Spaccare i bit in 4

Esempi di uso degli operatori di manipolazione dei bit del linguaggio C.

Una variabile di tipo intero (`int`, `short`, `long`, `char`,...) può essere usata per memorizzare molteplici informazioni in singoli bit o in sequenze di bit.

La lettura o modifica delle informazioni in queste variabili avviene tramite gli operatori di gestione bit a bit e con le costanti binarie/ottali o esadecimali.

```
// Costanti:
    int d = 42;
    int o = 052;
    int x = 0x2a;
    int X = 0X2A;
    int b = 0b101010; // ammesso solo da gcc recenti, diventerà standard in C23

    int bit0 = 1 << 0; // equivale a 1  oppure 0x1  01 o  0b1
    int bit1 = 1 << 1; // equivale a 2  oppure 0x2  02 o  0b10
    int bit2 = 1 << 2; // equivale a 4  oppure 0x4  04 o  0b100
    int bit3 = 1 << 3; // equivale a 8  oppure 0x8  010 o 0b1000
    int bit4 = 1 << 4; // equivale a 16 oppure 0x10 020 o 0b10000
    int bit5 = 1 << 5; // equivale a 32 oppure 0x20 040 o 0b100000
    //... la forma 1 << n è più leggibile
    // in decimale occorre ricordare le potenze di 2
    // in ottale e esadecimale è compatto ma occorre contare i bit
    // in binario è facile sbagliare a contare gli zeri

// Operazioni:
    A | B -> nel risultato sono "accesi" tutti i bit che sono accessi in A o in B
    A & B -> nel risultato sono "accesi" tutti i bit che sono accessi sia in A sia in B
    ~A -> nel risultato sono accesi i bit che in A sono spenti e viceversa.

    A |= B; // accende in A i bit di B
    A &= ~B; // spegne in A i bit di B.
```

esempio

Prendiamo per esempio il campo `st_mode` della struttura `struct stat`, usata dalla system call `stat`.

```
  15  14  13  12  11  10  9   8   7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   file type   | U | G | T | R | W | X | R | W | X | R | W | X |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
                            |   user    |   group   |   other   |
```

Il file incluso `#include <stat.h>` fornisce fra le altre queste definizioni (semplificate a scopo didattico)

```
#define S_IFMT  0170000 /* These bits determine file type.  */
#define S_IFDIR 0040000 /* Directory.  */
#define S_IFCHR 0020000 /* Character device.  */
#define S_IFBLK 0060000 /* Block device.  */
#define S_IFREG 0100000 /* Regular file.  */
#define S_IFIFO 0010000 /* FIFO.  */
#define S_IFLNK 0120000 /* Symbolic link.  */
#define S_IFSOCK  0140000 /* Socket.  */

#define __S_ISTYPE(mode, mask)  (((mode) & S_IFMT) == (mask))

#define S_ISDIR(mode)  __S_ISTYPE((mode), S_IFDIR)
#define S_ISCHR(mode)  __S_ISTYPE((mode), S_IFCHR)
#define S_ISBLK(mode)  __S_ISTYPE((mode), S_IFBLK)
#define S_ISREG(mode)  __S_ISTYPE((mode), S_IFREG)
#define S_ISFIFO(mode) __S_ISTYPE((mode), S_IFIFO)
#define S_ISLNK(mode)  __S_ISTYPE((mode), S_IFLNK)
#define S_ISSOCK(mode) __S_ISTYPE((mode), S_IFSOCK)

#define S_ISUID 04000 /* Set user ID on execution.  */
#define S_ISGID 02000 /* Set group ID on execution.  */
#define S_ISVTX 01000 /* Save swapped text after use (sticky).  */
#define __S_IREAD 0400  /* Read by owner.  */
#define __S_IWRITE  0200  /* Write by owner.  */
#define __S_IEXEC 0100  /* Execute by owner.  */

#define S_IRUSR __S_IREAD /* Read by owner.  */
#define S_IWUSR __S_IWRITE  /* Write by owner.  */
#define S_IXUSR __S_IEXEC /* Execute by owner.  */
/* Read, write, and execute by owner.  */
#define S_IRWXU (__S_IREAD|__S_IWRITE|__S_IEXEC)

#define S_IRGRP (S_IRUSR >> 3)  /* Read by group.  */
#define S_IWGRP (S_IWUSR >> 3)  /* Write by group.  */
#define S_IXGRP (S_IXUSR >> 3)  /* Execute by group.  */
/* Read, write, and execute by group.  */
#define S_IRWXG (S_IRWXU >> 3)

#define S_IROTH (S_IRGRP >> 3)  /* Read by others.  */
#define S_IWOTH (S_IWGRP >> 3)  /* Write by others.  */
#define S_IXOTH (S_IXGRP >> 3)  /* Execute by others.  */
/* Read, write, and execute by others.  */
#define S_IRWXO (S_IRWXG >> 3)
```

