Input/output

- Come già detto, input e output non sono parte integrante del C
- L'interazione con l'ambiente è demandato alla libreria standard
  $\Longrightarrow$ un insieme di funzioni a uso dei programmi C
- La libreria `stdio.h` implementa un semplice modello di ingresso e uscita di dati testuali
- un testo è trattato come un successione (stream) di caratteri, ovvero
  $\Longrightarrow$ una sequenza di caratteri organizzata in righe, ciascuna terminata da "`\n`"
- al momento dell'esecuzione, al programma vengono connessi automaticamente 3 stream:
  - standard input: di solito la tastiera
  - standard output: di solito lo schermo
  - standard error: di solito lo schermo

Input/output (cont.)

- Compito della libreria è fare in modo che tutto il trattamento dei dati in ingresso e uscita si conformi a questo modello

  $\Longrightarrow$ il programmatore non si deve preoccupare di come ciò sia effettivamente realizzato
- Ogni volta che si effettua una operazione di lettura attraverso `getchar` viene acquisito il prossimo carattere dallo standard input e viene restituito il suo valore

  (analogamente per `scanf` che comporta l'acquisizione di uno o più caratteri a seconda delle specifiche di formato presenti ...)
- Ogni volta che si effettua una operazione di scrittura (attraverso `putchar` o `printf`) tutti i valori coinvolti vengono convertiti in sequenze di caratteri e queste ultime vengo accodate allo standard output.
- Tipicamente il sistema operativo consente di redirigere gli stream standard, ad esempio su uno o più file.

Funzioni `printf` e `scanf`

- Le funzioni `printf` e `scanf` sono le funzioni di libreria per l'output e per l'input. La lettera `f` alla fine dei nomi delle due funzioni sta per "formatted", ovvero "con formato"
- Sia `printf` che `scanf` ricevono una stringa di controllo, che può contenere le specifiche di conversione indicate con il simbolo % (segnaposto), e una serie di parametri, che possono essere ad esempio le variabili da stampare o leggere.
  ```
  printf(stringa di controllo, lista di variabili)
  scanf(stringa di controllo, lista di variabili)
  ```
- Per stampare delle variabili di un determinato tipo dobbiamo utilizzare i segnaposto relativi (`%d` per `int`,`%c` per `char`,...)
- NOTA: come per i comandi della shell, anche per le funzioni di libreria standard possiamo usare il comando `man`.

Formattazione dell'output con `printf`

- Riepilogo specificatori di formato principali:
  - interi: `%d`, `%o`, `%u`, `%x`, `%X`
    per `short`: si antepone `h`
    per `long`: si antepone `l` (minuscola)
  - reali: `%e`, `%f`, `%g`
    per `double`: non si antepone nulla
    per `long double`: si antepone `L`
  - caratteri: `%c`
  - stringhe: `%s` (le vedremo più avanti)
  - puntatori: `%p` (li vedremo più avanti)
- Flag: messi subito dopo il "`%`"
  - "`-`": allinea a sinistra
  - altri flag (non ci interessano)
- Sequenze di escape: `\%`, `\'`, `\"`, `\\`, `\a`, `\b`, `\n`, `\t`, ...
- Per stampare il carattere `'%'` è necessario raddoppiarlo: `%%`

Formattazione dell'input con `scanf`

- Specificatori di formato: come per l'output, tranne che per i reali
  - `double`: si antepone `l`
  - `long double`: si antepone `L`
- **Soppressione dell'input:** mettendo "`*`" subito dopo "`%`"
  Non ci deve essere un argomento corrispondente allo specificatore di formato: il campo deve essere letto, ma il valore non deve essere assegnato ad alcuna variabile.

