Capitolo 1

Introduzione al linguaggio C

ESEMPIO 1: file `hello.c`

```
1  /*
2   * hello.c
3   *
4   * programma che stampa sul video la frase
5   * "Salve, mondo!"
6   */
7  #include <stdio.h>
8
9  int main()
10 {
11   printf("Salve, Mondo!\n");
12   return 0;
13 }
```

ESEMPIO 2: file `sum.c`

```
1  /*
2   * somma.c
3   *
4   * programma per il calcolo di una somma
5   * stampa sul video del risultato
6   */
7  #include <stdio.h>
8
9  int main()
10 {
11   int a, b, somma;
12
```

```
13    a = 10;
14    b = 12;
15    somma = a + b;
16    printf("La somma e' %i\n", somma);
17
18    return 0;
19  }
```

ESEMPIO 3: file `erone.c`

```
1  /*
2   * espr.c
3   *
4   * calcolo di una semplice espressione
5   */
6  #include <stdio.h>
7  #include <math.h>
8
9  int main()
10 {
11   double c1, c2, ipo, p, area;
12
13   c1 = 3;
14   c2 = 4;
15   ipo = sqrt(c1 * c1 + c2 * c2);
16
17   /* calcolo dell'area con la formula di Erone */
18   p = (c1 + c2 + ipo) / 2;
19   area = sqrt(p * (p - c1) * (p - c2) * (p - ipo));
20
21   printf("Ipotenusa:       %lf\n", ipo);
22   printf("Area (Erone):    %lf\n", area);
23   printf("Area (classica): %lf\n", c1 * c2 / 2);
24
25   return 0;
26 }
```

Capitolo 2

Il preprocessore

Nessun file di esempio disponibile.

Questo capitolo è stato incluso
per mantenere la coerenza
con la numerazione dei capitoli
nel libro di testo.

Capitolo 3

Istruzioni e strutture di controllo

ESEMPIO 4: file `sumpar.c`

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int main()
 5 {
 6         char s[10];
 7         int a, b;
 8
 9         fgets(s, sizeof(s), stdin);
10         a = atoi(s);
11         fgets(s, sizeof(s), stdin);
12         b = atoi(s);
13         printf("%i + %i = %i\n", a, b, a + b);
14         return 0;
15 }
```

ESEMPIO 5: file `oddeven.c`

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int main()
 5 {
 6   char s[80];
 7   int i, j;
 8
 9   fgets(s, sizeof(s), stdin);
10   i = atoi(s);
```

```
11    j = i / 2;
12
13    if (j * 2 == i) {
14       printf("%i: pari\n", i);
15       return 0;
16    }
17    printf("%i: dispari\n", i);
18    return 0;
19 }
20
```

ESEMPIO 6: file `oddeven2.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6    char s[80];
7    int i;
8
9    fgets(s, sizeof(s), stdin);
10   i = atoi(s);
11
12   if (i % 2)
13     printf("%i: dispari\n", i);
14   else
15     printf("%i: pari\n", i);
16
17   return 0;
18 }
19
```

ESEMPIO 7: file `leapyear.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6    char s[80];
7    int a;
8
9    fgets(s, sizeof(s), stdin);
10   a = atoi(s);
11
```

```
12   if (((a % 4 == 0) && !(a % 100 == 0)) || (a % 400 == 0))
13     printf("Il %d e` bisestile.\n", a);
14   else
15     printf("Il %d e` non bisestile.\n", a);
16
17   return 0;
18 }
```

ESEMPIO 8: file `fib.c`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6   char s[80];
7   int f1 = 1;
8   int f2 = 1;
9   int f3, i, n;
10
11   fgets(s, sizeof(s), stdin);
12   n = atoi(s);
13   if (n >= 1)
14     printf("%i\n", f1);
15   if (n >= 2)
16     printf("%i\n", f2);
17   for (i = 3; i <= n; i++) {
18     f3 = f1 + f2;
19     printf("%i\n", f3);
20     f1 = f2;
21     f2 = f3;
22   }
23   return 0;
24 }
```

ESEMPIO 9: file `collatz1.c`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6   char s[80];
7   int i, j = 0;
8
9   fgets(s, sizeof(s), stdin);i
```

```
10   i = atoi(s);
11
12   while (i != 1) {
13     printf("[%2i] %i\n", j, i);
14     if (i % 2)
15       i = 3 * i + 1;
16     else
17       i = i / 2;
18     j = j + 1;
19   }
20   return 0;
21 }
```

ESEMPIO 10: file `collatz2.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6    char s[80];
7    int i, j = 0;
8
9    fgets(s, sizeof(s), stdin);
10   i = atoi(s);
11
12   do {
13     printf("[%2i] %i\n", j, i);
14     if (i % 2)
15       i = 3 * i + 1;
16     else
17       i = i / 2;
18     j++;
19   } while (i != 1);
20   return 0;
21 }
```

ESEMPIO 11: file `calc.c`

```
1  /*
2   * switch.c
3   *
4   * Esempio di utilizzo della struttura switch.
5   */
6
7  #include <stdio.h>
```

```
 8 #include <stdlib.h>
 9
10 int main()
11 {
12   char s[80];
13   double n1, n2, r;
14   int sel;
15   char op;
16
17   puts("Inserire due valori separati da INVIO:");
18   fgets(s, sizeof(s), stdin);
19   n1 = atof(s);
20   fgets(s, sizeof(s), stdin);
21   n2 = atof(s);
22   puts("Operazione: 1) somma            2) sottrazione");
23   puts("            3) moltiplicazione  4) divisione");
24   fgets(s, sizeof(s), stdin);
25   sel = atoi(s);
26
27   switch (sel) {
28     case 1 :
29       r = n1 + n2;
30       op = '+';
31       break;
32     case 2 :
33       r = n1 - n2;
34       op = '-';
35       break;
36     case 3 :
37       r = n1 * n2;
38       op = 'x';
39       break;
40     case 4 :
41       r = n1 / n2;
42       op = ':';
43       break;
44     default:
45       return 1;
46   }
47   printf("%lf %c %lf = %lf\n", n1, op, n2, r);
48
49   return 0;
50 }
```

ESEMPIO 12: file `break_continue.c`

```
1 /*
```

```
 2  * break_continue.c
 3  *
 4  * programma di esempio per l'utilizzo delle istruzioni break e
 5  * continue per il controllo del flusso del programma.
 6  */
 7 #include <stdio.h>
 8 #include <stdlib.h>
 9 int main()
10 {
11   char s[80];
12   int x, y;
13   do {
14     puts("Inserisci x");
15     fgets(s, sizeof(s), stdin);
16     x = atoi(s);
17     if (x < -10) {
18       printf("  x = %d < -10: break...\n", x);
19       break;
20     }
21     if (x > 10) {
22       printf("  x = %d > 10: continue\n", x);
23       continue;
24     }
25     while (1) {
26       puts("Inserisci y");
27       gets(s);
28       y = atoi(s);
29       if (y < -10) {
30         printf("  y = %d < -10: break...\n", y);
31         break;
32       }
33       if (y > 10) {
34         printf("  y = %d > 10: continue\n", y);
35         continue;
36       }
37       printf("  valore immesso: y = %d\n", y);
38     }
39     printf("  valore immesso: x = %d\n", x);
40   } while (1);
41   return 0;
42 }
```

ESEMPIO 13: file `primes.c`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
```

```
 4 int main()
 5 {
 6   char s[80];
 7   int i, max, nut;   /* nut = number under test */
 8
 9   fgets(s, sizeof(s), stdin);
10   max = atoi(s);
11   for (nut = 2; nut < max; nut = nut + 1) {
12     for (i = 2; i < nut; i = i + 1) {
13       if (nut % i == 0)
14         break;
15     }
16     if (i == nut)
17       printf("%i\n", nut);
18   }
19   return 0;
20 }
```

ESEMPIO 14: file `goto.c`

```
 1 /*
 2  * goto.c
 3  *
 4  * Programma di esempio per il costrutto goto.
 5  *
 6  */
 7
 8 #include <stdio.h>
 9 #include <stdlib.h>
10 #include <math.h>
11
12 int main()
13 {
14   char s[80];
15   int val = 0;
16
17   while (1) {
18     while (1) {
19       printf("\nvalore: ");
20       fgets(s, sizeof(s), stdin);
21       val = atoi(s);
22       if (val < 0) goto uscita;
23
24       printf("\nsqrt(%d) = %lf\n", val, sqrt(val));
25     }
26   }
27
```

```
28    uscita:
29    if (val < 0) {
30      printf("\nInserito valore non valido.\n");
31      return 1;
32    }
33
34    return 0;
35  }
```

Capitolo 4

I tipi di dati

ESEMPIO 15: file `ascii.c`

```
1 #include <stdio.h>
2 int main()
3 {
4   int i;
5   for (i = 0; i < 255; i++)
6     printf("%02x %3i '%c'\n", i, i, i);
7   return 0;
8 }
```

ESEMPIO 16: file `fib_rev.c`

```
1 /*
2  * fibonacci_rev.c
3  *
4  * Programma che stampa i primi N numeri della
5  * sequenza di Fibonacci in ordine inverso.
6  *
7  * E` necessario prima memorizzare i valori
8  * in un vettore, per poi poterli stampare, dal
9  * momento che la sequenza di Fibonacci e`
10 * definita in senso solo crescente.
11 *
12 * Si presti attenzione agli indici e ai test
13 * utilizzati nei cicli for.
14 */
15
16 #include <stdio.h>
17 #include <stdlib.h>
```

```
18
19 #define MAXVAL 20
20
21 int main()
22 {
23   char s[80];
24   int num[MAXVAL];
25   int i, max;
26
27   fgets(s, sizeof(s), stdin);
28   max = atoi(s);
29   if (max > MAXVAL)
30     max = MAXVAL;
31
32   num[0] = 1;
33   num[1] = 1;
34
35   for (i = 2; i < max; i++) {
36     num[i] = num[i - 1] + num[i - 2];
37   }
38
39   for (i = max - 1; i >= 0; i--) {
40     printf("%d\n", num[i]);
41   }
42
43   return 0;
44 }
```

