

## C - MORE ABOUT PASSING VARIABLES

Some differing methods for passing variables become important when figuring out how to interface a peripheral chip to a PIC microcontroller. Following are three that I have found to be useful.

### OUTPUT

As we have seen, a single-purpose function can be written which has the variable of interest contained in it. It may be useful to make the function general or universal by designing it so that a different variable of interest may be passed to it by the main program each time the function is called.

A very simple example using one of the PIC microcontroller's own ports will be used to illustrate this point. We will create a "driver" which is used to display a byte via LEDs. The byte to be displayed (variable, called a parameter in the calling function), is passed to the receiving function as part of the function call. The function receives the byte (variable, called an argument in the receiving function), and displays it.

```
////pass parameter to a function demo                Cns27                8/15/09
#include <16F818.h>
#fuses INTRC_IO

display (BYTE value)                                //display function
{
    output_b (value);                                //display data passed to function
}

main()
{
    display (0x01);                                  //call function - pass data to be
                                                    // displayed
    while (TRUE);                                    //circle, always
}
```

The byte "value" declaration is contained in the argument part of the receiving function name.

## Sending A Character String To An LCD

A character string may be sent directly to an LCD for display using a driver function. This will be described in detail in a later chapter.

Handling a string is not as straight forward as handling a single character. To review, C does not have a string data type (string of characters). String data is stored as one character per element in an array called a "string array". A string terminator called the null character or null zero is added automatically by the compiler as the last character in the string array.

The string elements are declared in the program that calls the driver. The string copy `strcpy` function is used for this purpose. You might think of this as loading the characters into the string array just prior to use. The code is universal in the sense that different characters may be loaded into the array at a later time for another use.

```
strcpy (msg, "HELLO");
```

The name of the string is "msg" in this example.

Note that the string cannot be defined as part of the driver function argument as can be done for a character (although that is what it looks like)(position in the LCD example discussed later). The string must be defined ahead of use. This is done ahead of the driver function.

```
////LCD test                                CNS22c                                8/25/09
//send character string to first/left half of 1x16 LCD

//snip

char msg[9];                                //declare string array

//snip

display(BYTE posit, char msg[9])           //display string function - driver
// declare LCD character position,
// byte data type and string array
// named "msg", char data type
{
//snip
}

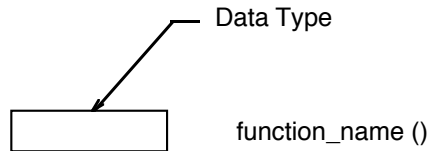
main()
{
//snip

strcpy (msg, "first 8");                    //load string array
display(0x80, msg);                        //call function to send display position
// and 8 characters to first half
// of display

//snip
}
```

## INPUT

In a situation where the "driver" is used to acquire or read data from an external peripheral device such as a A/D converter or compass, it may be convenient to use a scheme where the driver function name is used as the name of the variable returned by the function. HUuh?? That's what I thought.



In the following simple example, a function used to read port A is called.

```
read_a();
```

The function itself is identified as:

```
int8 read_a()
```

This behaves as a data declaration where `read_a()` is an `int8` data type. `read_a()` is also the function name.

The `read_a()` function reads port A and passes the contents of port A to the `main()` function which then displays the data via LEDs connected to port B.

```
output_b (read_a());
```

This methodology makes drivers easy to use once you understand how this works. You will find this method used frequently in CCS drivers.

```
////driver method demo                                CNS28b    9/4/09
#include <16F818.h>

int8 read_a()                                         //simple "driver" - read port A,
                                                       // pass variable = port A contents
                                                       // to calling function.
                                                       // use driver name as variable

name
{
  int8 read;
  read = input_a();                                  //read port A
  return read;                                       //pass variable to calling function
}
main()
{
  read_a();                                          //call "driver" function
  output_b (read_a());                               //use driver name as variable name
  while (TRUE);                                     //circle
}
```