Contents

1 Binary **1**

1.1 Binary numbering 1

1.2 A look at variables being stored in binary in C 2

1.2.1 Bit terminology 3

1.2.2 Endianness 3

2 Hexadecimal **3**

2.1 Hexadecimal numbering 3

3 Different base literals in C **4**

4 Converting different base numbers to decimal **5**

1 Binary

1.1 Binary numbering

Binary is a base 2 numbering system. This means that there are only two digits: 0 and 1. In contrast, we generally use a base 10 numbering system, or **decimal**, which has 10 digits: 0 through 9.

Just like how adding 1 to 9 in decimal results in 10, adding 1 to 1 in binary results in 10. This is because, once the highest digit of a number is surpassed, a larger digit is incremented from 0 and added to the left. To show this off more, here is an example of counting in binary:

Decimal number	Binary number
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000

1.2 A look at variables being stored in binary in C

As a reminder, C stores the values of variables in computer memory, which is one long series of **bits**, or spaces that can hold the value 0 or 1. On most modern machines, every 8 bits constitutes one **byte**, and variables in C take up a whole number of bytes.

C stores numbers in binary directly in the bytes where a variable's value is stored. Within a single byte, the binary value can be read left to right:

```
// Comments here will indicate x's value in binary

unsigned char x = 0;
// x is 00000000

x = 1
// x is 00000001

x = 2
// x is 00000010

x = 4
// x is 00000100
```

1.2.1 Bit terminology

The **most significant bit** or **MSB** is the largest digit, or the leftmost bit when reading a binary number from left to right. It is called the most sigificant bit because changing it will result in a significant change in value.

The **least significant bit** or **LSB** is the smallest digit, or the rightmost bit when reading a binary number from left to right. It is called the least significant bit because changing it will result in an insignificant change in value.

$$^{\mathrm{MSB}\rightarrow}00010111^{\leftarrow\mathrm{LSB}}$$

1.2.2 Endianness

When a variable takes up multiple bytes, the byte with the most significant digits is the **most significant byte** and the byte with the least significant digits is the **least significant byte**. Not all computers store the bytes of a variable in the same order, however.

Endianness is a property that indicates the order that bytes are stored in on a machine. On a **big endian** machine, the most significant byte of a variable is stored first. On a **little endian** machine, the least significant byte is stored first.

If you are writing a program that only runs on one computer, you don't have to worry about endianness, but if you are writing a program that writes files to be read on another computer or a program that communicates with others computers over a network, you might have to, just incase the computers have different endianness.

2 Hexadecimal

2.1 Hexadecimal numbering

Hexadecimal is a base 16 numbering system. It incorporates digits 0 through 9 and A through F, where A=10, B=11, C=12, D=13, E=14 and F=15.

This table shows counting in hexadecimal:

Decimal number	Hexadecimal number
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	a
11	b
12	c
13	d
14	e
15	f
16	10
17	11
18	12
19	13
20	14

3 Different base literals in C

To write a number literal in hexadecimal, prefix your literal with `0x`

```
int x;

// These two lines do the same thing
x = 15;
x = 0xf;

// You can write the letter digits of a hexadecimal number
// in uppercase or lowercase
x = 0xF + 0xa + 0xee;
```

To write a number literal in octal (base 8), prefix your literal with `0`. Octal isn't used often, but it's good to know incase you see it.

```
int x;
```

```
// This will store 35 in OCTAL in x,
// not 35 in DECIMAL
x = 035;
```

4 Converting different base numbers to decimal

To convert a number of a different base to decimal:

1. Number each digit from right to left starting from 0
2. Multiply each digit by the base of the number to the power of the number placed above each digit
3. Sum each product

Binary to decimal:

$$
\begin{aligned}
&01101101 \\
&0^7 1^6 1^5 0^4 1^3 1^2 0^1 1^0 \\
&0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\
&= 0 + 64 + 32 + 0 + 8 + 4 + 0 + 1 \\
&= 109
\end{aligned}
$$

Hexadecimal to decimal:

$$
\begin{aligned}
&43\mathrm{fe} \\
&4^3 3^2 \mathrm{f}^1 \mathrm{e}^0 \\
&4 \cdot 16^3 + 3 \cdot 16^2 + 15 \cdot 16^1 + 14 \cdot 16^0 \\
&= 16384 + 768 + 240 + 14 \\
&= 17406
\end{aligned}
$$