ESEMPIO 17: file `strings.c`

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define DIM 20
5
6 int main()
7 {
8   char nome[DIM], cognome[DIM], s[DIM];
9
10  /* Inizializzazione del contenuto delle stringhe */
11  strcpy(s, "");
12  strcpy(nome, "Albert");
13  strcpy(cognome, "Einstein");
14
15  puts(s);
16  puts(nome);
17  puts(cognome);
```

```
18
19    /* Composizione della stringa complessiva */
20    strcat(s, nome);
21    strcat(s, " ");
22    strcat(s, cognome);
23
24    /* Stampa della stringa */
25    puts(s);
26
27    /* Calcolo e stampa della sua lunghezza */
28    printf("Lunghezza: %zu\n", strlen(s));
29
30    /* Identificazione di una sottostringa */
31    puts(strstr(s, "in"));
32
33    /* Identificazione di un carattere */
34    puts(strchr(s, 'e'));
35    puts(strrchr(s, 'e'));
36
37    return 0;
38 }
```

Capitolo 5

Le funzioni

ESEMPIO 18: file `hypotenuse.c`

```
1  /*
2   * ipotenusa.c
3   *
4   * programma di esempio per la dichiarazione e definizione di
5   * funzioni.
6   */
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <math.h>
10
11 double leggi_double(void);
12 double quad(double a);
13 double ipot(double a, double b);
14
15 int main()
16 {
17   double a, b;
18
19   a = leggi_double();
20   b = leggi_double();
21   printf("Ipotenusa: %lf\n", ipot(a, b));
22
23   return 0;
24 }
25
26 /*
27  * Richiede un valore all'utente e lo restituisce
28  * convertito in double.
29  */
```

```
30 double leggi_double(void)
31 {
32   char buf[100];
33
34   fgets(buf, sizeof(buf), stdin);
35   return atof(buf);
36 }
37
38 double quad(double a)
39 {
40   return a * a;
41 }
42
43 /*
44  * La funzione calcola e restituisce l'iponenusa
45  * di un triangolo rettangolo i cui cateti sono a e b.
46  */
47 double ipot(double a, double b)
48 {
49   return sqrt(quad(a) + quad(b));
50 }
```

ESEMPIO 19: file `swap.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void swap_errato(int a, int b)
5  {
6    int tmp;
7
8    tmp = a;
9    a = b;
10   b = tmp;
11 }
12
13 void swap(int *a, int *b)
14 {
15   int tmp;
16
17   tmp = *a;
18   *a = *b;
19   *b = tmp;
20 }
21
22 int main()
23 {
```

```
24   char buf[100];
25   int n1, n2;
26
27   fgets(buf, sizeof(buf), stdin);
28   n1 = atoi(buf);
29   fgets(buf, sizeof(buf), stdin);
30   n2 = atoi(buf);
31
32   printf("n1 = %i -- n2 = %i\n", n1, n2);
33   swap(&n1, &n2);
34   printf("n1 = %i -- n2 = %i\n", n1, n2);
35   swap(&n1, &n2);
36   printf("n1 = %i -- n2 = %i\n", n1, n2);
37
38   /* errore in compilazione: passati puntatori invece di int */
39 //  swap_errato(&n1, &n2);
40   swap_errato(n1, n2);
41   printf("n1 = %i -- n2 = %i (errato)\n", n1, n2);
42
43   return 0;
44 }
```

ESEMPIO 20: file `chist.c`

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4
5  #define N ('z' - 'a' + 1)
6
7  int chist(const char testo[], int *h)
8  {
9    int len;
10   int i;
11
12   len = strlen(testo);
13   for (i = 0; i < len; i++) {
14     if (isalpha(testo[i])) {
15       h[toupper(testo[i]) - 'A']++;
16     }
17   }
18
19   return len;
20 }
21
22 int main()
23 {
```

```
24    int h[N] = { 0 };
25    char testo[80];
26    int i;
27
28    while (fgets(testo, sizeof(testo), stdin)) {
29      chist(testo, h);
30    }
31    for (i = 0; i < N; i++) {
32      printf("Cont(%c) = %d\n", 'A' + i, h[i]);
33    }
34
35    return 0;
36  }
```

ESEMPIO 21: file `printarg.c`

```
1  /*
2   * printarg.c
3   *
4   * Programma di esempio per l'accesso ai dati
5   * passati attraverso la linea di comando di
6   * un programma.
7   */
8
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 int main(int argc, char *argv[])
13 {
14   int i;
15
16   for (i = 0; i < argc; i = i + 1)
17     printf("arg %i/%i: \"%s\"\n", i, argc, argv[i]);
18   return 0;
19 }
```

ESEMPIO 22: file `dice.c`

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int lancio(void);
5
6  int main(int argc, char *argv[])
7  {
8    int seme = 0;
```

```
 9
10   if (argc >= 2)
11     seme = atoi(argv[1]);
12   srand(seme);
13   printf("%d %d\n", lancio(), lancio());
14
15   return 0;
16 }
17
18 int lancio(void)
19 {
20   return (1 + rand() % 6);
21 }
```

ESEMPIO 23: file `drand.c`

```
 1 /*
 2  * drand.c
 3  *
 4  * Esempio di utilizzo della struttura switch
 5  * per impostare i valori di parametri inseriti
 6  * dalla linea di comando.
 7  */
 8
 9 #include <stdio.h>
10 #include <stdlib.h>
11
12 double drand(double min, double max)
13 {
14   return (min + (max - min) *
15                 ((double)rand() / (double)(RAND_MAX)));
16 }
17
18 int main(int argc, char *argv[])
19 {
20   double a = 0.0, b = 1.0, seed = 1, c;
21
22   switch(argc) {
23     case 1 :
24       /* in assenza di parametri usa i valori di default */
25       break;
26     case 4 :
27       seed = atoi(argv[3]);
28       /* no break */
29     case 3 :
30       b = atof(argv[2]);
31       /* no break */
```

```
32      case 2 :
33        a = atof(argv[1]);
34        break;
35      default :   /* massimo 3 parametri ammessi */
36        printf("Uso: drand [ a ] [ b ] [ seed ]\n");
37        return 1;
38    }
39
40    srand(seed);
41    c = drand(a, b);
42    printf("%lf in [%lf,%lf]\n", c, a, b);
43
44    return 0;
45  }
```

ESEMPIO 24: file `fibonacci.c`

```
1  /*
2   * fibonacci.c
3   *
4   * Calcolo dell'i-esimo numero di Fibonacci
5   * usando una funzione ricorsiva.
6   *
7   * Il valore di i viene letto da linea di comando.
8   */
9
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 long fibonacci(long n)
14 {
15   if (n <= 1) return n;
16   return fibonacci(n - 1) + fibonacci(n - 2);
17 }
18
19 int main(int argc, char *argv[])
20 {
21   long n;
22
23   if (argc != 2) {
24     printf("numero di parametri errato\n");
25     return 1;
26   }
27
28   if ((n = atoi(argv[1])) > 0)
29     printf("fibonacci(%ld) = %ld\n", n, fibonacci(n));
30
```

```
31   return 0;
32 }
```

ESEMPIO 25: file `fibonacci_nr.c`

```
 1 /*
 2  * fibonacci_nr.c
 3  *
 4  * programma che stampa l'i-esimo numero di Fibonacci
 5  * in versione non ricorsiva
 6  */
 7
 8 #include <stdio.h>
 9 #include <stdlib.h>
10
11 long fibonacci_nr(long n)
12 {
13   long prev = -1;
14   long result = 1;
15   long sum;
16   int i;
17
18   for (i = 0; i <= n; ++i) {
19     sum = result + prev;
20     prev = result;
21     result = sum;
22   }
23
24   return result;
25 }
26
27 int main(int argc, char *argv[])
28 {
29   long n;
30
31   if (argc != 2) {
32     printf("numero di parametri errato\n");
33     return 1;
34   }
35
36   if ((n = atoi(argv[1])) > 0)
37     printf("fibonacci(%ld) = %ld\n", n, fibonacci_nr(n));
38
39   return 0;
40 }
```

ESEMPIO 26: file `strtol_man_gcc.c`

```
1  #include <stdlib.h>
2  #include <limits.h>
3  #include <stdio.h>
4  #include <errno.h>
5
6  int main(int argc, char *argv[])
7  {
8    int base;
9    char *endptr, *str;
10   long val;
11
12   if (argc < 2) {
13     printf("Uso: %s str [base]\n", argv[0]);
14     return EXIT_FAILURE;
15   }
16
17   str = argv[1];
18   base = (argc > 2) ? atoi(argv[2]) : 10;
19
20   /* per distinguere tra successo e fallimento dopo la chiamata */
21   errno = 0;
22   val = strtol(str, &endptr, base);
23
24   /* controllo su vari tipi di errore */
25   if ((errno == ERANGE && (val == LONG_MAX || val == LONG_MIN))
26       || (errno != 0 && val == 0)) {
27     perror("strtol");
28     return EXIT_FAILURE;
29   }
30
31   if (endptr == str) {
32     printf("Nessuna cifra trovata!\n");
33     return EXIT_FAILURE;
34   }
35
36   /*
37    * a questo punto strtol() ha convertito il
38    * numero con successo
39    */
40   printf("strtol() ha ritornato %ld\n", val);
41
42   if (*endptr != '\0')   /* non necessariamente un errore... */
43     printf("Altri caratteri dopo il numero: %s\n", endptr);
44
```

```
45   return EXIT_SUCCESS;
46 }
```

