1 Ordinamento dei bytes nei tipi multi-byte (Rev. 2.0.2)

La struttura naturale di indirizzamento della memoria utilizzata dal linguaggio C è quella in cui ogni singolo carattere (o byte) della memoria può essere indirizzato singolarmente. Questo vuol dire che ogni byte della memoria ha un suo indirizzo che può essere assegnato ad un puntatore. Computers che usano questo tipo di indirizzamento sono chiamati byte-addressable computers.

I tipi di dimensione più grande di un byte, detti anche tipi *multi-byte*, occupano blocchi di memoria composti da più caratteri contigui ed il loro indirizzo in memoria è di solito dato dall'indirizzo del *primo carattere* del blocco definito come carattere del blocco con l'indirizzo più basso. Tuttavia l'ordine con cui i diversi bytes di un tipo multi-byte sono disposti nei vari caratteri del blocco di memoria può dipendere dall'architettura del computer e quindi il contenuto del primo carattere del blocco in genere dipende dal computer.

Le due architetture più usate sono l'architettura big-endian o left-to-right e little-endian o right-to-left.

- Nell'architettura big-endian i caratteri del blocco sono occupati con il byte più significativo nel primo carattere del blocco (big-end-first). In questa architettura l'indirizzo del tipo multi-byte corrisponde all'indirizzo del suo byte più significativo.
- Nell'architettura little-endian i caratteri del blocco sono occupati con il byte meno significativo nel primo carattere del blocco (little-end-first). In questa architettura l'indirizzo del tipo multi-byte corrisponde all'indirizzo del suo byte meno significativo.

Siccome la memoria viene occupata a partire dalle locazioni con indirizzo minore nell'ordinamento big-endian il tipo multi-byte viene messo in memoria partendo dal suo byte più significativo e quindi da sinistra a destra. Invece nell'ordinamento little-endian il tipo multi-byte viene messo in memoria partendo dal suo byte meno significativo e quindi da destra a sinistra.

Per illustrare i due diversi ordinamenti consideriamo il numero intero 123456 memorizzato in un tipo intero di 32 bit, ovvero di 4 byte:

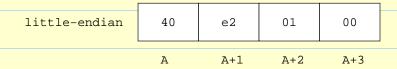
$$123456 = 00000000 00000001 11100010 01000000 = 00 01 e2 40$$

A seconda dell'architettura i 4 bytes sono organizzati in memoria a partire dal byte più a sinistra o più a destra:

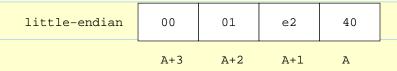
Indirizzo		little-endian	big-endian	
A	\Rightarrow	01000000	00000000	
A+1	\Rightarrow	11100010	0000001	
A+2	\Rightarrow	00000001	11100010	
A+3	\Rightarrow	00000000	01000000	

Possiamo visualizzare i due ordinamenti come segue:

big-endian	00	01	e2	40
	А	A+1	A+2	A+3



ovvero



da cui risulta evidente l'ordinamento da sinistra a destra o da destra a sinistra delle due architetture.

I seguente programma determina l'ordinamento usato dal computer scrivendo l'ordinamento in memoria di una variabile di tipo int.

Programma: test-endian.c

```
- Descrizione : Determina l'ordinamento in memoria di un int
             Usa il modulo bit_utils.c
- $Id: test-endian.c v 1.2 01.11.04 AC
*************************
#include <stdio.h>
* modulo bit_utils.c
#include "bit_utils.h"
int main(void)
           n, add;
 int
 unsigned char *c_p;
       line[81];
 printf("Numero intero: ");
 fgets(line, sizeof(line), stdin);
 sscanf(line,"%d", &n);
 printf("\nbit-rep: ");
 print_bit((unsigned char *) &n, sizeof(int));
 printf("\nbyte-rep: ");
 print_byte((unsigned char *) &n, sizeof(int));
 printf("\nmem-ord: \n");
 c_p = (unsigned char *) &n;
```

Per accedere ai singoli bytes della variabile di tipo int n si usa il puntatore c_p al tipo unsigned char a cui viene assegnato l'indirizzo della variabile n ossia l'indirizzo del primo carattere del blocco di memoria occupato dalla variabile n.

```
char *c_p;
...
c_p = (unsigned char *) &n;
```

In questo modo, siccome la dimensione di un char è 1, è possibile accedere ai singoli bytes di n semplicemente incrementando c_p. Il cast nell'assegnazione è necessario in quanto i puntatori sono a tipi diversi.

Chiaramente il risultato dell'ordinamento dipende dal computer, ad esempio sul mio ottengo:

```
Numero intero: 123456

bit-rep: 00000000 00000001 11100010 01000000

byte-rep: 00 01 e2 40

mem-ord:
0xbffffacc : 40
0xbffffacd : e2
0xbfffface : 01
0xbffffacf : 00
```

che mostra chiaramente l'ordinamento little-endian.

Vi sono molti modi di determinare l'ordinamento dei bytes utilizzato, ad esempio utilizzando una unione come mostra il programma seguente.

Programma: test-endian_1.c

```
int main(void)
{
    u.n = 1;

    if (u.a[0] == 1) {
        printf("Ordinamento little-endian\n");
    }
    else if (u.a[sizeof(unsigned int) - 1] == 1) {
        printf("Ordinamento big-endian\n");
    }
    else {
        printf("Ordinamento non riconosciuto\n");
    }
    return 0;
}
```

La filosofia del programma è la stessa del precedente. Il campo a dell'unione è un array di tipo unsigned char di dimensione pari alla dimensione di un tipo int. Ciascun elemento corrisponde ad un carattere del blocco in cui viene memorizzato il contenuto del campo n di tipo int dell'unione. Il campo a è l'equivalente di A nelle figure precedenti. Di conseguenza se assegnamo il valore 1 al campo u.n questo si troverà nell'elemento a[0] nel caso di ordinamento little-endian (little-end-first) e nell'elemento a[sizeof(unsigned int) - 1] nel caso di ordinamento big-endian (big-end-first). Se il valore non si trova in nessuno di questi due l'ordinamento non è nè big-endian nè little-endian.

I termini big-endian e little-endian derivano dai Lillipuziani dei Viaggi di Gulliver, il cui problema principale era se le uova bollite dovessero essere aperte dal lato grande (big-endian) o da quello piccolo (little-endian).

Il problema della conversione tra un ordinamento e l'altro è noto come il *NUXI problem*. Infatti se la parola UNIX fosse memorizzata in due variabili di 2 byte questa apparirebbe in memoria come UNIX in un sistema con architettura big-endian e NUXI in un sistema con architettura little-endian.

Osserviamo che l'ordinamento big-endian/little-endian si riscontra anche in altri campi, come ad esempio nel modo di scrivere le date. Gli Europei scrivono la data come gg/mm/aa quindi un ordinamento little-endian, mentre i Giapponesi la scrivono come aa/mm/gg e quindi usano un ordinamento big-endian. Per contro gli Americani la scrivono come mm/gg/aa e quindi ne big-endian ne little-endian, ma con ordinamento middle-endian. Architetture middle-endian con ordinamenti perversi dei bytes si possono trovare anche su alcuni computers, ad esempio i computers PDP.

©AC 2003