Exercises Two (Sept. 18): *Selected* Exercises.

(Try to solve the Following Problems as many as you can & submit your programs only to email program06@yeah.net in two weeks.)

1.3.3 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;
```

1.3.5 Write a program `RollLoadedDie` that prints the result of rolling a loaded die such that the probability of getting a 1, 2, 3, 4, or 5 is 1/8 and the probability of getting a 6 is 3/8.

1.3.7 Suppose that `i` and `j` are both of type `int`. What is the value of `j` after each of the following statements is executed?

a.
```
for (i = 0, j = 0; i < 10; i++)  j += i;
```
b.
```
for (i = 0, j = 1; i < 10; i++)  j += j;
```
c.
```
for (j = 0; j < 10; j++) j += j;
```
d.
```
for (i = 0, j = 0; i < 10; i++) j += j++;
```

1.3.8 Rewrite `TenHellos` to make a program `Hellos` that takes the number of lines to print as a command-line argument. You may assume that the argument is less than `1000`. Hint: Use `i % 10` and `i % 100` to determine when to use `st`, `nd`, `rd`, or `th` for printing the `i`th `Hello`.

1.3.9 Write a program that, using one `for` loop and one `if` statement, prints the integers from 1,000 to 2,000 with five integers per line. Hint: Use the `%` operation.

1.3.11 Describe what happens when you try to print a ruler function (see the table on page 57) with a value of `n` that is too large, such as `100`.

1.3.12 Write a program `FunctionGrowth` that prints a table of the values $\log n$, n, $n \log_e n$, n^2, n^3, and 2^n for $n = 16, 32, 64, \ldots, 2{,}048$. Use tabs (`\t` characters) to align columns.

1.3.13 What are the values of `m` and `n` after executing the following code?

```
int n = 123456789;
int m = 0;
while (n != 0)
{
   m = (10 * m) + (n % 10);
   n = n / 10;
}
```

1.3.14 What does the following code fragment print?

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

Solution. Even an expert programmer will tell you that the only way to understand a program like this is to trace it. When you do, you will find that it prints the values 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 134, 233, 377, and 610. These numbers are the first sixteen of the famous *Fibonacci sequence*, which are defined by the following formulas: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n > 1$.

1.3.15 How many lines of output does the following code fragment produce?

```
for (int i = 0; i < 999; i++);
{  System.out.println("Hello");  }
```

Solution. One. Note the spurious semicolon at the end of the first line.

1.3.18 Unlike the harmonic numbers, the sum $1/1^2 + 1/2^2 + ... + 1/n^2$ *does* converge to a constant as n grows to infinity. (Indeed, the constant is $\pi^2/6$, so this formula can be used to estimate the value of π.) Which of the following `for` loops computes this sum? Assume that `n` is an `int` variable initialized to `1000000` and `sum` is a `double` variable initialized to `0.0`.

a. `for (int i = 1; i <= n; i++) sum += 1 / (i*i);`

b. `for (int i = 1; i <= n; i++) sum += 1.0 / i*i;`

c. `for (int i = 1; i <= n; i++) sum += 1.0 / (i*i);`

d. `for (int i = 1; i <= n; i++) sum += 1 / (1.0*i*i);`

1.3.21 Modify `Binary` to get a program `Kary` that takes two integer command-line arguments `i` and `k` and converts `i` to base `k`. Assume that `i` is an integer in Java's `long` data type and that `k` is an integer between 2 and 16. For bases greater than 10, use the letters `A` through `F` to represent the 11th through 16th digits, respectively.

1.3.43 *Median-of-5.* Write a program that takes five distinct integers as command-line arguments and prints the median value (the value such that two of the other integers are smaller and two are larger). *Extra credit*: Solve the problem with a program that compares values fewer than 7 times for any given input.

1.3.44 *Sorting three numbers.* Suppose that the variables `a`, `b`, `c`, and `t` are all of the type `int`. Explain why 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; }
```

1.3.22 Write a code fragment that puts the binary representation of a positive integer `n` into a `String` variable `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. Working from PROGRAM 1.3.7, we get the solution

```
String s = "";
int power = 1;
while (power <= n/2) power *= 2;
while (power > 0)
{
   if (n < power) { s += 0;             }
   else           { s += 1; n -= power; }
   power /= 2;
}
```

A simpler option is to work from right to left:

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

Both of these methods are worthy of careful study.