ESEMPIO 27: file `day.c`

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int days(int d, int m)
 5 {
 6   int dxm[] = {31, 28, 31, 30, 31, 30,
 7                31, 31, 30, 31, 30, 31};
 8   int i, days;
 9
10   days = d;
11   for (i = 0; i < m - 1; i = i + 1) {
12     days = days + dxm[i];
13   }
14   return days;
15 }
16
17 int main(int argc, char *argv[])
18 {
19   int d, m;
20
21   if (argc < 3)
22     return 1;
23
24   d = atoi(argv[1]);
25   m = atoi(argv[2]);
26   printf("Numero di giorni: %i\n", days(d, m));
27   return 0;
28 }
```

Capitolo 6

I tipi di dati derivati

ESEMPIO 28: file `scalar_prod.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void stampa_vett(double v[], int n)
5  {
6    int i;
7
8    for (i = 0; i < n; i++)
9      printf("%lf ", v[i]);
10   putchar('\n');
11
12 }
13
14 /*
15  * In questo esempio ci basta generare un vettore
16  * di numeri casuali abbastanza piccoli da poter
17  * verificare facilmente i calcoli.
18  */
19 void leggi_valori(double v[], int n)
20 {
21   int i;
22
23   for (i = 0; i < n; i++)
24     v[i] = rand() % 10;
25
26 }
27
28 double prodotto_scalare(int n)
29 {
```

```
30   double a[n], b[n], p;
31   int i;
32
33   leggi_valori(a, n);
34   leggi_valori(b, n);
35   stampa_vett(a, n);
36   stampa_vett(b, n);
37
38   for (i = 0; i < n; i++)
39     p += a[i] * b[i];
40
41   return p;
42 }
43
44 int main(int argc, char *argv[])
45 {
46   if (argc != 2)
47     return 1;
48
49   printf("%lf\n", prodotto_scalare(atoi(argv[1])));
50
51   return 0;
52 }
```

ESEMPIO 29: file `datecmp.c`

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  struct data {
5    int g;
6    int m;
7    int a;
8  };
9
10 int datecmp(const struct data *d1, const struct data *d2)
11 {
12   int ret;
13   if (!(ret = (d1->a - d2->a)))
14     if (!(ret = (d1->m - d2->m)))
15       ret = (d1->g - d2->g);
16
17   return ret;
18 }
19
20 int main(int argc, char *argv[])
21 {
```

```
22    struct data d1, d2;
23    int n1, n2, cmp;
24
25    if (argc != 3) {
26      printf("Numero di parametri errato.\n");
27      return -1;
28    }
29
30    n1 = sscanf(argv[1], "%d-%d-%d", &d1.a, &d1.m, &d1.g);
31    if (n1 != 3) {
32      printf("Conversione data 1 errata\n");
33      return -1;
34    }
35    n2 = sscanf(argv[2], "%d-%d-%d", &d2.a, &d2.m, &d2.g);
36    if (n2 != 3) {
37      printf("Conversione data 2 errata\n");
38      return -1;
39    }
40
41    cmp = datecmp(&d1, &d2);
42
43    if (cmp < 0) {
44      printf("%s < %s\n", argv[1], argv[2]);
45    } else if (cmp == 0) {
46      printf("%s = %s\n", argv[1], argv[2]);
47    } else {
48      printf("%s > %s\n", argv[1], argv[2]);
49    }
50
51    return 0;
52  }
```

ESEMPIO 30: file `typedef_define.c`

```
1  /*
2   * typedef_define.c
3   *
4   * esempi di utilizzo di typedef e define
5   */
6  #include <stdio.h>
7
8  #define T_CHAR_D char
9  #define T_CHARP_D char *
10
11 typedef char t_char_t;
12 typedef char * t_charp_t;
13
```

```
14 typedef char t_char_vect[];
15
16 int main()
17 {
18   t_char_vect v = "la mia stringa\n";
19
20   /* due dichiarazioni equivalenti */
21   t_char_t n2;
22   T_CHAR_D n1;
23
24   //unsigned t_char_t n3;  /* questo non si  p u  fare!!! */
25   unsigned T_CHAR_D n4;
26
27   t_charp_t p1, p2, p3;   /* sono tutti puntatori a char */
28   T_CHARP_D p4, p5, p6;   /* solo il primo    un puntatore! */
29
30   printf("%ld %ld %ld\n", sizeof(p1), sizeof(p2), sizeof(p3));
31   printf("%ld %ld %ld\n", sizeof(p4), sizeof(p5), sizeof(p6));
32
33   return 0;
34 }
```

Capitolo 7

I file

ESEMPIO 31: file `falign.c`

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 void falign(FILE * infile, FILE * outfile, int maxcol)
 5 {
 6   char ch;
 7   int col = 0;
 8
 9   while ((ch = getc(infile)) != EOF) {
10     if (ch == '\n')
11       putc(' ', outfile);
12     else
13       putc(ch, outfile);
14
15     ++col;
16     if (col >= maxcol) {
17       putc('\n', outfile);
18       col = 0;
19     }
20   }
21   putc('\n', outfile);
22 }
23
24 int main(int argc, char *argv[])
25 {
26   FILE *fp;
27   int maxcol;
28
29   if (argc != 3) {
```

```
30      printf("Uso: falign file colonne\n");
31      return 1;
32    }
33
34    maxcol = atoi(argv[2]);
35    if (!(fp = fopen(argv[1], "r"))) {
36      printf("Errore nell'apertura del file %s\n", argv[1]);
37      return 1;
38    }
39
40    int i;
41    for (i = 1; i <= maxcol; i++)
42            printf("%d", i % 10);
43    putc('\n', stdout);
44
45    falign(fp, stdout, maxcol);
46    fclose(fp);
47
48    return 0;
49  }
```

ESEMPIO 32: file `split_file.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define FNAME "outfile"
6  #define MAX_DIM 0xffff
7
8  int split_file(FILE *fin, int dim)
9  {
10   size_t nread;
11   int count = 0;
12   char fname[30];
13   char buffer[MAX_DIM];
14   FILE *fout;
15
16   if (dim <= 0 || dim > MAX_DIM)
17     return -1;
18   do {
19     nread = fread(buffer, 1, dim, fin);
20     if (nread > 0) {
21       sprintf(fname, "%s%d%s", FNAME, count, ".split");
22       if (!(fout = fopen(fname, "w"))) {
23         return -1;
24       }
```

```
25        fwrite(buffer, nread, 1, fout);
26        fclose(fout);
27        count++;
28      }
29    } while (nread == dim);
30    return 0;
31  }
32
33  int main(int argc, char *argv[])
34  {
35    FILE *fin;
36    int dim;
37
38    if (argc != 3) {
39      printf("Utilizzo: split_file infile dimensione\n");
40      return -1;
41    }
42
43    if (!(fin = fopen(argv[1], "r"))) {
44      printf("Impossibile aprire il file di input.\n");
45      return -1;
46    }
47    dim = atoi(argv[2]);
48
49    return split_file(fin, dim);
50  }
```

ESEMPIO 33: file `pgm_fgets.c`

```
1  /*
2   * pgm_fgets_1.c
3   * Lettura immagine formato PGM con fgets.
4   * Assunzioni:
5   *  - al piu` 10 numeri per riga
6   */
7
8  #include <stdio.h>
9
10 #define MAX_LEN         (1024)
11 #define MAX_LEVEL       (256)
12
13 struct immagine {
14   int magic_number;              // numero magico
15   int max_gray_level;            // massimo livello di grigio
16   int larg, alt;                 // larghezza e altezza dell'immagine [pixel]
17   int istogramma[MAX_LEVEL];     // vettore contenente l'istogramma
18 };
```

```
19
20 int pgm_istogramma(struct immagine *pgm, FILE *infile)
21 {
22   int n, i;
23   char buf[MAX_LEN];
24   int v[10];
25
26   for (i = 0; i < MAX_LEVEL; i++)
27     pgm->istogramma[i] = 0;
28
29   /* lettura e parsing della riga contenente il magic number */
30   fgets(buf, sizeof(buf), infile);
31   n = sscanf(buf, "P%i", &pgm->magic_number);
32   if ((n != 1) || (pgm->magic_number != 2)) return 1;
33
34   /* lettura e parsing della riga contenente le dimensioni */
35   fgets(buf, sizeof(buf), infile);
36   if (sscanf(buf, "%i %i", &pgm->larg, &pgm->alt) != 2) return 1;
37
38   /* lettura e parsing della riga contenente i livelli di grigio */
39   fgets(buf, sizeof(buf), infile);
40   if (sscanf(buf, "%i", &pgm->max_gray_level) != 1) return 1;
41
42   /* lettura e parsing del resto del file */
43   while (fgets(buf, sizeof(buf), infile)) {
44
45     /* usa variabili temporanee per evitare
46      * di scrivere oltre la dimensione della matrice
47      * se mancassero meno di 10 pixel da leggere */
48     n = sscanf(buf, "%i %i %i %i %i %i %i %i %i %i",
49                &v[0], &v[1], &v[2], &v[3], &v[4],
50                &v[5], &v[6], &v[7], &v[8], &v[9]);
51
52     /* copia nel vettore dei pixel soltanto i valori
53      * effettivamente convertiti */
54     for (i = 0; i < n; i++)
55       (pgm->istogramma[v[i]])++;
56   }
57
58   return 0;
59 }
60
61 void print_istogramma(struct immagine *pgm, FILE *outfile)
62 {
63   int livello;
64
65   for (livello = 0; livello <= pgm->max_gray_level; livello++)
```

