Ciclo for

Il ciclo *for* corrisponde all'istruzione <for_ist>.

Sintassi:

<for_ist> ::= for (<espr_1>;<espr_2>;<espr_3>)<istruzione>

Semantica:

(1). Viene valutata <espr_1> (inizializzazione del ciclo).

(2). Viene valutata <espr_2> (controllo uscita dal ciclo).
Se Valore(<espr>_2) $\neq 0$:
- Viene eseguita <istruzione>.
- Viene valuatata <espr_3> (tipicamente un incremento).
- Il controllo torna al punto (2).

Se Valore(<espr>_2) $= 0$:
- Il controllo passa all'istruzione che segue <for_ist>.

Il ciclo

```
for(<espr_1>; <espr_2>; <espr_3>)
   <istruzione>
```

equivale al ciclo

```
<espr_1>;
while (<espr_2>) {
      <istruzione>
      <espr_3>;
 }
```


Esempio 1

```
for(i=1; i<= 10; i++)
    printf("%d ",i);
```

stampa

```
1 2 3 4 5 6 7 8 9 10
```

Esempio 2

```
for( printf("E1 "); printf("E2 "); printf("E3 ") )
     printf("Istr ");
```

Poiché `printf(E2 )` restituisce 3 (numero dei caratteri stampati), il ciclo non termina e viene stampato

```
E1 E2 Istr E3 E2 Istr E3 E2 Istr E3 ...
```

Esempio 3

```
for(c=getchar(); isspace(c); c=getchar());  // salta gli spazi
```

Il ciclo legge continuamente dei caratteri da standard input, fino al prossimo carattere diverso da un carattere di spaziatura (si veda definizione di `isspace()` della libreria `ctype.h`). Al termine del ciclo c è il primo carattere letto da standard input diverso da uno spazio.

- Alcune o tutte le espressioni di <for_ist> possono essere omesse (i ';' devono però essere presenti).
- Se manca <espr_2> si assume che tale condizione sia diversa da 0.

Esempi

- Dopo il ciclo

```
i=1; sum=1;
for(; i<= 10;)
    sum += i++;   //  sum = sum + (i++);
```

la variabile `sum` vale 55 ($= 1 + 2 + 3 + \cdots + 10$) e `i` vale 11.

-
```
for(i=1; ;i++)
     printf("%d ",i);
```
è un ciclo infinito che stampa

```
1 2 3 4 5 ...
```

L'operatore virgola

- Nelle espressioni <espr_1> e <espr_3> di un ciclo `for` può essere utile usare più di una espressione.
  L'operatore '`,`' (*virgola*) permette di giustapporre due espressioni creando una nuova espressione.
- Dal punto di vista sintattico, la virgola è un operatore binario, avente la priorità più bassa nella tavola degli operatori e associatività $\Rightarrow$.

La valutazione dell'espressione

<espr_1> , <espr_2>

avviene come segue:

- Viene valuata *prima* <espr_1> e *poi* <espr_2>.
- Il valore e il tipo dell'intera espressione coincide con il valore e il tipo di <espr_2>.

All'uscita dal ciclo

```
for(n=1, fatt=1, exp=1 ; n<=5 ; fatt *= n, exp *= 2, n++);
```

(dove il '`;`' al termine della linea indica che l'istruzione da eseguire all'interno del ciclo è l'istruzione nulla) il valore di `fatt` è 120 (=5!), il valore di `exp` è 32 ($=2^5$), il valore di `n` è 6.

Nota

Non tutte le virgole hanno la funzione di "operatore virgola"!
Ad esempio, le virgole usate nelle seguenti linee di codice

```
int f(int x, int y){
    ...
}

int main(){
    int a,b;
    f(a, b);
     ...
 }
```

non sono operatori, ma svolgono il ruolo di separatori.

Attenzione alle virgole!

```
double d;
d = 2,5;    // (A)
```

ha come effetto quello di assegnare a `d` il valore 2.
Infatti, l'espressione `d = 2,5` equivale all'espressione `(d = 2), 5`.
Quindi, eseguire l'istruzione (A) significa valutare l'espressione `d=2` e poi l'espressione `5` (che non produce effetto).

Supponiamo che la variabile

```
int   **matr;
```

punti a una matrice allocata dinamicamente. Allora

```
matr[2, 3] = 0;
```

equivalente a

```
matr[3] = 0;
```

pone a 0 (ossia, a `NULL`) l'elemento `matr[3]`, perdendo in questo modo l'indirizzo del vettore che rappresenta la riga 3 della matrice.
Per assegnare 0 all'elemento della matrice in riga 2 e colonna 3 occorre invece fare

```
matr[2][3] = 0;
```


L'istruzione `break`

L'istruzione `break` all'interno di un ciclo causa l'uscita dal ciclo più interno.

```
printf("durante il ciclo i assume i valori: \n");
for(i = 1 ; i <= 10 ; i++){
     if (i == 5)
        break;
     printf("%d ", i);
}
printf("\ndopo il ciclo i vale: %d\n",i);
```

Risultato dell'esecuzione:

```
durante il ciclo i assume i valori:
1 2 3 4
dopo il ciclo i vale: 5
```

Si usa anche all'interno dell'istruzione `switch`.

