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 ++ e -- hanno priorità maggiore degli operatori binari aritmetici, relazionali e logici, per cui vengono considerati prima degli altri (Figura 3.1) ■.

```
                    !    -       ++      --

                       *      /       %

                            +      -

                    >      >=     <     <=

                           ==     !=

                               &&

                               ||

                               ?:

            =     +=      -=      *=     /=      %=
```

Figura 3.1 Tavola di priorità degli operatori esaminati

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.

Forma compatta	*Forma Classica*
`a *= 5;`	`a = a*5;`
`a -= b;`	`a = a-b;`
`a *= 4+b;`	`a = a*(4+b);`

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

1. 1. viene calcolata l'intera espressione posta a destra dell'assegnamento: `4+b`;
2. 2. viene moltiplicato il valore ottenuto per il valore della variabile posta a sinistra dell'assegnamento: `a*(4+b);`
3. 3. 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.