23 In the two previous exercises we used the value returned by time() to seed the random-number generator. In this exercise we want to use the value returned by time() to measure the time it takes to call rand(). Here is one way this can be done:

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

#define   NCALLS    10000000   /* number of fct calls */
#define   NCOLS     8          /* number of columns   */
#define   NLINES    3          /* number of lines     */

int main(void)
{
  int    i, val;
  long   begin, diff, end;

  begin = time(NULL);
  srand(time(NULL));
  printf("\nTIMING TEST: %d calls to rand()\n\n", NCALLS);
  for (i = 1; i <= NCALLS; ++i) {
    val = rand();
    if (i <= NCOLS * NLINES) {
      printf("%7d", val);
      if (i % NCOLS == 0)
          putchar('\n');
    }
    else if (i == NCOLS * NLINES + 1)
      printf("%7s\n\n", ".....");
  }
  end = time(NULL);
  diff = end - begin;
  printf("%s%ld\n%s%ld\n%s%ld\n%s%.10f\n\n",
    "          end time: ", end,
    "        begin time: ", begin,
    "      elapsed time: ", diff,
    "time for each call: ", (double) diff / NCALLS);
  return 0;
}
```

Here is the output on our system:

```
TIMING TEST: 10000000 calls to rand()

 11753  27287  12703   5493  23634  23237  24988  31011
 19570    432  12211   9712  30284  31480  32334  30292
 13830  27126  17405   4877  19045   7305   1114  28874
 .....

       end time: 868871916
     begin time: 868871900
   elapsed time: 16
time for each call: 0.0000016000
```

The intent of this program is to print out some of the values produced by the call to rand() but not all of them. After all, looking at ten million numbers on the screen is not too interesting. Experiment with this program by modifying some of the #defines so that you can see what their effects are. For example, try making the following changes:

```
#define   NCALLS    1000     /* number of fct calls */
#define   NCOLS     7        /* number of columns   */
#define   NLINES    7        /* number of lines     */
```

Caution: If you are on a time-shared machine, then the use of values returned by time() to time things can be misleading. Between your calls to time(), the machine may be servicing other requests, making your timing results inaccurate. The proper way to time C code is with the use of the clock() function. (See Section 11.16, “How to Time C Code,” on page 528.)

24 The function rand() returns values in the interval [0, RAND_MAX]. (See exercise 20, on page 100.) If we declare the variable median of type double and initialize it to have the value RAND_MAX/2.0, then rand() will return a value that is sometimes larger than median and sometimes smaller. On average, there should be as many values that are larger as there are values that are smaller. Test this hypothesis. Write a program that calls rand(), say 500 times, inside a for loop, increments the variable above_cnt every time rand() returns a value larger than median, and increments the variable below_cnt every time rand() returns a value less than median. Each time through the for loop, print out the value of the difference of above_cnt and below_cnt. This difference should oscillate about zero. Does it?

25 In this exercise we continue with the discussion started in exercise 20, on page 100. A call to rand() produces a value in the interval [0, RAND_MAX], and RAND_MAX typically has the value 32767. Since this value is rather small, rand() is not useful for many scientific problems. Most C systems on UNIX machines provide the programmer with the rand48 family of random-number generators, so called because 48-bit arithmetic gets used to generate the numbers. The function drand48(), for example, can be used to produce randomly distributed doubles in the range [0, 1], and the function