C Review and Dissection IV:

Pointers, Strings and Formatted Text Output

These lecture notes created by Mr. James B. (Jim) Carlson, NCSU

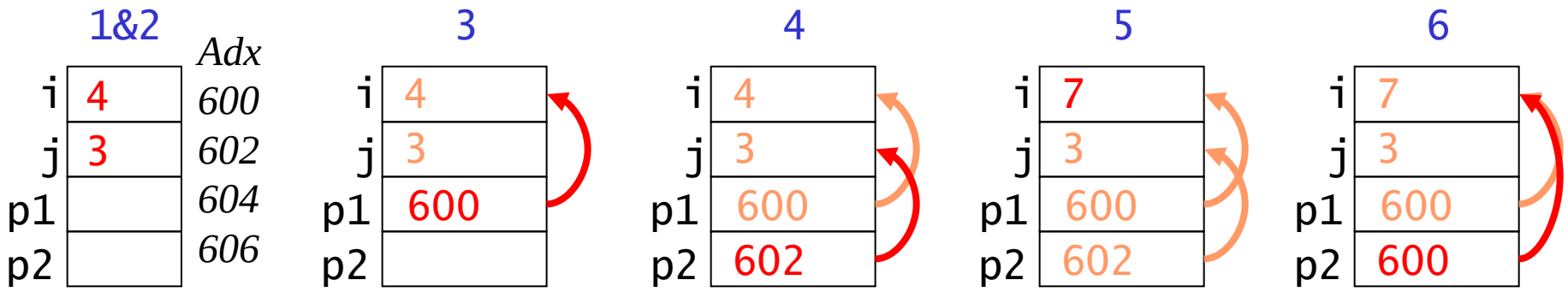
Today

- Pointers
- Strings
- Formatted Text Output

Pointers

- A *pointer* variable holds the *address* of the data, rather than the *data* itself
- To make a pointer point to variable **a**, we can specify the *address* of **a**
  - address operator **&**
- The data is accessed by *dereferencing* (following) the pointer
  - indirection operator ***** works for reads and writes
- Assigning a new value to a pointer variable changes *where the variable **points**, **not** the data*

```
void main ( ) {
 int i, j;
 int *p1, *p2;
1 i = 4;
2 j = 3;
3 p1 = &i;
4 p2 = &j;
5 *p1 = *p1+*p2;
6 p2 = p1;
}
```



More about Pointers

- Incrementing and decrementing pointers to array elements
  - Increment operator ++ makes pointer advance to next element (next larger address)
  - Decrement operator -- makes pointer move to previous element (next smaller address)
  - These use the size of the variable's base type (e.g. int, char, float) to determine what to add
    - **p1++** corresponds to **p1 = p1 + sizeof(int);**
    - sizeof is C macro which returns size of type in bytes

```
int a[18];
int * p;
p = &a[5];
*p = 5; /* a[5]=5 */
p++;
*p = 7; /* a[6]=7 */
p--;
*p = 3; /* a[5]=3 */
```

- Pre and post
  - Putting the ++/-- **before** the pointer causes inc/dec **before** pointer is used
    - int *p=100, *p2;
      - **p2 = ++p;** assigns **102** to integer pointer **p2**, and **p** is **102** afterwards
  - Putting the ++/-- **after** the pointer causes inc/dec **after** pointer is used
    - char *q=200, *q2;
      - **q2 = q--;** assigns **200** to character pointer **q2**, and **q** is **199** afterwards

Pointer Example

Consider the following code. What are the contents as the code executes? Assume the Static Base is defined as 0x008000. The address assignment grows in a positive direction.

```
                    #define right 0
                    #define left  1
//Globals
                      int left_speed, right_speed
                      char wheel_off [2];
                      char wheel_on [2];
                      int *left_sp, *right_sp;
// MAIN
                    void main (void) {
          1           left_speed = 301;
          2           right_speed = 378;
          3           Wheel_on[right] = wheel_on[left] = 200;
          4           Wheel_off[right] = wheel_off[left] = 0;
          5           left_sp = &left_speed;
          6           right_sp = &right_speed;
          7           *left_sp = *left_sp + *right_sp;
          8           right_sp = left_sp;
                    }
```

Pointer Example

Code			Address			Variables	Data
		#define right 0	0x00	0x80	0x00		
		#define left 1	0x00	0x80	0x01		
//Globals			0x00	0x80	0x02		
		int left_speed, right_speed	0x00	0x80	0x03		
		char wheel_off [2];	0x00	0x80	0x04		
		char wheel_on [2];	0x00	0x80	0x05		
		int *left_sp, *right_sp;	0x00	0x80	0x06		
// MAIN			0x00	0x80	0x07		
		void main (void) {	0x00	0x80	0x08		
	1	left_speed = 301;	0x00	0x80	0x09		
	2	right_speed = 378;	0x00	0x80	0x0A		
	3	Wheel_on[right] = wheel_on[left] = 200;	0x00	0x80	0x0B		
	4	Wheel_off[right] = wheel_off[left] = 0;	0x00	0x80	0x0C		
	5	left_sp = &left_speed;	0x00	0x80	0x0D		
	6	right_sp = &right_speed;	0x00	0x80	0x0E		
	7	*left_sp = *left_sp + *right_sp;	0x00	0x80	0x0F		
	8	right_sp = left_sp;	0x00	0x80	0x10		
		}	0x00	0x80	0x11		