```
66      fprintf(outfile, "%3i: %i\n", livello, pgm->istogramma[livello]);
67  }
68
69  int main(int argc, char *argv[])
70  {
71    struct immagine pgm;
72    FILE *infile;
73
74    if (argc != 2) {
75      printf("uso: %s <nome_file_pgm>\n", argv[0]);
76      return 1;
77    }
78    if (!(infile = fopen(argv[1], "r"))) {
79      printf("Errore nell'apertura di %s\n", argv[1]);
80      return 1;
81    }
82    if (pgm_istogramma(&pgm, infile) != 0) {
83      puts("Errore nella lettura del file.");
84      return 1;
85    }
86
87    print_istogramma(&pgm, stdout);
88
89    return 0;
90  }
```

ESEMPIO 34: file `parse_sscanf.c`

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    const char linea[] = "1975/03/03 06:00:00;info1;info2;info3";
6    const char *ptr = linea;
7    char campo[32];
8    int n;
9
10   while (sscanf(ptr, "%31[^;]%n", campo, &n) == 1) {
11     printf("campo = \"%s\"\n", campo);
12     ptr += n; /* aumenta ptr del numero di caratteri letti */
13     if (*ptr != ';') {
14       break; /* delimitatore non trovato; fine! */
15     }
16     ++ptr; /* salta il delimitatore */
17   }
18   return 0;
19 }
```

ESEMPIO 35: file `pgm_fgets_parser.c`

```
1  /*
2   * pgm_fgets_parser.c
3   * Lettura immagine formato PGM con fgets.
4   * Assunzioni:
5   *  - massima lunghezza della riga pari a 1024 caratteri
6   *  - nessun commento presente
7   */
8
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 #define MAX_LEN   (1024)
13 #define MAX_LEVEL (256)
14
15 struct immagine {
16   int magic_number;             // numero magico
17   int max_gray_level;           // massimo livello di grigio
18   int larg, alt;                // larghezza e altezza dell'immagine [pixel]
19   int istogramma[MAX_LEVEL];    // vettore contenente l'istogramma
20 };
21
22 /*
23  * Attenzione che buf deve essere un puntatore, non buf[], in quanto
24  * deve poter essere modificato.
25  */
26 void parsing_riga(char *buf, struct immagine *pgm)
27 {
28   char pixel[32];
29   int n;
30
31   while (sscanf(buf, "%31[^ ]%n", pixel, &n) == 1) {
32     (pgm->istogramma[atoi(pixel)])++;
33     buf += n;  /* aumenta ptr del numero di caratteri letti */
34     if (*buf != ' ')
35       break;   /* delimitatore non trovato; fine! */
36     while (*buf == ' ' || *buf == '\n')
37       ++buf;   /* salta il delimitatore e i successivi*/
38   }
39 }
40
41 int pgm_istogramma(struct immagine *pgm, FILE *infile)
42 {
43   int n, i;
44   char buf[MAX_LEN];
```

```
45
46    for (i = 0; i < MAX_LEVEL; i++)
47      pgm->istogramma[i] = 0;
48
49    /* lettura e parsing della riga contenente il magic number */
50    fgets(buf, sizeof(buf), infile);
51    n = sscanf(buf, "P%i", &pgm->magic_number);
52    if ((n != 1) || (pgm->magic_number != 2)) return 1;
53
54    /* lettura e parsing della riga contenente le dimensioni */
55    fgets(buf, sizeof(buf), infile);
56    if (sscanf(buf, "%i %i", &pgm->larg, &pgm->alt) != 2) return 1;
57
58    /* lettura e parsing della riga contenente i livelli di grigio */
59    fgets(buf, sizeof(buf), infile);
60    if (sscanf(buf, "%i", &pgm->max_gray_level) != 1) return 1;
61
62    /* lettura e parsing del resto del file */
63    while (fgets(buf, sizeof(buf), infile)) {
64      parsing_riga(buf, pgm);
65    }
66
67    return 0;
68  }
69
70  void print_istogramma(struct immagine *pgm, FILE *outfile)
71  {
72    int livello;
73
74    for (livello = 0; livello < pgm->max_gray_level; livello++)
75      fprintf(outfile, "%3i: %i\n", livello, pgm->istogramma[livello]);
76  }
77
78  int main(int argc, char *argv[])
79  {
80    struct immagine pgm;
81    FILE *infile;
82
83    if (argc != 2) {
84      printf("uso: %s <nome_file_pgm>\n", argv[0]);
85      return 1;
86    }
87    if (!(infile = fopen(argv[1], "r"))) {
88      printf("Errore nell'apertura di %s\n", argv[1]);
89      return 1;
90    }
91    if (pgm_istogramma(&pgm, infile) != 0) {
```

```
92      puts("Errore nella lettura del file.");
93      return 1;
94    }
95
96    print_istogramma(&pgm, stdout);
97
98    return 0;
99  }
```

ESEMPIO 36: file `pgm_fscanf.c`

```
1  /*
2   * pgm_fscanf_1.c
3   * Lettura immagine formato PGM senza commenti
4   * utilizzando fscanf.
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10
11 #define MAX_LEN                 (1024)
12 #define MAX_LEVEL               (256)
13
14 struct immagine {
15   int magic_number;             // numero magico
16   int max_gray_level;           // massimo livello di grigio
17   int larg, alt;                // larghezza e altezza dell'immagine [pixel]
18   int istogramma[MAX_LEVEL];    // vettore contenente l'istogramma
19 };
20
21 int pgm_istogramma(struct immagine *pgm, FILE *infile)
22 {
23   int n, i;
24   int val;
25
26   for (i = 0; i < MAX_LEVEL; i++)
27     pgm->istogramma[i] = 0;
28
29   /* lettura e parsing della riga contenente il magic number */
30   n = fscanf(infile, "P%i", &pgm->magic_number);
31   if ((n != 1) || (pgm->magic_number != 2))
32     return 1;
33
34   /* lettura e parsing della riga contenente le dimensioni */
35   if (fscanf(infile, "%i %i", &pgm->larg, &pgm->alt) != 2)
36     return 1;
```

```
37
38    /* lettura e parsing della riga contenente i livelli di grigio */
39    if (fscanf(infile, "%i", &pgm->max_gray_level) != 1)
40      return 1;
41
42    /* lettura e parsing del resto del file */
43    while (fscanf(infile, "%i", &val) != EOF) {
44      pgm->istogramma[val]++;
45    }
46
47    return 0;
48  }
49
50  void print_istogramma(struct immagine *pgm, FILE * outfile)
51  {
52    int livello;
53
54    for (livello = 0; livello <= pgm->max_gray_level; livello++)
55      fprintf(outfile, "%3i: %i\n", livello, pgm->istogramma[livello]);
56  }
57
58  int main(int argc, char *argv[])
59  {
60    struct immagine pgm;
61    FILE *infile;
62
63    infile = stdin;
64    switch (argc) {
65    case 2:
66      if (!(infile = fopen(argv[1], "r"))) {
67        printf("Errore nell'apertura di %s\n", argv[1]);
68        return 1;
69      }
70      break;
71    case 1:
72      printf("uso: %s <nome_file_pgm>\n", argv[0]);
73      return 1;
74    default:
75      puts("Numero di parametri eccessivo.");
76      return 1;
77    }
78
79    if (pgm_istogramma(&pgm, infile) != 0) {
80      puts("Errore nella lettura del file.");
81      return 1;
82    }
83
```

```
84   print_istogramma(&pgm, stdout);
85
86   return 0;
87 }
```

ESEMPIO 37: file `buf_scanf.c`

```
1  /*
2   * buf_scanf.c
3   *
4   * Esempio di programma che effettua letture con
5   * la scanf. Si provino varie combinazioni di
6   * inserimento, separando gli elementi con spazi
7   * o andate a capo, e inserendo per ciascuna istruzione
8   * piu` o meno del numero di elementi atteso.
9   */
10
11 #include <stdio.h>
12
13 int main()
14 {
15   int n1, n2, n3, n4, n5, n6;
16
17   scanf("%d %d %d", &n1, &n2, &n3);
18   printf("valori inseriti: %d %d %d\n", n1, n2, n3);
19
20   scanf("%d %d %d", &n4, &n5, &n6);
21   printf("valori inseriti: %d %d %d\n", n4, n5, n6);
22
23   return 0;
24 }
```

ESEMPIO 38: file `buf_fgets.c`

```
1  /*
2   * buf_fgets.c
3   *
4   * Esempio di programma che effettua letture
5   * con la fgets. Si provino varie combinazioni di
6   * inserimento, separando gli elementi con spazi o
7   * andate a capo, e inserendo per ciascuna istruzione
8   * piu` o meno del numero di elementi atteso.
9   */
10
11 #include <stdio.h>
12
```

```
13 int main()
14 {
15   char s[100];
16   int n1, n2, n3, n4, n5, n6;
17
18   fgets(s, sizeof(s), stdin);
19   sscanf(s, "%d %d %d", &n1, &n2, &n3);
20   printf("valori inseriti: %d %d %d\n", n1, n2, n3);
21
22   fgets(s, sizeof(s), stdin);
23   sscanf(s, "%d %d %d", &n4, &n5, &n6);
24   printf("valori inseriti: %d %d %d\n", n4, n5, n6);
25
26   return 0;
27 }
```

