11.1 Tipi e variabili

Un *tipo* rappresenta un insieme di valori e di operazioni che possono essere svolte su quei valori. Una *variabile* è una entità che conserva un valore del corrispondente tipo; inserire un nuovo valore in una variabile significa distruggere il vecchio valore presente in essa.
In C vi sono due categorie di tipi: *fondamentali* e *derivati*. I tipi fondamentali sono:

```
char   int   enum   float   double
```

Questi vengono detti anche tipi aritmetici, poiché corrispondono a valori che possono essere interpretati come numeri. I tipi derivati, invece, sono costruiti a partire dai tipi fondamentali, e sono:

```
    void   array   funzioni   puntatori   strutture   unioni
```

Alcuni tra i tipi fondamentali e derivati sono già stati trattati. In questo capitolo riprenderemo le definizioni precedentemente introdotte e le completeremo aggiungendo i qualificatori di tipo, definendo il tipo `enum` e un nuovo insieme di possibili operazioni.
Prima di procedere alla descrizione dettagliata dei tipi fondamentali è necessario richiamare alcuni concetti di carattere generale, quali la dichiarazione e la definizione di un nome.
In C un qualunque nome deve essere dichiarato prima di poter essere usato all'interno del programma; dichiarare un nome significa specificare a quale tipo appartiene quel nome e quindi, indirettamente, quanta memoria dovrà essere riservata per quel nome, e quali sono le operazioni ammesse su di esso. Esempi di dichiarazione di un nome sono:

```
char   c;
int    conta = 0;
double pot(float, int);
char   *animale = "Anatra";
double cubo(float c) { return c*c*c }
extern int codice_errore;
```

Alcune di queste dichiarazioni sono anche delle definizioni, cioè hanno l'effetto di creare la relativa regione di memoria, detta "oggetto"; esattamente si tratta di:

```
char   c;
int    conta = 0;
char   *animale = "Anatra";
double cubo(float c) { return c*c*c }
```

Le prime due riservano una determinata zona di memoria della dimensione di un `char` e di un `int` cui associare rispettivamente il nome `c` e il nome `conta`. L'ultima associa al nome `cubo` una porzione di programma, corrispondente alle istruzioni che formano la funzione `cubo`. Le dichiarazioni:

```
double pot(float, int);
extern int codice_errore;
```

invece non corrispondono ad alcuna allocazione di memoria. Entrambe le istruzioni introducono un nome, ma rimandano la definizione del nome, cioè l'allocazione dell'oggetto corrispondente, a un'altra parte del programma. Nel Capitolo 7 abbiamo già visto la differenza tra dichiarazione e definizione di una funzione; questa differenza può esistere anche tra variabili che non siano funzioni, come nel caso della variabile `codice_errore` il cui tipo è `int`, e la cui definizione è `extern` (esterna) cioè definita in un qualche altro file, diverso da quello in cui è dichiarata per mezzo dell'istruzione .

```
extern int codice_errore;
```

Si ricordi che tutte le dichiarazioni che specificano un valore per il nome che introducono sono sempre anche delle definizioni. Per esempio:

```
int    conta = 0;
char   *animale = "Anatra";
double cubo(float c) { return c*c*c }
```

sono delle definizioni poiché accanto ai nomi `conta`, `animale` e `cubo` vengono specificati dei valori. Il valore iniziale associato a `conta` e `animale` può essere cambiato nel corso del programma, poiché si tratta di inizializzazioni; il valore associato alla funzione `cubo`, ossia il suo blocco di istruzioni, è permanente, cioè non può più essere cambiato nel corso del programma.

✓ NOTA

È molto importante che il programmatore C acquisisca piena padronanza dell'uso della memoria dell'elaboratore; deve quindi saper correttamente distinguere tra dichiarazioni che definiscono un'entità e dichiarazioni che introducono un nome senza definire l'entità corrispondente.