```c
/*
 * Il programma determina se l'architettura dell'elaboratore è di tipo big endian o little endian
 *
 * La endianness è una proprietà della piattaforma di esecuzione (processore)
 * essa determina con quale ordine vengono memorizzati i byte di un valore formato da più byte
 * - big-endian:
 *      la memorizzazione inizia dal byte più significativo (ovvero inizia dall'estremità grande, The Big End)
 *      per finire col meno significativo
 * - little-endian:
 *      la memorizzazione inizia dal byte meno significativo (ovvero inizia dall'estremità piccola, The Little End)
 *      per finire col più significativo
 *
 * perché big/little endian? vedi I viaggi di Gulliver!
 *
 */
#include <stdio.h>

int main(int argc, char *argv[])
{
    /* metodo 1 - uso di union*/
    /*
```

```
22        * La definizione di una union è molto simile ad una struct:
          viene usata la parola chiave union invece di struct
23        * con le union tutti i campi sono sovrapposti, ovvero associati
          allo stesso indirizzo di memoria
24        * si riserva spazio in memoria sufficiente solo per il dato più
          grande
25        * fra i vari campi descritti  --> sizeof(un)==4
26        */
27       union {
28           int i;
29           char c[sizeof(int)];
30       } un = { .i = 123 };
31       /*
32       struct {
33           int i;
34           char c[sizeof(int)];
35       } un; --> sizeof(un)==8
36       */
37
38       if (un.c[0] == 123) {
39           printf ("little endian\n");
40       } else {
41           printf ("big endian\n");
42       }
43       /*
44        * little endian: |123| 0 | 0 | 0 |
```

```
45          * big endian:    | 0 | 0 | 0 |123|
46          */
47
48         /* metodo 2 - uso di un puntatore */
49         int i = 123;
50         char *p = (char *) &i;
51         if (*p == 123) {
52             printf ("little endian\n");
53         } else {
54             printf ("big endian\n");
55         }
56         /* metodo 3 - uso semplicemente un cast opportuno - come vedere
           la codifica in bit di un float */
57         float f = 1.0;
58         printf("1.0 --> %x\n", *((int *) &f));
59         return 0;
60     }
61
```