

3.1 Istruzione for

Quando si desidera ripetere una operazione un determinato numero di volte, si può riscrivere sequenzialmente l'istruzione corrispondente. Per esempio, se si vuole sommare tre volte alla variabile `somma`, inizializzata a 0, il valore 7, si può scrivere:

```
somma = 0;
somma = somma+7;
somma = somma+7;
somma = somma+7;
```

Risulta però decisamente più comodo inserire in un ciclo l'istruzione che si ripete:

```
somma = 0;
for(i=1; i<=3; i=i+1)
    somma = somma+7;
```

L'esempio precedente è volutamente semplice per concentrare l'attenzione sulle caratteristiche del costrutto `for`. Alla variabile `somma` viene dato il valore 0. L'istruzione `for` assegna il valore 1 alla variabile `i`. L'operazione

`i=1`

compresa tra la parentesi tonda aperta e il primo punto e virgola è detta *inizializzazione* e non verrà mai più eseguita. Successivamente l'esecuzione prosegue così:

1. se `i<=3` allora vai al passo 2 altrimenti termina
2. `somma=somma+7`
3. `i=i+1`, vai al passo 1

Inizialmente la condizione risulta vera, in quanto 1 è minore o uguale a 3, e quindi viene eseguito il corpo del ciclo, che in questo caso è composto dalla sola istruzione `somma=somma+7`. La variabile `somma` assume il valore 7. Viene incrementato di 1 il valore di `i`, che quindi assume il valore 2.

Alla fine del terzo ciclo la variabile `somma` ha il valore 21 e `i` vale 4. Nuovamente viene verificato se `i` è minore o uguale a 3. La condizione risulta falsa e l'iterazione ha termine; il controllo passa all'istruzione successiva del programma.

Il formato del costrutto `for` è il seguente:

```
for(esp1; esp2; esp3)
    istruzione
```

Si faccia attenzione ai punti e virgola all'interno delle parentesi. Il ciclo inizia con l'esecuzione di `esp1`, la quale non verrà mai più eseguita. Quindi viene esaminata `esp2`. Se `esp2` risulta vera, viene eseguita `istruzione`, altrimenti il ciclo non viene percorso neppure una volta.

Successivamente viene eseguita `esp3` e di nuovo valutata `esp2` che se risulta essere vera dà luogo a una nuova esecuzione di `istruzione`. Il processo si ripete finché `esp3` non risulta essere falsa. Nell'esempio precedente,

```
for(i=1; i<=3; i=i+1)
    somma=somma+7;
```

`esp1` era `i=1`, `esp2` `i<=3`, `esp3` `i=i+1` e `istruzione` era `somma=somma+7`.

Nella sintassi del `for`, `istruzione`, così come nel costrutto `if`, può essere un blocco, nel qual caso deve iniziare con una parentesi graffa aperta e terminare con parentesi graffa chiusa. Supponiamo di voler ottenere la somma di tre numeri interi immessi dall'utente: si può scrivere:

```
somma = 0;
scanf("%d", &numero);
somma = somma+numero;
scanf("%d", &numero);
somma = somma+numero;
scanf("%d", &numero);
somma = somma+numero;
```

La variabile `somma` che conterrà, di volta in volta, la somma degli interi letti, viene inizializzata a zero. Successivamente, per tre volte è richiesta l'immissione di un valore che viene immagazzinato nella variabile `numero`. A ogni lettura corrisponde un incremento di `somma` del valore di `numero`. Lo stesso risultato lo si ottiene in una forma più sintetica con il seguente codice:

```
somma = 0;
for(i=1; i<=3; i =i+1) {
    scanf("%d", &numero);
    somma = somma+numero;
}
```

Nel Listato 3.1 osserviamo un programma che calcola la somma di cinque numeri interi immessi dall'utente.

```
/* Esempio di utilizzo dell'istruzione for
   Calcola la somma di cinque numeri interi
   immessi dall'utente */

#include <stdio.h>

int i, somma, numero;

main()
{
printf("SOMMA 5 NUMERI\n");
somma = 0;

for(i=1; i<=5; i=i+1) {
    printf("Inser. intero: ");
    scanf("%d", &numero);
    somma = somma + numero;
}

printf("Somma: %d\n", somma);
}
```