  Esempio: Lettura di una data in formato gg/mm/aaaa oppure gg-mm-aaaa.

```
int g, m, a;
scanf("%d%*c%d%*c%d%*c", &g, &m, &a);
```

Espressioni booleane

- Il linguaggio deve consentire di descrivere espressioni booleane (guardie di condizionali e iterazione).
- Come già sappiamo, in C non esiste un tipo Booleano $\Longrightarrow$ si usa il tipo **int** :

  falso $\Longleftrightarrow$ 0

  vero $\Longleftrightarrow$ 1 (in realtà qualsiasi valore diverso da 0)

  Esempio: `2 > 3` ha valore 0 (ossia falso)

  `5 > 3` ha valore 1 (ossia vero)

- **Operatori relazionali del C**
  - `<`, `>`, `<=`, `>=` (minore, maggiore, minore o uguale, maggiore o uguale) — priorità alta
  - `==`, `!=` (uguale, diverso) — priorità bassa

  Esempio: `temperatura <= 0` `velocita > velocita_max`
  `voto == 30` `anno != 2000`

Operatori logici

- In ordine di priorità:
  - `!` (negazione) — priorità alta
  - `&&` (congiunzione)
  - `||` (disgiunzione) — priorità bassa

Semantica:

a	b	!a	a && b	a || b
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

0 ...falso

1 ...vero (o qualsiasi valore $\neq 0$)

Esempio:

`(a >= 10) && (a <= 20)` vero (`1`) se $10 \leq a \leq 20$

`(b <= -5) || (b >= 5)` vero se $|b| \geq 5$

- Le espressioni booleane vengono valutate da sinistra a destra:
  - con `&&`, appena uno degli operandi è falso, restituisce falso senza valutare il secondo operando
  - con `||`, appena uno degli operandi è vero, restituisce vero senza valutare il secondo operando
- **Priorità** tra operatori di diverso tipo:
  - not logico — priorità alta
  - aritmetici
  - relazionali
  - booleani (and e or logico) — priorità bassa

Esempio:

```
a+2 == 3*b || !trovato && c < a/3
```

è equivalente a

```
((a+2) == (3*b)) || ((!trovato) && (c < (a/3)))
```

Selezione doppia: istruzione **if-else**

Sintassi:

```
if     (espressione)
       istruzione1
else   istruzione2
```

- `espressione` è un'espressione booleana
- `istruzione1` rappresenta il ramo then (deve essere un'unica istruzione)
- `istruzione2` rappresenta il ramo else (deve essere un'unica istruzione)

Semantica:

1. viene prima valutata `espressione`
2. se `espressione` è vera viene eseguita `istruzione1`
   altrimenti (ovvero se `espressione` è falsa) viene eseguita `istruzione2`

```c
int temperatura;

printf("Quanti gradi ci sono? "); scanf("%d", &temperatura);
if (temperatura >= 25)
  printf("Fa caldo\n");
else
  printf("Si sta bene\n");

printf("Arrivederci\n");
```

```
=> Quanti gradi ci sono? 30 ↩
Fa caldo
Arrivederci
=>
```

```
=> Quanti gradi ci sono? 18 ↩
Si sta bene
Arrivederci
=>
```

Selezione singola: istruzione **if**

- È un'istruzione **if-else** in cui manca la parte else.

Sintassi:

```
if (espressione)
   istruzione
```

Semantica:

1. viene prima valutata `espressione`
2. se `espressione` è vera viene eseguita `istruzione` altrimenti non si fa alcunché

Esempio:

```
int temperatura;
scanf("%d", &temperatura);
if (temperatura >= 25)
  printf("Fa caldo\n");
printf("Arrivederci\n");
```

```
=> 18 ↩
Arrivederci
```

```
=> 30 ↩
Fa caldo
Arrivederci
```

Blocco

- La sintassi di **if-else** consente di avere un'unica istruzione nel ramo then (o nel ramo else).
- Se in un ramo vogliamo eseguire più istruzioni dobbiamo usare un blocco.

  Sintassi:

```
{
    istruzione-1
    …
    istruzione-n
}
```

- Come già sappiamo e come rivedremo più avanti, un blocco può contenere anche dichiarazioni.

