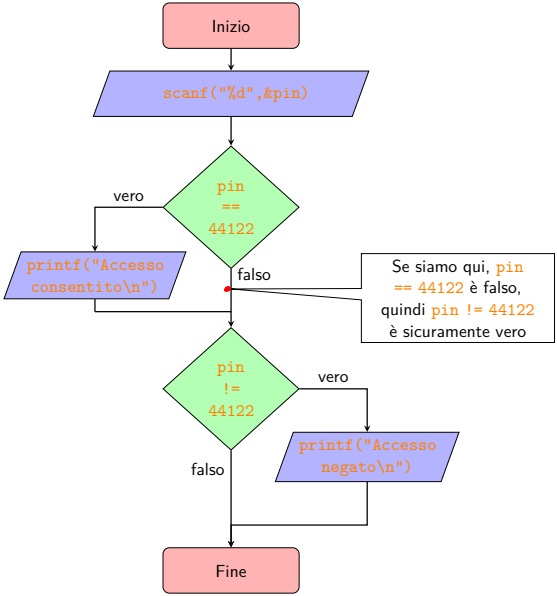


Diagramma di flusso

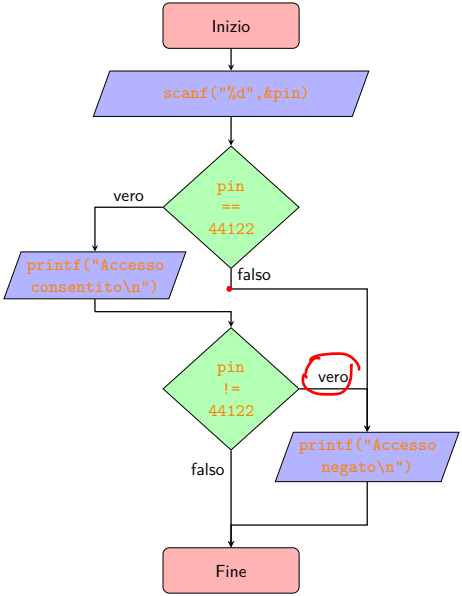


Diagramma di flusso

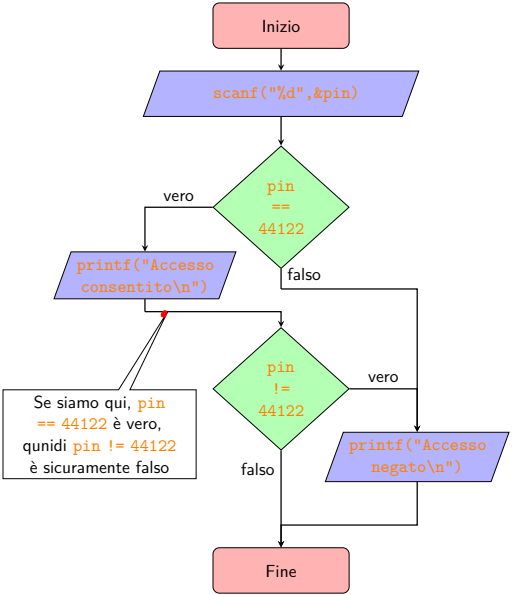


Diagramma di flusso

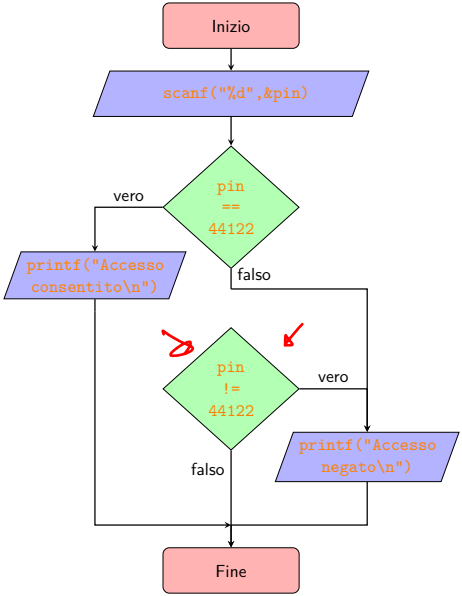


Diagramma di flusso

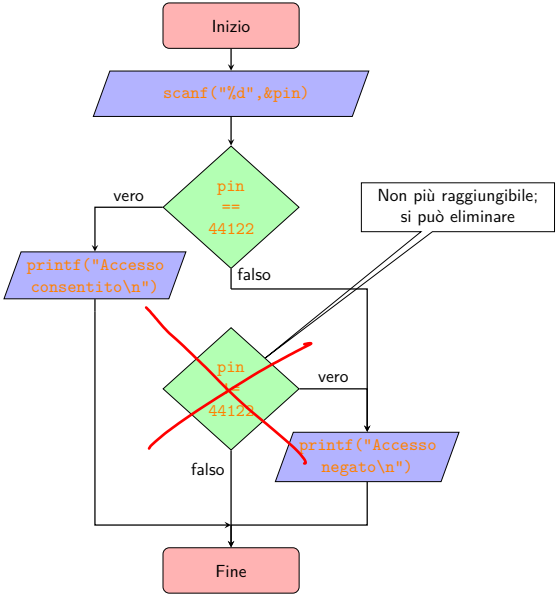
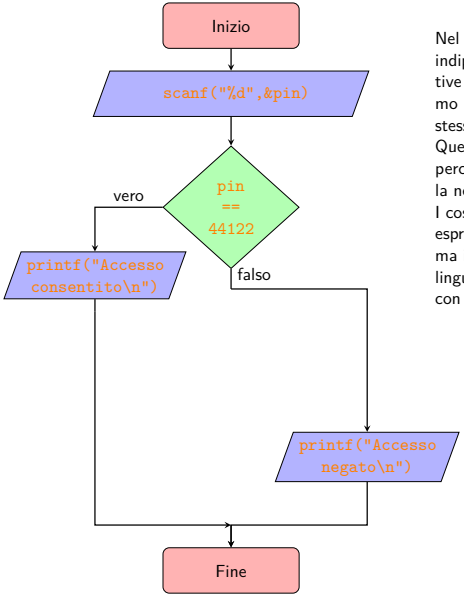


Diagramma di flusso

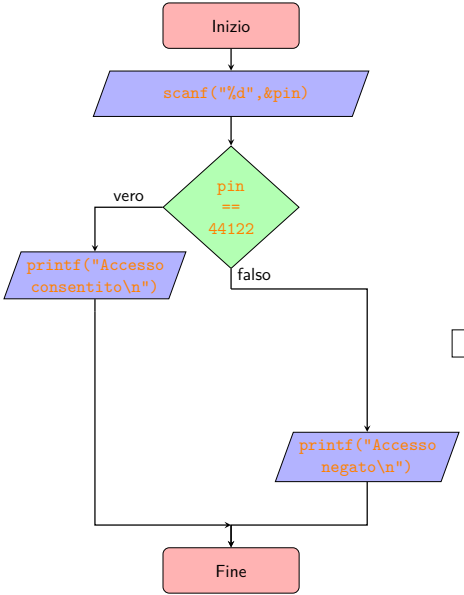


Nel diagramma di flusso, anziché eseguire indipendentemente due azioni se le rispettive condizioni si verificano, ora eseguiamo una o l'altra azione, a seconda che la stessa condizione sia vera o falsa.
Questa trasformazione è stata possibile perché le due condizioni iniziali erano una la negazione dell'altra.
I costrutti visti finora non permettono di esprimere questo tipo di controllo di flusso, ma il caso d'uso è tanto comune che il linguaggio C (come quasi tutti) lo supporta con un costrutto apposito.

if - else

SE ALTRIMENTI

Diagramma di flusso




```
main() {

scanf("%d", &pin);

if (pin == 44122)
printf("Accesso consentito\n");

else
printf("Accesso negato\n");

}
```

COND

RAMO VERO

La nuova parola chiave

ALTRIMENTI

RAMO FALSO

Istruzione `if-else`

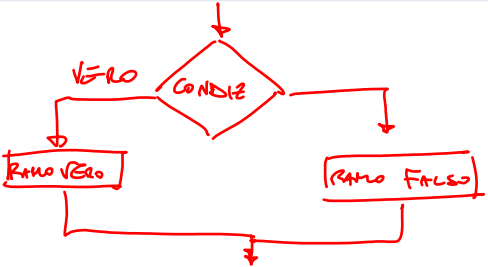
Sintassi

⟨*istruzioneCondizionale*⟩ ::= **`if`** **`(`**⟨*espressione*⟩**`)`** ⟨*istruzione*⟩
| **`if`** **`(`**⟨*espressione*⟩**`)`** ⟨*istruzione*⟩ **`else`** ⟨*istruzione*⟩

CONDIZIONE RAMO VERO RAMO FALSO

Semantica `if-else`

Se ⟨*espressione*⟩ è vera (cioè se ha valore diverso da 0) esegui l'⟨*istruzione*⟩ che precede **`else`** (ramo vero), altrimenti esegui l'⟨*istruzione*⟩ se segue **`else`** (ramo falso).



Soluzione

050_controllo_di_flusso/pin-else.c

```c
#include <stdio.h>

main() {
  int pin;
  scanf("%d", &pin);
  if (pin == 44122)
    printf("Accesso consentito\n");
  else
    printf("Accesso negato\n");
}
```

COND

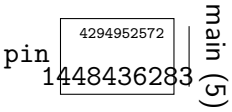
RAMO VERO

RAMO FALSO

COMPARE UNA SOLA VOLTA

IF ELSE

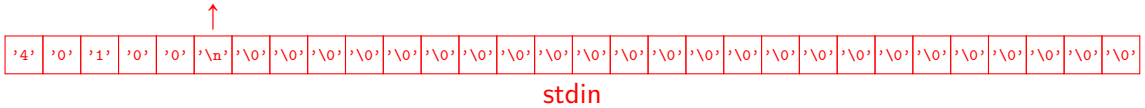
PIN con else




```
1  #include <stdio.h>
2
3  main() {
4    int pin;
5    scanf("%d", &pin);
6    if (pin == 44122)
7      printf("Accesso consentito\n");
8    else
9      printf("Accesso negato\n");
10 }
```

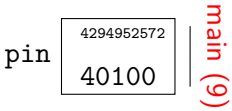
PIN con else

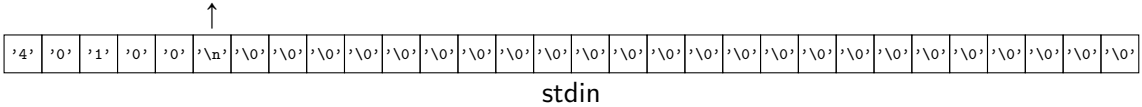
```
1   #include <stdio.h>
2
3   main() {
4     int pin;
5     scanf("%d", &pin);
6     if (pin == 44122)
7       printf("Accesso consentito\n");
8     else
9       printf("Accesso negato\n");
10  }
```



PIN con else

```
1   #include <stdio.h>
2
3   main() {
4     int pin;
5     scanf("%d", &pin);
6     if (pin == 44122)
7       printf("Accesso consentito\n");
8     else
9       printf("Accesso negato\n");
10  }
```





DRY

Nel programma alla slide 14, compaiono due condizioni: quella per scrivere "Accesso consentito" e quella per scrivere "Accesso negato".
Nella vita del programma, le condizioni potrebbero cambiare (ad esempio se cambia il PIN).
Le due condizioni sono una la negazione dell'altra. Nella realtà della nostra applicazione, una sola è indipendente (la prima condizione), mentre l'altra è derivata.
Nel programma alla slide 14, che cosa succede se il PIN cambia? Devo modificare due condizioni:

1. Lavoro di più
2. Posso dimenticarne una PROGRAMMA INCOERENTE

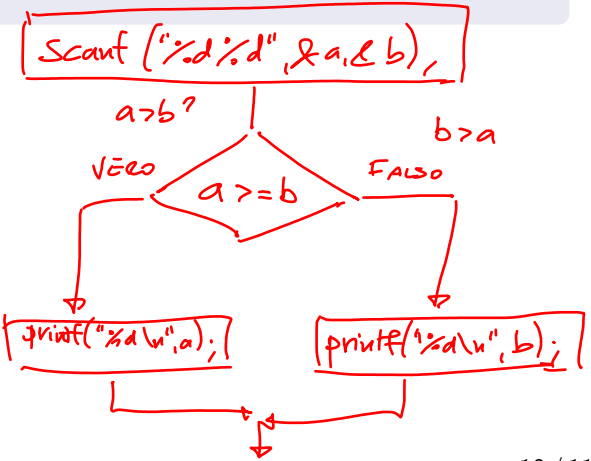
Nella versione alla slide 17, la condizione non è ripetuta. Se la cambio, è il sistema (la semantica del linguaggio) che garantisce lo stesso effetto senza che io debba fare nulla.
Preferire la versione con `if-else` alla versione con due `if`, anche se sono entrambe funzionanti, è una prima applicazione del principio DRY (Don't Repeat Yourself, non ripeterti).

