EXERCISES

1.1.1 Give the value of each of the following expressions:

a. `( 0 + 15 ) / 2`

b. `2.0e-6 * 100000000.1`

c. `true && false || true && true`

1.1.2 Give the type and value of each of the following expressions:

a. `(1 + 2.236)/2`

b. `1 + 2 + 3 + 4.0`

c. `4.1 >= 4`

d. `1 + 2 + "3"`

1.1.3 Write a program that takes three integer command-line arguments and prints `equal` if all three are equal, and `not equal` otherwise.

1.1.4 What (if anything) is wrong with each of the following statements?

a. `if (a > b) then c = 0;`

b. `if a > b { c = 0; }`

c. `if (a > b) c = 0;`

d. `if (a > b) c = 0 else b = 0;`

1.1.5 Write a code fragment that prints `true` if the `double` variables `x` and `y` are both strictly between `0` and `1` and `false` otherwise.

1.1.6 What does the following program print?

```
int f = 0;
int g = 1;
for (int i = 0; i <= 15; i++)
{
   StdOut.println(f);
   f = f + g;
   g = f - g;
}
```

1.1.7 Give the value printed by each of the following code fragments:

a.
```
double t = 9.0;
while (Math.abs(t - 9.0/t) > .001)
   t = (9.0/t + t) / 2.0;
StdOut.printf("%.5f\n", t);
```

b.
```
int sum = 0;
for (int i = 1; i < 1000; i++)
    for (int j = 0; j < i; j++)
        sum++;
StdOut.println(sum);
```

c.
```
int sum = 0;
for (int i = 1; i < 1000; i *= 2)
   for (int j = 0; j < N; j++)
       sum++;
StdOut.println(sum);
```

1.1.8 What do each of the following print?

a. `System.out.println('b');`

b. `System.out.println('b' + 'c');`

c. `System.out.println((char) ('a' + 4));`

Explain each outcome.

1.1.9 Write a code fragment that puts the binary representation of a positive integer `N` into a `String s`.

Solution: Java has a built-in method `Integer.toBinaryString(N)` for this job, but the point of the exercise is to see how such a method might be implemented. Here is a particularly concise solution:

```
String s = "";
for (int n = N; n > 0; n /= 2)
   s = (n % 2) + s;
```

EXERCISES *(continued)*

1.1.10 What is wrong with the following code fragment?

```
int[] a;
for (int i = 0; i < 10; i++)
   a[i] = i * i;
```

Solution: It does not allocate memory for `a[]` with `new`. This code results in a `variable a might not have been initialized` compile-time error.

1.1.11 Write a code fragment that prints the contents of a two-dimensional boolean array, using * to represent `true` and a space to represent `false`. Include row and column numbers.

1.1.12 What does the following code fragment print?

```
int[] a = new int[10];
for (int i = 0; i < 10; i++)
   a[i] = 9 - i;
for (int i = 0; i < 10; i++)
   a[i] = a[a[i]];
for (int i = 0; i < 10; i++)
   System.out.println(i);
```

1.1.13 Write a code fragment to print the *transposition* (rows and columns changed) of a two-dimensional array with *M* rows and *N* columns.

1.1.14 Write a static method `lg()` that takes an `int` value `N` as argument and returns the largest `int` not larger than the base-2 logarithm of `N`. Do *not* use `Math`.

1.1.15 Write a static method `histogram()` that takes an array `a[]` of `int` values and an integer `M` as arguments and returns an array of length `M` whose `i`th entry is the number of times the integer `i` appeared in the argument array. If the values in `a[]` are all between `0` and `M–1`, the sum of the values in the returned array should be equal to `a.length`.

1.1.16 Give the value of `exR1(6)`:

```
public static String exR1(int n)
{
   if (n <= 0) return "";
   return exR1(n-3) + n + exR1(n-2) + n;
}
```

1.1.17 Criticize the following recursive function:

```
public static String exR2(int n)
{
   String s = exR2(n-3) + n + exR2(n-2) + n;
   if (n <= 0) return "";
   return s;
}
```

Answer: The base case will never be reached. A call to `exR2(3)` will result in calls to `exR2(0)`, `exR2(-3)`, `exR3(-6)`, and so forth until a `StackOverflowError` occurs.

1.1.18 Consider the following recursive function:

```
public static int mystery(int a, int b)
{
   if (b == 0)     return 0;
   if (b % 2 == 0) return mystery(a+a, b/2);
   return mystery(a+a, b/2) + a;
}
```

What are the values of `mystery(2, 25)` and `mystery(3, 11)`? Given positive integers `a` and `b`, describe what value `mystery(a, b)` computes. Answer the same question, but replace `+` with `*` and replace `return 0` with `return 1`.

1.1.19 Run the following program on your computer:

```
public class Fibonacci
{
   public static long F(int N)
   {
      if (N == 0) return 0;
      if (N == 1) return 1;
      return F(N-1) + F(N-2);
   }

   public static void main(String[] args)
   {
      for (int N = 0; N < 100; N++)
         StdOut.println(N + " " + F(N));
   }
}
```

EXERCISES *(continued)*

What is the largest value of `N` for which this program takes less 1 hour to compute the value of `F(N)`? Develop a better implementation of `F(N)` that saves computed values in an array.

1.1.20 Write a recursive static method that computes the value of ln $(N!)$

1.1.21 Write a program that reads in lines from standard input with each line containing a name and two integers and then uses `printf()` to print a table with a column of the names, the integers, and the result of dividing the first by the second, accurate to three decimal places. You could use a program like this to tabulate batting averages for baseball players or grades for students.

