Chapter 2: Numbering Systems

Notes

- Least/most significant bit.
- Basic mathematics (addition, subtraction, multiplication, division) with binary and hexadecimal.
- Hex is a more convenient notation than binary because its numbers are smaller and each digit contains more than two values.

Introduction

Before moving on to explain how a computer stores information (such as instructions, or your program's *source* files), I must explain numbers and numbering systems. I must warn you that this can be a confusing topic to many people and a great hurdle that must be leapt over. The concepts covered in this chapter *must* be understood if you are to understand programming. Only *extremely* basic math skills are required (I should know, I'm in that category!); you only need to be able to add and subtract. If you feel stumped and this chapter does not help you, I encourage you to write me so we can work out what it is that is blocking you from understanding.

What is a *number*? Ever since you learned to think for yourself you've been pummeled with these things called *numbers*. In school you're forced to learn how to write them, add them, and do any number of torturous tasks with them. Later in life you'll use them to balance your checkbook or see how much in debt you are. But what *are* they? A number in is simply a representation of a value or count. You can represent a numeric value using one of many *numbering systems*. These systems define how a value is represented, obviously. Two that you may know of already are decimal and Roman Numerals. But then there are others with no names. If I held up two fingers I would be representing a count or *amount* of two fingers which is the same as the number 2. As you can see, I can write in long-hand, hold up fingers, write numbers, or use any number (pun pun) of ways to communicate a numeric value. It might be thought of, then, as a universal language. You'll even note that numbers everywhere seem to be represented the same (in decimal) even though words are completely different.

Most of our numeric abilities are automatic ever since elementary school. You see 6 and you can automatically discern that it represents an amount of six or six things:

Six = ☺ ☺ ☺ ☺ ☺ ☺

Regardless of how six is represented, the *value* still stays the same. In decimal we represent this as '6', long-hand as 'six', and Roman Numerals 'VI'. They all look differently, but mean the same thing. This is an important thing to remember. *How you display a numeric value does not change the amount it represents.*

The Decimal Numbering System

Decimal is a very logical system for representing numeric values and as such it has spawned many other numbering systems, for good or ill. Because of this I will first try to explain how decimal actually works.

The decimal numbering system uses *Arabic* numerals. A numeral is a character or glyph that represents a single numeric amount. The Arabic numerals are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The amounts they represent are below:

0 = <nothing>
1 = ☺
2 = ☺☺
3 = ☺☺☺
4 = ☺☺☺☺
5 = ☺☺☺☺☺
6 = ☺☺☺☺☺☺
7 = ☺☺☺☺☺☺☺
8 = ☺☺☺☺☺☺☺☺
9 = ☺☺☺☺☺☺☺☺☺

A decimal number is comprised of one or more of these numerals strung together. Each column's numeral is known as a *digit*. A decimal digit can, then, only be one of these *ten* different numerals and so multiple digits are needed to represent values greater than the largest of these numerals (nine).

In the case of multiple digits, the value is produced by adding the values of each digit together. However, there is a catch. Each digit's value is its numeral amount (as shown previously) *multiplied by its weight*. A digit's *weight* is ten to the power of the digit's column position. The right-most digit is in position zero (0). The next digit, to the left, is in position one (1) and so forth. The first digit is always the right-most and the last is the left-most. In the decimal number 1234, the positions are as so:

Decimal Digit:	1	2	3	4
Position:	3	2	1	0

So then the *weight* of each of these digits would be:

Weight:	10^3	10^2	10^1	10^0

And therefore each digit's value would be calculated as such:

Calculation:	$(1*10^3)$	$(2*10^2)$	$(3*10^1)$	$(4*10^0)$

So the value of each digit would be:

Value:	1000	200	30	4

Add them together (1000 + 200 + 30 + 4) and you get 1234 or "one thousand, two hundred thirty-four". Now, you already know this because you've used decimal all your life. And I bet you've taken it and this process completely for granted.

Base Systems

Decimal sure does use a lot of "ten" doesn't it? Yes! Decimal is known as "base ten"! The base of a logical numbering system like decimal determines the following:

- The weight of each digit is the base number to the power of the column position.
- The amount of numerals used by the system, starting with the numeral 0 (zero).

Other numbering systems I will be describing follow all the same rules, but they have a different *base*. In these numbering systems zero is counted. Each digit then can hold as many values as the base of the numbering system. Decimal is base ten and therefore can hold one of ten values in each digit. This is because, again, zero is counted. A digit in decimal can hold one of its ten Arabic numerals and therefore represent one of ten different values. However, the maximum *amount* of each is always one *less* than the base (nine in the case of decimal).

