Character and String Functions and Library Functions in C

Character Library Functions

- There are several C library functions that allow to identify the type of characters and their case.
- To use these functions, you should include <ctype.h> :
  - isdigit (ch) -- Returns true if ch is a digit (0-9)
  - islower(ch) -- Returns true if ch is a lower case letter (a-z)
  - ispunct(ch) -- Returns true if ch is a punctuation
  - iscntrl(ch) -- Returns true if ch is the control key
  - isspace(ch) -- Returns true if ch is the control key
  - toupper (ch) -- Returns uppercase ch
  - tolower(ch) -- Returns lowercase ch
  - isalnum(ch) -- Returns true if ch is alphanumeric character
- A Few Examples:

```
char mychar = ‘b’;
printf(“%c”,  toupper(mychar));  // prints B
printf(“%d”,  islower(mychar));  // prints 1 (true), as mychar holds a lower case char
printf(“%d”,  isdigit(mychar));    // prints 0 (false)
```

String Library Functions

Library Functions to Manipulate C-strings

- C doesn't support predefined type called `string` like in C++, or `String` in Processing. Therefore, you cannot use operators such =, +=, ==, >=, etc. to copy, concatenate, or compare c-stings.
- As stated earlier, a null-terminated array of characters represents as a c-string

```
char s1 [10] = "Apple";
char s2[ 10] = "Orange"
s1  =  s2;            // illegal statement
s1 += s2;             // illegal statement
If (s1 > s2) {  ...}  // Compares address. Not a Lexicographic comparison
```

- There are several library function for string manipulation. To use these functions you need to include `<string.h>`. Some of the C-string library functions include:

  strlen(s) -- Returns the length of a string. Examples:

```
char s[20] = "ABCD";
printf("%lu", strlen(s));        // prints 4
```

  Note: size of s is 20 bytes but its string length is 4.

C-Strings – Library Functions

strcmp(s1, s2) - Compares s1 and s2: Returns zero if two strings are identical. Otherwise returns a positive integer if s1 is greater than s2, or a negative integer if s1 is less than s2.

```
char s1[20] = “BCC”;
char s2[20]  = “BBC”;
if (strcmp (s1,  s2) >  0)
        printf(“%s is lexicographically greater than %s.”, s1, s2);
```

strcpy(s1, s2) -- Copies s2 into s1:

```
char s1[20] = “ABCD”;
char s2[20];
Strcpy(s2, s1);
printf(“%s”, s2);              // prints ABCD
```

strcat(s1, s2) -- Appends s2 to the end of s1.

```
char s1[20] = “ABCD”;
char s2[20]  = “XY;
strcat(s2, s1);
printf(“%s”, s2);              // prints: XYABCD
```

Strings Functions that Return a char* Pointer

- Functions **strcpy** and **strcat** also return a char pointer (char*).
- The returned pointer points to the first argument of the function (in the following call to strcpy, s1), and can be used for different purposes:

```
char s1[5] = “Red”;
char s2[5];
printf( “%s”, strcpy(s2, s1));
```

- Or:

```
char s1[5] = “CM”;
char s2[8] = “EN”;
printf( “%s”, strcat(s2, strcat(s1, “-339”));
```

  - First, function **strcat** appends string “-339” to the end of s1 (“CM”) and returns “CM-339” to the outer call of **strcat** that receives s2 as its first argument. Then it concatenates string “CM-339” to the end of s2 (which is “EN”).
  - Therefore the final output is: ENCM-339