Choosing Symbolic Constants Over Integer Variables in Source Code

Roy Vanegas
Interactive Telecommunications Program
New York University
r.o.y@nyu.edu

March 6, 2007

Using symbolic constants [1] in place of integer variables makes source code easier to read and creates programs that require less memory. The naming convention of using all capitals in the declarations of constants, such as `LED` vs `led`, establishes a clear distinction between constants and variables. The former are defined as such:

```
#define name replacement_text
```

When `name` is encountered in the compile process, it is replaced by `replacement_text`. Consider the following Arduino code fragment, which uses a constant:

```
#define LED 13

int loopCount = 0;

void setup()
{
   pinMode( LED, OUTPUT );
}

void loop()
{
   digitalWrite( LED, HIGH );
   delay( 1000 );
   digitalWrite( LED, LOW );
   delay( 1000 );
   ++loopCount;
}
```

[1] I will refer to symbolic constants as simply constants throughout this paper.

Now consider its analog, using a variable:

```
int ledPin = 13;

void setup()
{
   pinMode( ledPin, OUTPUT );
}

void loop()
{
   digitalWrite( ledPin, HIGH );
   delay( 1000 );
   digitalWrite( ledPin, LOW );
   delay( 1000 );
}
```

In the first program, every occurrence of `LED` is *replaced* by 13; memory is not set aside for it. (`HIGH`, `LOW`, and `OUTPUT` are "pre-defined constants" endemic to the Arduino language [2].) The capitals indicate that `LED` is a constant whose value will not change throughout the program, distinguishing it from a variable[3] such as `loopCount`, whose value *does* change in the program.

In the second code fragment, the value assigned to variable `ledPin` on the first line remains unchanged throughout the program. However, it takes an integer's worth of memory unnecessarily. (An integer is 4 bytes in Java. On a machine with 1GB of RAM, that is negligible; on an Arduino with 8K or 16K of RAM, it's considerable.)

Another case for the constant: code debugging. Using constants in place of integers where values don't change protect against accidentally modifying the constants' value. For instance, if the second program contained hundreds of lines of source code, and a programmer had modified the program somewhere deep in the code tree, he or she could have accidentally modified the `ledPin` variable unknowingly to another value. The compiler would not have thrown an error, leading to logic errors. Employing a constant in its place, however, would have generated a compiler error[4], and thus would have guarded against accidentally changing that constant's value.

In closing, if you are using variables in your Arduino source code where their values remain unchanged throughout your program, employ constants in their place to conserve memory and enhance readability.

[2] `http://www.arduino.cc/en/Reference/HomePage`

[3] The definition of a variable is that it's value will change, hence the title variable.

[4] A compiler error is thrown when the grammar of a language is defeated, causing the compiler to halt the final build of the executable or binary. In contrast, a logic error is one that defeats logic, not grammar. Consequently, the compiler completes the process of building the executable or binary, but yields unexpected results in logic.

Acknowledgements

Many thanks to Tom Igoe and Nick Hasty for edits and suggestions during the revision process of this paper.