Esempio: Dati mese ed anno, calcolare mese ed anno del mese successivo.

```
int mese, anno, mesesucc, annosucc;

if (mese == 12)
{
  mesesucc = 1;
  annosucc = anno + 1;
}
else
{
  mesesucc = mese + 1;
  annosucc = anno;
}
```

Esempio: Dato un anno, controllare se si tratta di un anno bisestile. Ricordiamo che un anno è *bisestile* se lo si può dividere per 4, ma non per 100, ad eccezione degli anni divisibili per 400, che sono bisestili.

```
int anno;

if ((anno%4 == 0 && anno%100 ! = 0) || (anno%400 == 0))
printf(%d e' un anno bisestile \n, anno);
else
printf(%d non e' un anno bisestile \n, anno);
```

Selezione multipla: If annidati (in cascata)

- Quando l'istruzione del ramo then o else è un'istruzione **if o if-else.**

  Esempio: Data una temperatura, stampare un messaggio secondo la seguente tabella:

temperatura t	messaggio
$30 < t$	Molto caldo
$20 < t \leq 30$	Caldo
$10 < t \leq 20$	Gradevole
$0 < t \leq 10$	Freddo
$t \leq 0$	Molto freddo

```
 if (temperatura > 30)
  printf("Molto caldo\n");
else if (temperatura > 20)
  printf("Caldo\n");
else if (temperatura > 10)
  printf("Gradevole\n");
else if (temperatura > 0)
  printf("Freddo\n");
else printf("Molto freddo\n");
```

Osservazioni:

- si tratta di un'unica istruzione **if-else**

```
if (temperatura > 30)
  printf("Molto caldo\n");
else ...
```

- non serve che la seconda condizione sia composta

```
if (temperatura > 30) printf("Molto caldo\n");
else /* il valore di temperatura e' <= 30 */
    if (temperatura > 20)
```

  Non c'è bisogno di una congiunzione del tipo

```
(t <= 30) && (t > 20)
```

  (analogamente per gli altri casi).

- **Attenzione**: il seguente codice

```
if (temperatura > 30) printf("Molto caldo\n");
if (temperatura > 20) printf("Caldo\n");
```

  ha ben altro significato (quale?)

Ambiguità dell'else

```
if (a >= 0) if (b >= 0) printf("b positivo");
else printf("???");
```

- `printf("???")` può essere la parte **else**
  - del primo **if** $\Longrightarrow$ `printf("a negativo");`
  - del secondo **if** $\Longrightarrow$ `printf("b negativo");`
- L'ambiguità sintattica si risolve considerando che un **else** fa sempre riferimento all'**if** più vicino, dunque

```
if (a > 0)
  if (b > 0)
    printf("b positivo");
  else

    printf("b negativo");
```

- Perché un **else** si riferisca ad un **if** precedente, bisogna inserire quest'ultimo in un blocco

```
 if (a > 0)
  { if (b > 0) printf("b positivo"); }
else

  printf("a negativo");
```

Esercizio

Leggere un reale e stampare un messaggio secondo la seguente tabella:

gradi alcolici g	messaggio
$40 < g$	superalcolico
$20 < g \leq 40$	alcolico
$15 < g \leq 20$	vino liquoroso
$12 < g \leq 15$	vino forte
$10.5 < g \leq 12$	vino normale
$g \leq 10.5$	vino leggero

Esempio: Dati tre valori che rappresentano le lunghezze dei lati di un triangolo, stabilire se si tratti di un triangolo equilatero, isoscele o scaleno.

