CHAPTER 3

Arduino C Data Types

Arduino C supports most of ANSI C’s data types with a few notable exceptions. Also, there’s a little hanky-panky going on with floating point numbers, but it shouldn’t be a problem as long as you are aware of what’s going on “under the hood.”

As mentioned in Chapter 2, a variable is little more than a chunk of memory that has been given a name. When you define a variable, you must also tell the compiler what type of data is to be associated with that variable. The data type of the variable is important because it determines how many bytes of memory are dedicated to that variable, and what kind of data can be stored in the variable. As you will learn later in this chapter, there are two basic types of data: value types and reference types. If the variable is defined as a value type, then there is a very specific range of values possible, too.

A list of the basic value data types is presented in Table 3-1.

Table 3-1. Arduino C Value Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Byte length</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>1</td>
<td>Limited to logic true and false</td>
</tr>
<tr>
<td>char</td>
<td>1</td>
<td>Range: –128 to +127</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1</td>
<td>Range: 0 to 255</td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
<td>Range: 0 to 255</td>
</tr>
<tr>
<td>int</td>
<td>2</td>
<td>Range: –32,768 to 32,767</td>
</tr>
<tr>
<td>unsigned int</td>
<td>2</td>
<td>Range: 0 to 65,535</td>
</tr>
<tr>
<td>word</td>
<td>2</td>
<td>Range: 0 to 65,535</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>Range: –2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4</td>
<td>Range: 0 to 4,294,967,295</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>Range: –3.4028235E+38 to 3.4028235E+38</td>
</tr>
<tr>
<td>double</td>
<td>4</td>
<td>Range: –3.4028235E+38 to 3.4028235E+38</td>
</tr>
<tr>
<td>string</td>
<td>?</td>
<td>A null (‘\0’) terminated reference type data build from a character array</td>
</tr>
<tr>
<td>String</td>
<td>?</td>
<td>An reference data type object</td>
</tr>
<tr>
<td>array</td>
<td>?</td>
<td>A sequence of a value type that is referenced by a single variable name</td>
</tr>
</tbody>
</table>
Keywords in C

Each of the types shown in Table 3-1 (i.e., boolean, char, int, etc.) are keywords in C. A keyword is any word that has special meaning to the C compiler. Because keywords are reserved for the compiler's use, you cannot use them for your own variable or function names. If you do, the compiler will flag it as an error. If the compiler didn't flag such errors, then the compiler would be confused as to which use of the keywords to use in any given situation.

Variable Names in C

If you can't use keywords for variable or function names, then what can you use? There are three general rules for naming variables or functions in C: Valid variable names may contain:

1. Characters a through z and A through Z
2. The underscore character (_)
3. Digit characters 0 through 9, provided they are not used as the first character in the name.

Just about everything else is not acceptable, including C keywords. That also means that punctuation, and other special non-printing characters are not allowed either. Valid variable names might include:

- jane
- Jane
- ohm
- ampere
- volt
- money
- day1
- Week50
- _system
- XfXf

Using the same rules, the following would not be valid names:

- ^carat
- 4July
- -negative
- @URL
- %percent
- not-Good
- This&That
- what?

Given these limits, how does one create a “good” variable name? As a general rule, I like variable names that are long enough to give me a clue as to what they do in a program but short enough that I don’t get tired of typing their name. Another convention a lot of programmers used is a variant of what’s called camel notation. Using this notation, variable names begin with a lowercase letter with each subword capitalized. Examples using this style might be:

- myFriend
- togglePrinter
- reloadEmptyPaperTray
- closeDriveDoor

I think this style makes it easy to read the variable names. C could care less which style you use. However, keep in mind that it is unlikely that you will write perfect (error-free) code every time you write a program. Using variable names that make sense and are easy to read makes debugging just that much easier. (Keep in mind that C is case sensitive, which means that myData and MyData are two different variables.)

With that in mind, let’s examine the common data types available for use in your C programs.