Esercizio

Maggiore

Scrivere un programma che richieda all'utente due numeri interi e stampi il maggiore dei due.

INPUT		OUTPUT
2	3	3
3	2	3
2	2	2



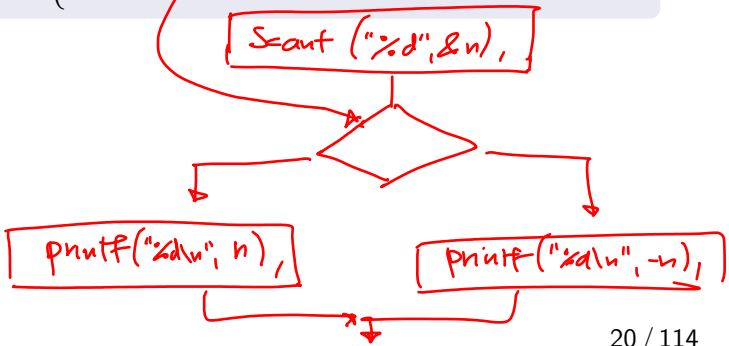
Esercizio

Valore assoluto

Scrivere un programma che richieda all'utente un numero intero e ne stampi il valore assoluto.

$$|n| = \begin{cases} n & \text{se } n \geqslant 0 \\ -n & \text{altrimenti} \end{cases}$$

INPUT	OUTPUT
2	2
-2	2
0	0



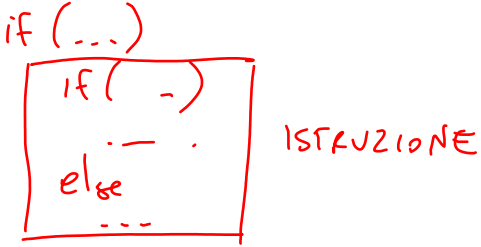
If annidati NESTED

Riguardiamo la sintassi:

Sintassi `if` e `if-else`

⟨*istruzioneCondizionale*⟩ ::= **`if`** (⟨*espressione*⟩) ⟨*istruzione*⟩
| **`if`** (⟨*espressione*⟩) ⟨*istruzione*⟩ **`else`** ⟨*istruzione*⟩

Osservazione: i rami vero e falso sono istruzioni; nulla impedisce che una delle due, o entrambe, siano a loro volta istruzioni condizionali.

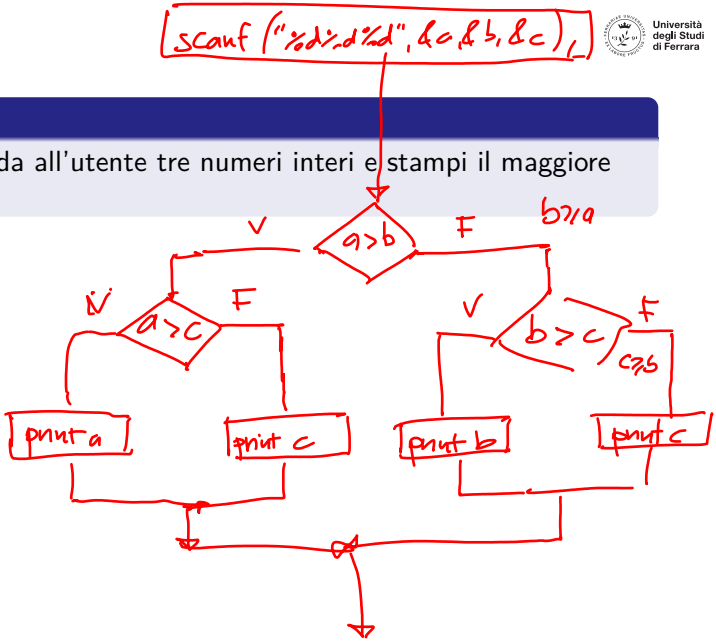


Esercizio

Maggiore di tre

Scrivere un programma che richieda all'utente tre numeri interi e stampi il maggiore dei tre.

INPUT	OUTPUT
5 3 2	5
2 5 3	5
3 2 5	5
3 5 5	5
a b c	



Esercizio

Scrivere un programma che richieda all'utente un intero e scriva

- `compreso` se il numero è compreso fra 2 e 5;
- `non compreso` altrimenti.

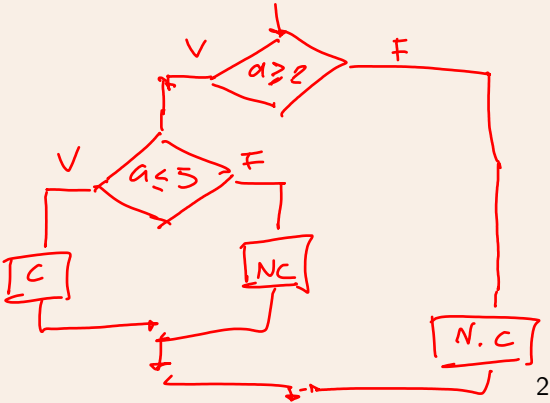
2 3 4 5 compreso
altrimenti non compreso

050_controllo_di_flusso/compreso.c

```c
#include <stdio.h>

main() {
  int a;
  scanf("%d", &a);
  if (a >= 2)
    if (a <= 5)
      printf("Compreso\n");
    else
      printf("Non compreso\n");
  else
    printf("Non compreso\n");
}
```

RIP

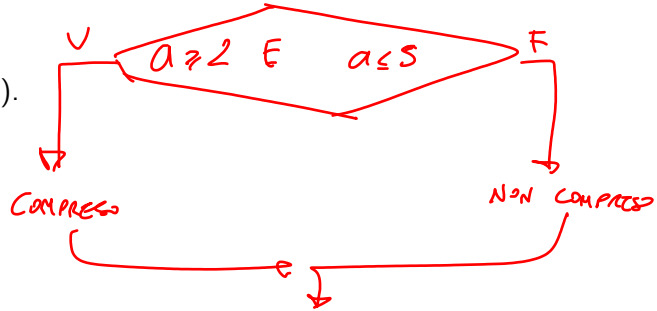


Connettivi logici

Il programma alla slide 23 utilizza due istruzioni condizionali annidate (una con condizione `a >= 2` e una con condizione `a <= 5`), ma potrebbe utilizzarne una sola se fosse possibile esprimere combinare le due condizioni in una unica.
Questo è possibile con i connettivi logici, ossia operatori che consentono di combinare espressioni logiche componendone i valori secondo la logica classica.
I connettivi logici possono essere

- unari (negazione)
- binari (congiunzione e disgiunzione).

Sono definiti da tabelle di verità.



Connettivo logico unario

espressione

- ! (negazione, *non* logico): !p è
  - vero se p è falso,
  - falso se p è vero.

	p	!p	
FALSO	0	1	VERO
VERO	non 0	0	FALSO

Tabella: Tabella di verità dell'operatore di negazione (!). Ad esempio la seconda riga dice che se p ha valore diverso da 0, allora !p vale 0.

Connettivi logici binari

a>2 && a<5

AND

- && (congiunzione, e logico): p && q è
  - vero se p e q sono entrambi veri,
  - falso se almeno uno fra p e q è falso.
- || (disgiunzione, *o inclusivo* logico): p || q è
  - falso se p e q sono entrambi falsi,
  - vero se almeno uno fra p e q è vero.

OR

3 > 2 || FERRARA È IN FRANCIA

VERO FALSO

p	q	p && q	p || q
0	0	0	0
0	non 0	0	1
non 0	0	0	1
non 0	non 0	1	1

Tabella: Tabelle di verità degli operatori di congiunzione (&&) e disgiunzione (||). Ad esempio la seconda riga dice che se p vale 0 e q ha valore diverso da 0, allora p && q vale 0 e p || q vale 1.

Esercizio

a 2
b -3

Se a e b sono variabili intere rispettivamente di valore 2 e −3, qual è il valore delle seguenti espressioni?

- `! (a > 0)` → 0
- `(!a) > 0` → 0
- `a > 0 && b < 0` → 1
- `a > 0 && b > 0` → 0
- `a > 0 || b > 0` → 1
- `a < 0 || 2` → 1

Verificare la correttezza delle risposte con un programma.

Esercizio

int a, int b;

Esprimere con un'espressione in linguaggio C ciascuna delle seguenti proposizioni:

- *a* è compreso fra 3 e 5 — a>=3 && a<=5
- il quadrato di *a* è compreso fra 3 e 5 — a*a >=3 && a*a <=5
- *a* e *b* sono entrambi minori di 5; — a<5 && b<5
- almeno uno fra *a* e *b* è minore di 5; — a<5 || b<5
- esattamente uno fra *a* e *b* è maggiore di 0;

(a>0 && b<=0) || (a<=0 && b>0)

Esercizio

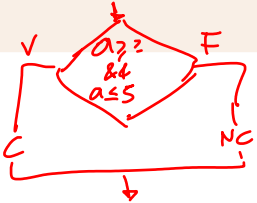
Soluzione preferibile dell'esercizio alla slide 23:

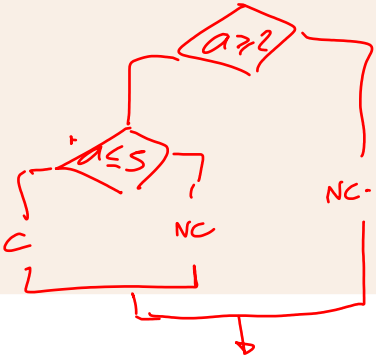
050_controllo_di_flusso/compreso-con-and.c

```c
#include <stdio.h>

main() {
  int a;
  scanf("%d", &a);
  if (a >= 2 && a <= 5)
    printf("Compreso\n");
  else
    printf("Non compreso\n");
}
```

COMPRESO





VECCHIO DIAGRAMMA

Valutazione in cortocircuito

Nel valutare la congiunzione `__cond1__ && __cond2__`, se `__cond1__` è falsa è inutile valutare anche `__cond2__`, perché la congiunzione sarà comunque falsa (come si verifica dalla tabella di verità).

Per questo motivo la macchina astratta C valuta `__cond2__` solo se il risultato della valutazione di `__cond1__` è vero. Questa operazione si chiama valutazione in cortocircuito.

Analogamente, nel valutare la disgiunzione `__cond1__ || __cond2__`, la macchina astratta valuta `__cond2__` solo se `__cond1__` è falsa, altrimenti la disgiunzione è sicuramente vera.

Esempio

Il seguente programma che richiede all'utente un numero intero e stampa il messaggio "Verificato" se il quoziente fra 10 e quel numero è maggiore di 2.

050_controllo_di_flusso/input-denominatore.c

```c
#include <stdio.h>

main() {
    int denominatore;
    scanf("%d", &denominatore);
    if (10 / denominatore > 2)
        printf("Verificato\n");
}
```

Che cosa stampa il programma per gli input 6, 2, 5 e 0?

ERRORE

Correzione

DENOMINATORE
10

DENOMINATORE + 5 → 15
(DENOMINATORE != 0) + 5 → 6

Il seguente programma funziona anche per input 0; eppure l'espressione che comporta la divisione per 0 è presente.

050_controllo_di_flusso/input-denominatore-corretto.c

```c
#include <stdio.h>

main() {
    int denominatore;
    scanf("%d", &denominatore);
    if (denominatore != 0 && 10 / denominatore > 2)
        printf("Verificato\n");
}
```

NON VALUTATA

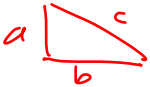
0 0

EQUIVALENTI RISPETTO AL VALORE DI VERITÀ

Perché non genera eccezione? VALUTAZIONE IN CORTOCIRCUITO

Si può semplificare la condizione? denominatore && 10/denominatore > 2

Esercizio



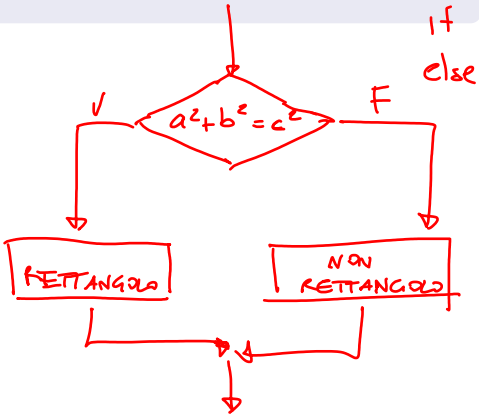
$a^2 + b^2 = c^2$

Triangolo rettangolo

Scrivere un programma che richieda all'utente i tre lati di un triangolo scriva se il triangolo è rettangolo oppure no.