Listato 3.1 Iterazione con l'istruzione for

Durante l'esecuzione del programma l'utente sarà chiamato a introdurre cinque valori interi:

```
SOMMA 5 NUMERI
Inser. intero: 32
Inser. intero: 111
Inser. intero: 2
Inser. intero: 77
Inser. intero: 13
Somma: 235
```

In questo caso l'utente ha inserito 32, 111, 2, 77 e 13. Naturalmente il numero dei valori richiesti può variare: se si vogliono accettare 100 valori si deve modificare soltanto la condizione di fine ciclo *esp2* nel for:

```
for(i=1; i<=100; i=i+1)
```

Potrebbe risultare utile far apparire il numero d'ordine d'inserimento; a tale scopo si deve modificare la prima printf:

```
printf("\nInser. intero n.%d: ", i);
```

che genererà

```
Inser. intero n.1: 32
Inser. intero n.2: 111
Inser. intero n.3: 2
...
```

Nel costrutto for

```
for(esp1; esp2; esp3)
    istruzione
```

esp1, come *esp2* ed *esp3*, può essere una qualsiasi espressione ammessa in C ■.

Per ora limitiamoci a vederne alcune applicazioni classiche:

```
for(i=5; i>=1; i=i-1)
```

Il ciclo viene ripetuto cinque volte ma la variabile che controlla il ciclo viene inizializzata al valore massimo (5) e decrementata di uno a ogni passaggio; l'ultima iterazione avviene quando il valore assunto è 1. Se si desidera far assumere alla variabile che controlla un ciclo, ripetuto quattro volte, i valori 15, 25, 35 e 45 si potrà scrivere

```
for(i=15 ; i<=45; i=i+10)
```

Analogamente, se i valori devono essere 7, 4, 1, -2, -5, -8 si avrà:

```
for(i=7; i>=-8; i=i-3)
```

Quando si predispose la ripetizione ciclica di istruzioni si deve fare molta attenzione a che l'iterazione non sia infinita, come nell'esempio seguente:

```
for(i=5; i>=5; i=i+1)
```

Il valore di *i* viene inizializzato a 5; è dunque verificata la condizione $i \geq 5$. Successivamente *i* viene incrementato di una unità e assume di volta in volta i valori 6, 7, 8 ecc. che risulteranno essere sempre maggiori di 5: il ciclo è infinito. Il compilatore non segnalerà nessun errore ma l'esecuzione del programma probabilmente non farà ciò che si desidera.

Ognuna delle *esp1*, *esp2* ed *esp3* può essere l'espressione nulla, nel qual caso comunque si deve riportare il punto e virgola corrispondente. Vedremo nei prossimi paragrafi alcuni esempi significativi. Anche l'istruzione del *for* può essere nulla, corrispondere cioè a un punto e virgola, come nell'esempio:

```
for(i=1; i<1000; i=i+100)
;
```

che incrementa il valore *i* di 100 finché *i* risulta minore di 1000. Si osservi che al termine dell'esecuzione dell'istruzione *i* avrà valore 1001: è chiaro perché?

3.2 Incrementi e decrementi

L'incremento unitario del valore di una variabile è una delle operazioni più frequenti. Si ottiene con l'istruzione

```
somma = somma+1;
```

In C è possibile ottenere lo stesso effetto mediante l'operatore ++, costituito da due segni di addizione non separati da nessuno spazio. L'istruzione

```
++somma;
```

incrementa di uno il valore di *somma*, esattamente come faceva l'istruzione precedente. Lo stesso ragionamento vale per l'operazione di decremento, per cui

```
somma = somma-1;
```

è equivalente a

```
--somma;
```

L'operatore --, costituito da due segni di sottrazione, decrementa la variabile di una unità. Dunque anche l'istruzione *for*

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

può essere trasformata in

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

Osserviamo come viene modificato il ciclo del programma esaminato precedentemente grazie all'utilizzo dell'operatore ++:

```
for(i=1; i<=5; ++i) {
    printf("\nInser. intero: ");
    scanf("%d", &numero);
    somma = somma+numero;
}
```

Si noti come il codice C si faccia via via più compatto.

