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

```
 4  #if VERSIONE == 1
 5  int **alloca_matrice(int rig, int col)
 6  {
 7      int *data, **mat;
 8      int i;
 9
10      data = malloc(rig * col * sizeof(*data));
11      if (data == NULL) return NULL;
12
13      mat = malloc(rig * sizeof(*mat));
14      if (mat == NULL) return NULL;
15
16      for (i = 0; i < rig; i++, data += col) {
17          mat[i] = data;
18      }
19      return mat;
20  }
21
22  void libera_matrice(int **mat)
23  {
24      free(mat[0]);
25      free(mat);
26  }
27  #endif
```

```
28  #if VERSIONE == 2
29  int **alloca_matrice(int rig, int col)
30  {
31      int i;
32      int **mat = malloc(rig * sizeof(*mat));
33      if (mat == NULL) return NULL;
34
35      for (i = 0; i < rig; i++) {
36          /* ogni riga viene allocata separatamente */
37          mat[i] = malloc(col * sizeof(**mat));
38      }
39      return mat;
40  }
41
42  void libera_matrice(int **mat, int rig)
43  {
44      int i;
45      /* Devono prima essere deallocate le singole righe */
46      for (i=0; i<rig; i++) free(mat[i]);
47      /* Poi viene deallocato il vettore di righe */
48      free(mat);
49  }
50  #endif
51
```

```
52  #if VERSIONE == 3
53  /* S è la dimensione di ogni singolo data */
54  /* con calloc la memoria viene azzerata */
55  #define ALLOCA_MATRICE(R, C, S) calloc((R) * (C), (S))
56
57  #define LIBERA_MATRICE free
58
59  /* si possono definire due funzioni per accedere ai singoli
    elementi, si noti che il numero di righe non è necessario */
60  int get(int *mat, int nc, int i, int j) {
61      return mat[i*nc+j];
62  }
63
64  void set(int *mat, int nc, int i, int j, int value) {
65      mat[i*nc+j] = value;
66  }
67
68  /* oppure si possono usare ancora delle macro */
69  /* va fatto ad hoc per ogni matrice */
70  #define MAT(I,J) (mat[(I)*3+(J)])
71
72
73
74
75
76
```

```
 77
 78  int main(int argc, char *argv[])
 79  {
 80      int *mat = ALLOCA_MATRICE(4, 3, sizeof(*mat));
 81      MAT(2,3) = 5;
 82      printf("%d\n", MAT(2,3));
 83      LIBERA_MATRICE(mat);
 84      return 0;
 85  }
 86  #endif
 87
```

```
 88  #if VERSIONE == 4
 89  /* versione "quasi" ad oggetti */
 90
 91  /* viene definita una struttura "opaca" */
 92  struct _matrice_;
 93  /* si utilizzano solo puntatori alla struttura */
 94  typedef struct _matrice_ *matrice;
 95
 96  /* la struttura è usabile solo attraverso funzioni di libreria */
 97  matrice mat_alloca(int nr, int nc);
 98  matrice mat_prodotto(matrice m1, matrice m2);
 99  void mat_stampa(matrice m);
100  void mat_libera(matrice m);
101
102  void test()
103  {
104      matrice m = mat_alloca(4,4);
105      matrice q = mat_prodotto(m, m);
106      mat_stampa(q);
107      mat_libera(m);
108      mat_libera(q);
109  }
110  #endif
111
```