IN ORDINE CRESCENTE

INPUT			OUTPUT
3	4	5	RETTANGOLO
4	4	4	NON RETTANGOLO
5	12	13	RETTANGOLO
a	b	c	



if

else

Esercizio

$(a == b) == c \rightarrow 0$

Università degli Studi di Ferrara

Triangolo

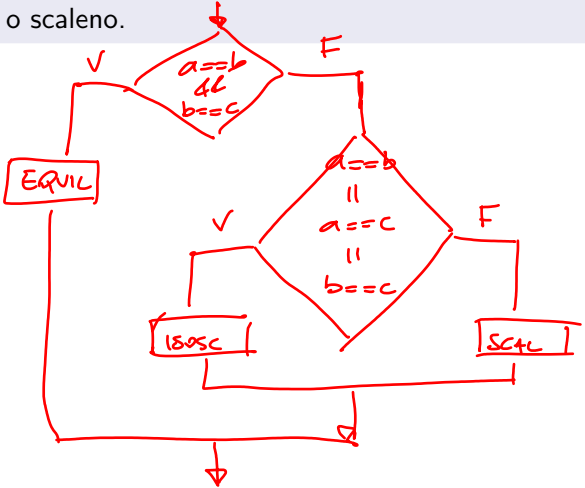
Scrivere un programma che richieda all'utente i tre lati di un triangolo e classifichi il triangolo come equilatero, isoscele o scaleno.

3 3 3 EQUIL

3 3 4 ISOSCELE

3 4 5 SCALENO

a b c



Esercizio

Frazioni

Leggere quattro numeri che rappresentano numeratore e denominatore di due frazioni e dire se le due frazioni rappresentano lo stesso numero.

$\frac{a}{b}$ $\frac{c}{d}$

INPUT

a	b	c	d
2	3	4	6
2	3	4	5

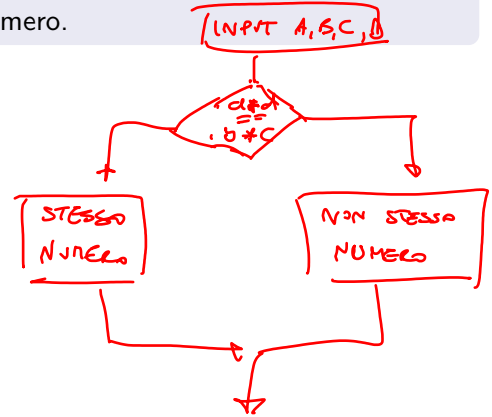
$\frac{2}{3} = \frac{4}{6}$

$\frac{2}{3} \neq \frac{4}{5}$ OUTPUT

STESSO NUMERO

NON STESSO NUMERO

$\frac{\cancel{b}ad}{\cancel{b}} = \frac{bc\cancel{d}}{\cancel{d}}$



Esercizio

Giorno

Leggere da tastiera un numero e, se è compreso fra 1 e 7, visualizzare il giorno della settimana corrispondente (ad esempio se l'utente digita 2 bisogna scrivere "martedì")

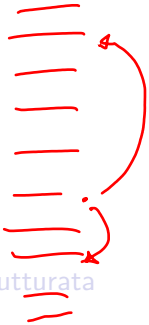
INPUT	OUTPUT
1	Lunedì
2	martedì

0 lunedì
1 martedì
...
...

```
int g;
scanf("%d", &g);
if (g==1)
    printf("Lunedì\n");
if (g==2)
    printf("Martedì\n");
```

Sommario

1. Istruzioni condizionali (o di selezione)
2. Salti
3. Blocchi
4. Iterazione
5. Programmazione strutturata
6. Selezione multipla



Esercizio

Scrivere un programma che simuli la scelta del numero da chiamare in una rubrica telefonica. In particolare, il programma deve

1. stampare l'elenco dei contatti, ognuno preceduto da un identificativo intero progressivo
2. richiedere all'utente l'identificativo del contatto da chiamare
3. stampare un messaggio che indichi che la chiamata al contatto selezionato è in corso.

Soluzione esercizio slide 37

050_controllo_di_flusso/rubrica.c

```c
#include <stdio.h>

main() {
  int scelta;
  printf("1. Tizio 335-1234567\n");
  printf("2. Caio 347-1234567\n");
  printf("Scegliere un contatto\n");
  scanf("%d", &scelta);
  printf("Chiamata del contatto %d in corso\n", scelta);
  printf("Fine\n");
}
```

Variante esercizio slide 37

Fare in modo che il programma inizialmente chieda all'utente un PIN, ed esegua le successive istruzioni solo se il PIN immesso è uguale a 44122. Il seguente programma funziona?

050_controllo_di_flusso/rubrica-pin.c

```
1  #include <stdio.h>
2
3  main() {
4    int pin, scelta;
5    scanf("%d", &pin);
6    if (pin == 44122)
7      printf("Accesso consentito\n");
8    printf("1. Tizio 335-1234567\n");
9    printf("2. Caio 347-1234567\n");
10   printf("Scegliere un contatto\n");
11   scanf("%d", &scelta);
12   printf("Chiamata al contatto %d in corso\n", scelta);
13   printf("Fine\n");
14 }
```

Tentativo

050_controllo_di_flusso/rubrica-pin.c

```
1  #include <stdio.h>
2
3  main() {
4    int pin, scelta;
5    scanf("%d", &pin);
6    if (pin == 44122)
7      printf("Accesso consentito\n");
8    printf("1. Tizio 335-1234567\n");
9    printf("2. Caio 347-1234567\n");
10   printf("Scegliere un contatto\n");
11   scanf("%d", &scelta);
12   printf("Chiamata al contatto %d in corso\n", scelta);
13   printf("Fine\n");
14 }
```

UNICA ISTRUZIONE CONDIZIONATA

Non funziona, perché l'istruzione `if` condiziona solo l'esecuzione dell'istruzione alla riga 7, mentre le successive vengono eseguite comunque.

Soluzione funzionante

050_controllo_di_flusso/rubrica-pin-molti-if.c

```
1  #include <stdio.h>
2
3  main() {
4    int pin, scelta;
5    scanf("%d", &pin);
6    if (pin == 44122) printf("Accesso consentito\n");
7    if (pin == 44122) printf("1. Tizio 335-1234567\n");
8    if (pin == 44122) printf("2. Caio 347-1234567\n");
9    if (pin == 44122) printf("Scegliere un contatto");
10   if (pin == 44122) scanf("%d", &scelta);
11   if (pin == 44122)
12     printf("Chiamata al contatto %d\n", scelta);
13   printf("Fine\n");
14 }
```

if (pin!=44122)

fine

Questa soluzione funziona, ma contiene molte ripetizioni della condizione. Sarebbe meglio verificare la condizione una sola volta e, in caso fosse falsa, saltare all'istruzione alla riga 13.

goto

L'effetto si ottiene con l'istruzione goto, o istruzione di salto.
Per indicare il punto del programma dove saltare si utilizza un'etichetta (label).

Sintassi

$\langle \textit{etichetta} \rangle ::= \langle \textit{identificatore} \rangle$ **:**

$\langle \textit{istruzioneSalto} \rangle ::=$ **goto** $\langle \textit{identificatore} \rangle$ **;**

Semantica

L'istruzione `goto __identificatore__;` comporta il salto del flusso all'istruzione immediatamente successiva all'etichetta `__identificatore__:`.

Soluzione dell'inconveniente alla slide 41

050_controllo_di_flusso/rubrica-pin-salto.c

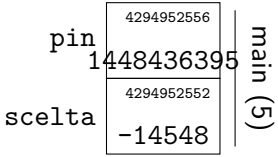
```c
#include <stdio.h>

main() {
  int pin, scelta;
  scanf("%d", &pin);
  if (pin != 44122)
    goto fine;
  printf("Accesso consentito\n");
  printf("1. Tizio 335-1234567");
  printf("2. Caio 347-1234567");
  printf("Scegliere un contatto\n");
  scanf("%d", &scelta);
  printf("Chiamata al contatto %d\n", scelta);
fine:
  printf("Fine\n");
}
```

Un salto soggetto a una condizione si chiama salto condizionato.

PIN corretto

```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```



PIN corretto

```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```

0

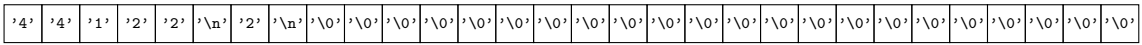
pin	4294952556 44122	main (6)
scelta	4294952552 -14548	

'4'	'4'	'1'	'2'	'2'	'\n'	'2'	'\n'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'

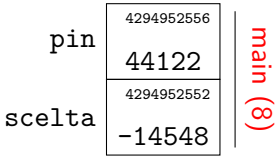
stdin

PIN corretto

```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```

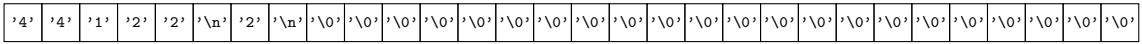


stdin

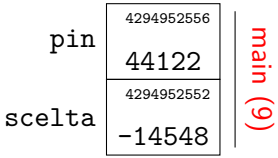


PIN corretto

```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```

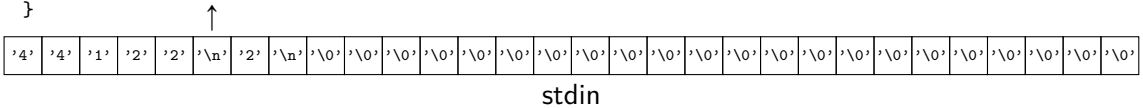


stdin



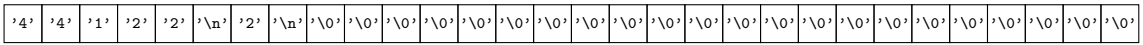
PIN corretto

```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```

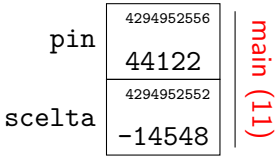


PIN corretto

```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```

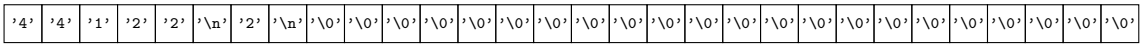


stdin

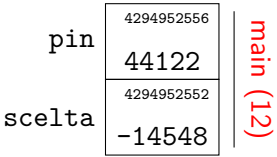


PIN corretto

```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```

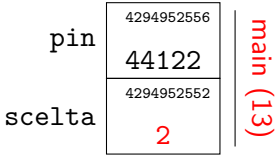


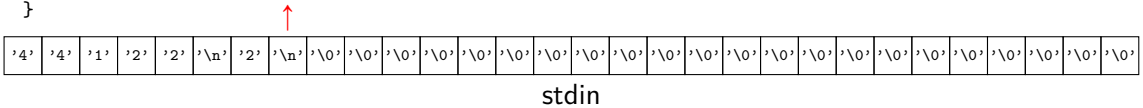
stdin



PIN corretto

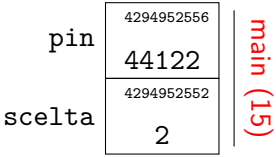
```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```

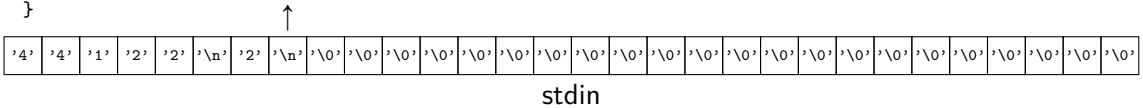




PIN corretto

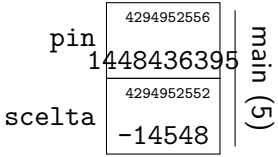
```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```





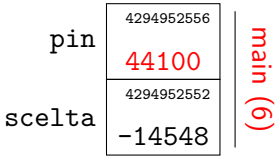
PIN errato

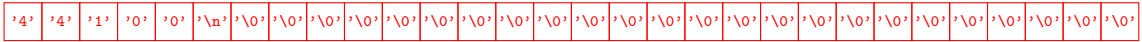
```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```



PIN errato

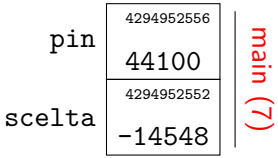
```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```

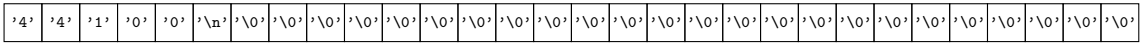




PIN errato

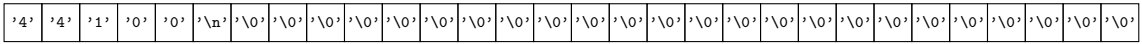
```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```



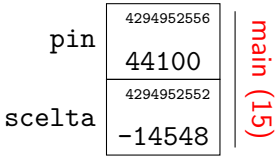


PIN errato

```
1   #include <stdio.h>
2
3   main() {
4     int pin, scelta;
5     scanf("%d", &pin);
6     if (pin != 44122)
7       goto fine;
8     printf("Accesso consentito\n");
9     printf("1. Tizio 335-1234567");
10    printf("2. Caio 347-1234567");
11    printf("Scegliere un contatto\n");
12    scanf("%d", &scelta);
13    printf("Chiamata al contatto %d\n", scelta);
14  fine:
15    printf("Fine\n");
16  }
```



stdin



Sommario

1. Istruzioni condizionali (o di selezione)
2. Salti
3. Blocchi
4. Iterazione
5. Programmazione strutturata
6. Selezione multipla

Blocco

In realtà, la ripetizione alla slide 41 si può eliminare senza ricorrere a goto, poiché il linguaggio C consente di mettere una sequenza di istruzioni dovunque si può mettere un'istruzione.

