Dati elementari, operatori, strutture di controllo

Salvatore Filippone
salvatore.filippone@uniroma2.it

Tipi di dati elementari

Come abbiamo visto in "Hello, world", occorre *dichiarare* le variabili:

Tipi primitivi: `char`, `int`, `float`, `double`;

Modificatori: `signed`, `unsigned`, `short`, `long`;

Speciale: `void`

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  int times=100;

  printf("Hello, world!\n");
  printf("Welcome here %d times \n",times);
  return(0);
}
```

Le variabili hanno delle regole di *visibilità*, tipicamente dentro un *blocco* {}

Operatori

Sulle variabili agiscono degli *operatori* per costruire delle *espressioni*:

Operatore di assegnazione: `=`

Operatori aritmetici: gli usuali `+ - * /`

Operatori relazionali: `< > == !=`

Operatori di incremento: `++ --`

Operatori logici: `| || & && ~`

Ogni espressione ha un valore; inoltre, gli operatori hanno una precedenza di associazione (ma si possono usare le parentesi).

Operatori

Attenzione agli operatori interi:

```
1    int main(int argc, char *argv[]) {
2      double result = 1 / 2 * 2;
3      printf("Result is %lf\n", result);
4      return 0;
5    }
```

Operatore di *cast*:

```
1    int main(int argc, char *argv[]) {
2      double result = ((double) 1) / 2 * 2;
3      printf("Result is %lf\n", result);
4      return 0;
5    }
```

Praticamente qualunque programma richiede di *alterare* il flusso delle istruzioni in dipendenza dai dati. Primo esempio: `if`

```
1  if(TEST) {
2    CODE;
3  } else if(TEST) {
4    CODE;
5  } else {
6    CODE;
7  }
```

```
if (a != 0.0) {
  c = b / a;
} else if (a == 0.0) {
  fprintf(stderr,"Non dividere per zero!\n");
}
```

Strutture di controllo: `switch`

```
1     switch (OPERAND) {
2       case CONSTANT1:
3        CODE;
4        break;
5       case CONSTANT2:
6        CODE;
7        break;
8       default:
9        CODE;
10    }
```

L'argomento di `switch` deve avere un valore intero o carattere.

Strutture di controllo: ciclo `for`

Spesso un blocco di istruzioni deve essere *ripetuto* più volte

```
1  for (INIT; TEST; POST) {
2    CODE;
3  }
```

Calcolo della potenza x^n

```
1  y=1;
2  for (i=0; i<n; i++) {
3    y = y * x ;
4  }
```

Il ciclo `for` è appropriato quando il numero di iterazioni è prevedibile.

Strutture di controllo: ciclo `while`

Quando il numero di iterazioni non è prevedibile, si usa `while`

```
while(TEST) {
  CODE;
}
```

Iterazione $3 \cdot n + 1$

```
while (n>1) {
  if (n%2 == 0) {
    n /= 2;
  } else {
    n = 3*n + 1;
  }
}
```

Strutture di controllo: `break` e `continue`

Con l'istruzione `break` si *esce* dal ciclo circostante:

```
while (TEST) {
  if (OTHER TEST) {
    break;
  }
  CODE;
}
OTHER CODE;
```

Con l'istruzione `continue` si passa alla *prossima iterazione* del ciclo

```
for (INIT; TEST; POST) {
  if (OTHER TEST) {
    continue;
  }
  CODE;
}
```

Chiamate a funzioni

Una *funzione* racchiude un insieme di istruzioni da eseguire su dati di volta in volta diversi

```
1  TYPE NAME(ARG1, ARG2)
2  {
3    CODE;
4    return VALUE;
5  }
```

Vantaggi essenziali

- Scrivere il codice una volta, riusarlo più volte;
- Separare interfaccia da implementazione;

Gli argomenti di una funzione vengono normalmente passati *per valore*

Chiamate a funzioni

