11.3 Costanti

Si dice costante un valore che non può essere variato durante l'esecuzione di un programma. A una costante può essere associato un nome simbolico come sua rappresentazione.
Esistono tre principali tipi di costanti: carattere, intere, e floating point; un ulteriore tipo è rappresentato dalle stringhe di caratteri, che vengono trattate come costanti di tipo `char[]`. Ecco quindi alcuni esempi di costante:

```
int                  43, 452, -6573
long           -433L, 0L, 547343771
double               32.45, -34.4, -0.33e-3, 54.3e4
char           'f', 'G', 't', '#'
"stringa"      "Bobo","Frog","ranocchio"
```

In C sono previste quattro categorie di costanti di tipo intero:

- le costanti decimali;
- le costanti ottali;
- le costanti esadecimali;
- le costanti carattere.

Le costanti decimali sono quelle più comunemente usate e sono della forma:

```
0        1234        976         12345678901234567890
```

Il tipo di una costante decimale è `int` se `int` è sufficiente a contenerla, altrimenti è `long`. Il compilatore avverte se si stanno trattando costanti più grandi di quelli rappresentabili sulla macchina. Per facilitare la programmazione di basso livello il linguaggio consente la definizione di costanti numeriche in sintassi ottale ed esadecimale (purtroppo manca la sintassi binaria):

- ottale aggiungere una cifra `0` prima della costante
- esadecimale aggiungere `0x` o `0X` prima della costante

Per esempio: `012`, `077`, `-05` sono costanti ottali, mentre `0xAA`, `0xddL`, `0xF` sono costanti esadecimali.
Una costante floating point è di tipo `double`. Il compilatore, in genere, avverte anche in questo caso se si stanno trattando costanti più grandi di quelli rappresentabili sulla macchina. Esempi di costanti floating point sono:

```
1.23     .23      0.23     1.    1.0      1.2e10   1.23e-5
```

Non si possono inserire spazi nel mezzo di una costante floating point. Per esempio, la sequenza di caratteri

```
65.43 e - 21
```

non è una costante floating point ma rappresenta quattro simboli diversi: `65.43`, `e`, `-`, `21`, che provocheranno un errore sintattico.
In C non esiste propriamente un tipo carattere, ma piuttosto un tipo intero che può contenere caratteri. Esiste una speciale convenzione per indicare le costanti di tipo carattere: esse vengono racchiuse tra apici:

```
'a'      '0'      'A'
```

Tali costanti carattere sono realmente delle rappresentazioni simboliche, che corrispondono ai valori interi associati a ogni carattere nell'insieme dei caratteri della macchina. Se il sistema usa la codifica ASCII il valore di `'0'` è 48, mentre se usa la codifica EBCDIC il suo valore è 240.
È possibile definire costanti carattere non limitate ai caratteri alfanumerici ma estese a tutte le 256 combinazioni ottenibili con gli 8 bit di un byte. La sintassi di definizione è:

```
\ooo
```

dove `ooo` rappresenta una maschera di bit interpretata secondo le convenzioni dell'aritmetica ottale. Così, per esempio, si hanno le seguenti corrispondenze.

Maschera ottale	Rappresentazione binaria	Rappresentazione decimale
\201	01000001	65
\012	00001010	10
\11	00001001	9
\0	00000000	0
\377	11111111	254

Se consideriamo la prima linea, l'assegnamento:

```
lettera = '\201';
```

nel codice ASCII corrisponde a:

```
lettera = 'A';
```

Questa rappresentazione è comoda quando si devono utilizzare combinazioni non alfanumeriche, come quelle presenti nelle linee della tabella successive alla prima. Si provi a verificarne la rappresentazione nel codice ASCII.
Per i compilatori conformi allo standard ANSI è prevista anche la codifica esadecimale per le costanti `char`:

```
\xhhh
```

che si ottiene facendo precedere il valore da `\x`: `'\xfa'`, `'\1a'`. Comunque, poiché esistono molte costanti carattere non alfanumeriche di uso comune, il C aiuta ulteriormente il programmatore nella loro definizione.

\'	Apice singolo
\"	Doppio apice
\?	Punto interrogativo
\\	Backslash – carattere \
\a	"Bell"
\b	"Backspace" (^H)
\f	"Form feed" (^L)
\n	New line (^J)
\r	Carriage return (^M)
\t	"Tab" (^I)
\v	Tabulazione verticale (^V)

Si ricorda che, nonostante l'apparenza, questi sono tutti caratteri singoli.
Le costanti cui si fa riferimento con un nome (o simbolo) sono dette costanti *simboliche*. Una costante simbolica è quindi un nome il cui valore non può essere ridefinito nell'ambito di validità del nome stesso. In C esistono tre tipi di costanti simboliche:

1. 1. costanti rese simboliche per mezzo della parola chiave *const*;
2. 2. una collezione di costanti elencato da una enumerazione;
3. 3. costanti simboliche corrispondenti ai nomi di array e funzioni.

La parola chiave `const` può essere aggiunta alla dichiarazione di un oggetto in modo da rendere quell'oggetto una costante invece di una variabile:

```
const int modello = 145;
const int v[] = {1, 2, 3, 4};
```

Il lettore osservi come, non potendosi effettuare assegnazioni a una costante nel corso di un programma, si debba necessariamente procedere alla inizializzazione in fase di dichiarazione:

```
modello = 165;      /* errore */
modello++;          /* errore */
```

Dichiarando un nome `const` si fa in modo che il suo valore non cambi nell'ambito di validità del nome. Dunque `const` è un modificatore di tipo, nel senso che limita i modi in cui un oggetto può essere usato, senza di per sé specificare il tipo di quell'oggetto.
Non è richiesta alcuna memoria dati per allocare una costante, per il semplice motivo che il compilatore conosce il suo valore e lo sostituisce all'interno del programma prima di generare l'eseguibile. L'inizializzatore di un nome di tipo `const` può essere anche una espressione di tipo costante che comunque viene valutata durante la compilazione. Il tipo `const` è utilizzato anche per ottenere costanti in virgola mobile di tipo `float`, non disponibili di per sé:

```
const float pi8 = 3.14159265;
```

La dichiarazione `const int` può essere abbreviata da `const`.
Esiste un metodo alternativo di definire costanti intere, spesso più conveniente dell'uso di `const`. Per esempio:

```
enum { QUI, QUO, QUA };
```

definisce tre costanti intere dette enumeratori alle quali assegna implicitamente dei valori interi sequenziali e crescenti a partire da zero; `enum { QUI, QUO, QUA }` è equivalente a:

```
const QUI = 0;
const QUO = 1;
const QUA = 2;
```

A una enumerazione può essere associato un nome:

```
enum papero { QUI, QUO, QUA };
```

che non è un nuovo tipo ma un sinonimo di `int`. Agli enumeratori possono essere anche assegnati esplicitamente valori:

```
enum valore_simbolo {
       NOME, NUMERO,  FINE,
       PIU = '+', MENO = '-', PER = '*', DIV = '/'
     };
```

La dichiarazione

```
valore_simbolo x;
```

costituisce un utile suggerimento sia per il lettore sia per il compilatore!