quindi equivale a:

```
#define S_IFMT   0b1111000000000000 /* 0xF000 */
```

```
#define S_IFDIR  0b0100000000000000 /* 0x4000 */
#define S_IFCHR  0b0010000000000000 /* 0x2000 */
#define S_IFBLK  0b0110000000000000 /* 0x6000 */
#define S_IFREG  0b1000000000000000 /* 0x8000 */
#define S_IFIFO  0b0001000000000000 /* 0x1000 */
#define S_IFLNK  0b1010000000000000 /* 0xA000 */
#define S_IFSOCK 0b1100000000000000 /* 0xC000 */

#define S_ISUID  0b0000100000000000 /* 0x0800 i.e. 1 << 11 */
#define S_ISGID  0b0000010000000000 /* 0x0400 i.e. 1 << 10 */
#define S_ISVTX  0b0000001000000000 /* 0x0200 i.e. 1 <<  9 */
#define S_IRUSR  0b0000000100000000 /* 0x0100 i.e. 1 <<  8 */
#define S_IWUSR  0b0000000010000000 /* 0x0080 i.e. 1 <<  7 */
#define S_IXUSR  0b0000000001000000 /* 0x0040 i.e. 1 <<  6 */
#define S_IRWXU  0b0000000111000000 /* 0x01C0 */
#define S_IRGRP  0b0000000000100000 /* 0x0020 i.e. 1 <<  5 */
#define S_IWGRP  0b0000000000010000 /* 0x0010 i.e. 1 <<  4 */
#define S_IXGRP  0b0000000000001000 /* 0x0008 i.e. 1 <<  3 */
#define S_IRWXG  0b0000000000111000 /* 0x0038 */
#define S_IROTH  0b0000000000000100 /* 0x0004 i.e. 1 <<  2 */
#define S_IWOTH  0b0000000000000010 /* 0x0002 i.e. 1 <<  1 */
#define S_IXOTH  0b0000000000000001 /* 0x0001 i.e. 1 <<  1 */
#define S_IRWXO  0b0000000000000111 /* 0x0007 */
```

Per controllare il tipo del file si selezionano i soli bit che rappresentano il tipo del file e si confrontano con il valore che rappresenta il tipo specifico (dir, file regolare, file speciale etc.):

```
#define S_ISDIR(mode)  (mode & 0b1111000000000000) == 0b0100000000000000
#define S_ISCHR(mode)  (mode & 0b1111000000000000) == 0b0010000000000000
#define S_ISBLK(mode)  (mode & 0b1111000000000000) == 0b0110000000000000
#define S_ISREG(mode)  (mode & 0b1111000000000000) == 0b1000000000000000
#define S_ISFIFO(mode) (mode & 0b1111000000000000) == 0b0001000000000000
#define S_ISLNK(mode)  (mode & 0b1111000000000000) == 0b1010000000000000
#define S_ISSOCK(mode) (mode & 0b1111000000000000) == 0b1100000000000000
```