Capitolo 8

I puntatori

ESEMPIO 39: file `void_ptr.c`

```
1  #include <stdio.h>
2
3  void stampa_bit(void *ptr)
4  {
5    int i, x;
6
7    for (i = 0; i < 32; i++) {
8      x = (*(int *)(ptr) >> i) & 0x01;
9      printf("%d", x);
10   }
11   printf("\n");
12 }
13
14 int main()
15 {
16   int a = 10;
17   float b = 3.14;
18   void *ptr;   /* dichiarazione del puntatore a void */
19
20   /* assegna al void pointer l'indirizzo dell'int */
21   ptr = &a;
22   printf("Valore intero 0x%p = %d\n", ptr, *((int*)(ptr)));
23   stampa_bit(ptr);
24
25   /* assegna al void pointer l'indirizzo del float */
26   ptr = &b;
27   printf("Valore float  0x%p = %f\n", ptr, *((float*)(ptr)));
28   stampa_bit(ptr);
29   return 0;
```

```
30 }
```

ESEMPIO 40: file `vect_ptr.c`

```
1  /*
2   * vett_ptr.c
3   *
4   * Esempi di utilizzo dei puntatori e dualita`
5   * tra i vettori e i puntatori stessi.
6   */
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <math.h>
10
11 #define N 10
12
13 int main()
14 {
15   double v[N];
16   double *p;
17   int i;
18
19   p = v;
20
21   for (i = 0; i < N; i++)
22     *(p + i) = sqrt(i);
23
24   for (i = 0; i < N; i++)
25     printf("%4.2f ", v[i]);
26   printf("\n");
27
28   /* si accede al vettore sommando un offset */
29   p = v + (N / 2);
30
31   /* si usano le parentesi quadre in abbinamento al puntatore */
32   for (i = 0; i < N / 2; i++)
33     printf("%4.2f ", p[i]);
34   printf("\n");
35
36   /* errore: vett e' una costante! */
37   //v = v + 1;
38
39   return 0;
40 }
```

ESEMPIO 41: file `matr_print.c`

```c
/*
 * matrice_stampa.c
 *
 * effettua l'inizializzazione di
 * due matrici di dimensione prefissata
 * con valori generati casualmente in un range
 * e le stampa a video
 * definisce due funzioni che ricevono come parametro
 * l'indirizzo del primo elemento della matrice
 * dichiarato utilizzando le due notazioni alternative
 */

#include <stdio.h>
#include <stdlib.h>

#define NROW1 2
#define NROW2 5
#define NCOL 4
#define MINRANGE 1
#define MAXRANGE 10

double randn(double min, double max)
{
  return (((double)rand() / (double)RAND_MAX) * (max - min)) + min;
}

void init_mat(double m[][NCOL], int r, int c)
{
  int i, j;

  for (i = 0; i < r; i++)
    for (j = 0; j < c; j++)
      m[i][j] = randn(MINRANGE, MAXRANGE);
}

void print_mat(double (*m)[NCOL], int r, int c)
{
  int i, j;

  printf("\n");
  for (i = 0; i < r; i++) {
    for (j = 0; j < c; j++) {
      printf("%6.2lf ", m[i][j]);
    }
    printf("\n");
  }
  printf("\n");
```

```
48 }
49
50 int main()
51 {
52   double matA[NROW1][NCOL], matB[NROW2][NCOL];
53
54   init_mat(matA, NROW1, NCOL);
55   init_mat(matB, NROW2, NCOL);
56   print_mat(matA, NROW1, NCOL);
57   print_mat(matB, NROW2, NCOL);
58   return 0;
59 }
```

ESEMPIO 42: file `day2.c`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6   int dxm[] = {31, 28, 31, 30, 31, 30,
7                31, 31, 30, 31, 30, 31};
8   char *mese[] = {"gennaio", "febbraio", "marzo", "aprile",
9      "maggio", "giugno", "luglio", "agosto", "settembre",
10     "ottobre", "novembre", "dicembre"};
11  char st[10];
12  int m;
13
14  fgets(st, sizeof(st), stdin);
15  m = atoi(st);
16  printf("Il mese di %s e` composto da %i giorni\n",
17         mese[m - 1], dxm[m - 1]);
18  return 0;
19 }
```

ESEMPIO 43: file `pgm_fgets_malloc.c`

```
1 /*
2  * pgm_fgets_malloc.c
3  * Lettura immagine formato PGM con fgets.
4  * Assunzioni:
5  *  - al piu` 10 numeri per riga
6  *  - numero massimo di livelli di grigio non noto a priori
7  */
8
9 #include <stdio.h>
```

```
10 #include <stdlib.h>
11
12 #define MAX_LEN         (1024)
13
14 struct immagine {
15   int magic_number;        // numero magico
16   int max_gray_level;      // massimo livello di grigio
17   int larg, alt;           // larghezza e altezza dell'immagine [pixel]
18   int *istogramma;         // vettore contenente l'istogramma
19 };
20
21 int pgm_istogramma(struct immagine *pgm, FILE *infile)
22 {
23   int n, i;
24   char buf[MAX_LEN];
25   int v[10];
26
27   /* lettura e parsing della riga contenente il magic number */
28   fgets(buf, sizeof(buf), infile);
29   n = sscanf(buf, "P%i", &pgm->magic_number);
30   if ((n != 1) || (pgm->magic_number != 2)) return 1;
31
32   /* lettura e parsing della riga contenente le dimensioni */
33   fgets(buf, sizeof(buf), infile);
34   if (sscanf(buf, "%i %i", &pgm->larg, &pgm->alt) != 2) return 1;
35
36   /* lettura e parsing della riga contenente i livelli di grigio */
37   fgets(buf, sizeof(buf), infile);
38   if (sscanf(buf, "%i", &pgm->max_gray_level) != 1) return 1;
39
40   /* alloca il vettore per contenere l'istogramma */
41   pgm->istogramma = malloc(pgm->max_gray_level * sizeof(*(pgm->istogramma)));
42   if (pgm->istogramma == NULL)
43     return 1;
44
45   /* azzera i valori dell'istogramma */
46   for (i = 0; i < pgm->max_gray_level; i++)
47     pgm->istogramma[i] = 0;
48
49   /* lettura e parsing del resto del file */
50   while (fgets(buf, sizeof(buf), infile)) {
51
52     /* usa variabili temporanee per evitare
53      * di scrivere oltre la dimensione della matrice
54      * se mancassero meno di 10 pixel da leggere */
55     n = sscanf(buf, "%i %i %i %i %i %i %i %i %i %i",
56                &v[0], &v[1], &v[2], &v[3], &v[4],
```

```
57                &v[5], &v[6], &v[7], &v[8], &v[9]);
58
59      /* copia nel vettore dei pixel soltanto i valori
60       * effettivamente convertiti */
61      for (i = 0; i < n; i++)
62        (pgm->istogramma[v[i]])++;
63    }
64
65    return 0;
66  }
67
68  void print_istogramma(struct immagine *pgm, FILE *outfile)
69  {
70    int livello;
71
72    for (livello = 0; livello <= pgm->max_gray_level; livello++)
73      fprintf(outfile, "%3i: %i\n", livello, pgm->istogramma[livello]);
74  }
75
76  int main(int argc, char *argv[])
77  {
78    struct immagine pgm;
79    FILE *infile;
80
81    if (argc != 2) {
82      printf("uso: %s <nome_file_pgm>\n", argv[0]);
83      return 1;
84    }
85    if (!(infile = fopen(argv[1], "r"))) {
86      printf("Errore nell'apertura di %s\n", argv[1]);
87      return 1;
88    }
89    if (pgm_istogramma(&pgm, infile) != 0) {
90      puts("Errore nella lettura del file.");
91      return 1;
92    }
93
94    print_istogramma(&pgm, stdout);
95
96    free(pgm.istogramma);
97
98    return 0;
99  }
```

ESEMPIO 44: file `matrix_alloc.c`

```
1 #include <stdio.h>
```

```
 2 #include <stdlib.h>
 3
 4 void fill_matr(int **mat, int rig, int col, int val)
 5 {
 6   int i, j;
 7
 8   for (i = 0; i < rig; i++) {
 9     for (j = 0; j < col; j++) {
10       mat[i][j] = val;
11     }
12   }
13 }
14
15 int **alloc_matr(int rig, int col)
16 {
17   int *data, **mat;
18   int i;
19
20   data = malloc(rig * col * sizeof(*data));
21
22   if (data == NULL)
23     return NULL;
24
25   mat = malloc(rig * sizeof(*mat));
26   if (mat == NULL)
27     return NULL;
28
29   for (i = 0; i < rig; i++, data += col) {
30     mat[i] = data;
31   }
32
33   return mat;
34 }
35
36 void free_matr(int **mat, int rig, int col)
37 {
38   free(*mat);
39   free(mat);
40 }
41
42 int **alloc_matr2(int rig, int col)
43 {
44   int i;
45   int **mat;
46
47   mat = malloc(rig * sizeof(*mat));
48   if (mat == NULL)
```