Gli operatori ++ e -- possono precedere una variabile in un'espressione:

```
int a, b, c;
a = 5;
b = 7;
c = ++a + b;
printf("%d \n", a);
printf("%d \n", b);
printf("%d \n", c);
```

Nell'espressione ++a+b, la variabile *a* viene incrementata di una unità (++a) e sommata alla variabile *b*. Successivamente il risultato viene assegnato a *c*. Le tre istruzioni `printf` visualizzeranno rispettivamente 6, 7 e 13.

Gli operatori di incremento e decremento possono sia precedere sia seguire una variabile:

```
++somma;
somma++;
```

Le due istruzioni precedenti hanno lo stesso effetto, ma se gli operatori vengono utilizzati all'interno di espressioni che coinvolgono più elementi valgono le seguenti regole:

- se l'operatore ++ (--) precede la variabile, prima il valore della variabile viene incrementato (decrementato) e poi viene valutata l'intera espressione;
- se l'operatore ++ (--) segue la variabile, prima viene valutata l'intera espressione e poi il valore della variabile viene incrementato (decrementato).

Per esempio:

```
int a, b, c;
a = 5;
b = 7;
c = a++ + b;
printf("%d \n", a);
printf("%d \n", b);
printf("%d \n", c);
```

non produce la stessa visualizzazione della sequenza precedente. La variabile *a* viene sommata a *b* e il risultato viene assegnato a *c*, successivamente *a* viene incrementata di una unità. Le istruzioni `printf` visualizzeranno rispettivamente 6, 7 e 12. Si osservi l'identità dei due cicli `for`:

```
for(i=1; i<=3; ++i)          for(i=1; i<=3; i++)
```

poiché `esp3` è da considerarsi un'istruzione a sé stante. Viceversa

```
for(i=1; ++i<=3;)          for(i=1; i++<=3;)
```

sono diversi in quanto nel caso di sinistra *i* viene incrementata prima della valutazione di `espr2`, per cui nel primo ciclo *i* acquista valore 2, nel secondo 3 e il terzo ciclo non verrà mai eseguito dato che *i* ha già valore 4. Nel caso di destra *i* assume valore 4 solamente dopo il confronto operato nel terzo ciclo, che quindi verrà portato a termine; per verificarlo si provino le successive due sequenze.

```
j=0;
for(i=1; ++i<=3;)
    printf("Ciclo: %d\n", ++j);
printf("Cicli:%d i:%d\n", j, i);
```

```

j=0;
for(i=1; i++<=3;)
    printf("Ciclo: %d\n", ++j);
printf("Cicli:%d i:%d\n", j, i);

```

Le visualizzazioni prodotte saranno rispettivamente

```

Ciclo:1          Ciclo:1
Ciclo:2          Ciclo:2
Cicli:2 i:4     Ciclo:3
                  Cicli:3 i:5

```

È chiaro perché *i* ha valore 4 nel caso di sinistra e 5 in quello di destra?

Non è generalmente permesso in C, ed è comunque sconsigliato per gli effetti che ne possono derivare, utilizzare in un'espressione la stessa variabile contemporaneamente incrementata e non incrementata come in `a=b+(++b)`.

Nel caso che una variabile debba essere incrementata o decrementata di un valore diverso da uno, oltre che con il metodo classico

```
somma = somma+9;
```

si può usufruire dell'operatore `+=`:

```
somma += 9;
```

che nell'esempio incrementa di nove unità il valore di *somma*. La forma generalizzata è

```
variabile [operatore]= espressione
```

Dove [*operatore*] può essere `+` `-` `*` `/` `%` ed *espressione* una qualsiasi espressione lecita. La forma compatta appena vista è utilizzabile quando una variabile appare sia a sinistra sia a destra di un operatore di assegnamento ed è equivalente a quella classica:

```
variabile = variabile[operatore]espressione
```

Si hanno pertanto le seguenti equivalenze.