1.3.35 *Checksum.* The International Standard Book Number (ISBN) is a 10-digit code that uniquely specifies a book. The rightmost digit is a checksum digit that can be uniquely determined from the other 9 digits, from the condition that $d_1 + 2d_2 + 3d_3 + ... + 10d_{10}$ must be a multiple of 11 (here d_i denotes the ith digit from the right). The checksum digit d_1 can be any value from 0 to 10. The ISBN convention is to use the character `'X'` to denote 10. As an example, the checksum digit corresponding to `020131452` is `5` since 5 is the only value of x between `0` and `10` for which

$$10{\cdot}0 + 9{\cdot}2 + 8{\cdot}0 + 7{\cdot}1 + 6{\cdot}3 + 5{\cdot}1 + 4{\cdot}4 + 3{\cdot}5 + 2{\cdot}2 + 1{\cdot}x$$

is a multiple of 11. Write a program that takes a 9-digit integer as a command-line argument, computes the checksum, and prints the ISBN number.

1.3.36 *Counting primes.* Write a program `PrimeCounter` that takes an integer command-line argument `n` and finds the number of primes less than or equal to `n`. Use it to print out the number of primes less than or equal to 10 million. *Note*: If you are not careful, your program may not finish in a reasonable amount of time!

1.3.38 *Exponential function.* Assume that `x` is a positive variable of type `double`. Write a code fragment that uses the Taylor series expansion to set the value of `sum` to $e^x = 1 + x + x^2/2! + x^3/3! + \ldots$.

Solution. The purpose of this exercise is to get you to think about how a library function like `Math.exp()` might be implemented in terms of elementary operators. Try solving it, then compare your solution with the one developed here.

We start by considering the problem of computing one term. Suppose that `x` and `term` are variables of type `double` and `n` is a variable of type `int`. The following code fragment sets `term` to $x^n / n!$ using the direct method of having one loop for the numerator and another loop for the denominator, then dividing the results:

```
double num = 1.0, den = 1.0;
for (int i = 1; i <= n; i++) num *= x;
for (int i = 1; i <= n; i++) den *= i;
double term = num/den;
```

A better approach is to use just a single `for` loop:

```
double term = 1.0;
for (i = 1; i <= n; i++) term *= x/i;
```

Besides being more compact and elegant, the latter solution is preferable because it avoids inaccuracies caused by computing with huge numbers. For example, the two-loop approach breaks down for values like $x = 10$ and $n = 100$ because 100! is too large to represent as a `double`.

To compute e^x, we nest this `for` loop within another `for` loop:

```
double term = 1.0;
double sum = 0.0;
for (int n = 1; sum != sum + term; n++)
{
   sum += term;
   term = 1.0;
   for (int i = 1; i <= n; i++) term *= x/i;
}
```

The number of times the loop iterates depends on the relative values of the next term and the accumulated sum. Once the value of the sum stops changing, we leave the loop. (This strategy is more efficient than using the loop-continuation condition `(term > 0)` because it avoids a significant number of iterations that do not change the value of the sum.) This code is effective, but it is inefficient because the inner `for` loop recomputes all the values it computed on the previous iteration of the outer `for` loop. Instead, we can make use of the term that was added in on the previous loop iteration and solve the problem with a single `for` loop:

```
double term = 1.0;
double sum = 0.0;
for (int n = 1; sum != sum + term; n++)
{
   sum += term;
   term *= x/n;
}
```