```
49      return NULL;
50
51    for (i = 0; i < rig; i++) {
52      mat[i] = malloc(col * sizeof(**mat));
53    }
54
55    return mat;
56  }
57
58  void free_matr2(int **mat, int rig, int col)
59  {
60    int i;
61
62    for (i = 0; i < rig; i++) {
63      free(mat[i]);
64    }
65    free(mat);
66  }
67
68  void print_matr(int **mat, int rig, int col)
69  {
70    char no_sep[] = "";
71    char ok_sep[] = " ";
72    char *separator;
73    int i, j;
74
75    for (i = 0; i < rig; i++) {
76      separator = no_sep;
77      for (j = 0; j < col; j++) {
78        printf("%s%d", separator, mat[i][j]);
79        separator = ok_sep;
80      }
81      putchar('\n');
82    }
83  }
84
85  int main(int argc, const char *argv[])
86  {
87    int rig, col, val;
88    int **mat;
89
90    if (argc != 4)
91      return 1;
92
93    rig = atoi(argv[1]);
94    col = atoi(argv[2]);
95    val = atoi(argv[3]);
```

```
96
97    if ((rig <= 0) || (col <= 0))
98      return 1;
99
100   mat = alloc_matr(rig, col);
101   fill_matr(mat, rig, col, val);
102   print_matr(mat, rig, col);
103   free_matr(mat, rig, col);
104
105   return 0;
106 }
```

ESEMPIO 45: file `fgets_buffer.c`

```
1  #include <stdio.h>
2
3  int main()
4  {
5    char buffer[11];
6
7    while (fgets(buffer, sizeof(buffer), stdin)) {
8      printf("'%s'\n", buffer);
9    }
10   return 0;
11 }
```

ESEMPIO 46: file `pgm_fgets_realloc.c`

```
1  /*
2   * pgm_fgets_realloc.c
3   * Lettura immagine formato PGM con fgets.
4   * Assunzioni:
5   *  - lunghezza della riga illimitata
6   *  - leggi_riga alloca la memoria del buffer caricato
7   */
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12
13 #define MAX_LEVEL (256)
14
15 struct immagine {
16   int magic_number;             // numero magico
17   int max_gray_level;           // massimo livello di grigio
18   int larg, alt;                // larghezza e altezza dell'immagine [pixel]
```

```
19   int istogramma[MAX_LEVEL];    // vettore contenente l'istogramma
20 };
21
22 /*
23  * Attenzione che buf deve essere un puntatore, non buf[], in quanto
24  * deve poter essere modificato.
25  */
26 void analizza_riga(char *buf, struct immagine *pgm)
27 {
28   char pixel[32];
29   int n;
30
31   while (sscanf(buf, "%31[^ ]%n", pixel, &n) == 1) {
32     (pgm->istogramma[atoi(pixel)])++;
33     buf += n;  /* aumenta ptr del numero di caratteri letti */
34     if (*buf != ' ')
35       break;   /* delimitatore non trovato; fine! */
36     while (*buf == ' ' || *buf == '\n')
37       ++buf;   /* salta il delimitatore e i successivi*/
38   }
39 }
40
41 /**
42  * FIO20-C. Avoid unintentional truncation when using fgets() or fgetws()
43  * da https://www.securecoding.cert.org/confluence/pages/viewpage.action?pag
44  */
45 char *leggi_riga(FILE *infile) {
46   char temp[32];
47   char *ret = malloc(sizeof(temp));
48   char *end = ret;
49   if (!ret) {
50     return NULL;
51   }
52
53   while (fgets(temp, sizeof(temp), infile)) {
54     size_t len = strlen(temp);
55     const size_t full_size = end - ret + len;
56     char *r_temp = realloc(ret, full_size + 1); /* NTBS */
57     if (r_temp) {
58       ret = r_temp;
59       strcat(ret, temp);
60       end = ret + full_size;
61     } else {
62       break;
63     }
64
65     if ((feof(infile)) || (temp[len - 1] == '\n')) {
```

```
66        return ret;
67      }
68    }
69    free(ret);
70    return NULL;
71  }
72
73  int pgm_istogramma(struct immagine *pgm, FILE *infile)
74  {
75    int n, i;
76    char *buf = NULL;
77
78    for (i = 0; i < MAX_LEVEL; i++)
79      pgm->istogramma[i] = 0;
80
81    /* lettura e parsing della riga contenente il magic number */
82    buf = leggi_riga(infile);
83    n = sscanf(buf, "P%i", &pgm->magic_number);
84    if ((n != 1) || (pgm->magic_number != 2)) return 1;
85    free(buf);
86
87    /* lettura e parsing della riga contenente le dimensioni */
88    buf = leggi_riga(infile);
89    if (sscanf(buf, "%i %i", &pgm->larg, &pgm->alt) != 2) return 1;
90    free(buf);
91
92    /* lettura e parsing della riga contenente i livelli di grigio */
93    buf = leggi_riga(infile);
94    if (sscanf(buf, "%i", &pgm->max_gray_level) != 1) return 1;
95    free(buf);
96
97    /* lettura e parsing del resto del file */
98    while ((buf = leggi_riga(infile))) {
99      analizza_riga(buf, pgm);
100     free(buf);
101   }
102
103   return 0;
104 }
105
106 void print_istogramma(struct immagine *pgm, FILE *outfile)
107 {
108   int livello;
109
110   for (livello = 0; livello < pgm->max_gray_level; livello++)
111     fprintf(outfile, "%3i: %i\n", livello, pgm->istogramma[livello]);
112 }
```

```
113
114 int main(int argc, char *argv[])
115 {
116   struct immagine pgm = {.alt = 0, .larg = 0, .max_gray_level = 0};
117   FILE *infile;
118
119   if (argc != 2) {
120     printf("uso: %s <nome_file_pgm>\n", argv[0]);
121     return 1;
122   }
123   if (!(infile = fopen(argv[1], "r"))) {
124     printf("Errore nell'apertura di %s\n", argv[1]);
125     return 1;
126   }
127   if (pgm_istogramma(&pgm, infile) != 0) {
128     puts("Errore nella lettura del file.");
129     return 1;
130   }
131
132   print_istogramma(&pgm, stdout);
133
134   return 0;
135 }
```

ESEMPIO 47: file `func_ptr.c`

```
1  /*
2   * func_ptr.c
3   *
4   * Esempio di uso dei puntatori a funzione.
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9
10 double somma(double, double);
11 double sottrazione(double, double);
12 double prodotto(double, double);
13 double divisione(double, double);
14
15 int main(int argc, char *argv[])
16 {
17   double a, b, c;
18   int scelta;
19   double (*operazione)(double, double);
20
21   if (argc != 4) {
```

```
22      printf("Numero di parametri errato (%d)\n", argc);
23      printf("Uso: ptr_func num1 op num2\n");
24      printf("     op = { + - x : }\n");
25      return 1;
26    }
27
28    a = atof(argv[1]);
29    b = atof(argv[3]);
30    scelta = argv[2][0];
31
32    switch(scelta) {
33      case '+' :
34        operazione = somma;
35        break;
36      case '-' :
37        operazione = sottrazione;
38        break;
39      case 'x' :
40        /* non uso '*' per evitare problemi sulla linea di comando */
41        operazione = prodotto;
42        break;
43      case ':' :
44        operazione = divisione;
45        break;
46      default :
47        printf("Operazione non supportata\n");
48        return 1;
49    }
50
51    c = operazione(a, b);
52    printf("Il risultato vale %lf\n", c);
53
54    return 0;
55  }
56
57  double somma(double a, double b)
58  {
59    return a + b;
60  }
61
62  double sottrazione(double a, double b)
63  {
64    return a - b;
65  }
66
67  double prodotto(double a, double b)
68  {
```

```
69    return a * b;
70  }
71
72  double divisione(double a, double b)
73  {
74    return (a / b);
75  }
76
```

Capitolo 9

Suddivisione in moduli di un programma

ESEMPIO 48: file `string_change.c`

```
 1 /*
 2  * string_change.c
 3  *
 4  * Esempio di programma il cui codice sorgente e`
 5  * scritto in diversi file sorgente.
 6  * Gli altri file che compongono il programma sono
 7  * pers.c, pers.h, strchg.c, strchg.h, debug.h e il
 8  * makefile.
 9  * Il programma viene anche impiegato per mostrare
10  * l'utilizzo di make e dei makefile.
11  */
12
13 #include <stdlib.h>
14 #include "debug.h"
15 #include "pers.h"
16 #include "strchg.h"
17
18 int main(int argc, char *argv[])
19 {
20   struct pers *per;
21
22   TRACE();
23   srand(argc == 2 ? atoi(argv[1]) : 3);
24
25   if (!(per = pers_alloc("Albert", "Einstein")))
26     return 1;
27
```

```
28    pers_print(per);
29
30    TRACE();
31    anagram(per->nome);
32    anagram(per->cognome);
33    pers_print(per);
34
35    capitalize(per->nome);
36    capitalize(per->cognome);
37    pers_print(per);
38
39    pers_free(per);
40    TRACE();
41
42    return 0;
43  }
```

ESEMPIO 49: file `debug.h`

```
1  /*
2   * debug.h
3   *
4   * Dichiarazione della macro TRACE usata
5   * da vari moduli di string_change.c.
6   */
7
8  #ifndef __DEBUG_H__
9  #define __DEBUG_H__
10
11 #include <stdio.h>
12
13 #define TRACE() fprintf(stderr, "%s (%d): %s\n", __FILE__, __LINE__, __func_
14
15 #endif
```

ESEMPIO 50: file `pers.h`

```
1  /*
2   * pers.h
3   *
4   * File di intestazione che contiene la
5   * dichiarazione di strutture e funzioni
6   * utilizzate da string_change.c.
7   */
8
9  #ifndef __PERS_H__
```