<i>Forma compatta</i>	<i>Forma Classica</i>
<code>a *= 5;</code>	<code>a = a*5;</code>
<code>a -= b;</code>	<code>a = a-b;</code>
<code>a *= 4+b;</code>	<code>a = a*(4+b);</code>

L'ultima linea evidenzia quale sia la sequenza di esecuzione nella forma compatta:

- viene calcolata l'intera espressione posta a destra dell'assegnamento: `4+b`;
- viene moltiplicato il valore ottenuto per il valore della variabile posta a sinistra dell'assegnamento: `a*(4+b)`;
- viene assegnato il risultato ottenuto alla variabile posta a sinistra dell'assegnamento: `a=a*(4+b)`.

Questo funzionamento è coerente con la bassa priorità degli operatori `+=`, `-=`, `*=`, `/=` e `%=` che hanno lo stesso livello dell'assegnamento semplice `=` (Figura 3.1). Per esempio, dopo la sequenza di istruzioni

```

a = 3;
b = 11;
c = 4;
c -= a*2+b;

```

La variabile *c* ha valore -13.

3.3 Calcolo del fattoriale

Utilizziamo il costrutto `for` per il calcolo del fattoriale, indicato con *n!*, di un intero *n*, definito da

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \dots \cdot 2 \cdot 1$$

dove $1!$ e $0!$ sono per definizione uguali a 1. Avremo, per esempio, che

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$$
$$6! = 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 720$$

```
/* Calcolo di n! (n fattoriale) */
#include <stdio.h>

main()
{
int n, fat, m;

printf("CALCOLO DI N!\n\n");
printf("Inser. n: ");
scanf("%d", &n);

fat = n;
for(m=n; m>2; m--)
    fat = fat*(m-1);

printf("Il fattoriale di: %d ha valore: %d\n", n, fat);
}
```

Listato 3.2 Calcolo del fattoriale di n

Nell'ipotesi di non considerare il caso $n = 0$, un algoritmo possibile è quello del Listato 3.2. Se viene passato in ingresso il valore 4, fat assume tale valore:

```
fat = n;
```

Il ciclo `for` inizializza 4 a m e controlla che sia maggiore di 2. Viene eseguito una prima volta il ciclo

```
fat = fat*(m-1);
```

e fat assume il valore 12. Di nuovo il controllo dell'esecuzione passa al `for` che decrementa il valore di m e verifica se $m > 2$, cioè se $3 > 2$. Viene eseguito il corpo del ciclo

```
fat = fat*(m-1);
```

e fat assume il valore 24. Il `for` decrementa m e verifica se $m > 2$, cioè se $2 > 2$. Questa volta l'esito è negativo e le iterazioni hanno termine. Utilizzando l'operatore `*=`, al posto di `fat=fat*(m-1)` avremmo potuto scrivere

```
fat *= m-1;
```

Per considerare anche il caso in cui sia compreso il fattoriale di zero, prima di iniziare il ciclo ci si deve chiedere se n ha tale valore, nel qual caso il risultato è 1.

```
fat = n;
if(n==0)
    fat = 1;
else
    for(m=n; m>2; m--)
        fat = fat*(m-1);
```

L'uso della variabile m è necessario perché si desidera mantenere il valore iniziale di n per stamparlo nella `printf` finale, altrimenti se ne potrebbe fare a meno utilizzando al suo posto direttamente n :

```
fat = n;
if(n==0)
    fat=1;
else
    for(n=n; n>2; n--)
        fat = fat*(n-1);
```

L'inizializzazione all'interno del `for` `n=n` è del tutto superflua, per cui si può scrivere

```
for(; n>2; n--)  
    fat = fat*(n-1);
```

