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
```

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

```
esp3;
```

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

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

```
numero = 1;
     somma = 0;
     for(; numero!=0;) {
       printf("Inser. intero: ");
       scanf("%d", &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!=0` (`n` diverso da 0) mentre *esp1* ed *esp3* corrispondono a espressioni vuote. La presenza dei punti e virgola è naturalmente obbligatoria.

✓ **NOTA**

L'istruzione `for`, con la sua chiarezza d'intenti, l'enorme potenza e compattezza, è largamente utilizzata dai programmatori C.

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 condizione 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;