Algoritmo: `determina tipo di triangolo`

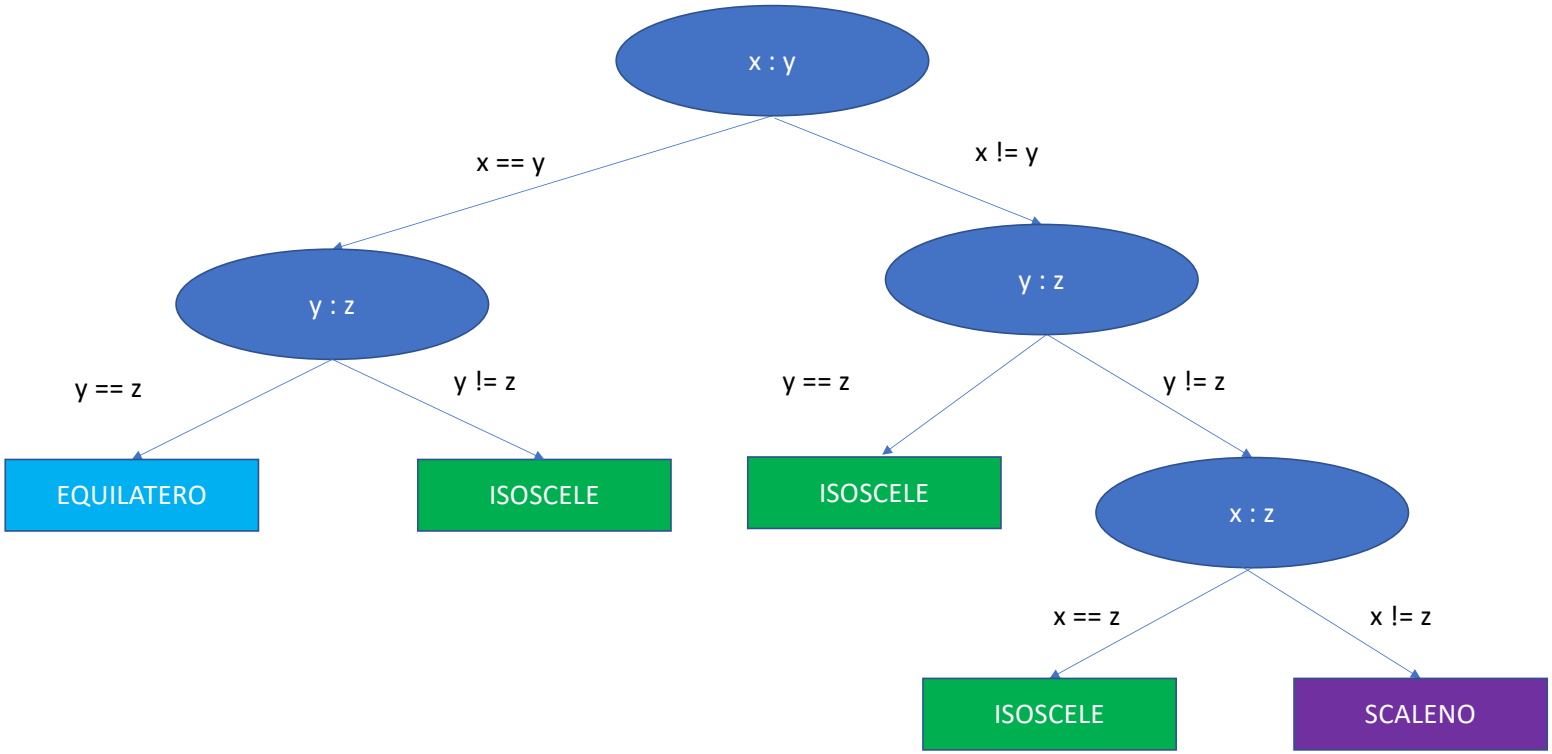
```
leggi i tre lati
confronta i lati a coppie, fin quando non
  hai raccolto una quantità di informazioni
  sufficiente a prendere la decisione
stampa il risultato
```

```
main()   {
float primo, secondo, terzo;

printf("Lunghezze lati triangolo ? ");
scanf("%f%f%f", &primo, &secondo, &terzo);

if (primo == secondo) {
  if (secondo == terzo)
    printf("Equilatero\n");
  else
    printf("Isoscele\n");
}
else {
  if (secondo == terzo)
    printf("Isoscele\n");
  else if (primo == terzo)
    printf("Isoscele\n");
  else
    printf("Scaleno\n");
}     }
```

Relativo albero delle decisioni



Per stabilire la natura del triangolo si fanno al più 3 confronti.