Questa sintassi è permessa e indica che `esp1` è vuota; il punto e virgola è obbligatorio. Un altro metodo è quello di eseguire le moltiplicazioni successive a partire dal basso: $n! = 2 \cdot 3 \dots (n-1) \cdot n$, inizializzando `fat` a 1 e utilizzando una variabile ausiliaria intera (Listato 3.3). Si noti come con questa soluzione sia già incluso il caso di 0!. Anche questa volta invece di `fat=fat*aux` avremmo potuto scrivere `fat*=aux` ■.

```
/* Calcolo n! (n fattoriale) */  
  
#include <stdio.h>  
  
main()  
{  
    int n, fat, aux;  
  
    printf("CALCOLO DI N!\n\n");  
    printf("Inser. n: ");  
    scanf("%d", &n);  
  
    fat = 1;  
    for(aux=2; aux<=n; aux++) fat = fat*aux;  
  
    printf("Il fattoriale di: %d ha valore: %d\n", n, fat);  
}
```

Listato 3.3 Un'altra possibilità per il calcolo del fattoriale

3.4 Istruzione while

Anche l'istruzione `while`, come l'istruzione `for`, permette di ottenere la ripetizione ciclica di una istruzione:

```
while(esp)  
    istruzione
```

Viene verificato che `esp` sia vera, nel qual caso viene eseguita `istruzione`. Il ciclo si ripete fintantoché `esp` risulta essere vera. Naturalmente, ancora una volta, `istruzione` può essere un blocco. Riprendiamo il programma che calcola la somma dei valori immessi dall'utente e modifichiamolo in modo da controllare il ciclo con `while`:

```
    i = 1;  
    while(i<=5) {  
        printf("Inser. intero: ");  
        scanf("%d", &numero);  
        somma = somma+numero;  
        i++;  
    }  
  
    for(i=1; i<=5; i++) {  
        printf("Inser. intero: ");  
        scanf("%d", &numero);  
        somma = somma+numero;  
    }
```

L'inizializzazione della variabile che controlla il ciclo deve precedere l'inizio del `while` e l'incremento della stessa variabile deve essere inserito come ultima istruzione del blocco. In generale, quando il numero d'iterazioni è noto a priori, per passare da un `for` a un `while` vale la seguente equivalenza:

```
    esp1;  
    while(esp2)  
        corpo_del_ciclo  
    esp3;  
  
    for(esp1; esp2; esp3)  
        corpo_del_ciclo
```

Nel programma precedente si poteva inserire l'incremento della variabile di controllo del ciclo all'interno della condizione logica presente tra parentesi tonde. Si ha infatti la seguente corrispondenza:

```

i = 1;
while(i<=5){
    printf("Inser. intero: ");
    scanf("%d", &numero);
    somma = somma+numero;
    i++;
}

i = 1;
while(i++<=5) {
    printf("Inser. intero: ");
    scanf("%d", &numero);
    somma = somma+numero;
}

```

Grazie all'operatore ++ la variabile *i* viene incrementata automaticamente ad ogni ciclo. È obbligatorio posporre l'operatore alla variabile perché si desidera che l'incremento venga fatto dopo il confronto tra il valore di *i* e 10. In caso contrario il numero di iterazioni sarebbe uguale a nove. Quando si deve ripetere *n* volte un ciclo la migliore soluzione è ancora un'altra:

```

i = n;
while(i-->0)
    corpo_del_ciclo

```

Come abbiamo visto nel capitolo precedente, la condizione logica diviene falsa quando *i* assume valore zero. Nell'esempio precedente si ha:

```

i = 5;
while(i-->0) {
    printf("Inser. intero: ");
    scanf("%d", &numero);
    somma = somma+numero;
}

```

Osserviamo, ancora una volta, come il codice si faccia sempre più compatto.