$\langle \textit{istruzione} \rangle ::= \langle \textit{istruzioneSemplice} \rangle$
$\quad | \quad \langle \textit{istruzioneDiControllo} \rangle$

$\langle \textit{istruzioneDiControllo} \rangle ::= \langle \textit{istruzioneCondizionale} \rangle$ if , if-else
$\quad | \quad \langle \textit{istruzioneSalto} \rangle$ goto
$\quad | \quad \langle \textit{blocco} \rangle$

{ a=3; printf(...), ... } { }

$\langle \textit{blocco} \rangle ::= \texttt{\{} \; [\![\langle \textit{istruzione} \rangle]\!]^* \; \texttt{\}}$

Un blocco è semplicemente una sequenza di istruzioni racchiuse fra parentesi graffe.

Semantica

L'esecuzione di un blocco consiste nell'esecuzione di tutte le istruzioni che lo compongono, dalla prima all'ultima.

Esempio blocco

In questo esempio, il ramo vero dell'`if` è l'intero blocco; quindi se la condizione è falsa è l'intero blocco a non essere eseguito.

050_controllo_di_flusso/rubrica-pin-blocco.c

```c
#include <stdio.h>

main() {
  int pin, scelta;
  scanf("%d", &pin);
  if (pin == 44122) {
    printf("Accesso consentito\n");
    printf("1. Tizio 335-1234567\n");
    printf("2. Caio 347-1234567\n");
    printf("Scegliere un contatto\n");
    scanf("%d", &scelta);
    printf("Chiamata al contatto %d\n", scelta);
  }
  printf("Fine\n");
}
```

Blocco nel ramo falso

Anche il ramo falso di un `if-else` può essere un blocco.

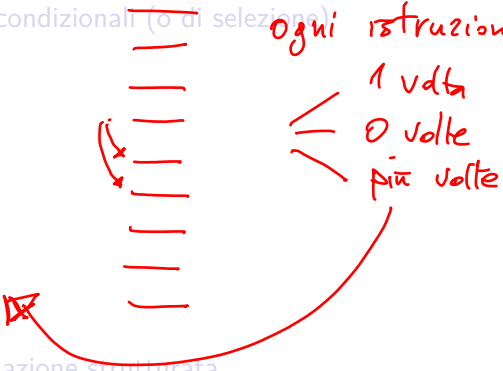
050_controllo_di_flusso/rubrica-pin-blocco-else.c

```
#include <stdio.h>

main() {
  int pin, scelta;
  scanf("%d", &pin);
  if (pin == 44122) {
    printf("Accesso consentito\n");
    printf("1. Tizio 335-1234567\n");
    printf("2. Caio 347-1234567\n");
    printf("Scegliere un contatto");
    scanf("%d", &scelta);
    printf("Chiamata al contatto %d\n", scelta);
  } else {
    printf("Accesso negato\n");
    printf("Il programma sta per terminare\n");
  }
  printf("Fine\n");
}
```

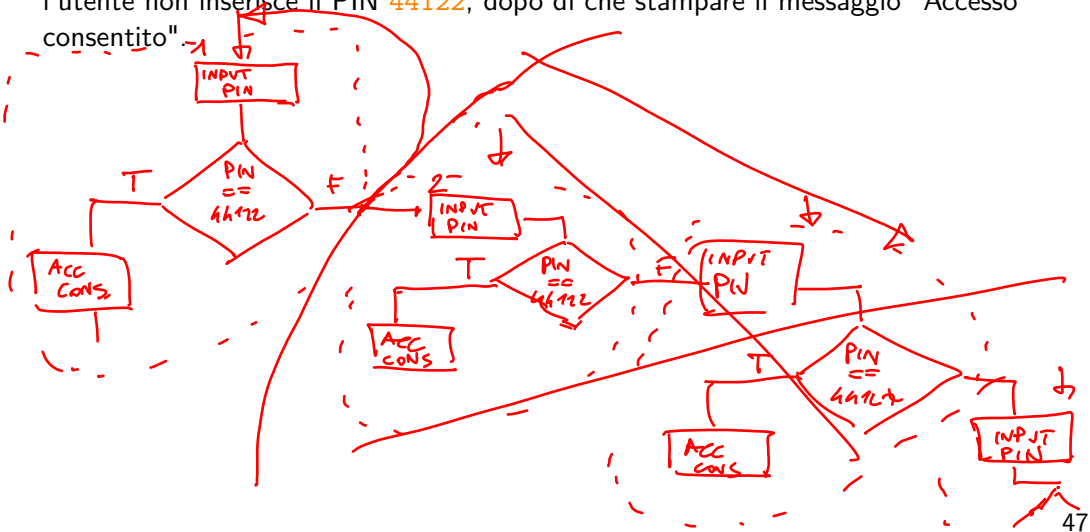
Sommario

1. Istruzioni condizionali (o di selezione)
2. Salti
3. Blocchi
4. Iterazione
5. Programmazione strutturata
6. Selezione multipla



Esercizio

Scrivere un programma che richieda all'utente un PIN, ripetendo la richiesta finché l'utente non inserisce il PIN 44122, dopo di che stampare il messaggio "Accesso consentito".



Richiesta PIN ripetuta

PIN

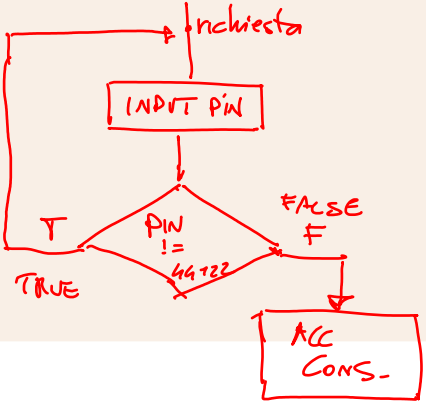
050_controllo_di_flusso/pin-goto.c

```c
#include <stdio.h>

main() {
  int pin;
richiesta:
  scanf("%d", &pin);
  if (pin != 44122)
    goto richiesta;
  printf("Accesso consentito\n");
}
```

CORPO

COND. PERM.



SCHEMA

Pattern ciclo

In programmazione, un pattern è una soluzione generale a un problema ricorrente.
Il pattern ciclo risolve il problema di dover ripetere l'esecuzione di certe istruzioni fino a che non si verifica una condizione.
E' caratterizzato (staticamente) da

FRAMMENTO DI CODICE

- corpo: le istruzioni da eseguire zero o più volte SCANF
- condizione di permanenza: espressione che, se falsa, comporta l'uscita dal ciclo (e quindi nessuna ulteriore esecuzione del ciclo)

PIN SBAGLIATO

Dinamicamente, ogni iterazione di un ciclo è una singola esecuzione del corpo del ciclo.
L'esecuzione del ciclo comporta, in generale, zero o più iterazioni.

Esercizio: identificazione di pattern

050_controllo_di_flusso/pin-goto.c

```
1  #include <stdio.h>
2
3  main() {
4    int pin;
5  richiesta:
6    scanf("%d", &pin);
7    if (pin != 44122)
8      goto richiesta;
9    printf("Accesso consentito\n");
10 }
```

CORPO

CONDIZIONE DI PERMANENZA

Questo programma contiene un'implementazione del pattern ciclo? SÌ Se sì:

- quale problema risolve? CONTINUARE A RICHIEDERE IL PIN FINCHÈ NON È ESATTO
- qual è il corpo del ciclo? RIGA 6
- qual è la condizione di permanenza? pin != 44122
- quante iterazioni vengono eseguite? ALMENO UNA

Problema

Scrivere un programma che stampi a video cinque asterischi e poi vada a capo.

050_controllo_di_flusso/star5.c

```
#include <stdio.h>

main() {
  printf("*****\n");
}
```

```
printf("*");
printf("*");
printf("*");
⋮
```

Fa quanto richiesto, ma sarebbe meglio evitare la ripetizione (e se dovessimo stampare 5000 asterischi?)

Evitare la ripetizione con i salti

050_controllo_di_flusso/star-loop.c

```c
#include <stdio.h>

main() {
scrivi:
  printf("*");
  goto scrivi;
  printf("\n");
}
```

****

Qual è il problema di questo programma? LOOP INFINITO

CICLO NON TERMINA

Variabile di controllo

050_controllo_di_flusso/star5-salto.c

```c
#include <stdio.h>

main() {
  int i = 1;
scrivi:
  printf("*");
  i++;
  if (i <= 5)
    goto scrivi;
  printf("\n");
}
```

VARIABILE DI CONTROLLO

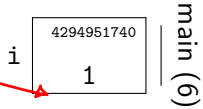
AGGIORNAMENTO

Qui, nel pattern ciclo, 1 funge da variabile di controllo, ossia

- compare nella condizione di permanenza (e quindi ne determina il valore), e
- viene modificata nel corpo del ciclo, in modo che la condizione di permanenza, prima o poi, diventi falsa (onde evitare loop infiniti).

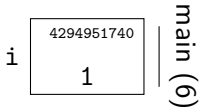
Stampa 5 asterischi

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

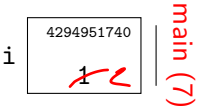


Stampa 5 asterischi

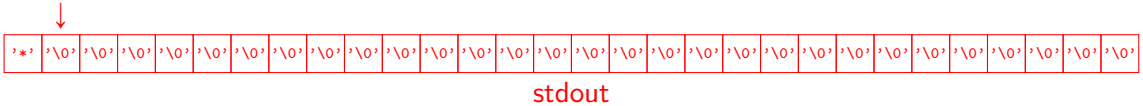
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```



Stampa 5 asterischi

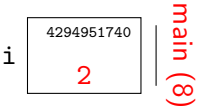


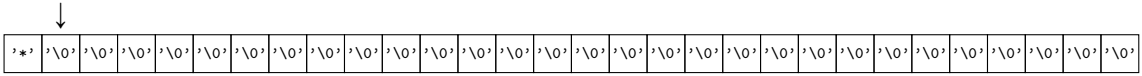

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```



Stampa 5 asterischi

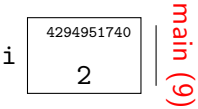
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

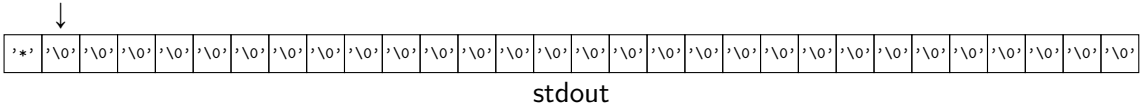




Stampa 5 asterischi

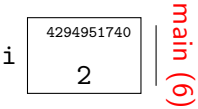
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

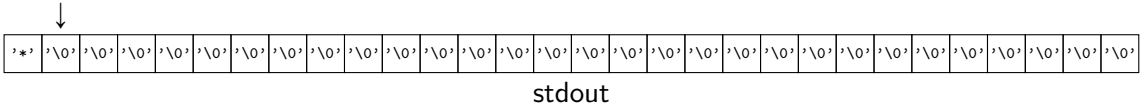




Stampa 5 asterischi

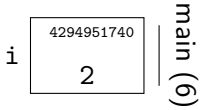
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

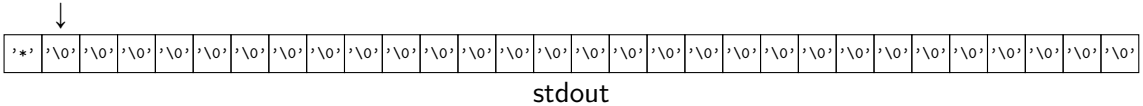




Stampa 5 asterischi

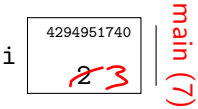
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