Esercizio

Risolvere il problema del triangolo utilizzando il seguente algoritmo:

```
Algoritmo: determina tipo di triangolo con conteggio
leggi i tre lati
confronta i lati a coppie contando
  quante coppie sono uguali
if le coppie uguali sono 0
  è scaleno
  else if le coppie uguali sono 1
    è isoscele
    else è equilatero
```

Selezione a più vie: istruzione **switch**

- **Sintassi:**

```
switch (espressione) {
  case valore-1: istruzioni-1
                 break;
  ...
  case valore-n: istruzioni-n
                 break;
  default : istruzioni-default
}
```

Semantica:

1. viene valutata `espressione`
2. viene cercato il primo `i` per cui il valore di `espressione` è uguale a `valore-i`
3. se si trova tale `i`, allora vengono eseguite `istruzioni-i` altrimenti vengono eseguite `istruzioni-default`

Equivale a `if (espressione == valore-1) istruzioni-1 else if (espressione == valore-2) istruzione-2 else... istruzioni-default`

Esempio:

```
int giorno;
...
switch (giorno) {
  case 1: printf("Lunedi'\n");
          break;
  case 2: printf("Martedi'\n");
          break;
  case 3: printf("Mercoledi'\n");
          break;
  case 4: printf("Giovedi'\n");
          break;
  case 5: printf("Venerdi'\n");
          break;
  default : printf("Week end\n");
}
```

- Se abbiamo più valori a cui corrispondono le stesse istruzioni, possiamo raggrupparli come segue:

```
case valore-1: case valore-2:
                    istruzioni
                    break;
```

Esempio:

```
int giorno;
...
switch (giorno) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: printf("Giorno lavorativo\n");
          break;
  case 6:
  case 7: printf("Week end\n");
  default : printf("Giorno non valido\n");
}
```

Osservazioni sull'istruzione **switch**

- L'`espressione` usata per la selezione può essere una qualsiasi espressione C che restituisce un valore intero.
- I valori specificati nei vari **case** devono invece essere costanti (o meglio valori noti a tempo di compilazione). In particolare, non possono essere espressioni in cui compaiono variabili.

  Esempio: Il seguente frammento di codice è sbagliato:

```
int a;
switch (a) {
  case a<0: printf("negativo\n");
            /* ERRORE: a<0 non è una costante*/
  case 0: printf("nullo\n");
  case a>0: printf("positivo\n");
            /* ERRORE: a>0 non è una costante*/
}
```

- In realtà il C non richiede che nei **case** di un'istruzione **switch** l'ultima istruzione sia **break**.

  Quindi, in generale la **sintassi** di un'istruzione **switch** è:

```
switch (espressione) {
  case valore-1: istruzioni-1
  ...
  case valore-n: istruzioni-n
  default : istruzioni-default
}
```

Semantica:

1. viene prima valutata `espressione`
2. viene cercato il primo `i` per cui il valore di `espressione` è pari a `valore-i`
3. se si trova tale `i`, allora si eseguono in sequenza `istruzioni-i`, `istruzioni-(i+1)`, ..., fino a quando non si incontra **break** o è terminata l'istruzione **switch**,
   altrimenti vengono eseguite `istruzioni-default`

Esempio: più **case** di uno **switch** eseguiti in sequenza (corretto)

```
int lati;
printf("Immetti il massimo numero di lati del poligono (al piu` 6): ");
scanf("%d", &lati);
printf("Poligoni con al piu` %d lati: ", lati);

switch (lati) {
  case 6: printf("esagono, ");
  case 5: printf("pentagono, ");
  case 4: printf("rettangolo, ");
  case 3: printf("triangolo\n");
          break;
  case 2: case 1: printf("nessuno\n");
                  break;
  default : printf("\nErrore: valore immesso > 6.\n");
}
```

- N.B. Quando si omettono i **break**, diventa rilevante l'ordine in cui vengono scritti i vari **case**. Questo può essere facile causa di errori.

