5

Types

Exercises

5.1 See program `sizes.c` in the *ch05code* directory for the language C. As reported on a Power PC G5 Mac under OS/X 10.4 using gcc: short int - 2, int - 4, long int - 4. float - 4 double - 8, long double - 16.

5.2 Java supports an unordered boolean type that is not ordered. However, in Pascal this type was basically an enum in which false was less than true: false was equivalent to zero and true one. This proved to be not useful and moreover it has no basis in mathematical logic.

5.3 A common error in C-like languages is to use an = in a test where == is wanted. E.g., when searching an array for a value we can easily write:

```
index = search(a, SIZE, value);
if (index = -1) /* should be == -1 */
```

By insisting that tests be of type boolean, Java compilers are able to detect all such errors.

5.4 See `Float2Bits.java` or `float2bits.cpp` in the *ch05code* directory.

5.5
```
0.2 = 0011 1110 0100 1100 1100 1100 1100 1101
0.5 = 0011 1111 0000 0000 0000 0000 0000 0000
0.3 = 0011 1110 1001 1001 1001 1001 1001 1010
1.0 = 0011 1111 1000 0000 0000 0000 0000 0000
```

5.6
```
0.1 = 0011 1101 1100 1100 1100 1100 1100 1101
```

5.7 See the program `Int2Bits.java` in the *ch05code* directory.

5.8
```
0 =  0000 0000 0000 0000 0000 0000 0000 0000
1 =  0000 0000 0000 0000 0000 0000 0000 0001
```

```
2 =  0000 0000 0000 0000 0000 0000 0000 0010
3 =  0000 0000 0000 0000 0000 0000 0000 0011

-1 =  1111 1111 1111 1111 1111 1111 1111 1111
-2 =  1111 1111 1111 1111 1111 1111 1111 1110
-3 =  1111 1111 1111 1111 1111 1111 1111 1101
```

5.9 A *big-endian* machine, such as the Motorola M680x0 series, numbers both bits and bytes from left to right; i.e., from the most significant byte to the least.. So the bytes for a 32-bit integer are numbered: 0 1 2 3.

In contrast, a *little-endian* machine, such as the DEC Vax, numbered both bits and bytes from right to left; i.e., from least significant to most. So the bytes for a 32-bit integer are numbered: 3 2 1 0.

Consider the value 0x01234567. On a big endian machine this value would be treated as: 01 23 45 67, whereas on a little endian machine the value would be treated as: 67 45 23 01.

The data given in Table 5.2 assumes a big-endian machine.

5.10 In Ada both `mod` and `rem` compute the integer remainder when dividing. Both operators give the same result when both quantities have the same sign. They differ when one quantity is positive and the other negative. The `mod` operator provides the mathmatical mod operator. while the `rem` operator provides whatever the hardware does. The later is more efficient when you know that both operands are non-negative.

5.11 For example, the rules for C are given in [Kernighan and Ritchie, 1988, Chapter A6.5, p. 198] which defines precisely the usual arithmetic conversions. After specifying a number of special cases, the default case converts both operands to type `int`.

5.12
```
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
```

5.13 In C/C++ `enum`s are merely names given to integer values; for example, it is possible to take the largest value in an `enum` and add one to it producing

a value that is not in the specified `enum`. Nor do C/C++ support I/O on `enum`s.

In contrast, Java 1.5 `enum`s are classes, which provides type-safety, compile-time checking, and output.

5.14 The assignment statement `a = b` generates a compile error. One possible interpretation is that the statement copies the string `b` to the variable `a`. A second interpretation is that the statement assigns the value of the pointer `b` to the pointer `a`. This interpretation would be allowed if both variables had been declared `int *`.

5.15 (a) A special feature of one-dimensional arrays in Ada is the ability to denote a slice of an array. For example, if `T` is delcared as `String`, then the array slice `T[2..7]` extracts the second through seventh characters of `T`. The bounds can be any expression. The slice is null if the range is null.

(b) (1) Extracting a slice; similar to the `substring` operator for `Strings` in Java. (2) Assigning to a slice. (3) Passing a slice of an array as an argument in a function call.

5.16 a. Perl arrays are truly dynamic. Conisder the Perl fragment:

```
@a = {1, 2};
$a[7] = 8;
```

At the conclusion, the size of `a` is 8, with `a[2]` through `a[6]` all having the value `undef`.

b. Python requires that arrays be extended by using the `append` function. Thus, the Perl code above cannot be duplicated in a single Python statement.

An associative array is a hash table. Java's equivalent is a `HashMap`.

An example occurs in the Perl `mailgrades` program in which the userid is retrieved from the associative array `@ENV`.

In Perl and Python the index of an associative array is usually a string.

5.17 Independent of the languages chosen, such a report needs to be done carefully. For example, Scheme allows each element of a list to be a different type. In contrast, Haskell (following Pascal and Ada) requires all elements of a list to have the same type. This is very different from Java.

5.18

$$\textit{Structure} \rightarrow \text{struct } \textit{Identifier} \text{ \{ } \textit{Declarations} \text{ \}}$$

$$\ldots$$

$$\textit{StructureRef} \rightarrow \textit{StructureRef} \text{ . } \textit{Identifier} \mid \textit{Identifier}$$

The above EBNF defines a structre to be a list of variable declarations, enclosed in a `struct` declaration and given a name. A structure reference is a sequence of *Identifier*s separated by dots.

5.19 As indicated in the text, two records are structurally equivalent if they have the same structure independent of the names used. The algorithm merely has to step through the fields one at a time, checking to see if two records have the same number of fields and corresponding fields in both sequences have the same type.

5.20

5.21 Taking C as the example language and using [Kernighan and Ritchie, 1988, pp. 221-222] as the authoritative source, two types T1 and T2 are *type equivalent* if they contain the same set of type specifiers, taking into account that some specifiers can be implied by others. Structures, unions, and enumerations with different tags are distinct and a tagless union, structure, or enumeration specifies a unique type.

Two types are the same if their abstract declarators, after expanding any `typedef` types, and deleting any function parameter identifiers, are the same up to equivalence of type specifier lists. Array sizes and function parameter types are significant.

5.22