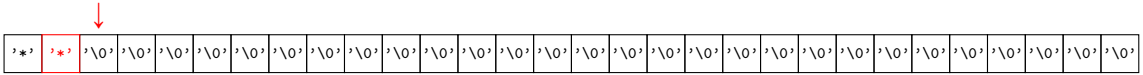




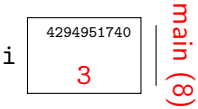
Stampa 5 asterischi

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```



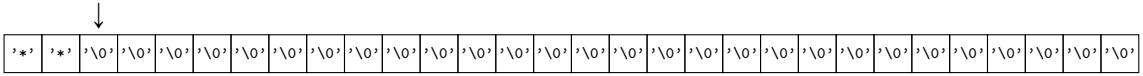


Stampa 5 asterischi



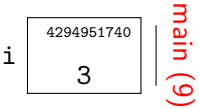

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

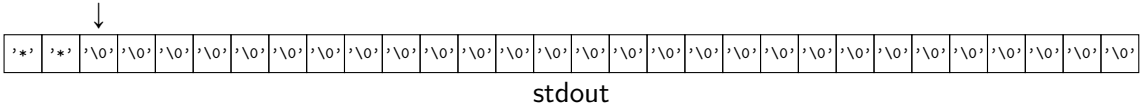
3



Stampa 5 asterischi

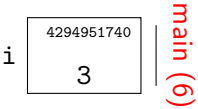
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```



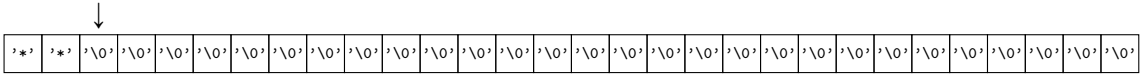


Stampa 5 asterischi

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

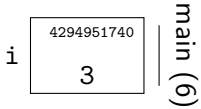


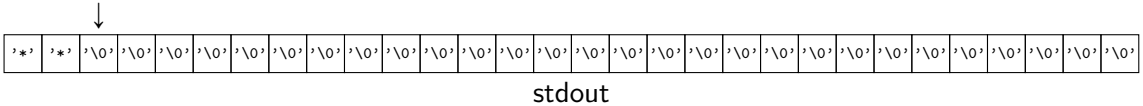
i	4294951740
	3



Stampa 5 asterischi

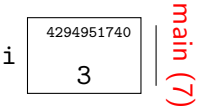
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

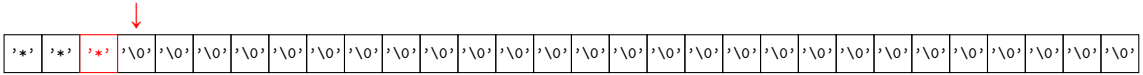




Stampa 5 asterischi

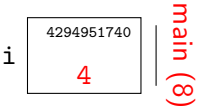
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

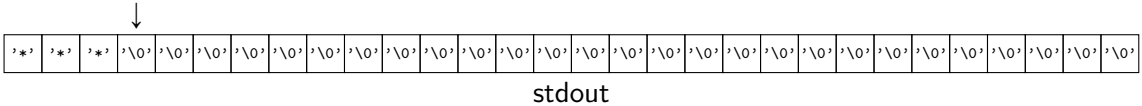




Stampa 5 asterischi

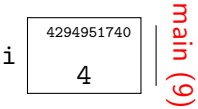
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

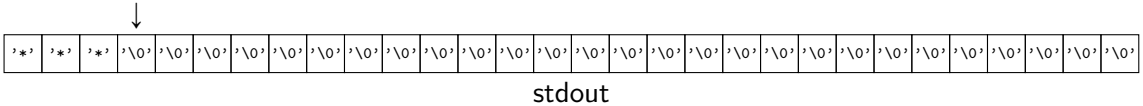




Stampa 5 asterischi

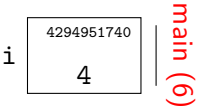
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```



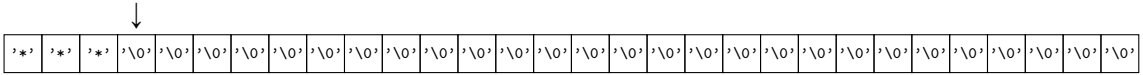


Stampa 5 asterischi

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

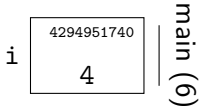


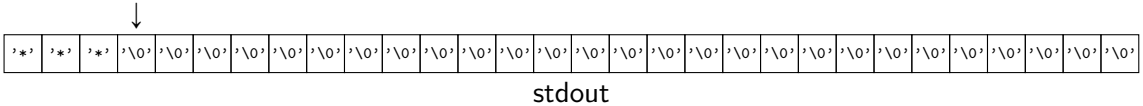
i	4294951740
	4



Stampa 5 asterischi

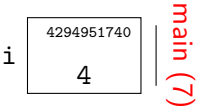
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

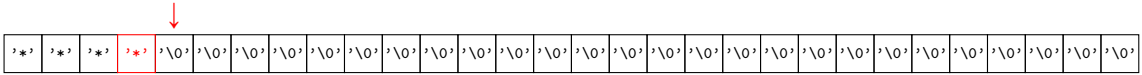




Stampa 5 asterischi

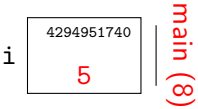
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

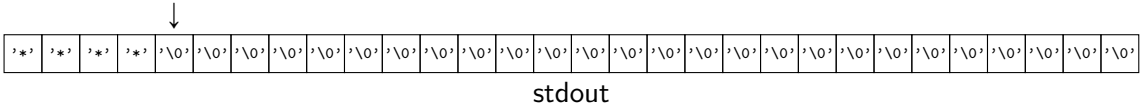




Stampa 5 asterischi

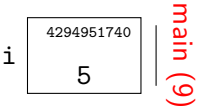
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;  5
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

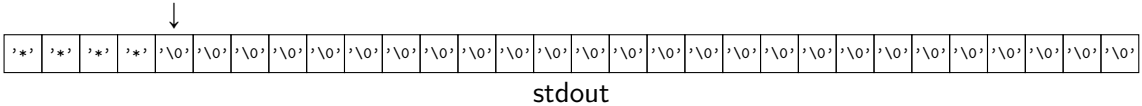




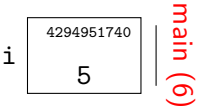
Stampa 5 asterischi

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

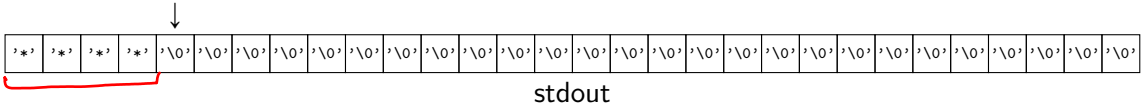




Stampa 5 asterischi

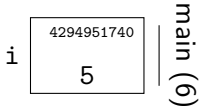


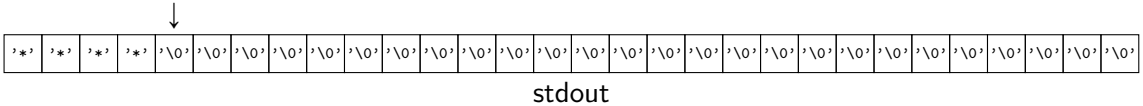

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```



Stampa 5 asterischi

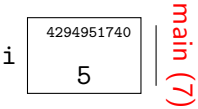
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

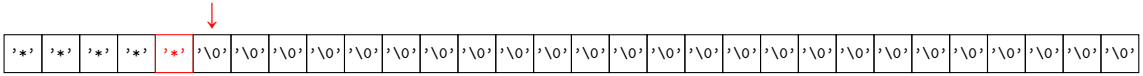




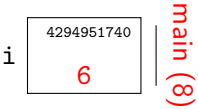
Stampa 5 asterischi

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

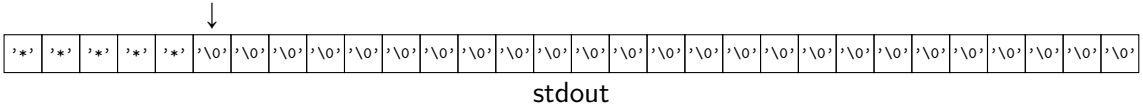




Stampa 5 asterischi

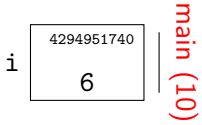


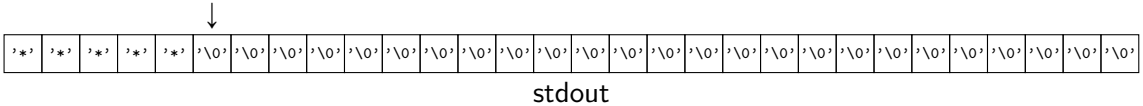

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```



Stampa 5 asterischi

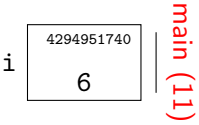
```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```

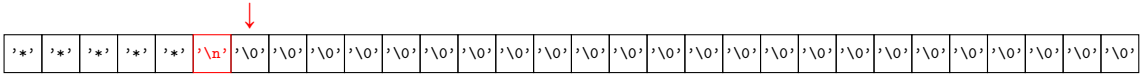




Stampa 5 asterischi

```
1   #include <stdio.h>
2
3   main() {
4     int i = 1;
5   scrivi:
6     printf("*");
7     i++;
8     if (i <= 5)
9       goto scrivi;
10    printf("\n");
11  }
```



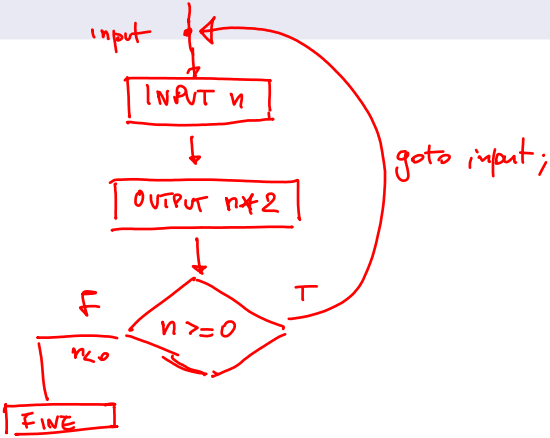


Esercizio

Doppio

Scrivere un programma che richieda all'utente numeri interi. Per ogni numero intero inserito, scriva il doppio. Il programma deve fermarsi quando l'utente scrive un numero negativo.

INPUT	OUTPUT
2	4
5	10
1	2
-3	-6?



Istruzioni di ciclo

IMPERATIVI

Il pattern ciclo è di uso tanto comune che tutti i linguaggi di programmazione lo supportano con costrutti appositi. Il C ne ha tre.

$\langle$*istruzioneDiControllo*$\rangle$ ::= $\langle$*istruzioneCondizionale*$\rangle$
| $\langle$*istruzioneSalto*$\rangle$
| $\langle$*blocco*$\rangle$
| $\langle$*ciclo*$\rangle$

$\langle$*ciclo*$\rangle$ ::= **do** $\langle$*istruzione*$\rangle$ **while** **(** $\langle$*espressione*$\rangle$ **);**
| **while** **(** $\langle$*espressione*$\rangle$ **)** $\langle$*istruzione*$\rangle$
| **for** **(** $\langle$*espressione*$\rangle$ **;** $\langle$*espressione*$\rangle$ **;** $\langle$*espressione*$\rangle$ **)** $\langle$*istruzione*$\rangle$

Sono di solito identificate con le rispettive parole chiave (`do-while`, `while` e `for`).

do-while

Sintassi

`do` ⟨*istruzione*⟩ `while (` ⟨*espressione*⟩ `);`

LOGICA

Semantica

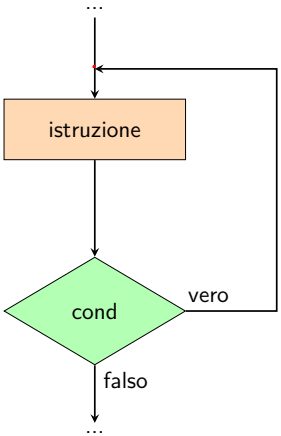
1. esegui ⟨*istruzione*⟩ CORPO
2. se ⟨*espressione*⟩ è vera, torna a 1

CONDIZIONE DI PERMANENZA

Quindi ⟨*istruzione*⟩ viene sempre eseguita almeno una volta.
Il numero di iterazioni di un ciclo `do-while`, in generale, non è prevedibile.
Bisogna fare sì che ~~[illegible]~~ istruzione renda falsa ⟨*espressione*⟩ in un numero finito di iterazioni, altrimenti l'esecuzione del ciclo continuerà all'infinito e il programma non terminerà.

LOOP

Diagramma di flusso `do...while`

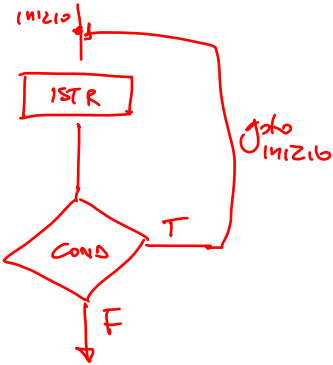



