File Systems and Data

Learn C With Babbo

File Systems

A **file system** is a system for the organization, storage, and retrieval of data on a computer. Almost invariably you will find that modern filesystems consist of **files** and **directories** (or, less formally, **folders**). Files are, fundamentally, named pieces of data on a file system. Files have a **size** indicating the number of **bytes** they consist of, which I will elaborate on in a later section. Directories, on the other hand, are components of your file system which have links to files and other directories. Likely you already have a good idea about files and directories just from navigating your file system with your operating system's file manager. On Windows, if you press the Windows key and type "My Computer" or "This PC" and selected the first option, you'll see your C drive (unrelated to the programming language), which you can navigate into and explore all the files and directories on your system. On Windows, this interface is called **Windows Explorer**. Mac users may navigate their file system using the **Finder** app, and on Linux systems, there are several graphical file system managers. As I said, you are probably already familiar with the file system manager that you use, but I want to bring it up so that you understand that when I am talking about file systems I am talking about the organization of files into directories that you find on modern computers.

Bits and Bytes

A byte is the smallest amount of data that may be stored in memory. Bytes themselves consist of 8 **bits**. Abstractly speaking, a bit is an on-off switch: it is always in one of two states: A bit is either "on" or "off", or one might say a bit indicates "yes" or "no", or that a bit is either "one" or "zero." Bits are the smallest pieces of information available on a computer. A byte is the smallest chunk of data that may be *stored*, which is reflected in the fact that the smallest non-empty file that may exist on a file system has a size of one byte, not one bit. All data on a computer, whether they are names of programs, programs themselves, numbers, words, or anything else, are represented as series of bytes. The next three sections will describe how numbers, words, and programs are represented this way.

Representation of Numbers

We normally represent numbers is through the base 10 **decimal** system, whereby the 10 digits from 0 to 9 are written in a sequence wherein they are multiplied by a power of ten according to their position in the sequence, and then summed together to represent a number. For example, the number 7401 may be digitally decomposed as follows:

$7401 = 7000 + 400 + 0 + 1 = 7 * 1000 + 4 * 100 + 0 * 10 + 1 * 1 = 7 * 10^3 + 4 * 10^2 + 0 * 10^1 + 1 * 10^0$

(Note that any number to the power of zero is 1). The decimal system is called base 10 because numbers are decomposed into powers of 10. Indeed, numbers may be decomposed in an almost identical way for any positive integer that is at least 2. A base 4 number system, for example, uses only the digts 0, 1, 2, and 3. The number 99 in base four, for example, is represented as 1203_4, where the subscript 4 is used to denote that this is a base 4 and not a base 10 representation. To see why, let's decompose 1203_4 into its base 4 digits:

$1203_4 = 1 * 1000_4 + 2 * 100_4 + 0 * 10_4 + 3 * 1_4 = 1 * 4^3 + 2 * 4^2 + 0 * 4^1 + 3 * 4^0 = 1 * 64 + 2 * 16 + 0 * 4 + 3 * 1 = 64 + 32 + 0 + 3 = 99$

Note carefully how $1000_4 = 4^3, 100_4 = 4^2, 10_4 = 4^1, 1_4 = 4^0$: this is completely analogous to how 1000, 100, 10, and 1 are powers of 10 in the decimal system.

So why am I even talking about bases? Because bits give us a convenient mechanism for representing numbers in base 2, commonly called **binary**. This is because every bit could be considered a base 2 digit, of which there are only 2: 0 and 1. The number 13 has the binary representation 1101_2 as we may see by expanding 1101_2 as

$1101_2 = 1*2^3+1*2^2+0*2^1+1*2^0 = 1*8+1*4+0*2+1*1 = 8+4+0+1 = 13$

This is indeed how numbers are primarily represented in computers, with variations for negative numbers and fractional (usually called "floating-point") numbers. When decimal (again, base 10) numbers are displayed on your computer screen, such as in a calculator app, your computer is converting bits, which may essentially be seen as base 2 digits, to base 10 digits, which itself would turn out to be a sequence of bytes where every byte represents one base 10 digit, more details of which we'll get into in the next section.

Representation of Text

It turns out that the ability of bits and bytes to represent numbers also gives us an easy way to represent text. One common way of doing so is through **ASCII**, which stands for American Standard Code for Information Interchange. In this system, each number which may be represented in one byte has a symbolic counterpart, which may be a letter, a textual representation of a digit, a symbol, or a space. Note that a byte, being composed of 8 bits, may represent a number as large as 255, which is $11111111_2 = 2^8 - 1$. This is analogous to the fact that the largest number that may be represented with 8 decimal digits is $99999999 = 10^8 - 1$. The symbolic counterparts may be found in an **ASCII Table**, shown below:

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

The two most relevant columns for us are "Dec" and "Chr", "Dec" corresponding to the numeric value of the byte, while "Chr" corresponds to the character representation. For example, note that 65 in the Dec column corresponds to "A" in the "Chr" column, so the same assortment of bits that represents the 65 may also represent the capital letter "A". Also, we can find that the number 53 corresponds to the digit "5": this gives us a way to represent base 10 numbers in binary. For example, to display the number "7301", we may use the bytes with numeric values 55, 51, 48, and 49, which correspond to the digits 7, 3, 0, and 1 respectively. Notice also that every symbol that may be typed with standard keyboards is also in the ASCII table, including the exclamation point, plus symbol, asterisk, etc, along with a space character, with value 32. Somewhere in any text editor you use, such as Microsoft Word, is a series of bytes corresponding to the contents of that program, and standard characters will be stored according to the ASCII table.

I would like to stress that bits and bytes fundamentally represent neither numbers nor text: as I said before, bits are fundamentally one of two states, and bytes are composed of 8 bits. How we interpret sequences of bits and bytes ultimately depends on context.

Representation of Programs

I'll conclude this lesson of how bits and bytes represent data by describing how they represent programs. Unlike numbers and text, programs are made to be understood by computers, not by humans (note in this case I am not talking about the source code used to create the program, but the program itself). In doing so, it uses bytes that represent different instructions natively understood by your computer, creating an execution blueprint that your computer understands. Because of this, the content of programs is not readable by most humans, though dedicated individuals could learn to read it in principle. Files not meant to be read by humans are usually called **binary files**, and include other kinds of files such as compressed data. At the risk of sounding repetitive, *all* data on a computer is composed of bytes.