```
10 #define __PERS_H__
11
12 struct pers {
13   char *nome;
14   char *cognome;
15 };
16
17 struct pers *pers_alloc(const char *nome, const char *cognome);
18 void pers_free(struct pers *p);
19 void pers_print(struct pers *p);
20
21 #endif
```

ESEMPIO 51: file `pers.c`

```
1  /*
2   * pers.c
3   *
4   * Definizione di una funzione utilizzata nel
5   * programma string_change.c.
6   */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include "debug.h"
12 #include "pers.h"
13
14 struct pers *pers_alloc(const char *nome, const char *cognome)
15 {
16   struct pers *p;
17   TRACE();
18
19   if (!nome || !cognome)
20     return NULL;
21
22   if (!(p = malloc(sizeof(*p))))
23     return NULL;
24
25   if (!(p->nome = strdup(nome))) {
26     free(p);
27     return NULL;
28   }
29
30   if (!(p->cognome = strdup(cognome))) {
31     free(p->nome);
32     free(p);
```

```
33      return NULL;
34    }
35
36    return p;
37  }
38
39  void pers_free(struct pers *p)
40  {
41    TRACE();
42    free(p->nome);
43    free(p->cognome);
44    free(p);
45  }
46
47  void pers_print(struct pers *p)
48  {
49    TRACE();
50    if (p && p->nome && p->cognome)
51      printf("%s %s\n", p->nome, p->cognome);
52  }
```

ESEMPIO 52: file `strchg.h`

```
1  /*
2   * strchg.h
3   *
4   * Dichiarazione di funzioni di elaborazione delle stringhe.
5   */
6
7  #ifndef __STRCHG_H__
8  #define __STRCHG_H__
9
10 int anagram(char *s);
11 void capitalize(char *s);
12
13 #endif
```

ESEMPIO 53: file `strchg.c`

```
1  /*
2   * strchg.c
3   *
4   * Definizione di funzioni di elaborazione delle stringhe.
5   */
6
7  #include <string.h>
```

```
 8 #include <stdlib.h>
 9 #include <ctype.h>
10 #include "debug.h"
11 #include "strchg.h"
12
13 void swap(char *a, char *b)
14 {
15   char c;
16   c = *a;
17   *a = *b;
18   *b = c;
19 }
20
21 int anagram(char *s)
22 {
23   int len;
24   int i;
25   int a, b;
26
27   TRACE();
28   len = strlen(s);
29   for (i = 0; i < len / 2; i++) {
30     a = rand() % len;
31     b = rand() % len;
32     swap(&s[a], &s[b]);
33   }
34
35   return 0;
36 }
37
38 void capitalize(char *s)
39 {
40   int len, i;
41
42   TRACE();
43   len = strlen(s);
44   if (len < 1)
45     return;
46
47   s[0] = toupper(s[0]);
48   for (i = 1; i < len; i++)
49     s[i] = tolower(s[i]);
50 }
```

Capitolo 10

Gli operatori

ESEMPIO 54: file `notes.c`

```
1  /*
2   * notes.c
3   *
4   * Programma che stampa a video le note
5   * musicali in notazione anglosassone, con
6   * indicazione dell'ottava di appartenenza.
7   */
8  #include <stdio.h>
9
10 #define NOTE 12
11
12 int main()
13 {
14   int i, j, n;
15   char *note[] = {"C",  "Db", "D",  "Eb", "E",  "F",
16                   "Gb", "G",  "Ab", "A",  "Bb", "B"};
17
18   for (i = 0; i < 127; i++) {
19     n = printf("%s%i ", note[i % NOTE], (i / NOTE) - 1);
20     for (j = n; j < 5; j++) putchar(' ');
21     if (!((i + 1) % NOTE)) putchar('\n');
22   }
23   putchar('\n');
24   return 0;
25 }
```

ESEMPIO 55: file `and.c`

```
1  #include <stdio.h>
2
3  struct t_str;
4
5  struct t_methods {
6    int (* print)(struct t_str *str);
7  };
8
9  struct t_str {
10   int id;
11   struct t_methods *methods;
12 };
13
14 int print(struct t_str *p)
15 {
16   printf("%d\n", p->id);
17   return 1;
18 }
19
20 int main()
21 {
22   struct t_str str = {100, NULL};
23   struct t_methods methods = {NULL};
24   struct t_str *strptr = NULL;
25
26   methods.print = print;
27   strptr = &str;
28   if (strptr)
29     strptr->methods = &methods;
30
31   if (strptr &&
32       strptr->methods &&
33       strptr->methods->print)
34    strptr->methods->print(strptr);
35
36   return 0;
37 }
```

ESEMPIO 56: file `or.c`

```
1  #include <stdio.h>
2
3  #define N 5
4
5  int fill_item(int **v, int n)
6  {
7    static int m = 10;
```

```
 8
 9   if (n < 3) *v = &m;
10   else v = NULL;
11
12   return (int)(v);
13 }
14
15 int set_default(int **v)
16 {
17   static int m = 100;
18
19   *v = &m;
20
21   return (int)(v);
22 }
23
24 int main()
25 {
26   int *v[N];
27   int i;
28
29   for (i = 0; i < N; i++) v[i] = NULL;
30   v[2] = &i;
31
32   for (i = 0; i < N; i++) {
33     if (v[i] || fill_item(&v[i], i) || set_default(&v[i])) {
34       printf("%d\n", *v[i]);
35     }
36   }
37
38   return 0;
39 }
```

Capitolo 11

Le classi di memorizzazione

ESEMPIO 57: file `scope.c`

```
1  /*
2   * scope.c
3   *
4   * esempi di visibilit  delle variabili
5   */
6  #include <stdio.h>
7
8  int c = 1;
9
10 void func1(int c)
11 {
12   printf("[func1, pre if ] %d\n", c);
13   if (1) {
14     int c = 5;
15     printf("[func1, if     ] %d\n", c);
16   }
17   printf("[func1, post if] %d\n", c);
18 }
19
20 void func2()
21 {
22   printf("[func2, pre if ] %d\n", c);
23   if (1) {
24     int c = 6;
25     printf("[func2, if     ] %d\n", c);
26   }
27   printf("[func2, post if] %d\n", c);
28 }
29
```

```
30 int main(int argc, char *argv[])
31 {
32   int c = 2;
33
34   printf("[main,  pre if ] %d\n", c);
35   if (1) {
36     int c = 3;
37     printf("[main,  if     ] %d\n", c);
38   }
39   printf("[main,  post if] %d\n", c);
40
41   func1(4);
42   func2();
43
44   return 0;
45 }
```

ESEMPIO 58: file `static.c`

```
1 #include <stdio.h>
2
3 int c1 = 1;
4
5 void func() {
6   int c3 = 3;
7   static int c4 = 1;
8
9   printf("func: c1 %2d   c3 %2d   c4 %2d\n", c1, c3, c4);
10  c4++;
11  c3++;
12 }
13
14 int main(int argc, char *argv[])
15 {
16   int i;
17
18   /* questa istruzione genera errore */
19   //printf("main: c3 %d\n", c3);
20
21   for (i = 0; i < 5; i++) {
22     c1 = c1 + 2;
23     func();
24   }
25
26   return 0;
27 }
```

Capitolo 12

Ricerca e ordinamento

ESEMPIO 59: file `bubble_sort.c`

```
 1 /*
 2  * bubble_sort.c
 3  *
 4  * esempio di programma di ordinamento bubblesort;
 5  *
 6  * alcuni concetti illustrati:
 7  * - direttive del pre-processore
 8  * - suddivisione in funzioni di un programma
 9  * - passaggio di parametri per indirizzo
10  * - cicli for annidati
11  * - generazione di numeri casuali
12  */
13 #include <stdio.h>
14 #include <stdlib.h>
15
16 #define MAX 20
17
18 void swap(int *a, int *b)
19 {
20   int tmp;
21
22   tmp = *a;
23   *a = *b;
24   *b = tmp;
25 }
26
27 void bubblesort(int l[], int n)
28 {
29   int i, j;
```

```
30
31    for (i = 0; i < n; i++)
32      for (j = i+1; j < n; j++)
33        if (l[i] > l[j])
34          swap(&l[i], &l[j]);
35  }
36
37  void init(int l[], int n)
38  {
39    int i;
40
41    for (i = 0; i < n; i++)
42      l[i] = rand() % MAX;
43  }
44
45  void stampa(int l[], int n)
46  {
47    int i;
48
49    for (i = 0; i < n; i++)
50      printf("%d ", l[i]);
51    printf("\n");
52  }
53
54  int main(int argc, char *argv[])
55  {
56    int *l, n;
57
58    if (argc != 3) {
59      puts("Uso: bubble_sort seed size");
60      return 1;
61    }
62    n = atoi(argv[2]);
63
64    if (!(l = malloc(n * sizeof(int)))) {
65      puts("Impossibile allocare il vettore");
66      return 1;
67    }
68
69    srand(atoi(argv[1]));
70
71    init(l, n);
72    stampa(l, n);
73    bubblesort(l, n);
74    stampa(l, n);
75
76    free(l);
```

