Characters in a computer.

Computers can only work with numbers, so each character is assigned a number. The assignment is known as ASCII (American Standard Coding Information Interchange) encoding.

This encoding is documented at www.asciitable.com which shows the relationship between the 7-bit numbers and characters. Included here is a similar table showing the Decimal, Hexadecimal, Octal and character.

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

If you look at the table, there are a variety things I would like to point out. First note that numbers 0 to 9 are best viewed as hexadecimal numbers 0x30 to 0x39. In other words, the last digit is equal to the number in question.

Similarly if we look at the capital letters, we see that 'A' is 0x41. However lower case 'a' is 0x61. Write these out in binary we have

0100 0001 'A'
0110 0001 'a'

We see that the difference is bit 5 is set for lower case, this is true for all characters.

Note there are a lot of these types of relationships and a lot of ways that ASCII characters can be manipulated. As you work more with characters you will learn a lot of tricks.

Characters in c programming. The c language has a type named char, which are eight bit numbers,

```
char a, b[5], c[] = "String of Characters";
int digit = 5;

a = 'A'; // Single quote means just that character.
a = 65; // This will load a with 'A'
a = 0x41; // This also loads a with 'A'

b = "A0";  // WILL NOT WORK

b[0] = 'A';  // Loads first element with A
b[1] = '0';   // Loads second element with 0
b[2] = 0;    // Terminates string

// Operations can be applied to these numbers.

b[0] |= 0x20;    //  Oper. in binary    0100 0001  'A'
                 //  Mask               0010 0000   0x20
                 //  OR force bit high  0110 0001  'a'

b[0] &= ~0x20;   // Binary              0110 0001  'a'
                 // Mask                1101 1111  ~0x20
                 // AND force bit low   0100 0001  'A'

b[1] = '0' + digit; // Written out      0011 0000   '0'
                    // Adding in digit  0000 0101   5
                    // Results          0011 0101   '5'
```

As an example of how we can manipulate characters and some new programming constructs, we will write two function that will convert a string of characters to upper case or lower case.

```
// function that will convert a string to all
// upper case, using a for loop.
void ToUpperCase(char string[])
{
      int index; // variable that indexes into array.
      // for loop to move through string
      for (index = 0; string[index] != 0; index++)
      {
            // check to see if current character
            // is a lower case letter.
            if (string[index] >= 'a'
                  && string[index] <= 'z')
            {
                  string[index] &= ~0x20; // Converts to Upper Case
            } // end of lower case if

      }// end of for loop

}// end of ToUpperCase function
```

Note the "for" loop has three parts, "index = 0;" is the initialization step, "string[index] != 0" is the test if the loop should continue, and "index++" is the operation that is done prior to the next pass of the loop. This operation can also be done with a "while" loop as shown in the next function, which converts a string to all lower case.

```
// function that will convert a string to all
// lower case, using a while loop.
void ToLowerCase(char string[])
{
	int index; // variable that indexes into array.
	// Start at first character
	index = 0;
	// while loop to move through string
	while (string[index] != 0)
	{
		// check to see if current character
		// is a upper case letter.
		if (string[index] >= 'A'
			&& string[index] <= 'Z')
		{
			string[index] |= 0x20; // Converts to lower Case
		} // end of upper case if

		// move to next character
		index++;

	}// end of while loop

}// end of ToLowerCase function
```

Some programming groups, prefer the for loop over a while, since it readily documents how the loop is set up. In the while, it is often easy to forget the initialization or the end of loop statement.

Note functions can return a number to the calling operation. For example, lets look at some code that will look at a character array and return to us the length of the string held in the character array. Remember that the end of string is marked with a NULL or zero character. Thus we simply look through the array until we find a NULL character.

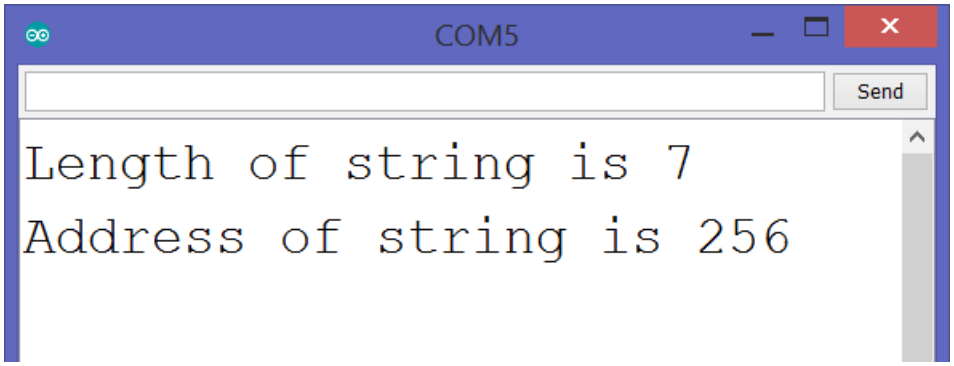
```
// function to return the length of a character string
int StrLen(char inString[])
{
	int k = 0; // Start at 0, or first character.
	while (inString[k] != 0) // if the k'th character is not zero
	{
		k++;			// move on to next character.
	} // End of for loop.
	return k; // Once we hit the zero character, we return the k value
	// which is the number of none zero characters,
} // End of StrLen

int SinglePassFlag = 1;
void setup()
{
	// put your setup code here, to run once:
	Serial.begin(9600); // Set Baud rate for serial.
} // End of setup

// put your main code here, to run repeatedly:
void loop()
{
	// String to test StrLen function.
	char TestString[] = "AbCd Ef";

	if (SinglePassFlag == 1)
	{
		// Test StrLen with 7 character fixed string,
		// printing results.
		Serial.print("Length of string is ");
		Serial.println(StrLen(TestString));
		Serial.print("Address of string is ");
		Serial.println((int)TestString);
		SinglePassFlag = 0; // clear flag to stop printing.
	} // End of SinglePassFlag test

} // End of loop.
```



Results Printed from Previous Test

The following is an example of how you would use a character string in conjunction with the Serial port.

```
// Input buffer and its pointer.
char Input[32];
int InputPointer;
// function that will convert a string to all
// upper case, using a for loop.
void ToUpperCase(char string[])
{
	int index;  // variable that indexes into array.
	// for loop to move through string
	for (index = 0; string[index] != 0; index++)
	{
		// check to see if current character
		// is a lower case letter.
		if (string[index] >= 'a' && string[index] <= 'z')
		{
			string[index] &= ~0x20; // Converts to Upper Case
		} // end of lower case if
	}// end of for loop
}// end of ToUpperCase function

// Start up code
void setup()
{
	// Start the serial port
	Serial.begin(9600);
	InputPointer = 0;   // start pointer at 0.
} // End of setup.

// Continuously called loop.
void loop()
{
	// Check to see if a character has come in.
	if (Serial.available())
	{
		// Read in the character.
		char local = Serial.read();
		// Check for end of string (new line character).
		if (local != '\n')
		{
			// If not the end of string
			Input[InputPointer] = local; // Place character in string
			InputPointer++; // Advance the pointer
		}
		else // Once we receive the new line '\n'
		{
			Input[InputPointer] = 0; // Add terminator.
			Serial.println(Input); // Send out string
			ToUpperCase(Input);    // Convert to upper case
			Serial.println(Input); // send out converted string
			InputPointer = 0;        // Reset pointer for new string.
		} // End of new line check
	} // End of serial input check.
} // End of loop.
```

Results – Test of Serial code.