CORSO DI PROGRAMMAZIONE
A.A. 2014-15

Dispensa 4

Laboratorio

Dott. Mirko Ravaioli
e-mail: mirko.ravaioli@unibo.it

4.1 Operatori compatti

Gli operatori di assegnamento compatti del C forniscono un metodo veloce di combinare operazioni matematiche binarie a operazioni di assegnamento. Ad esempio, si supponga di voler incrementare di 5 il valore di x, o in altre parole voler sommare 5 a x e assegnare il risultato a x. Si può scrivere:

```
x = x + 5;
```

Utilizzando un operatore di assegnamento compatto, che si può considerare come un metodo compatto di assegnamento, si scrive:

```
x += 5;
```

In generale gli operatori di assegnamento compatti hanno la sintassi seguente (dove *op* sta per operatore binario):

```
exp1 op= exp2;
```

che equivale a scrivere:

```
exp1 = exp1 op exp2;
```

Si possono creare degli operatori di assegnamento compatti utilizzando I cinque operatori matematici binari. Alcuni esempi:

Scrittura Compatta	***Scrittura equivalente***
X *= Y	X = X * Y
Y -= Z * 1	Y = Y – Z * 1
A /= B	A = A / B
X += Y / 8	X = X + Y / 8
Y %= 3	Y = Y % 3

4.2 Operatore virgola

La virgola viene utilizzata nel C come semplice carattere di iterputazione che separa le dichiarazioni delle variabili, gli argomenti delle funzioni e così via. In alcune situazioni però la virgola diventa un vero e proprio operatore: è possibile formare un'espressione separando due sottoespressioni con una virgola. Il risultato è che:

- entrambe le espressioni vengono valutate, l'espressione a sinistra viene valutata per prima
- l'intera espressione assume il valore dell'espressione di destra.

Ad esempio, l'istruzione seguente assegna a x il valore di b, quindi incrementa a e successivamente incrementa b:

```
x = (a++, b++);
```

Dato che l'operatore ++ è impiegato in modalità postfissa, a x viene assegnato il valore di b prima che questo venga incrementato. L'uso delle parentesi è necessario dato che l'operatore virgola ha una precedenza molto bassa.

4.3 Costanti

4.3.1 Direttiva #define

Una costante simbolica è una costante rappresentata da un nome o simbolo all'interno del programma. La direttiva #define è una direttiva del preprocessore e ha la seguente sintassi:

```
#define NOMECOSTATNE letterale
```

In questo modo si definisce una costante con il nome NOMECOSTANTE il cui valore è letterale.

Il funzionamento della direttiva #define è quello di istruire il compilatore in modo da comportarsi nel modo seguente: "nel codice sorgente, sostituisci tutte le occorrenze di NOMECOSTANTE con letterale".

4.3.2 Parola chiave const

Il secondo modo di definire costanti simboliche consiste nell'utilizzo della parola chiave **const**. *Const* è un modificatore che può essere utilizzato in aggiunta a qualsiasi dichiarazione di variabile. Una variabile dichiarata *const* non può venire modificata durante l'esecuzione dei programmi, ma può solo essere inizializzata durante la sua dichiarazione, ad esempio:

```
const int conta = 100;
const float pi = 3.14159;
const long debito = 1200000, float tasso = 0.21;
```

L'effetto di *const* si propaga su tutte le variabili della riga di dichiarazione. Nell'ultima riga debito e tasso sono due costanti simboliche. Se il programma cerca di modificare una variabile *const*, il compilatore genera un messaggio di errore, come nel caso seguente:

```
const int count = 100;
count = 200; /*Il programma non viene compilato! E' impossibile rassegnare
            il valore di una costante!*/
```

Quali sono le differenze pratiche tra le costanti simboliche create con la direttiva #define e quelle create con il modificatore *const*? Le differenze riguardano l'uso dei puntatori e l'ambito delle variabili.