CMPE1250 – Strings, Pointers, and the IDE

You will often wish to send strings to the LCD. Before you can do that, you need to review some of the details on using string with this platform.

Strings in C are simply an array of char, with the last element being a literal zero. This is often referred to as a "NULL-terminated string". This is a terrible name, as NULL is typically reserved for pointer with no valid address, or NULL has special meaning in database terminology. In C, NULL does take on the value zero, and the literal zero in the ASCII table is called 'NUL', so the confusion intensifies.

Hex	Value	Hex	Value	Hex	Value	Hex	Valu
00	NUL	10	DLE	20	SP	30	0
01	SOH	11	DC1	21	!	31	1
02	STX	12	DC2	22	"	32	2
03	ETX	13	DC3	23	#	33	3
04	EOT	14	DC4	24	$	34	4

Zero is a good choice to terminate a string, as it will never appear as a valid ASCII code for anything you would put in a string. Any functions that operate the string may look for the 'NUL' to locate the end of the string.

Creating a string in C may be done with the following syntax, using a string literal:

```
char const StrHello [] = "Hello!";
```

String literals (the part in double quotes), will automatically be NUL terminated. You must always ensure you leave enough room in arrays/buffers/allocations to accommodate the NUL character, which is the string length + 1. Care must also be taken to ensure that a programmatically created string contains a NUL character!

Strings are passed to functions as pointer to char, and should have const modifiers on either the pointer or target, if the function is not meant to modify those parts. These modifiers on the formal argument may be more restrictive than the actual argument for the target, but not less. Placing the correct const modifier on the target will prevent accidental modification to the target in the calling scope.

The above string would be passed to a function as shown. This function does not intend to modify any part the string, nor alter the pointer during execution of the function. Remember that the name of an array is a pointer to the first element of the array!

```
char const StrHello [] = "Hello!";

void foo (char const * const pStr)
{
  lcd_StringXY (5, 1, pStr);
}
```

If a function intends to use the formal argument for operational code, it may leave the pointer as non-const, as it is getting a copy of the pointer, and can't modify the actual argument anyway:

```
char const StrHello [] = "Hello!";

int foo (char const * pStr)
{
  int iStrLen = 0;
  while (*(pStr++))
    ++iStrLen;
  return iStrLen;
}

// calling code
Segs_16D (foo (StrHello), 0);  // outputs 0006
```

This permits the formal argument to manipulate the copy of the actual pointer it receives to do its work but does not need to modify the target (the actual characters of the string).

Of course, you could just call a function with a string literal if it is appropriate to do so:

```
// calling code
Segs_16D (foo ("How long am I?"), 0);  // outputs 0014
```

You may create strings programmatically as well. The following code creates a pseudo-random five-character string, and appropriately terminates it with a NUL:

```
void GenStr (char * const pStr)
{
  int i;
  for (i = 0; i < 5; ++i)
    pStr[i] = rand() % ('Z' + 1 - 'A') + 'A';
  pStr[i] = 0;
}
```

Note: a function would never return a pointer to stack memory (a pointer created in the function). The target buffer must persist beyond the function scope, so it would typically be creating in the calling scope and be populated by the function.

Calling code:

```
{
  char buff [6] = {0};
  GenStr (buff);
  lcd_StringXY (0, 0, buff);
}
```

Initializing an array with NULs is an effective way to ensure that the string will be correctly terminated. If the modifying code only modifies array size – 1 elements, then the residual NUL from initialization will terminate the string. This function example does not do this, but it is a possibility.

You may also create formatted strings with `sprintf` from the `stdio.h` library.

The `sprintf` functions is somewhat analogous to string interpolation in C#, but much less sophisticated. A format string is interpreted for specifiers that indicate argument substitutions within the string. The format specifiers are removed and the arguments are formatted and put in their place. The entire string is written to a provided buffer.