```
do __istruzione__
```

CORPO

```
while (__cond__);
```

CONDIZIONE DI PERMANENZA



Programma della slide 50 con do-while

050_controllo_di_flusso/pin-do.c

```c
#include <stdio.h>

main() {
  int pin;
  do
    scanf("%d", &pin);
  while (pin != 44122);
  printf("Accesso consentito\n");
}
```

Esercizio

Asterischi con do-while

Riscrivere il programma alla slide 53, riportato di seguito, sostituendo il pattern ciclo con un ciclo `do-while`.

050_controllo_di_flusso/star5-salto.c

```
1  #include <stdio.h>
2
3  main() {
4    int i = 1;
5  scrivi:
6    printf("*");
7    i++;
8    if (i <= 5)
9      goto scrivi;
10   printf("\n");
11 }
```

CONTATORE

CICLO CON GOTO

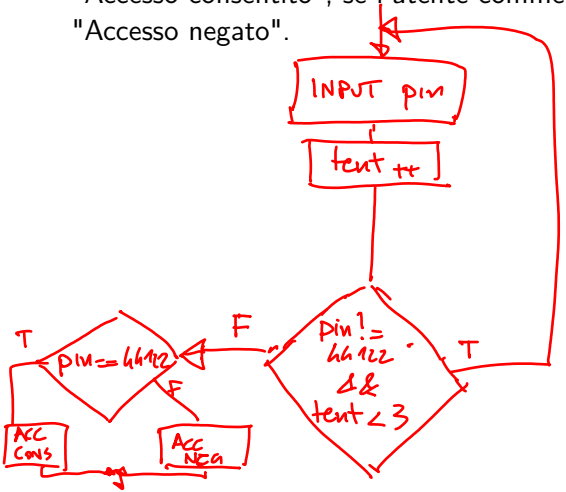
CORPO

COND. PERM

do <corpo>
while (<cond perm>);

Esercizio

Scrivere un programma che richieda all'utente un PIN al massimo tre volte. Se l'utente inserisce il PIN esatto (uguale a 44122) il programma deve stampare il messaggio "Accesso consentito"; se l'utente commette tre errori, deve stampare il messaggio "Accesso negato".



Tre tentativi

050_controllo_di_flusso/pin-do-var.c

```c
#include <stdio.h>

main() {
  int pin;
  int tentativi = 0;
  do {
    tentativi++;
    scanf("%d", &pin);
  } while (tentativi < 3 && pin != 44122);
  if (pin == 44122)
    printf("Accesso consentito\n");
  else
    printf("Accesso negato\n");
}
```

Sintassi

while (⟨*espressione*⟩) ⟨*istruzione*⟩

COND. PERM. CORPO

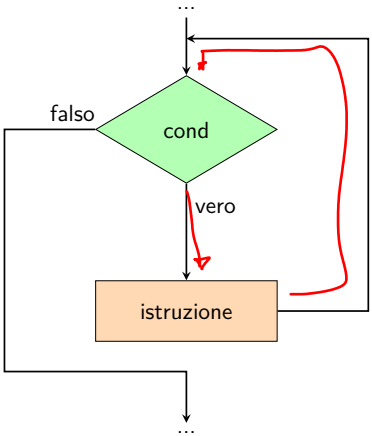
Semantica

1. se ⟨*espressione*⟩ è falsa, esci dal ciclo
2. esegui ⟨*istruzione*⟩
3. vai a 1

COND. PERM. CORPO

Quindi ⟨*istruzione*⟩ non viene mai eseguita, se ⟨*espressione*⟩ è inizialmente falsa. Anche per i cicli `while`, in generale, il numero di iterazioni non è prevedibile e il programmatore deve assicurarsi che ⟨*espressione*⟩, prima o poi, diventi falsa.

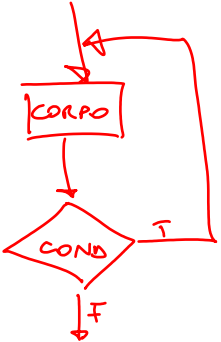
Diagramma di flusso `while`




```
while (__cond__)
```

```
__istruzione__
```

CORPO



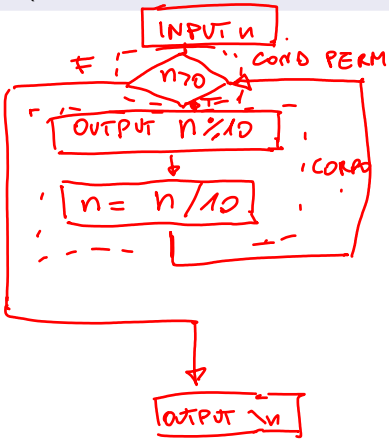
Esercizio

n

26 % 10 → 6
50 % 10 → 0

Inversione cifre

Scrivere un programma che richieda all'utente un numero intero e lo stampi da destra a sinistra (ad esempio, se l'utente scrive 251017 il programma deve scrivere 710152).



251017

for

Sintassi

for (⟨*init*⟩ ; ⟨*cond*⟩ ; ⟨*update*⟩) <istruzione> CORPO

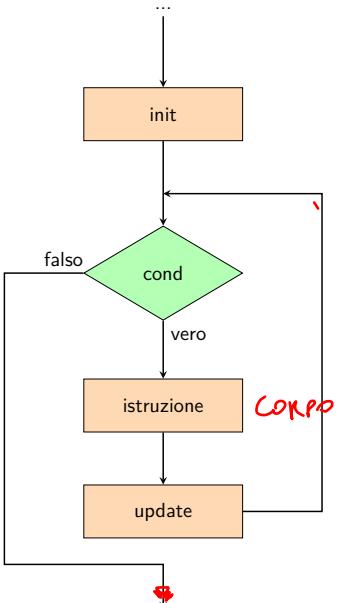
⟨*init*⟩, ⟨*cond*⟩ e ⟨*update*⟩ sono tutte espressioni.

Semantica

1. valuta ⟨*init*⟩
   COND. PERM.
2. se ⟨*cond*⟩ è falsa, esci dal ciclo
3. esegui ⟨*istruzione*⟩ CORPO
4. valuta ⟨*update*⟩
5. vai a 2

F

Diagramma di flusso `for`




```
for (__init__, __cond__, __update__)
```

```
__istruzione__
```

Uso di `for`

1. 2 3 4 5 6 7 8 9 10

Un utilizzo tipico del ciclo `for` è far assumere a una variabile di controllo (spesso usata nell'istruzione) una sequenza predefinita di valori (tipicamente un intervallo intero).
In questo caso (ad esempio se si vuol far variare `i` fra `2` e `10`):

- `__init__` imposta il valore iniziale della variabile (`i = 2`)
- `__cond__` verifica che il valore della variabile di controllo sia nell'intervallo desiderato (`i <= 10`)
- `__istruzione__` esegue qualche operazione (utilizzando o meno la variabile di controllo) (`printf("%d\n", i);`)
- `__update__` aggiorna il valore della variabile di controllo (`i++`)

for (init, cond, update) istr (corpo)
i=2 ; i<=10 ; i++ printf("%d\n", i);

Stampa di asterischi (slide 51) con for

050_controllo_di_flusso/star-for.c

```
#include <stdio.h>

main() {
  int i;
  for (i = 0; i < 5; i++)
    printf("*");
  printf("\n");
}
```

INIT COND UPDATE

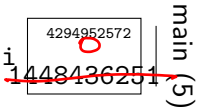
Qui i è la variabile di controllo e non è utilizzata direttamente nell'istruzione; il suo ruolo è solo determinare il numero di iterazioni.

- Quante sono le iterazioni? 5
- In quale intervallo varia i, ossia quali sono i valori che assume? 0 1 2 3 4 5
- Come bisogna modificare il programma affinché assuma i valori fra 5 e 9? Come cambia in questo caso l'output del programma?

for (i=5 ; i<9; i++) printf("*"),

Stampa asterischi

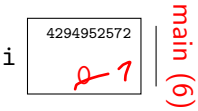
```
1  #include <stdio.h>
2
3  main() {
4    int i;
5    for (i = 0; i < 5; i++)
6      printf("*");
7    printf("\n");
8  }
```



Stampa asterischi

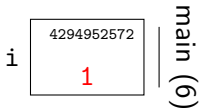
```
1  #include <stdio.h>
2
3  main() {
4    int i;
5    for (i = 0; i < 5; i++)
6      printf("*");
7    printf("\n");
8  }
```

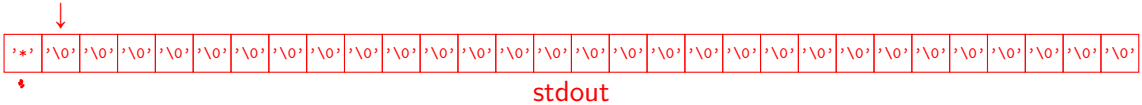
UPDATE



Stampa asterischi

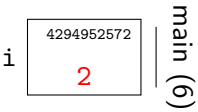
```
1  #include <stdio.h>
2
3  main() {
4    int i;
5    for (i = 0; i < 5; i++)
6      printf("*");
7    printf("\n");
8  }
```

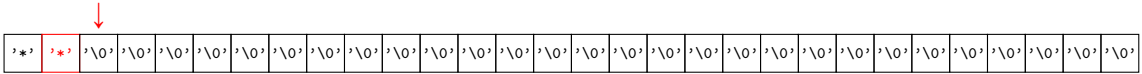




Stampa asterischi

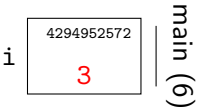
```
1  #include <stdio.h>
2
3  main() {
4    int i;
5    for (i = 0; i < 5; i++)
6      printf("*");
7    printf("\n");
8  }
```

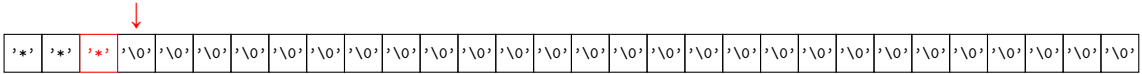




Stampa asterischi

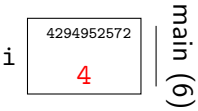
```
1  #include <stdio.h>
2
3  main() {
4    int i;
5    for (i = 0; i < 5; i++)
6      printf("*");
7    printf("\n");
8  }
```

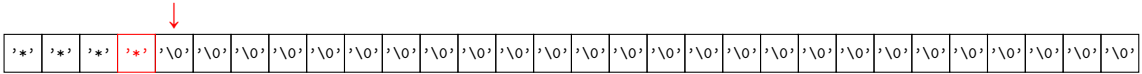




Stampa asterischi

```
1  #include <stdio.h>
2
3  main() {
4    int i;
5    for (i = 0; i < 5; i++)
6      printf("*");
7    printf("\n");
8  }
```





Stampa asterischi

```
1  #include <stdio.h>
2
3  main() {
4    int i;
5    for (i = 0; i < 5; i++)
6      printf("*");
7    printf("\n");
8  }
```

COND

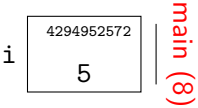
i	4294952572	main (7)
	5	

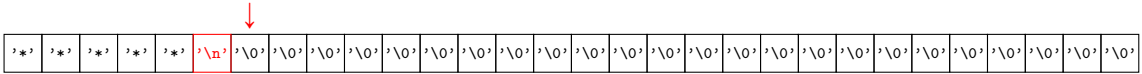
'*'	'*'	'*'	'*'	'*'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'

stdout

Stampa asterischi