```
77
78   return 0;
79 }
```

ESEMPIO 60: file `search_seq.c`

```
1  /*
2   * ricerca_seq.c
3   *
4   * esempio di algoritmo di ricerca sequenziale.
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include "ricerca_util.h"
10
11 #define MAX 10
12
13 int ricerca_seq(int l[], int x, int n);
14
15 int main(int argc, char *argv[])
16 {
17   int *l, x, n;
18
19   if (argc != 2) {
20     printf("Specificare la dimensione del vettore\n");
21     return 1;
22   }
23
24   n = atoi(argv[1]);
25   l = malloc(n * sizeof(*l));
26
27   init(l, n, MAX);
28   stampa(l, n);
29
30   printf("chiave: ");
31   scanf("%d", &x);
32   printf("ricerca_seq(l, %d, %d) = %d\n",
33          x, N, ricerca_seq(l, x, n));
34
35   free(l);
36
37   return 0;
38 }
39
40 int ricerca_seq(int l[], int x, int n)
41 {
```

```
42   int i = 0;
43
44   while ((i < n) && (x != l[i])) i++;
45
46   if (i < n)
47     return i;
48
49   return -1;
50 }
```

ESEMPIO 61: file `search_sent.c`

```
1  /*
2   * ricerca_sent.c
3   *
4   * Esempio di algoritmo di ricerca sequenziale
5   * con valore sentinella.
6   */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include "ricerca_util.h"
12
13 #define MAX 10
14
15 int ricerca_sent(int l[], int x);
16
17 int main(int argc, char *argv[])
18 {
19   int *l, x, n;
20
21   if (argc != 2) {
22     printf("Specificare la dimensione del vettore\n");
23     return 1;
24   }
25
26   n = atoi(argv[1]);
27   l = malloc(n * sizeof(*l));
28
29   memset(l, 0, n * sizeof(*l));
30   init(l, n - 1, MAX);
31   stampa(l, n);
32
33   printf("chiave: ");
34   scanf("%d", &x);
35   printf("ricerca_sent(l, %d) = %d\n", x, ricerca_sent(l, x));
```

```
36   free(l);
37
38   return 0;
39 }
40
41 /*
42  * Ricerca il numero x nel vettore l.
43  * I numeri da ricercare sono tutti diversi da 0.
44  * Il valore sentinella e` il valore 0.
45  */
46 int ricerca_sent(int l[], int x)
47 {
48   int i = 0;
49
50   while (l[i] != 0) {
51     if (l[i] == x)
52       return i;
53     i++;
54   }
55
56   return -1;
57 }
```

ESEMPIO 62: file `search_bin_ric.c`

```
1  /*
2   * ricerca_bin_ric.c
3   *
4   * Esempio di algoritmo di ricerca binaria ricorsiva.
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include "ricerca_util.h"
10
11 #define N   15
12 #define MAX 50
13
14 int ricerca_bin_ric(int l[], int x, int a, int b);
15 int cmp_int(const void *a, const void *b);
16
17 int main(int argc, char *argv[])
18 {
19   int l[N], x;
20
21   init(l, N, MAX);
22   stampa(l, N);
```

```
23
24    /*
25     * Per usare la ricerca binaria il
26     * vettore deve essere ordinato.
27     */
28    qsort(l, sizeof(l) / sizeof(int), sizeof(int), cmp_int);
29
30    stampa(l, N);
31
32    printf("chiave: ");
33    scanf("%d", &x);
34    printf("ricerca_bin_ric(%d, %d) %d\n",
35           x, N, ricerca_bin_ric(l, x, 0, N));
36
37    return 0;
38  }
39
40  int ricerca_bin_ric(int l[], int x, int a, int b)
41  {
42    int m;
43
44    /* l'elemento desiderato non c'e' */
45    if ((b < a) || (a < 0) || (b < 0))
46      return -1;
47
48    m = (a + b) / 2;   /* elemento centrale */
49    if (x < l[m]) {
50      /* ricerca nella parte inferiore */
51      return ricerca_bin_ric(l, x, a, m - 1);
52    } else if (x > l[m]) {
53      /* ricerca nella parte superiore */
54      return ricerca_bin_ric(l, x, m + 1, b);
55    } else {
56      /* indice dell'elemento desiderato */
57      return m;
58    }
59  }
60
61  int cmp_int(const void *a, const void *b)
62  {
63    return *(int *)(a) - *(int *)(b);
64  }
```

ESEMPIO 63: file `search_bin_nr.c`

```
1  /*
2   * ricerca_bin_nr.c
```

```
 3  *
 4  * esempio di algoritmo di ricerca binaria NON ricorsiva.
 5  */
 6
 7 #include <stdio.h>
 8 #include <stdlib.h>
 9 #include "ricerca_util.h"
10
11 #define N   15
12 #define MAX 10
13
14 int ricerca_bin_nr(int l[], int x, int n);
15 int cmp_int(const void *a, const void *b);
16
17 int main(int argc, char *argv[])
18 {
19   int l[N], x;
20
21   init(l, N, MAX);
22   stampa(l, N);
23
24   /*
25    * Per usare la ricerca binaria il vettore
26    * deve essere ordinato
27    */
28   qsort(l, sizeof(l) / sizeof(int), sizeof(int), cmp_int);
29
30   stampa(l, N);
31
32   printf("chiave: ");
33   scanf("%d", &x);
34   printf("ricerca_bin_nr(%d, %d) %d\n",
35          x, N, ricerca_bin_nr(l, x, N));
36
37   return 0;
38 }
39
40 int ricerca_bin_nr(int l[], int x, int n)
41 {
42   int a, b, m;
43
44   a = 0;
45   b = n - 1;
46
47   while (a <= b) {
48     m = (a + b) / 2;
49     if(l[m] == x)
```

```
50        return m;   /* valore x trovato alla posizione m */
51      if(l[m] < x)
52        a = m + 1;
53      else
54        b = m - 1;
55    }
56
57    return -1;
58  }
59
60  int cmp_int(const void *a, const void *b)
61  {
62    return *(int *)(a) - *(int *)(b);
63  }
```

ESEMPIO 64: file `strange_people.c`

```
1  /*
2   * people.c
3   *
4   * Programma che legge mostra l'uso delle funzioni
5   * di libreria qsort e bsearch.
6   */
7  #include <stdlib.h>
8  #include <stdio.h>
9  #include <string.h>
10
11 struct people_t {
12   char nome[101];
13   char specie[101];
14   int anno;
15   int mese;
16   int giorno;
17 };
18
19 void stampa(const struct people_t *c);
20 void stampa_tutti(const struct people_t *c, int n);
21 struct people_t *trova_nome(struct people_t *people,
22                 int count, const char *nome);
23 struct people_t *load(int *count);
24
25 int cmp_nome(const void *c1p, const void *c2p);
26
27 int main(int argc, char *argv[])
28 {
29   int count, i;
30   struct people_t *people, *result = NULL;
```

```
31
32    people = load(&count);
33    if (!people) {
34      puts("Errore nel caricamento dati");
35      return -1;
36    }
37    stampa_tutti(people, count);
38    puts("******************");
39
40    qsort(people, count, sizeof(*people), cmp_nome);
41    stampa_tutti(people, count);
42
43    for (i = 1; i < argc; i++) {
44      result = trova_nome(people, count, argv[i]);
45      if (result) {
46        printf("[TROVA] ");
47        stampa(result);
48      } else {
49        printf("[TROVA] <%s> non trovato\n", argv[i]);
50      }
51    }
52
53    free(people);
54    return 0;
55  }
56
57  /*
58   * Funzione di comparazione per ordinamento e ricerca.
59   */
60  int cmp_nome(const void *c1p, const void *c2p)
61  {
62    const struct people_t *c1 = c1p, *c2 = c2p;
63
64    return strcmp(c1->nome, c2->nome);
65  }
66
67  /*
68   * Stampa il contenuto di una singola struttura.
69   */
70  void stampa(const struct people_t *c)
71  {
72    printf("%s %s %d %d %d\n", c->nome, c->specie,
73           c->giorno, c->mese, c->anno);
74  }
75
76  /*
77   * Stampa il contenuto di tutto il vettore.
```

```
 */
void stampa_tutti(const struct people_t *c, int n)
{
  int i;

  for (i = 0; i < n; i++)
    stampa(c + i);
}

/*
 * Ricerca nel vettore ordinato.
 */
struct people_t *trova_nome(struct people_t *people,
                int count, const char *nome)
{
  struct people_t target, *result;

  strcpy(target.nome, nome);
  result = bsearch(&target, people, count,
                   sizeof(*people), cmp_nome);

  return result;
}

struct people_t *load(int *count)
{
  struct people_t *v;
  int conv, dim;
  char buf[1000];

  *count = 0;
  dim = 4;
  if (!(v = malloc(dim * sizeof(*v)))) {
    return NULL;
  }

  while (fgets(buf, sizeof(buf), stdin)) {
    conv = sscanf(buf, "%100s %100s %d %d %d",
                  ((struct people_t *)(v) + *count)->nome,
                  ((struct people_t *)(v + *count))->specie,
                  &((struct people_t *)(v + *count))->anno,
                  &((struct people_t *)(v + *count))->mese,
                  &((struct people_t *)(v + *count))->giorno);

    /*
     * Verifica che la sscanf abbia convertito correttamente
     * i 5 campi attesi.
```

```
125       */
126      if (conv < 5) {
127        free(v);
128        return NULL;
129      }
130
131      /*
132       * Nel caso in cui il vettore attualmente allocato
133       * sia stato completamente riempito, lo rialloca
134       * raddoppiando la dimensione.
135       */
136      if (*count + 1 >= dim) {
137        dim *= 2;
138        if (!(v = realloc(v, dim * sizeof(*v)))) {
139          free(v);
140          return NULL;
141        }
142      }
143      (*count)++;
144    }
145    return v;
146  }
147
```