ASCII Character Set

Computers can only store numbers. Humans need words to communicate, so how can we make that happen? We can assign a numeric code to each letter we wish to represent. That leads to the questions: who assigned the numbers and what letters should be included?

In the beginning, the were no answers, and anyone could make up the answers they wanted to use. IBM has what they called EBCDIC, which you don't hear of much anymore.

The manufacturers of equipment got together to figure out how to have one answer for everyone, resulting in the American Standard Code for Information Interchange (ASCII). This resulted in 128 possible characters (using seven bits for data and one more called the parity bit to be used to facilitate communications). This works real good if you you use the Roman alphabet. For some reason, it does not work so well with Chinese, Japanese, Arabic, or a bunch of others. (To overcome those problems, we now have Unicode, which is capable of expressing all alphabets, with ASCII as a subset.)

Normally, ASCII values 0 through 31_{10} are non-printable characters. So is the value 127_{10}. The space character has the value of 32_{10}, which is printable -- you just can't see it!

ASCII Chart

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
32	20		64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p

49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	U	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5A	Z	122	7A	z
59	3B	;	91	5B	[	123	7B	{
60	3C	<	92	5C	\	124	7C	|
61	3D	=	93	5D	]	125	7D	}
62	3E	>	94	5E	^	126	7E	~
63	3F	?	95	5F	_	127	7F	

Important Notes!

Numeric Conversion

One of the tricks of the trade is that the character '0' is not the value 0. But since the numbers run consecutively, you can get the binary value of a character by subtracted the value of the character '0'. Thus:

'9' - '0' = 9
$39_{16} - 30_{16} = 9$
$57_{10} - 48_{10} = 9$

You **MUST** remember, when you are entering numeric values from the keyboard, they are in character form! You **MUST** convert them to binary form before attempting any mathematically operation. In C, this conversion is so painless, you never think about it, but when you have something like,

```
scanf("%d", myInt);
```

that conversion is implied in the "%d"!

Case Conversion

Another trick is that you can convert a lowercase letter to a uppercase letter by subtraction and addition.

'j' - 'a' + 'A' = 'J'
$6A_{16} - 61_{16} + 41_{16} = 4A_{16}$
$106_{10} - 97_{10} + 65_{10} = 74_{10}$

The difference between any uppercase letter and the corresponding lower case letter is only one bit.

$a = 41_{16}$ or 01000001
$A = 61_{16}$ or 01100001

This means that the bitwise logical operators of AND, OR, and XOR can also be used for the conversions....This will be covered later in the course.

Previous | Next