HOW enum WORKS

Internally, the compiler assigns an integer to each of the items in an enum list. For example, the statement below does not print small, medium, large, or jumbo to the screen. It prints either 0, 1, 2, or 3.

```
cout << drink_size << '\n';
```

WARNING

Attempting to print the value of an enumerated type results in an error on some compilers. Enumerated types are best used in expressions and switch structures, rather than directly for output.

By default, the compiler begins assigning integers with zero. For example, in the sizes type, small = 0, medium = 1, large = 2, and jumbo = 3. You can, however, choose your own values. For example, suppose you wanted to use an enumerated type to assign quantities. You could use a statement like the one below to declare an enumerated type with the values 1, 2, 12, 48, and 144.

```
enum quantity {Single=1, Dozen=12, Full_Case=48, Gross=144};
```

As another example, suppose you want to create a type called month that is made up of the months of the year. Because the months are commonly numbered 1 through 12, you decide to have the compiler assign those numbers to your list. The assignment in the first value of the list sets the beginning value for the items in the list, as in the statement below. January will be assigned the value 1, February the value 2, etc.

```
enum month {January=1, February, March, April, May, June, July,
            August, September, October, November, December};
```

You can use the fact that enum uses integers to your advantage. For example, a statement like the one below can be used with the sizes type.

```
if (drink_size > medium)
  { cout << "This drink will not fit in your cup holder.\n"; }
```

USING typedef

Another C++ feature which is related to enum is typedef. You can use typedef to give a new name to an existing data type. For example, if you prefer to use the term *real* to declare variable of type `float`, you can give the data type an *alias* of real with typedef.

```
typedef float real;
```

You should, of course, use typedef sparingly because you may confuse the reader of your code.

You can use typedef for more than just changing the names of data types to fit your liking. For example, recall from Chapter 4 that some C++ compilers include a boolean data type and some compilers do not. You can use typedef and a couple of constants to create your own boolean data type.

```
typedef int bool;
const int TRUE = 1;
const int FALSE = 0;
```

The three statements above make it possible to declare variables of type `bool` and assign values to the variables using `TRUE` and `FALSE`.

```
bool acceptable;
acceptable = TRUE;
```

On the Net

For more information about typedef, including how typedef can be used to make code easier to move among compilers, see http://www.ProgramCPP.com. See topic 10.3.1.

SECTION 10.3 QUESTIONS

1. Write a statement that declares an enum type called speed that allows the values *stopped*, *slow*, and *fast*.
2. Write a statement that declares a variable named rabbit of the type you declared in question 1.
3. Write a statement that assigns the value fast to the *rabbit* variable you declared above.
4. What does the compiler use internally to represent the values of an enum type?
5. Write a statement that declares an enum type called temperature that allows the values *frigid*, *cold*, *cool*, *mild*, *warm*, *hot*, and *sizzling*. Have the list begin with the value 1.
6. What can be used to give a new name to an existing data type?

PROBLEM 10.3.1

Make a list of several different enumerated data types that could be useful in programs. For example, `enum TrueFalse { FALSE, TRUE}` or `enum Position {open, closed}`.

CHAPTER 10, SECTION 4
Structures

C++ *structures* allow variables to be grouped to form a new data type. The data elements in a structure are arranged in a manner that is similar to the way database programs arrange data.