Ad esempio se vogliamo vedere se il file con `mode = 0b1000000111101101` è un file regolare `S_ISREG(mode)` fa le seguenti operazioni:

```
mode    0b1000000111101101 &
S_IFMT  0b1111000000000000 =
        ------------------
        0b1000000000000000 ==
        0b1000000000000000
        ------------------
        true
```

Se si volesse modificare il tipo del file mantenendo invariati gli altri bit:

```
#define settype(newtype, mode) mode = ((mode) & ~S_IFMT) | (newtype)
```

Il primo termine `((mode) & ~S_IFMT)` azzera i bit di mode che rappresentano il tipo del file, il seguito della macro (`| (newtype)`) imposta il nuovo tipo. Applichiamo la macro al `mode` precedente (`mode = 0b1000000111101101`) per cambiare il tipo in link (`0b1010000000000000`):

```
mode    0b1000000111101101 &
~S_IFMT 0b0000111111111111
        ------------------
        0b0000000111101101 |
        0b1010000000000000 =
        ------------------
        0b1010000111101101
```

riassumendo

Pattern classici per alcune azioni comuni sono:

- accendere il bit n in v

```
v |= 1 << n
```

- invertire il bit n in v

```
v ^= 1 << n
```

- spegnere il bit n in v

```
v &= ~(1 << n)
```

- assegnare il valore del bit n di v alla variable mybit:

```
mybit = (v >> n) & 1
//oppure
mybit = !!(v & (1 << n))
```

- copiare il valore del bit n da v a w

```
w = (w & ~(1<<n)) | (v & (1<<n))
```

- fare qualcosa se il bit n di è acceso:

```
if (v & (1 << n)) ....
```

per fare qualcosa se il bit è spento basta negare la condizione

- accendere in w tutti i bit accesi in v:

```
w |= v
```

- spegnere in w i bit accesi in v:

```
w &= ~v
```

- copiare i bit selezionati dalla maschera m (quelli accesi in m) da v a w:

```
w = (w & ~m) | (v & m)
```

- fare qualcosa se almeno un bit di v è acceso in w:

```
if (v & w) ...
```

- fare qualcosa se i bit selezionati dalla maschera m sono uguali in v e in w:

```
if ((v & m) == (w & m)) ....
```

se i bit accesi di w sono un sottoinsieme di quelli accesi in m (come per esempio `S_IFDIR` rispetto a `S_IFMT`) si può scrivere:

```
if ((v & m) == w) ....
```

- selezionare i singoli byte:

```
unsigned int value;
unsigned int byte[4];
byte[3] = (value >> 24);         // oppure (value & 0xff000000) >> 24;
byte[2] = (value >> 16) & 0xff; // oppure (value & 0x00ff0000) >> 16;
byte[1] = (value >> 8) & 0xff;  // oppure (value & 0x0000ff00) >> 8;
byte[0] = value & 0xff;          // oppure (value & 0x000000ff) >> 0;
```

Se il la variabile byte è un array di interi a 8 bit si può evitare il mascheramento `& 0xff`:

```
unsigned int value;
unsigned char byte[4];
byte[3] = value >> 24;
byte[2] = value >> 16;
byte[1] = value >> 8;
byte[0] = value >> 0;
```

- operazioni veloci di "modulo potenze di 2".

molte strutture dati (buffer, tabelle di hash, ...) hanno dimensioni pari ad una potenza di 2 per poter selezionare velocemente gli elementi usando mascheramenti al posto dell'operatore modulo (%). Infatti `x % (2 ^ n)` equivale a `x & ((2 ^ n) - 1)`.

```
odd = num & 1; // invece di odd = num % 2
elem = index & 255; // invece di elem = index % 256
```

- inverti il valore di v con w:

```
v = v ^ w;
w = v ^ w;
v = v ^ w;
```