```
1   #include <stdio.h>
2
3   main() {
4     int i;
5     for (i = 0; i < 5; i++)
6       printf("*");
7     printf("\n");
8   }
```





Esercizio

OSS sequence

n $\boxed{6}$

$1 + 2 + 3 + 4 + 5 + 6 = 21$

Università degli Studi di Ferrara

Somma interi

Si scriva un programma che stampi la somma dei primi *n* numeri naturali, dove *n* è un numero richiesto all'utente.

INPUT	OUTPUT
6	21
4	10
100	5050

s1
1
2
3
4
5
6

reduce (+, 0, s1)

s $\boxed{\cancel{0}\ \cancel{1}\ \cancel{3}\ \cancel{6}\ \cancel{10}\ \cancel{15}\ 21}$

accumulatore

Terminologia

Nei cicli, spesso

- una variabile di controllo il cui valore corrisponde al numero di iterazioni del ciclo (cioè vale 1 alla prima iterazione, 2 alla seconda, etc.) è detta contatore.
- una variabile che contiene il risultato parziale di un calcolo, e che conterrà il risultato completo all'uscita dal ciclo, è detta accumulatore.

Nella soluzione dell'esercizio alla slide 69, ci sono contatori o accumulatori? Come si chiamano?

i s

Esercizio

Somma input

Si scriva un programma che legga da tastiera 5 numeri interi e ne stampi la somma.

Esercizio

Moltiplicazione

Stampare il prodotto di due numeri m e n richiesti all'utente, calcolandolo come somma di n addendi pari a m.

m	n	output
5	2	10
2	3	6

m=5 n=2

5+5 =10

s1 = 5, 5

m=2 n=3

2+2+2 = 6

s1 = 2, 2, 2

reduce (+, 0, s1)

Esercizio

m 12
n 5

Università degli Studi di Ferrara

Divisione

Stampare il quoziente e il resto di due numeri m e n richiesti all'utente, usando il metodo delle sottrazioni successive.
Il quoziente è il numero di volte che è necessario sottrarre n da m per ottenere un risultato negativo, diminuito di 1. Il resto è il sottraendo dell'ultima sottrazione.
Ad esempio, il quoziente intero di 12 e 5 è 2 perché sottraendo 5 da 12 3 volte si ottiene un numero negativo; il resto è 2, ossia il sottraendo della terza sottrazione.

conteggio 2 → quoziente

12 ≥5 -5
7 ≥5 -5
2 → resto

3 elem

7 / 2
5
3
1 → RESTO → quoziente

reduce (++, 0, s1)

Esercizio TEST DI PRIMALITÀ

Primo

Verificare se un numero, richiesto all'utente, è primo, ossia se non ha divisori interi a parte 1 e se stesso.

INPUT	DIVISIBILE PER	OUTPUT
5	1, 5	2 PRIMO
12	1, 2, 3, 4, 6, 12	6 NON PRIMO
8	1, 2, 4, 8	4 NON PRIMO
7	1, 7	2 PRIMO

8 % i == 0

1,	1
2,	2
3,	
4,	4
5,	
6,	8
7,	
8	

CONTARE → 4

1. SEQUENZA DIVISORI
2. CONTARE DIVISORI
3. SE CONTEGGIO È 2 ALLORA IL NUMERO È PRIMO
   ALTRIMENTI NON È PRIMO

se n non primo

$n = h \cdot k \qquad h \leq k$

$h^2 \leq \underbrace{h \cdot k}_{n}$

$h^2 \leq n$ h a un divisore <= della sua radice quadrata

Esercizio

Primi

Stampare tutti i numeri primi minori o uguali a un numero richiesto all'utente.

s_1 filter(primo, s_1) = s_2 10

2 2

3 3 map(

4 printf,

5 5 s_2)

6

7 7

8

9

10

Esercizio

FIZZ BUZZ

BimBum

Scrivere un programma che, per ogni numero compreso fra 1 e 100, scriva:

map

- "bim" se il numero è divisibile per 3; 6 bim
- "bum" se il numero è divisibile per 5; 10 bum
- "bimbum" se il numero è divisibile per 3 e per 5; bimbum
- altrimenti, il numero stesso. 4

1	1
2	2
3	bim
4	4
5	bum
6	bim
15	bim bum
100	bum

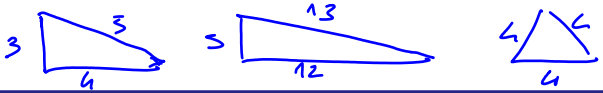
Esercizio

Fattorizzazione

Calcolare la fattorizzazione di un numero richiesto all'utente, ossia esprimere il numero come prodotto di potenze aventi come base numeri primi crescenti.
Ad esempio, la fattorizzazione di 300 è $2^2 \times 3 \times 5^2$.

INPUT output

Esercizio



Terne

Calcolare tutte le terne pitagoriche in cui l'ipotenusa è minore o uguale di un numero letto da tastiera.

10

FILTER terna pitagorica $a^2+b^2=c^2$ $a \le b \le c$

a	b	c
1	1	1
1	1	2
1	1	3
.	.	.
1	1	10
1	2	1
1	2	2
⋮		
3	4	5
6	8	10
10	10	10

3 4 5

6 8 10

Cicli annidati

Come i rami dell'istruzione `if-else`, anche il corpo di un ciclo è un'istruzione, senza restrizioni.
Può quindi essere un'istruzione semplice, un blocco, un condizionale o un altro ciclo, di qualunque tipo.
Non ci sono limiti all'annidamento di istruzione composte. Ad esempio:

```
while (...) {
  ...
  for (...;...;...)
  if (...) {
    do {
      ...
    } while (...);
  }
}
```

Esercizio

Tavola pitagorica

Scrivere un programma che stampi a video la tavola pitagorica, cioè la tabella di 10 righe e 10 colonne tale che il numero alla riga i e alla colonna j è il prodotto di i e j:

Colonna

	1	2	...	9	10
riga.	2	4	...	18	20
	...	...	...	...	...
	10	20	...	90	100

$i \cdot j$

Varianti dell'esercizio alla slide 80

printf("%3d", 2) stampa " 2"

Tavola pitagorica con allineamento

Stampare la tavola pitagorica allineando correttamente i numeri per colonne. Può essere utile lo specificatore di conversione `%__n__d`, dove `__n__` è un naturale, che fa sì che un intero venga stampato usando *almeno* `__n__` caratteri, aggiungendo spazi a sinistra se necessario. Ad esempio, `printf("%3d",2)` stampa " 2".

Tavola pitagorica con allineamento manuale

Stampare la tavola pitagorica allineando correttamente i numeri per colonne, senza usare lo specificatore di conversione appena introdotto. Solo "%d"

Tavola pitagorica di lato qualsiasi

Stampare la tavola pitagorica di lato (cioè numero di righe e colonne) scelto dall'utente.

Sommario

1. Istruzioni condizionali (o di selezione)
2. Salti
3. Blocchi
4. Iterazione
5. Programmazione strutturata
6. Selezione multipla

Go To Statement Considered Harmful

Crisi del software FINE ANNI '60

- Difficoltà dello scrivere programmi affidabili nel rispetto di tempi e budget
- Termine coniato alla NATO Software Engineering Conference del 1968
- Colpevole individuato nell'istruzione `goto`
- Motivo: `goto` rende più difficile ragionare sui programmi, e quindi modificarli per estenderne la funzionalità o correggerne gli errori.

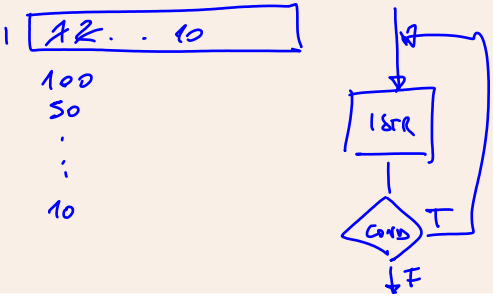
Ragionamento su codice

Il seguente programma mostra i quozienti interi di 100 per tutti gli interi compresi fra 1 e 10.

050_controllo_di_flusso/elenco-quozienti-do.c

```c
#include <stdio.h>

main() {
  int i = 1;
  do {
    printf("%d\n", 100 / i);
    i++;
  } while (i <= 10);
}
```



Nel corpo del ciclo do c'è pericolo che venga eseguita una divisione per 0? No, perché i vale inizialmente 1 e viene solo incrementato.

Ragionamento su codice

Stesso programma con il ciclo do sostituito da pattern ciclo:

050_controllo_di_flusso/elenco-quozienti-goto.c

```c
#include <stdio.h>

main() {
  int i = 1;
ciclo:
  printf("%d\n", 100 / i);
  i++;
  if (i <= 10)
    goto ciclo;
}
```

i=0;
goto ciclo;

Che cosa succede se, altrove nel programma, mettiamo le istruzioni `i = 0; goto ciclo;`?

Il problema è che al corpo del ciclo si può arrivare da più punti, quindi non siamo più sicuri del valore iniziale di `i`.

Eliminazione di goto

- In generale, i costrutti di selezione e iterazione di un programma senza salti possono essere considerati come un'istruzione unica, perché hanno un solo punto di ingresso e un solo punto di uscita. Questo rende più facile ragionare sull'effetto del programma.
- Teorema di Böhm-Jacopini (1966): qualsiasi programma con `goto` può essere riscritto usando solo blocchi, `if-else` e `while`.
- I programmi scritti così sono detti strutturati.
- Quindi se `goto` rende un programma difficile da leggere, trasformiamolo in un programma strutturato, come il teorema garantisce possibile! Problema risolto.

Eliminazione di `goto`

- In generale, i costrutti di selezione e iterazione di un programma senza salti possono essere considerati come un'istruzione unica, perché hanno un solo punto di ingresso e un solo punto di uscita. Questo rende più facile ragionare sull'effetto del programma.
- Teorema di Böhm-Jacopini (1966): qualsiasi programma con `goto` può essere riscritto usando solo blocchi, `if-else` e `while`.
- I programmi scritti così sono detti strutturati.
- Quindi se `goto` rende un programma difficile da leggere, trasformiamolo in un programma strutturato, come il teorema garantisce possibile! Problema risolto.
- Il problema è che a volte il programma trasformato in strutturato non è più leggibile di quello iniziale con `goto`!
  In altre parole, il teorema garantisce di poter sempre scrivere i programmi in modo strutturato, ma non garantisce che non esista nessun caso in cui valga la pena usare `goto`.

Problema

3 2 6 -3 → 11

Scrivere un programma che richieda all'utente dei numeri interi; quando l'utente inserisce un numero negativo, il programma deve stampare la somma dei numeri positivi inseriti e terminare.

Il seguente programma è corretto?

050_controllo_di_flusso/somma-interi-difettoso.c

```c
#include <stdio.h>

main() {
  int numero, somma = 0;
  do {
    printf("Digita un intero (negativo termina):\n");
    scanf("%d", &numero);
    somma += numero;
  } while (numero >= 0);
  printf("La somma e` %d\n", somma);
}
```

accumulatore

Correzione

050_controllo_di_flusso/somma-interi-do.c

```
1  #include <stdio.h>
2
3  main() {
4    int numero, somma = 0;
5    do {
6      printf("Digita un intero (negativo termina):\n");
7      scanf("%d", &numero);
8      if (numero >= 0)
9        somma += numero;
10   } while (numero >= 0);
11   printf("La somma e` %d\n", somma);
12 }
```

Questo programma funziona, ma la condizione di validità è ripetuta.

Eliminiamo la ripetizione della condizione di permanenza

050_controllo_di_flusso/somma-interi-while.c

```
#include <stdio.h>

main() {
  int numero, somma = 0;
  printf("Digita un intero (negativo termina):\n");
  scanf("%d", &numero);
  while (numero >= 0) {
    somma += numero;
    printf("Digita un intero (negativo termina):\n");
    scanf("%d", &numero);
  }
  printf("La somma e` %d\n", somma);
}
```

Ora però stiamo ripetendo le istruzioni di output e input.

Soluzione con goto

050_controllo_di_flusso/somma-interi-goto.c

```
1  #include <stdio.h>
2
3  main() {
4    int numero, somma = 0;
5  richiedi:
6    printf("Digita un intero (negativo termina):\n");
7    scanf("%d", &numero);
8    if (numero < 0)
9      goto fine;
10   somma += numero;
11   goto richiedi;
12 fine:
13   printf("La somma e` %d\n", somma);
14 }
```

Senza ripetizioni: goto permette di uscire dal ciclo in qualsiasi passo del corpo.

break

L'utilità di `goto` in casi come questo è riconosciuta; tuttavia molti linguaggi (tra cui il C) si preferisce rendere esplicito questo pattern con un costrutto apposito.

Sintassi

```
break;
```

Semantica

All'interno di un ciclo qualsiasi, l'esecuzione di `break;` comporta l'uscita dal ciclo (cioè il passaggio del flusso alla prima istruzione successiva all'istruzione di ciclo)

Soluzione con break

050_controllo_di_flusso/somma-interi-break.c

```c
#include <stdio.h>

main() {
  int numero, somma = 0;
  while (1) {
    printf("Digita un intero (negativo termina):\n");
    scanf("%d", &numero);
    if (numero < 0)
    break;
    somma += numero;
  }
  printf("La somma e` %d\n", somma);
}
```

continue

A volte l'istruzione `goto` sarebbe utile non per uscire dal ciclo, ma per terminare l'iterazione attuale, saltando alla successiva. Questo pattern è reso esplicito in C dall'istruzione `continue`.

Sintassi