  È buona norma mettere **break** come ultima istruzione di ogni **case**

Esempio: più **case** eseguiti in sequenza (scorretto)

```
int b;
printf("Immetti un numero tra 1 e 6: ");
scanf("%", &b);
switch (b) {
  case 1: case 2: case 3: case 5: printf("Numero primo\n");
  case 4: case 6:                  printf("Numero non primo\n");
  default :                         printf("Valore non valido!\n");
}
```

=> 3↩
Numero primo
Numero non primo
Valore non valido!
=>

=> 4 ↩
Numero non primo
Valore non valido!
=>

Istruzioni iterative

Esempio: Leggere 5 interi, calcolarne la somma e stamparli.

- Variante non accettabile: 5 variabili, 5 istruzioni di lettura, 5 ...

```
int i1, i2, i3, i4, i5;
scanf("%d", &i1):
...
scanf("%d", &i5);
printf("%d", i1 + i2 + i3 + i4 + i5);
```

- Variante migliore che utilizza solo 2 variabili:

```
int somma, i;
somma = 0;
scanf("%d", &i);
somma = somma + i;
...          /* per 5 volte */
scanf("%d", &i);
somma = somma + i;
printf("%d", somma);
```

$\Longrightarrow$ conviene però usare un'istruzione iterativa

Iterazione determinata e indeterminata

- Le istruzioni iterative permettono di ripetere determinate azioni più volte:
  - un numero di volte fissato $\Longrightarrow$ iterazione determinata
    Esempio:
    ```
    fai un giro del parco di corsa per 10 volte
    ```
  - finchè una condizione rimane vera $\Longrightarrow$ iterazione indeterminata
    Esempio:
    ```
    finché non sei sazio
    prendi una ciliegia dal piatto e mangiala
    ```

Istruzione **while**

Permette di realizzare l'iterazione in C.

Sintassi:

```
while (espressione)
    istruzione
```

- `espressione` è la guardia del ciclo
- `istruzione` è il corpo del ciclo (può essere un blocco)

Semantica:

1. viene valutata l'`espressione` (condizione di continuazione)
2. se è vera si esegue `istruzione` e si torna ad eseguire l'intero `while`
3. se è falsa si termina l'esecuzione del `while`

- Nota: se `espressione` è falsa all'inizio, il ciclo non fa nulla.

Iterazione determinata

Esempio: Stampa 100 asterischi.

- Si utilizza un contatore per contare il numero di asterischi stampati.

```
Algoritmo: stampa di 100 asterischi
inizializza il contatore a 0
while il contatore è minore di 100
{ stampa un ``*''
    incrementa il contatore di 1 }
```

- Implementazione:

```
int i;
i = 0;
while (i < 100) {
  putchar('*');
  i = i + 1;
}
```

- come già sappiamo, la variabile `i` viene detta variabile di controllo del ciclo.

Iterazione determinata

Esempio: Leggere 10 interi, calcolarne la somma e stamparla.

- Si utilizza un contatore per contare il numero di interi letti.

```
int conta, dato, somma;
printf("Immetti 10 interi: ");
somma = 0;
conta = 0;
while (conta < 10) {
  scanf("%d", &dato);
  somma = somma + dato;
  conta = conta + 1;
}
printf("La somma è %d\n", somma);
```

Esempio: Leggere un intero N seguito da N interi e calcolare la somma di questi ultimi.

- Simile al precedente: il numero di ripetizioni necessarie non è noto al momento della scrittura del programma ma lo è al momento dell'esecuzione del ciclo.

```
int lung, conta, dato, somma;
printf("Immetti la lunghezza della sequenza ");
printf("seguita dagli elementi della stessa: ");
scanf("%d", &lung);
somma = 0;
conta = 0;
while (conta < lung) {
  scanf("%d", &dato);
  somma = somma + dato;
  conta = conta + 1;
}
printf("La somma è%d\n", somma);
```

Esempio: Leggere 10 interi positivi e stamparne il massimo.

