Assegnazione di una variabile

Per scrivere un valore dentro una variabile si usa l'operatore di assegnazione, che è rappresentato dal simbolo **=**.

Quindi, se scrivo

```
int a;
a = 12;
```

assegno alla variabile intera a il valore numerico 12.

L'assegnazione ha sempre la forma:

```
variabile = espressione;
```

Ad esempio:

```
a = 2*b + c;
```

A differenza di quanto avviene in matematica, la scrittura

```
2*b + c = a;
```

non ha senso.

My first C++ program

Un semplice programma C++ ha un aspetto di questo genere:

```
main() {
    int a, b;
    a = 12;
    b = 2 + a*a;
}
```

Una sequenza di comandi (function) è identificata da un nome, una coppia di parentesi tonde che contiene i parametri, e una coppia di parentesi graffe che contiene le istruzioni.
Il nome main è quello che caratterizza la prima funzione che viene eseguita.

Questo programma dichiara due variabili intere, a e b. Assegna ad a il valore 12, e a b il valore ottenuto sommando due al quadrato di a (146).

My first C++ program

Un semplice programma C++ ha un aspetto di questo genere:

```
main() {
    int a, b;
    a = 12;
    b = 2 + a*a;
}
```

Dichiariamo due variabili intere a e b.
La dichiarazione è obbligatoria.
Il valore iniziale delle due variabili non è definito.

Questo programma dichiara due variabili intere, a e b. Assegna ad a il valore 12, e a b il valore ottenuto sommando due al quadrato di a (146).

My first C++ program

Un semplice programma C++ ha un aspetto di questo genere:

```
main() {
    int a, b;
    a = 12;
    b = 2 + a*a;
}
```

Assegnamo ad a il valore 12.

Questo programma dichiara due variabili intere, a e b. Assegna ad a il valore 12, e a b il valore ottenuto sommando due al quadrato di a (146).

My first C++ program

Un semplice programma C++ ha un aspetto di questo genere:

```
main() {
    int a, b;
    a = 12;
    b = 2 + a*a;
}
```

Assegnamo a b il valore pari a 2 più il quadrato del valore di a.

Questo programma dichiara due variabili intere, a e b. Assegna ad a il valore 12, e a b il valore ottenuto sommando due al quadrato di a (146).

Bello, ma a che serve?

Il nostro primo programma non serve ovviamente a nulla, nel senso che non ci dice il risultato e non ci consente di variare il valore iniziale di a. Per fare input/output di variabili si deve modificare così:

```
#include <iostream.h>
main() {
    int a, b;
    cout << "a = ";
    cin >> a;
    b = 2 + a*a;
    cout << "b = " << b << endl;
}
```

Il file iostream.h (che fa parte del linguaggio) contiene le definizioni degli identificatori dei canali di input/output che vengono associati alla tastiera e al terminale.

Bello, ma a che serve?

Il nostro primo programma non serve ovviamente a nulla, nel senso che non ci dice il risultato e non ci consente di variare il valore iniziale di a. Per fare input/output di variabili si deve modificare così:

```
#include <iostream.h>
main() {
    int a, b;
    cout << "a = ";
    cin >> a;
    b = 2 + a*a;
    cout << "b = " << b << endl;
}
```

Scriviamo sul terminale
a =

Bello, ma a che serve?

Il nostro primo programma non serve ovviamente a nulla, nel senso che non ci dice il risultato e non ci consente di variare il valore iniziale di a. Per fare input/output di variabili si deve modificare così:

```
#include <iostream.h>
main() {
    int a, b;
    cout << "a = ";
    cin >> a;
    b = 2 + a*a;
    cout << "b = " << b << endl;
}
```

Leggiamo il valore di a dalla tastiera

Bello, ma a che serve?

Il nostro primo programma non serve ovviamente a nulla, nel senso che non ci dice il risultato e non ci consente di variare il valore iniziale di a. Per fare input/output di variabili si deve modificare così:

```
#include <iostream.h>
main() {
    int a, b;
    cout << "a = ";
    cin >> a;
    b = 2 + a*a;
    cout << "b = " << b << endl;
}
```

Scriviamo il risultato (il valore di b) sul terminale. Si possono concatenare gli operatori <<.
Il simbolo endl fa andare a capo.