```c
#include <stdio.h>

int mcd(int m, int n); /* di solito si crea un file di include */

int main(int argc, char *argv[])
{
  int a,b,c ;
  a = 12;
  b = 25;
  c = mcd(a,b);
  printf("MCD: %d\n",c);
  return(0);
}
```

Variabili: regole di visibilità

Ogni variabile ha un ambito (*scope*) e una conseguente visibilità

- Variabli globali;
- Variabili locali;
- Variabili statiche.

Nel C89 le variabili *devono* essere dichiarate all'inizio di una funzione. Nelle versioni successive no; pratiche comuni:

- Dichiarare le variabili "importanti" della procedura all'inizio (per avere subito idea della complessità);
- Dichiarare variabili di uso locale e temporaneo nel blocco di codice che le usa (p.es. dentro un ciclo).

Tipi di dati elementari: i dettagli

Torniamo ai tipi di dati elementari: `char`, `int`, `float`, `double`, `short`, `long`;
In realtà *non è garantita la dimensione dei dati stessi* ma solo delle relazioni minime. In particolare abbiamo:

char	Minimo tipo intero, dimensione minima 8 bit;
short int	Dimensione minima 16 bit;
int	Dimensione maggiore o uguale di `short int`, quindi minima 16 bit;
long int	Dimensione minima 32 bit, maggiore o uguale di `int`;
long long int	Dimensione minima 64 bit;
float/double/long double	Numeri "reali" in precisione singola/doppia/estesa; lo standard del C non li specifica

Interi nel C99

Tipi interi a dimensione specificata esattamente

(richiedono `stdint.h`)

`int8_t` `uint8_t` Interi a 8 bit con o senza segno;

`int16_t` `uint16_t` Interi a 16 bit con o senza segno;

`int32_t` `uint32_t` Interi a 32 bit con o senza segno;

`int64_t` `uint64_t` Interi a 64 bit con o senza segno;

Ulteriori costanti e tipi:

(vedi anche `limits.h`)

`INT_MIN` Minimo valore intero (anche `SHRT_MIN`, `INT16_MIN` etc);

`INT_MAX` Massimo valore intero (anche `SHRT_MAX`, `INT32_MAX`, `UINT16_MAX` etc);

`intptr_t` un intero che è in grado di contenere un puntatore;

Reali e complessi nel C99

Tipi

float	Reali a (32) bit;
double	Reali a (64) bit;
long double	Reali con (più di 64) bit;
float, double, long double _Complex	Tipi complessi (C99) corrispondenti ai reali;

Ulteriori costanti:

FLT_EPSILON	il più piccolo x tale che $1.0 + x > 1.0$; (anche DBL)
FLT_MIN	Il più piccolo numero normalizzato; (anche DBL)
FLT_MAX	Il più grande numero normalizzato; (anche DBL)

Tipi booleani

Nel C base non esiste un tipo booleano:

- I valori logici vengono gestiti con gli interi;
- Il valore 0 è `false`, qualunque valore diverso da 0 è `true`.

Nel C11 viene definito il tipo `_Bool` (header `stdbool.h`)

- keyword `bool`, `true`, `false`
- `true` vale 1, `false` vale 0;

Tutti gli operatori

Operatori aritmetici

+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione
%	Modulo
++	Incremento
--	Decremento

Operatori Logici

&&	And
||	Or
!	Negazione
? :	Confronto ternario

Operatori di confronto

==	Eguaglianza
!=	Non Eguaglianza
>	Maggiore
>=	Maggiore o uguale
<	Minore
<=	Minore o uguale

Operatori bit a bit

&	and
|	or
^	xor
~	complemento a 1
<<	shift sinistra
>>	shift destra

Operatori di assegnazione

=	Assegnazione
+=	Somma e assegnazione
-=	Sottrazione e assegnazione
*=	Moltiplicazione e assegnazione
/=	Divisione e assegnazione
%=	Modulo e assegnazione
<<=	Shift e assegnazione
>>=	Shift e assegnazione
&=	And e assegnazione
|=	Or e assegnazione
^=	Xor e assegnazione