Trasformiamo adesso il programma in modo che la lunghezza della serie dei numeri in ingresso non sia determinata a priori ma termini quando viene inserito il valore zero. Non è possibile evidentemente risolvere il problema con una ripetizione sequenziale d'istruzioni in quanto il numero di valori non è noto, ma viene deciso a tempo d'esecuzione (Listato 3.4).

```

/* Calcola la somma dei valori interi passati dall'utente
   termina quando viene immesso il valore 0 (zero) */

#include <stdio.h>

main()
{
    int somma, numero;

    printf("SOMMA NUMERI\n");
    printf("zero per terminare\n");
    numero = 1;
    somma = 0;
    while(numero!=0) {
        printf("Inser. intero: ");
        scanf("%d", &numero);
        somma = somma+numero;
    }
    printf("Somma: %d\n",somma);
}

```

Listato 3.4 Esempio di utilizzo dell'istruzione `while`

Alla variabile `numero` si è assegnato il valore 1 per far in modo che il ciclo venga eseguito almeno una volta; ovviamente qualsiasi valore diverso da zero va bene. Una possibile esecuzione è la seguente:

```

SOMMA NUMERI
zero per terminare
Inser. intero: 105

```

```
Inser. intero: 1
Inser. intero: 70
Inser. intero: 0
Somma: 176
```

dove i valori passati dall'utente sono 105, 1, 70 e 0 per terminare l'inserzione.

Ogni istruzione `for` può essere sostituita da un'istruzione `while` se si ha cura di aggiungere le opportune inizializzazioni prima del ciclo e gli opportuni incrementi all'interno dello stesso. In C è vero anche l'inverso. Ogni istruzione `while` ha un suo corrispondente `for`, anche quando il numero d'iterazione non è noto a priori. Per esempio, la parte centrale del programma precedente può essere realizzata con un ciclo `for`:

```
numero = 1;           numero = 1;
somma = 0;           somma = 0;
while(numero!=0) {   for(; numero!=0;) {
    printf("Inser. intero: ");   printf("Inser. intero: ");
    scanf("%d", &numero);       scanf("%d", &numero);
    somma = somma+numero;       somma = somma+numero;
}                               }
```

Infatti, come si è già evidenziato, nel costrutto `for`

```
for(esp1; esp2; esp3)
```

è possibile sostituire *esp1*, *esp2* ed *esp3* con qualsiasi espressione, nella fattispecie *esp2* corrisponde al controllo $n \neq 0$ (n diverso da 0) mentre *esp1* ed *esp3* corrispondono a espressioni vuote. La presenza dei punti e virgola è naturalmente obbligatoria.

Supponiamo che oltre alla somma si desideri determinare il valore massimo della sequenza in ingresso, con la limitazione che i valori debbano essere tutti positivi. Una volta inizializzata la variabile intera `max` a zero il ciclo diventa il seguente:

```
while(numero!=0) {
    printf("Inser. intero positivo: ");
    scanf("%d", &numero);
    if(numero>max) max=numero;
    somma = somma+numero;
}
```

All'interno di un blocco è lecito inserire qualsiasi istruzione, quindi anche un `if`. La variabile `max` viene inizializzata a zero, che è minore di qualsiasi valore che l'utente possa inserire. A ogni iterazione del ciclo viene controllato se il valore inserito dall'utente, presente nella variabile `numero`, è maggiore di `max`, nel qual caso viene assegnato a `max` il nuovo valore. Se si desidera che i valori passati in ingresso non siano comunque superiori a certo numero, supponiamo 10, si può inserire una variabile contatore degli inserimenti e controllarne il valore all'interno del `while`:

```
while(numero!=0 && i<=10)
```

Le due condizioni logiche sono poste in AND, affinché l'iterazione continui: deve essere vero che `numero` è diverso da zero e che `i` è minore di 10 (Listato 3.5).

```
/* Determina somma e maggiore dei valori immessi */
#include <stdio.h>

main()
{
    int somma,numero,max,i;

    printf("SOMMA E MAGGIORE\n");
    printf("zero per finire\n");
    numero = 1;
    somma = 0;
    max = 0;

    i = 1;
```

```

while(numero!=0 && i<=10)
{
    printf("Valore int.: ");
    scanf("%d", &numero);
    if(numero>max)
        max = numero;
    somma = somma+numero;
    i++;
}
printf("Somma: %d\n", somma);
printf("Maggiore: %d\n", max);
}