```
continue;
```

Semantica

All'interno di un ciclo qualsiasi, l'esecuzione di `continue;` comporta la fine dell'iterazione (cioè il passaggio del flusso alla fine del corpo: alla verifica della condizione di permanenza per i cicli `while` e `do-while` e all'istruzione di update per i cicli `for`)

```
while (cond) {
    continue;
}
```

Esercizio

INPUT
3 2 -1 4 -6

OUTPUT
9

Università degli Studi di Ferrara

Scrivere un programma che richieda all'utente 5 numeri interi e stampi la somma dei soli numeri positivi inseriti.

050_controllo_di_flusso/somma-interi-positivi.c

```c
#include <stdio.h>

main() {
  int i, numero, somma = 0;
  for (i = 0; i < 5; i++) {
    printf("Digita un intero:\n");
    scanf("%d", &numero);

    if (numero <= 0) continue;
    somma += numero;
  }
  printf("La somma e` %d\n", somma);
}
```

Esercizio

Scrivere un programma che proceda come per stampare la tavola pitagorica di lato 10, ma fermandosi dopo aver stampato il primo numero maggiore di 50.

1 2 3 4 5 . 10
2 4 6 8 10 . . 20
.
10 100

```
for (i=1, i<=10, i++) {
    for (j=1, j<=10, j++)
        printf ("%d ", i*j);
    printf("\n"),
}
```

050_controllo_di_flusso/uscita-doppio.c

```c
#include <stdio.h>

main() {
  int i, j;

  for (i = 1; i <= 10; i++) {
    for (j = 1; j <= 10; j++) {
      printf("%3d ", i * j);
      if (i * j > 50)
        // qui vorrei terminare l'esecuzione
        ;
    }
    printf("\n");
  }
  printf("Finito\n");
}
```

Tentativo con break

050_controllo_di_flusso/uscita-doppio-break.c

```c
#include <stdio.h>

main() {
  int i, j;

  int finito = 0;
  for (i = 1; i <= 10; i++) {
    for (j = 1; j <= 10; j++) {
      printf("%3d ", i * j);
      if (i * j > 50)
        break;
    }
    printf("\n");
  }
  printf("Finito\n");
}
```

Funziona?

Soluzione con flag

050_controllo_di_flusso/uscita-doppio-flag.c

```c
#include <stdio.h>

main() {
  int i, j;
      FLAG
  int finito = 0;
  for (i = 1; !finito && i <= 10; i++) {
    for (j = 1; !finito && j <= 10; j++) {
      printf("%3d ", i * j);
      if (i * j > 50)
        finito = 1;  ← IMPOSTAZIONE FLAG
    }
    printf("\n");
  }
  printf("Finito\n");
}
```

Soluzione con goto

050_controllo_di_flusso/uscita-doppio-goto.c

```c
#include <stdio.h>

main() {
  int i, j;

  for (i = 1; i <= 10; i++) {
    for (j = 1; j <= 10; j++) {
      printf("%3d ", i * j);
      if (i * j > 50)
        goto finito;
    }
    printf("\n");
  }
finito:
  printf("Finito\n");
}
```

Non richiede introduzione di flag; è uno dei pochi utilizzi legittimi di goto in programmi strutturati.

Conclusioni sull'uso di `goto`

L'istruzione `goto`, in linguaggio C, non è quasi mai indispensabile.

- Dove si userebbe per implementare il pattern ciclo si possono usare `while`, `do-while` e `for` a seconda dei casi;
- Dove si userebbe per l'uscita dal ciclo o dall'iterazione attuale si possono usare `break` e `continue`;
- E' effettivamente la soluzione migliore per l'uscita da cicli annidati

L'utilizzo di `goto` quando è disponibile un costrutto più specifico è un errore di stile.

Differenze fra istruzioni di iterazione

- Il linguaggio C offre tre istruzioni di ciclo (`do-while`, `while` e `for`).
- E' lecito chiedersi: in cosa differiscono in termini di espressività? Ovvero, c'è qualcosa che si può fare con una istruzione e non con un'altra?
- Nelle prossime diapositive confronteremo l'espressività delle istruzioni di iterazione e del pattern ciclo. Faremo l'ipotesi che il corpo dei cicli non contenga etichette di salto (cioè non si possa entrare nel corpo tramite un'istruzione `goto`).

da while a do...while

Se non avessimo a disposizione il `while` ma solo il `do-while`, come potremmo scrivere codice equivalente?

```
while (i < 3) {
   printf("%d\n", i);
   i++;
}
```

i [~~2~~ 3]

output 2

i [4]

```
if(i<3){
do {
  printf("%d \n",i),
  i++;
} while (i<3);
}
```

i [~~2~~ 3]

output 2

i [4]

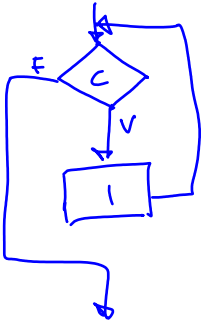
output

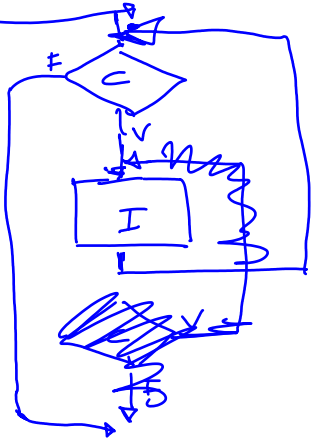
da while a do...while

In generale sono equivalenti:

```
while (__condizione__)
__istruzione__
```

```
if (__condizione__)
do __istruzione__
while (__condizione__);
```





da do-while a while

Se non avessimo a disposizione il do-while ma solo il while, come potremmo scrivere codice equivalente al seguente?

```
do {
  printf("%d\n", i);
  i++;
} while (i < 3)
```

{ printf("%d\n", i);
i++;
}
while (i < 3) {
printf("%d\n", i);
i++;
}

output "n

1 $\boxed{4}$

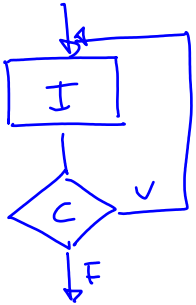
output 4

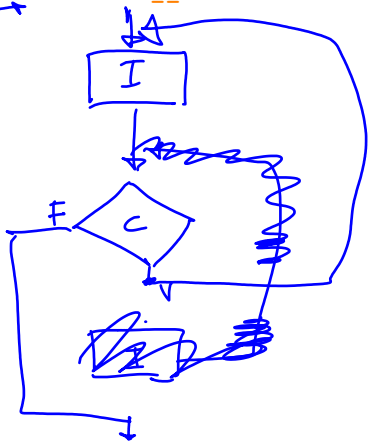
da do-while a while

In generale sono equivalenti:

```
do __istruzione__
while (__condizione__);
```

```
__istruzione__
while (__condizione__)
__istruzione__
```

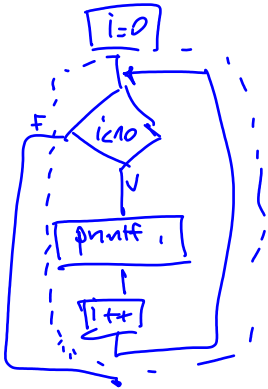




da for a while

Se non avessimo a disposizione il `for` ma solo il `while`, come potremmo scrivere codice equivalente al seguente?

```
for (i=0; i<10;i++)
printf("%d",i);
```




```
i=0;
while (i<10){
    printf i;
    i++;
}
```

da for a while

In generale sono equivalenti:

```
for (__init__,__cond__,__update__)
__istruzione__
```

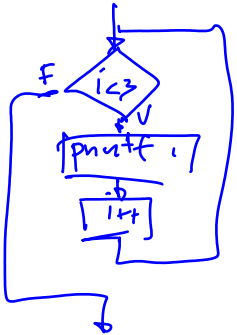
```
__init__;
while (__cond__)
{__istruzione__
__update__;}
```

da while a for

Università degli Studi di Ferrara

Se non avessimo a disposizione il `while` ma solo il `for`, come potremmo scrivere codice equivalente al seguente?

```
while (i < 3) {
  printf("%d\n", i);
  i++;
}
```




```
for( ; i<3 ; ){
  printf i;
  i++;
}
```

da while a for

In generale sono equivalenti:

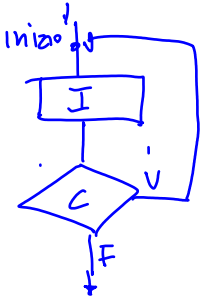
```
while (__condizione__)
__istruzione__
```

```
for (;__cond__;)
__istruzione__
```

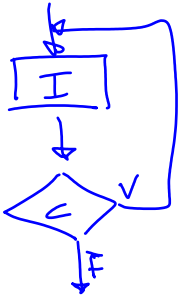
Da pattern ciclo a do-while e viceversa

In generale sono equivalenti:

```
inizio:
__istr__
if (__cond__)
goto inizio;
```




```
do
__istr__
while (__cond__);
```



Relazioni fra cicli

In conclusione, se `__istr1__` → `__istr2__` indica che `__istr1__` si può riscrivere come `__istr2__`, abbiamo mostrato che

pattern ciclo ↔ `do-while` ↔ `while` ↔ `for`

Quindi le tre istruzioni di ciclo e il pattern ciclo sono equivalenti? Sì, ma a seconda della situazione uno è più opportuno di un altro:

- `for` quando c'è una variabile di controllo che assume una sequenza di valori predefiniti (di solito in un intervallo intero)
- `do-while` quando si deve eseguire l'istruzione almeno una volta;
- `while` negli altri casi.
- Pattern ciclo: mai. E' sempre meglio usare il costrutto più specifico (quindi ad esempio un costrutto apposito piuttosto che un pattern equivalente), perché comunica più chiaramente l'intento del programmatore.

Sommario

1. Istruzioni condizionali (o di selezione)
2. Salti
3. Blocchi
4. Iterazione
5. Programmazione strutturata
6. Selezione multipla

IF

SWITCH

Giorno giuliano

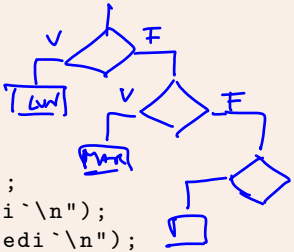
Supponiamo di voler stampare il giorno della settimana corrispondente a un giorno giuliano in lettere:

050_controllo_di_flusso/giorno-settimana-if.c

```c
#include <stdio.h>

main() {
  int jd = 1234567;

  if (jd % 7) == 0) printf("Lunedi`\n");
  else if (jd % 7 == 1) printf("Martedi`\n");
  else if (jd % 7 == 2) printf("Mercoledi`\n");
  else if (jd % 7 == 3) printf("Giovedi`\n");
  else if (jd % 7 == 4) printf("Venerdi`\n");
  else if (jd % 7 == 5) printf("Sabato\n");
  else printf("Domenica\n");
}
```



Selezione su valori della stessa espressione

- Nell'esempio alla slide 111 le condizioni delle istruzioni `if-else` sono in realtà molto simili: confrontano sempre la stessa espressione (`jd % 7`) con tutti i possibili valori.
- Esiste un costrutto apposito (`switch`) per i casi in cui a determinare il ramo scelto è uno dei possibili valori di una determinata espressione.
- In questi casi, quindi, è preferibile usare `switch` in quanto più specifico.

Sintassi

ESPR

switch(⟨*selettore*⟩ **){**

VALORE

⟦ **case** ⟨*espressione*⟩ **:** ⟦ ⟨*istruzione*⟩ ⟧* ⟦**break;** ⟧⟧*

⟦**default:** ⟦ ⟨*istruzione*⟩ ⟧* ⟧

}

⟨*selettore*⟩ è un'espressione.

Semantica

1. Valuta ⟨*selettore*⟩; VALORE
2. Salta tutti i rami **case** le cui ⟨*espressione*⟩ hanno tutte valore diverso da ⟨*selettore*⟩;
3. Esegue (quindi a partire dal primo ramo **case** in cui una ⟨*espressione*⟩ ha lo stesso valore di ⟨*selettore*⟩, o, se non ce ne sono, dal ramo ⟨*default*⟩, se presente) in sequenza tutte le istruzioni fino alla fine del blocco; se però incontra un'istruzione `break`, esce dal blocco.

Giorno giuliano con switch

050_controllo_di_flusso/giorno-settimana-switch.c

```
#include <stdio.h>

main() {
  int jd = 1234567;

  switch (jd % 7) {
    case 0: printf("Lunedi`\n"); break;
    case 1: printf("Martedi`\n"); break;
    case 2: printf("Mercoledi`\n"); break;
    case 3: printf("Giovedi`\n"); break;
    case 4: printf("Venerdi`\n"); break;
    case 5: printf("Sabato\n"); break;
    default: printf("Domenica\n"); break;
  }
}
```

SELETTORE

case 6