- Si utilizza un massimo corrente con il quale si confronta ciascun numero letto.

```
int conta, dato, massimo;
printf("Immetti 10 interi: ");
massimo = 0;
conta = 0;
while (conta < 10) {
  scanf("%d", &dato);
  if (dato > massimo)
    massimo = dato;
  conta = conta + 1;
}
printf("Il massimo è %d\n", massimo);
```

Esercizio

Leggere 10 interi arbitrari e stamparne il massimo.

Istruzione **for**

- I cicli visti fino ad ora hanno queste caratteristiche comuni:
  - utilizzano una variabile di controllo
  - la guardia verifica se la variabile di controllo ha raggiunto un limite prefissato
  - ad ogni iterazione si esegue un'azione
  - al termine di ogni iterazione viene incrementato (decrementato) il valore della variabile di controllo

  Esempio: Stampare i numeri pari da 0 a N.

```
i = 0; /* Inizializzazione della var. di controllo */
while (i <= N) { /* guardia */
  printf("%d ", i); /* Azione da ripetere */
  i=i+2; /* Incremento var. di controllo */
}
```

- L'istruzione **for** permette di gestire direttamente questi aspetti:

```
for (i = 0; i <= N; i=i+2)
  printf("%d", i);
```

Sintassi:

```
for (istr-1; espr-2; istr-3)
  istruzione
```

- `istr-1` serve a inizializzare la variabile di controllo
- `espr-2` è la verifica di fine ciclo
- `istr-3` serve ad aggiornare la variabile di controllo alla fine del corpo del ciclo
- `istruzione` è il corpo del ciclo

Semantica: l'istruzione **for** precedente è equivalente a

```
istr-1;
while (espr-2) {
  istruzione
  istr-3
}
```

Esempio:

```
for (i = 1; i <= 10; i=i+1)      ⟹   i: 1, 2, 3, ..., 10
for (i = 10; i >= 1; i=i-1)      ⟹   i: 10, 9, 8, ..., 2, 1
for (i = -4; i <= 4; i = i+2)    ⟹   i: -4, -2, 0, 2, 4
for (i = 0; i >= -10; i = i-3)   ⟹   i: 0, -3, -6, -9
```

- In realtà, la sintassi del **for** è

  for `(espr-1; espr-2; espr-3)`
  `istruzione`

  dove `espr-1`, `espr-2` e `espr-3` sono delle espressioni qualsiasi (in C anche l'assegnamento è un'espressione ...).
- È buona prassi:
  - usare ciascuna `espr-i` in base al significato descritto prima
  - non modificare la variabile di controllo nel corpo del ciclo
- Ciascuna delle tre `espr-i` può anche mancare:
  - i "`;`" vanno messi lo stesso
  - se manca `espr-2` viene assunto il valore vero
- Se manca una delle tre `espr-i` è meglio usare un'istruzione **while**

Esempio: Leggere 10 interi positivi e stamparne il massimo.

```
int conta, dato, massimo;
printf("Immetti 10 interi: ");
massimo = 0;
for (conta=0; conta<10; conta=conta+1)
{
  scanf("%d", &dato);
  if (dato > massimo)
    massimo = dato;
}
printf("Il massimo è %d\n", massimo);
```

Cicli annidati

- Il corpo di un ciclo può contenere a sua volta un ciclo.

  Esempio: Stampa della Tavola Pitagorica

  Algoritmo

```
for ogni riga tra 1 e 10
  { for ogni colonna tra 1 e 10
    stampa riga * colonna
  stampa un a capo }
```

- Traduzione in C

```
int riga, colonna;
const int Nmax = 10; /* indica il numero di righe e di colonne
*/
for (riga = 1; riga <= Nmax; riga=riga+1) {
  for (colonna = 1; colonna <= Nmax; colonna=colonna+1)
    printf("%d ", riga * colonna);
  putchar('\n'); }
```