```

Listato 3.5 Diramazione `if` all'interno di una iterazione `while`

L'incremento della variabile che conta il numero di valori immessi può essere inserito direttamente nella parte *espressione* di `while`:

```

while(numero!=0 && i++<=10) {
    printf("Inser. intero positivo: ");
    scanf("%d", &numero);
    if(numero>max) max=numero;
    somma+=numero;
}

```

L'incremento deve avvenire dopo il controllo `i<10`, per cui l'operatore `++` deve seguire e non precedere `i`. Il ciclo `while` esaminato nell'ultimo programma può essere, come sempre, realizzato con un `for`

```

for(i=1; numero!=0 && i<=10; i++) {
    printf("Inser. intero positivo: ");
    scanf("%d", &numero);
    if(numero>max) max=numero;
    somma+=numero;
}

```

Per far in modo che il programma comprenda anche il caso di numeri negativi, si deve provvedere all'immissione del primo dato in `max` anteriormente all'inizio del ciclo. Una soluzione alternativa è di inizializzare `max` al minimo valore negativo accettato da una variabile intera. Nell'ipotesi che fossero riservati quattro byte a un `int` potremmo quindi scrivere:

```
max = - 2147483648;
```

ma questo valore è dipendente dall'implementazione. Nella libreria `limits.h` sono definiti i valori limite definiti dall'implementazione; in essa sono presenti alcune costanti, fra cui `INT_MAX`, che contiene il massimo valore di un `int`, e `INT_MIN`, che contiene il minimo valore di un `int`. È sufficiente includere nel programma tale libreria per poter utilizzare le variabili in essa definite:

```
#include <limits.h>
```

Si potrà inizializzare `max` al minor intero rappresentabile con una variabile di tipo `int`:

```
max = INT_MIN;
```

3.5 Istruzione `do-while`

Quando l'istruzione compresa nel ciclo deve essere comunque eseguita almeno una volta, risulta più comodo utilizzare il costrutto

```
do
    istruzione
```

```
while (esp);
```

Viene eseguita *istruzione* e successivamente controllato se *esp* risulta essere vera, nel qual caso il ciclo viene ripetuto. Come sempre, l'iterazione può coinvolgere una istruzione composta (blocco).

Riprendiamo il programma che determina la somma e il maggiore tra i numeri immessi dall'utente e realizziamo il ciclo centrale con l'istruzione appena vista (Listato 3.6).

```
/* Determina somma e maggiore dei valori immessi
   (esempio uso do-while) */

#include <stdio.h>

main()
{
int somma,numero,max,i;

printf("SOMMA E MAGGIORE\n");
printf("zero per finire\n");
numero = 1;
somma = 0;
max = 0;

i = 1;
do {
printf("Valore int.: ");
scanf("%d", &numero);
if(numero>max)
max = numero;
somma = somma+numero;
i++;
}
while(numero!=0 && i<=10);

printf("Somma: %d\n", somma);
printf("Maggiore: %d\n", max);
}
```

Listato 3.6 Esempio di utilizzo del costrutto do-while

Il ciclo viene ripetuto fino a quando la condizione del `while` risulta essere vera, o in altre parole si esce dal ciclo quando la condizione del `while` risulta essere falsa. Per trasformare un `while` in `do-while` si deve semplicemente porre il `do` all'inizio del ciclo e il `while (esp)` alla fine dello stesso. Il ciclo poteva essere più sinteticamente espresso come segue:

```
do {
printf("Valore int.: ");
scanf("%d", &numero);
if(numero>max)
max = numero;
somma = somma+numero;
}
while(numero!=0 && ++i<=10);
```

In cui l'operatore `++` deve obbligatoriamente precedere il nome della variabile in quanto l'incremento deve avvenire prima del controllo `i<=10`.