C does not offer much protection from errors, and the buffer length, argument count, and argument types can all contribute to errors or unpredictable behavior, so you must be careful when constructing `sprintf` calls.

The following examples should be studied to see how `sprintf` can produce formatted output:

```
//#include <stdlib.h>
// useful functions: rand, srand, atoi, atof, atol

// rand will return a random number, from 0 to intmax (or 0x7FFF)
// srand can be used to seed the random number generator (same sequence per seed)

// atoX functions return numeric conversion of string representation, where possible

//#include <stdio.h>
// useful functions: sprintf, sprintf_s, scanf, scanf_s (also full file functions, but not here...)

// we use printf to send text to the console (if only we had one)
// we use sprintf to send text to a string (a buffer)
// sprintf uses text with special character sequences to specify formatting
// the % character specifies the start of such a sequence
// %d would mean decimal
// %f would mean floating point
// %s and %c exist as well (string, character)
// minimum width, padding, and precision can also be specified, and mean
//  different things depending on the format specifier used
// additional arguments are passed to the sprintf function as required to satisfy
//  the format specifiers
// width and precision are specified as W.P in the format specifier
// width means minimum width
// precision means padding for int, number of digits R of DP for float

// NOTE: errors or garbage will be the result of mismatch in args or arg types
// NOTE: the special sequence is swapped for the formatted output in the resultant string
// NOTE: other characters, like <CR><LF> are represented by \r or \n
// there are others too, like \t \b
// \\ would mean a \ actual, %% would mean % actual
```

```
void main(void)
{
  // create buffer for sprintf output (remember to include room for null)
  // this can be oversized, if output length is variable (within reason)
  char buff[21];

  // main entry point
  _DISABLE_COP();
  EnableInterrupts;

  PLL_To20MHz();
  Timer_Init(20E6, Timer_Prescale_32, 0, 0, Timer_Pin_Disco);
  lcd_Init();
  SWL_Init();

  for (;;)
  {
    // place formatted decimal output into this buffer
    // %d means substitute this part for the first argument, as decimal
    // rest of string is intact
    // sprintf returns the number of characters written, we don't care in this case
    (void)sprintf(buff, "Value: %d", 42); // outputs "Value: 42"
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;

    // %x HEX, with lowercase a-f where shown
    (void)sprintf(buff, "%x", 42); // outputs "2a"
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;

    // %X HEX, with uppercase A-F where shown
    (void)sprintf(buff, "%X", 42); // outputs "2A"
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;

    // other examples - floating point, width, precision, padding
    // %f floating point, natural (arg must be float)
    (void)sprintf(buff, "%f", 42 / 3.1f); // outputs "13.548387"
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;

    // %f floating point, natural width, 2 DP precison (arg must be float)
    (void)sprintf(buff, "%0.2f", 42 / 3.1f); // outputs "13.55"
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;

    // %f floating point, min 12 character width, 2 DP precison (arg must be float)
    (void)sprintf(buff, "%12.2f", 42 / 3.1f); // outputs "       13.55" (seven spaces + 5 chars == 12)
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;

    // %d, 4 character width, integer as decimal
    (void)sprintf(buff, "%4d", 42); // outputs "  42"
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;
```

```
    // %d, 4 character width, precision means padding with zeroes, integer as decimal
    (void)sprintf(buff, "%4.4d", 42); // outputs "0042"
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;

    // %f, 10 character minimum, left-aligned, 2 DP precision
    (void)sprintf(buff, "*%-10.2f*", 42 / 3.1f); // outputs "*13.55     *"
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;

    // %ld, for printing long
    (void)sprintf(buff, "%ld", (long)-20E6); // outputs "-20000000"
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;

    // %lu, for printing long
    (void)sprintf(buff, "%lu", (unsigned long)20E6); // outputs "20000000"
    lcd_Clear();
    lcd_StringXY(0, 0, buff);
    while (!SWL_Transition(SWL_CTR, SWL_DebOff))
      ;
  }
}
```