Un po' di sintassi

A dispetto delle apparenze, le istruzioni dei linguaggi di programmazione non sono totalmente arbitrarie, e quindi non vanno semplicemente imparate a memoria. Ci sono, come nei linguaggi tradizionali, delle regole sintattiche che sono valide in generale. Ne abbiamo visto già alcune:

- la coppia di parentesi graffe {} raggruppa blocchi di istruzioni; sintatticamente equivale ad una istruzione singola.
- ogni istruzione termina con un punto e virgola; il punto e virgola può essere omesso dopo la parentesi che chiude un blocco di istruzioni.
- ogni variabile deve essere dichiarata prima di essere utilizzata; la dichiarazione ha la forma

  tipo nome1, nome2, ...;

Un po' di sintassi

- Fino ad ora abbiamo accennato all'esistenza di variabili di tipo intero (int) e a virgola mobile (double). In realtà cin e cout sono variabili di tipo istream e ostream che identificano dispositivi di ingresso e uscita. Sono dichiarate all'interno di iostream.h. In C++ è possibile definire nuovi tipi di variabili.
- Esistono operatori che consentono di costruire delle espressioni; gli operatori più comuni sono quelli aritmetici (+,-, *, /). Abbiamo visto che esiste un operatore di assegnazione (=) e due operatori di lettura e scrittura da stream (>> e <<). In C++ è possibile definire nuovi operatori o ridefinire il comportamento di quelli esistenti su particolari tipi di varibili.
- Oltre alle variabili si possono utilizzare nelle espressioni delle costanti, sia numeriche (12, 4.566) che alfanumeriche ("a = ").

Il blocco if-then-else

Serve ad eseguire certe istruzioni solo se si verificano date condizioni

```
if (Espressione) Istruzione1; else Istruzione2;
```

Istruzione1 viene eseguita se Espressione ha un valore diverso da zero; Istruzione2 viene eseguita se Espressione vale zero.

Espressione può essere una espressione qualunque. Esistono però degli operatori speciali che valgono 0 o 1 a seconda del verificarsi di date condizioni logiche. I principali sono == (uguale), !=, (non uguale), >, <, >=, <=, && (and), || (or).

```
if (a > 0) {
   b = 2 + a*a;
} else {
   b = 2 – a*a;
}
```

Il ciclo while

Serve a ripetere un blocco di istruzioni fino a quando una determinata espressione vale zero;

Esiste in due forme:

```
while (Espressione) Istruzione;
```

oppure

```
do Istruzione while (Espressione);
```

Ad esempio:

```
int a;
cin >> a;
while (a >= 0) {
    a = a - 1;
}
```

```
int a, sum=0;
do {
    cin >> a;
    sum = sum + a;
} while (a != 0);
```

Incremento e decremento

Abbiamo visto che l'espressione

```
a = a + 1;
```

serve ad incrementare di uno il contenuto di a. In generale la presenza di una variabile sia a sinistra che a destra del segno di uguale indica che il valore di quella variabile viene modificato e che il nuovo valore dipende in qualche modo da quello vecchio; ad esempio:

```
a = 2*x – a*a;
```

In C/C++ esistono due operatori specifici per l'incremento (++) ed il decremento (--).

Di conseguenza le seguenti istruzioni sono equivalenti:

```
a = a + 1;        a++;
a = a – 1;        a--;
```

Il ciclo for

Serve a ripetere una serie di istruzioni un numero fissato di volte. In questa struttura si una usa una variabile intera, detta indice del loop, che ad ogni iterazione conta il numero di ripetizioni effettuate. La sintassi è:

```
for (assegnazione iniziale; condizione; modifica indice) {
   istruzioni;
}
```

Ad esempio:

```
int i;
int sum = 0;
for (i=0; i<10; i++) {
      sum = sum + i;
}
```

Eseguita prima del primo ciclo

Controllata ad ogni ciclo

Eseguita alla fine di ogni ciclo

Analogia for/while

Il ciclo for può essere realizzato anche con while.

```
int i=0;
while (i<10) {
        ...;
        i++;
}
```

è equivalente a

```
int i;
for (i=0; i<10; i++) {
        ...;
}
```

Vettori

Una collezione di n variabili si chiama vettore. Come al solito dobbiamo prendere le distanze dall'omonima entità matematica. Nel contesto dei linguaggi di programmazione un vettore è solo una collezione di scatole con un nome. Il nome identifica la collezione nel suo complesso, mentre ogni singola scatola è identificata dal nome e da un numero progressivo da 0 a n-1. Vediamo in parallelo la sintassi delle varie operazioni su un vettore e su una variabile scalare.

```
int a;          // Dichiarazione          int a[47];
int a = 0;      // Inizializzazione       int a[3] = { 0, 2, 4 };
a = 2;          // Assegnazione           a[2] = 19;
x = 2*a;        // Uso in espressioni     x = a[7]*a[9];
```

Utilizzo dei vettori nei cicli

L'uso dei vettori in programmazione si estende ben oltre la realizzazione di calcoli vettoriali in senso matematico. Si può dire che vettori e cicli sono due cose che vanno di pari passo.

In pratica è piuttosto raro dover ripetere n volte la stessa operazione sugli stessi dati. Usualmente si deve ripetere un'operazione su diversi insiemi di dati e ricavare un insieme di risultati distinti. I vettori consentono appunto di indicizzare dati, risultati e variabili intermedie in modo che siano facilmente utilizzabili all'interno dei loops.

```
int i, a[10], b[10];
for (i=0; i<10; i++) {
        cin >> a[i];
        b[i] = a[i]*a[i];
}
```

Lettura della componenti i-ma di a

Calcolo della componente i-ma di b