The Fivemal System

To explore this generic use of system *base* (not "boom boom boom" bass, dammit!), let's create an imaginary numbering system that is *base five* called Fivemal. I'm choosing this amount because it is half of ten and therefore might be pretty easy to understand. Because it's base five, the weight of each digit will be five to the power of the digit's position. All numbers written in Fivemal will have '(f)' appended to them so you can see which ones are not Decimal. For my first demonstration I will represent the value three (☺☺☺) in Fivemal:

3(f)

Wow, that was exceedingly painless. It still looks the same and only uses one digit because we have not equaled or exceeded our base which is five (☺☺☺☺☺). What happens if we do? What if we want to show the value six (☺☺☺☺☺☺)? It would like look the following in Fivemal:

11(f)

Woh, wait a minute! That looks like eleven doesn't it? In *decimal* that *would* be eleven, but that's because decimal is *base ten*. Our own numbering system, Fivemal, is only *base five*. The value six, then, must use two digits. Each digit's weight in Fivemal will be a

multiple of five. Simply "knowing" the numbers doesn't help much now does it? Now you have to figure out the value of the number by actually using the rules I tried to explain earlier. Let's break it down. First we'll look at the weight of each digit:

Fivemal Digit:	1	1
Weight:	5^1	5^0

Since the digit's value is its numeral amount multiplied by it's weight, the calculation for each digit's value would be:

Calculation:	$(1*5^1)$	$(1*5^0)$

Added together this would be 6 in decimal. Please remember that '6' in decimal is simply a way of representing the same value (☺☺☺☺☺☺) written as '11(f)' in Fivemal. Fortunately, any time we are converting from a foreign system like Fivemal, all numbers used in the calculation are typically written in decimal. The only one *not* written in decimal is the one whose digits are being converted. Let's look at the conversion to decimal from Fivemal from the top:

11(f) =
$(1*5^1) + (1*5^0)$ =
5 + 1 =
6 (☺☺☺☺☺☺)

Notice that the only number in this whole process that *isn't* decimal is the '11(f)' at the top. This process can be used to convert a number from an abstract base to decimal, or base ten. It can also be used to split up a decimal number as we saw earlier, but then it's yield will always be that same decimal number. For example, let's break down one hundred thirty-five in decimal using this process (and I'm not going to placate you by putting down that many smileys!):

135 =
$(1*10^2) + (3*10^1) + (5*10^0)$ =
100 + 30 + 5 =
130 + 5 =
135 (*many* smileys would go here …)

Distinguishing Numbering Systems

You need to know what numbering system a number is using to be able to use it effectively. But how do you know what system a number is in? Most systems add something to the display of the number in order to convey the system. In Fivemal I added '(f)' to the end of the number to show that it was represented "in Fivemal". Decimal is the *only* one that doesn't. This means that if there are no distinguishing marks on the number (strange spacing, leading/ending characters, etc.) then it is most definitely

a decimal number. For the numbering systems I will be describing I will also show you how they are written so they can be distinguished from decimal and their counterparts.

The Binary System (base 2)

This is the simplest system of all because its base is the lowest you can go. Yup, that's right. There is none lower. Base one can't possibly exist because each digit would *have* to be zero and therefore the highest number a base one system could represent would be zero (nada, nothing, zilch). In base two, as it is, each digit can only be either 1 or 0. For example, the number 5 in binary is:

0101b

First let me explain how binary is distinguished from other systems in writing. First off, the number of digits will almost always be a multiple of four. If that many digits are not required, the number is padded with zeroes. Secondly, any one of the following things may be used, so if you see any one of these on a number, it is probably written in binary:

- Every four digits (to the left) there is a space.
- The number ends with a 'b' or 'B'.
- The number begins with a percent sign '%' (don't ask me … I never use that one).

Now, let's look at some numbers written in binary. In fact, let's go back to our value five (☺☺☺☺☺) in binary:

Binary Digits:	0	1	0	1
Weight:	2^3	2^2	2^1	2^0
Value:	$(0*2^3)$	$(1*2^2)$	$(0*2^1)$	$(1*2^0)$
Calculation:	0 +	4 +	0 +	1

Each digit in binary is known as a *bit* (**BI**nary digi**T**) and the weight of each also has a label of *bit significance*. The two most common labels are "least significant bit" and "most significant bit". This significance is based on the weight of a particular binary digit. If the bit has the smallest weight, it is known as the least significant bit, and vice versus for the most significant. Thus the left-most bit is most significant because it has the largest weight and the right-most is least significant. This labeling is sometimes needed to determine the order when a binary number might be written backwards according to what I have shown above.[1]

Because each bit can only be a one (1) or zero (0), the value of a binary number can be calculated by simply adding up the weights of all of the *on* bits (i.e. bits containing one (1)). Thus, the maximum amount a binary number can be is limited to the sum of all of

[1] Binary numbers whose least and most significant bits are reversed (left = lease, right = most) are rare and will not be covered in this book.

the bit weights. For example, given four bits you can only have a value as large as fifteen (15 = 1111b = 8 + 4 + 2 + 1).

The Hexadecimal System (base 16)

The hexadecimal system is very important because it is a relative bridge between complete gibberish (binary) and that which we know by heart (decimal).

The Dead System (Octal, base 8)

Blah

Hex <> Binary Conversions

Converting between hexadecimal and binary is one of the easiest conversions, because both systems are based on the powers of two (2). Every four (4) binary digits represents *one* (1) hexadecimal digit and vice versa. Thus, to convert a binary number to hexadecimal, you can do it in blocks of four digits and to convert from hexadecimal to binary you expand every digit to four bits.

Arbitrary Conversions

You've already actually learned how to convert from any based system to decimal. But how to do you convert from any based system to another abstract base system or even from decimal to another based system? This is where it starts to get difficult.

To be done here:

- Similar and arbitrary (not so similar ☺) systems
- Converting between similar systems
- Converting between arbitrary systems