1.1.22 Write a version of `BinarySearch` that uses the recursive `rank()` given on page 25 and *traces* the method calls. Each time the recursive method is called, print the argument values `lo` and `hi`, indented by the depth of the recursion. *Hint*: Add an argument to the recursive method that keeps track of the depth.

1.1.23 Add to the `BinarySearch` test client the ability to respond to a second argument: + to print numbers from standard input that *are not* in the whitelist, - to print numbers that *are* in the whitelist.

1.1.24 Give the sequence of values of p and q that are computed when Euclid's algorithm is used to compute the greatest common divisor of 105 and 24. Extend the code given on page 4 to develop a program `Euclid` that takes two integers from the command line and computes their greatest common divisor, printing out the two arguments for each call on the recursive method. Use your program to compute the greatest common divisor or 1111111 and 1234567.

1.1.25 Use mathematical induction to prove that Euclid's algorithm computes the greatest common divisor of any pair of nonnegative integers p and q.

CREATIVE EXERCISES

1.1.26 *Sorting three numbers.* Suppose that the variables `a`, `b`, `c`, and `t` are all of the same numeric primitive type. Show that the following code puts `a`, `b`, and `c` in ascending order:

```
if (a > b) { t = a; a = b; b = t; }
if (a > c) { t = a; a = c; c = t; }
if (b > c) { t = b; b = c; c = t; }
```

1.1.27 *Binomial distribution.* Estimate the number of recursive calls that would be used by the code

```
public static double binomial(int N, int k, double p)
{
   if ((N == 0) || (k < 0)) return 1.0;
   return (1.0 - p)*binomial(N-1, k) + p*binomial(N-1, k-1);
}
```

to compute `binomial(100, 50)`. Develop a better implementation that is based on saving computed values in an array.

1.1.28 *Remove duplicates.* Modify the test client in `BinarySearch` to remove any duplicate keys in the whitelist after the sort.

1.1.29 *Equal keys.* Add to `BinarySearch` a static method `rank()` that takes a key and a sorted array of `int` values (some of which may be equal) as arguments and returns the number of elements that are smaller than the key and a similar method `count()` that returns the number of elements equal to the key. *Note*: If `i` and `j` are the values returned by `rank(key, a)` and `count(key, a)` respectively, then `a[i..i+j-1]` are the values in the array that are equal to `key`.

1.1.30 *Array exercise.* Write a code fragment that creates an *N*-by-*N* boolean array `a[][]` such that `a[i][j]` is `true` if `i` and `j` are relatively prime (have no common factors), and `false` otherwise.

1.1.31 *Random connections.* Write a program that takes as command-line arguments an integer `N` and a `double` value `p` (between 0 and 1), plots `N` equally spaced dots of size .05 on the circumference of a circle, and then, with probability `p` for each pair of points, draws a gray line connecting them.

CREATIVE EXERCISES *(continued)*

1.1.32 *Histogram.* Suppose that the standard input stream is a sequence of `double` values. Write a program that takes an integer N and two `double` values l and r from the command line and uses `StdDraw` to plot a histogram of the count of the numbers in the standard input stream that fall in each of the N intervals defined by dividing (l, r) into N equal-sized intervals.

1.1.33 *Matrix library.* Write a library `Matrix` that implements the following API:

```
public class Matrix
   static     double  dot(double[] x, double[] y)          vector dot product
   static double[][]  mult(double[][] a, double[][] b)     matrix-matrix product
   static double[][]  transpose(double[][] a)              transpose
   static   double[]  mult(double[][] a, double[] x)       matrix-vector product
   static   double[]  mult(double[] y, double[][] a)       vector-matrix product
```

Develop a test client that reads values from standard input and tests all the methods.

1.1.34 *Filtering.* Which of the following *require* saving all the values from standard input (in an array, say), and which could be implemented as a filter using only a fixed number of variables and arrays of fixed size (not dependent on N)? For each, the input comes from standard input and consists of N real numbers between 0 and 1.

- Print the maximum and minimum numbers.
- Print the median of the numbers.
- Print the kth smallest value, for k less than 100.
- Print the sum of the squares of the numbers.
- Print the average of the N numbers.
- Print the percentage of numbers greater than the average.
- Print the N numbers in increasing order.
- Print the N numbers in random order.

EXPERIMENTS

1.1.35 *Dice simulation.* The following code computes the exact probability distribution for the sum of two dice:

```
int SIDES = 6;
double[] dist = new double[2*SIDES+1];
for (int i = 1; i <= SIDES; i++)
   for (int j = 1; j <= SIDES; j++)
      dist[i+j] += 1.0;

for (int k = 2; k <= 2*SIDES; k++)
   dist[k] /= 36.0;
```

The value `dist[i]` is the probability that the dice sum to `k`. Run experiments to validate this calculation simulating N dice throws, keeping track of the frequencies of occurrence of each value when you compute the sum of two random integers between 1 and 6. How large does N have to be before your empirical results match the exact results to three decimal places?

1.1.36 *Empirical shuffle check.* Run computational experiments to check that our shuffling code on page 32 works as advertised. Write a program `ShuffleTest` that takes command-line arguments M and N, does N shuffles of an array of size M that is initialized with `a[i] = i` before each shuffle, and prints an M-by-M table such that row `i` gives the number of times `i` wound up in position `j` for all `j`. All entries in the array should be close to N/M.

1.1.37 *Bad shuffling.* Suppose that you choose a random integer between `0` and `N-1` in our shuffling code instead of one between `i` and `N-1`. Show that the resulting order is *not* equally likely to be one of the $N!$ possibilities. Run the test of the previous exercise for this version.

1.1.38 *Binary search versus brute-force search.* Write a program `BruteForceSearch` that uses the brute-force search method given on page 48 and compare its running time on your computer with that of `BinarySearch` for `largeW.txt` and `largeT.txt`.