7

Semantics

Exercises

7.1 They are compiled and executed, resulting in an infinite loop.

In general infinite loops cannot be detected by any compilation system; such loops are instances of the *halting problem* in computer science, which in the general case is unsolvable.

7.2 Any loop statement that defines an infinite loop is semantically undefined. Similarly, any recursive function that does not provide a *base case* (and thus overflows the run-time stack) is also semantically undefined.

7.3

a. Part 23 of the *Java Language Specification* [Gosling 1996] defines "infinity" as a float or double value that results from division by zero. The Java expression `1.0/0.0` gives the predefined constant POSITIVE_INFINITY as a result, and `-1.0/0.0` gives the constant NEGATIVE_INFINITY. This is consistent with the IEEE 754 floating point standard. Also useful is the predefined constant `NAN` (short for Not A Number) in the Float class, which signals an expression that has no well-defined numerical result like `0.0/0.0`.

b. When `i` and `j` are float and `j` is not 0, the computation `i=3/j` assigns to `i` the `float` quotient. If `j` is 0, `i` is assigned the value POSITIVE_INFINITY.

7.4 Ignoring the size aspect and considering only primitive data types, the `/` operator in the expression `x + y/2` must be division. So there are basically two interpretations: an int expression and a float expression. The `+` represents an addition and the `/` represents either integer or floating point division.

The arithmetic types in C include int (16 or 32 bit), short int (16 bit), long int (32 or 64 bit depending on the processor), char, enumerations, float (32 bit), double (64 bit), and long double (128 bit). For more details, see [Kernighan and Ritchie, 1988].

In C++ and Ada, the expression might denote vector or matrix arithmetic if the operators `+` and `/` are properly overloaded. See [Stroustrup, 1991, p. 236] for an example of overloading in C++.

7.5 Repeats exercise 5.12; integer overflow for N = 13.

```
N:   F
2:   2
3:   6
4:   24
5:   120
6:   720
7:   5040
8:   40320
9:   362880
10:  3628800
11:  39916800
12:  479001600
13:  1932053504 <------------OVERFLOW
```

7.6 No error is reported, nor is the value preserved without loss.

7.7 No error is reported, nor is the value preserved without loss. Try running `Char1.java` in the *ch07code* directory.

7.8 The value is not preserved; you do not get the same value back. Try running `Float1.java` in the *ch07code* directory.

7.9 This is a very difficult problem. Input/output has not had the same study as the other features of imperative and object-oriented languages. Instead the I/O facilities of various languages were defined to meet the need of the hardware input/output devices at the time the language began. There are many subtle, idiosyncratic differences between the formatting conventions of the languages cited above.

7.10 Again this is a difficult problem for much the same reasons as cited in the previous problem. Here, even the term *unformatted* I/O is not well defined. Does it include only text I/O but without format codes? Does it include both text I/O without format codes and binary I/O? Both sequential and random I/O?

7.11
a. As a code fragment rather than a compilable C program, the code is valid.

b.

c.

7.12 C does not provide an exception handling mechanism. The I/O errors defined are particular to each compiler.

By default, in C++ errors involving streams do not generate exceptions. Exceptions can be enabled individually. Early on, functions were provided to determine the success of a particular operation.

7.13
a. The Java and C++ models for exceptions are similar, in that both are objects. Both use `try-catch` blocks to catch exceptions. They differ in that Java has many predefined I/O exceptions which C++ lacks.

   Also in Java, all exceptions derive from a common ancestor `Throwable` which C++ lacks. Java also checks that appropriate exception handlers are provided, which C++ does not.

b. Ada exceptions are not objects, but merely parts of the control flow of the program. Since exceptions are not objects, they have no state. In creating new exceptions, one cannot add new elements to the state of an exception.

7.14 Replacing the `while (true)` with `for (int i=0; i<3; i++)` eliminates the infinite loop. Then you can either return a default value or after the `try` terminate the program if `i` has the value 2.

7.15 See `Average.java` in the *ch07code* directory.

7.16 See `Stack.java` in the *ch07code* directory.

7.17 See previous exercise; replace overflow with underflow. Replace the `s.push` with `s.pop`.