L'istruzione continue

L'istruzione `continue` causa l'interruzione dell'iterazione corrente del ciclo in cui è inserita e l'inizio dell'iterazione successiva.

```
printf("durante il ciclo i assume i valori: \n");
for(i = 1 ; i <= 10 ; i++){
    if (i == 5)
       continue;
    printf("%d ", i);
}
printf("\ndopo il ciclo i vale: %d\n",i);
```

Risultato esecuzione:

```
durante il ciclo i assume i valori:
1 2 3 4 6 7 8 9 10
dopo il ciclo i vale: 11
```


L'istruzione switch

È un'istruzione di *scelta plurima* che controlla se un'espressione assume un valore all'interno di un certo insieme di costanti intere ed esegue le istruzioni corrispondenti.

Esempio
Scrivere un programma che legge una sequenza di caratteri da standard input terminante con "end-of-file" e stampa il numero di cifre, di caratteri di spaziatura (spazi, a capo, tabulazioni) e il numero totale di caratteri letti.

Usiamo tre variabili contatore:

```
int   cifre = 0;  // conta il numero di cifre
int   spazi = 0;  // conta il numero di caratteri di spaziatura
int   altri = 0;  // conta tutti gli altri caratteri
```


Il ciclo di lettura ha la seguente struttura:

```
...
int c;   //  ultimo carattere letto
...
while ((c = getchar()) != EOF){
    switch(c){
    case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
        cifre++; // incrementa contatore cifre
        break;
    case ' ': case '\n': case '\t':
        spazi++; // incrementa contatore spazi
        break;
    default :
        altri++; // incrementa contatore altri caratteri
        break;
    } // end switch
} // end while
```

Per stampare i risultati:

```
printf("Cifre: %d, Spazi: %d, Totale: %d", cifre, spazi, cifre+spazi+altri);
```



```
switch(Espr){
  ...
}
```

viene eseguita nel seguente modo.

- Viene valutata l'espressione `Espr`.

  Sia `c` il valore di `Espr`; il controllo salta a:

  - Istruzione con etichetta `case c:`

    Oppure

  - Istruzione con etichetta `default:` se nessuna istruzione ha etichetta `case c:`

    Oppure

  - Istruzione successiva a `switch` se non esistono istruzioni aventi etichetta `case c:` oppure non esiste alcuna istruzione con etichetta `default:`.

- L'esecuzione del comando `break` comporta la terminazione di `switch` e l'esecuzione dell'istruzione successiva.

Uso di break

L' istruzioni `break` non è obbligatorio.
Se si omette `break` come conclusione di un "caso", vengono eseguite le successive istruzioni, qualunque sia l'etichetta `case` ad esse associate, fino al prossimo `break`.

Esempio

```
int n;
...
switch(n){
case 0: printf("0 ");
case 1: printf("1 ");
case 2: printf("2 ");
case 3: printf("3 ");
        break;
case 4: printf("4 ");
}
```

Se `n` vale 1, viene stampato

```
0 1 2 3
```

Se invece `n` vale 2, viene stampato

```
2 3
```


La direttiva #if

L'istruzione

```
#if  X
  ....
#endif
```

dove `X` deve essere definita da

```
#define X ..
```

è trattata dal preprocessore

- Se `X` ha valore 0, il preprocessore elimina le linee di codice comprese fra `#if` e `#endif`
- Altrimenti, le linee sono mantenute.

Vedere i dettagli sui manuali.

Esempio

```
#include <stdio.h>

#define X 0

int main(){
  printf("riga 1\n");
#if X
  printf("riga 2\n");
  printf("riga 3\n");
#endif
  printf("riga 4\n");
  return 0;
}
```

Compilando ed eseguendo il programma viene stampato

```
riga 1
riga 4
```


Ponendo invece

```
#define X 1
```

viene stampato

```
riga 1
riga 2
riga 3
riga 4
```

Notare che l'eliminazione del codice è fatta dal preprocessore. Per verificarlo, compilare il programma con opzione `-E` (codice prodotto dal preprocessore)

È utile per:

- Commentare codice (in C non è possibile commentare codice che contiene già dei commenti!).
- Aggiungere al codice di operazioni di stampa che possono essere facilmente eliminate

```
#define DEBUG 1  //  1: stampa i messaggi di debug

void  f(int x, int y){
  .....
#if DEBUG
  printf("x= %d  y= %d\n",x,y); // viene stampato solo se
                                // DEBUG e' diverso da 0
#endif
  ...
}
```