Pointer Example

Code			Address			Variables	Data
		#define right 0	0x00	0x80	0x00	wheel_off[0]	
		#define left 1	0x00	0x80	0x01	wheel_off[1]	
//Globals			0x00	0x80	0x02	wheel_on[0]	
		int left_speed, right_speed	0x00	0x80	0x03	wheel_on[1]	
		char wheel_off [2];	0x00	0x80	0x04		
		char wheel_on [2];	0x00	0x80	0x05		
		int *left_sp, *right_sp;	0x00	0x80	0x06		
// MAIN			0x00	0x80	0x07		
		void main (void) {	0x00	0x80	0x08		
	1	left_speed = 301;	0x00	0x80	0x09		
	2	right_speed = 378;	0x00	0x80	0x0A		
	3	Wheel_on[right] = wheel_on[left] = 200;	0x00	0x80	0x0B		
	4	Wheel_off[right] = wheel_off[left] = 0;	0x00	0x80	0x0C		
	5	left_sp = &left_speed;	0x00	0x80	0x0D		
	6	right_sp = &right_speed;	0x00	0x80	0x0E		
	7	*left_sp = *left_sp + *right_sp;	0x00	0x80	0x0F		
	8	right_sp = left_sp;	0x00	0x80	0x10		
		}	0x00	0x80	0x11		

Pointer Example

Code			Address			Variables	Data
		#define right 0	0x00	0x80	0x00	wheel_off[0]	
		#define left 1	0x00	0x80	0x01	wheel_off[1]	
//Globals			0x00	0x80	0x02	wheel_on[0]	
		int left_speed, right_speed	0x00	0x80	0x03	wheel_on[1]	
		char wheel_off [2];	0x00	0x80	0x04	left_speed	
		char wheel_on [2];	0x00	0x80	0x05	left_speed	
		int *left_sp, *right_sp;	0x00	0x80	0x06	right_speed	
// MAIN			0x00	0x80	0x07	right_speed	
		void main (void) {	0x00	0x80	0x08	*left_sp	
	1	left_speed = 301;	0x00	0x80	0x09	*left_sp	
	2	right_speed = 378;	0x00	0x80	0x0A	*right_sp	
	3	Wheel_on[right] = wheel_on[left] = 200;	0x00	0x80	0x0B	*right_sp	
	4	Wheel_off[right] = wheel_off[left] = 0;	0x00	0x80	0x0C		
	5	left_sp = &left_speed;	0x00	0x80	0x0D		
	6	right_sp = &right_speed;	0x00	0x80	0x0E		
	7	*left_sp = *left_sp + *right_sp;	0x00	0x80	0x0F		
	8	right_sp = left_sp;	0x00	0x80	0x10		
		}	0x00	0x80	0x11		

Pointer Example

Code			Address			Variables	Data
		#define right 0	0x00	0x80	0x00	wheel_off[0]	
		#define left 1	0x00	0x80	0x01	wheel_off[1]	
//Globals			0x00	0x80	0x02	wheel_on[0]	
		int left_speed, right_speed	0x00	0x80	0x03	wheel_on[1]	
		char wheel_off [2];	0x00	0x80	0x04	left_speed	0x2D
		char wheel_on [2];	0x00	0x80	0x05	left_speed	0x01
		int *left_sp, *right_sp;	0x00	0x80	0x06	right_speed	
// MAIN			0x00	0x80	0x07	right_speed	
		void main (void) {	0x00	0x80	0x08	*left_sp	
	1	left_speed = 301;	0x00	0x80	0x09	*left_sp	
	2	right_speed = 378;	0x00	0x80	0x0A	*right_sp	
	3	Wheel_on[right] = wheel_on[left] = 200;	0x00	0x80	0x0B	*right_sp	
	4	Wheel_off[right] = wheel_off[left] = 0;	0x00	0x80	0x0C		
	5	left_sp = &left_speed;	0x00	0x80	0x0D		
	6	right_sp = &right_speed;	0x00	0x80	0x0E		
	7	*left_sp = *left_sp + *right_sp;	0x00	0x80	0x0F		
	8	right_sp = left_sp;	0x00	0x80	0x10		
		}	0x00	0x80	0x11		

Pointer Example

Code			Address			Variables	Data
		#define right 0	0x00	0x80	0x00	wheel_off[0]	
		#define left 1	0x00	0x80	0x01	wheel_off[1]	
//Globals			0x00	0x80	0x02	wheel_on[0]	
		int left_speed, right_speed	0x00	0x80	0x03	wheel_on[1]	
		char wheel_off [2];	0x00	0x80	0x04	left_speed	0x2D
		char wheel_on [2];	0x00	0x80	0x05	left_speed	0x01
		int *left_sp, *right_sp;	0x00	0x80	0x06	right_speed	0x7A
// MAIN			0x00	0x80	0x07	right_speed	0x01
		void main (void) {	0x00	0x80	0x08	*left_sp	
	1	left_speed = 301;	0x00	0x80	0x09	*left_sp	
	2	right_speed = 378;	0x00	0x80	0x0A	*right_sp	
	3	Wheel_on[right] = wheel_on[left] = 200;	0x00	0x80	0x0B	*right_sp	
	4	Wheel_off[right] = wheel_off[left] = 0;	0x00	0x80	0x0C		
	5	left_sp = &left_speed;	0x00	0x80	0x0D		
	6	right_sp = &right_speed;	0x00	0x80	0x0E		
	7	*left_sp = *left_sp + *right_sp;	0x00	0x80	0x0F		
	8	right_sp = left_sp;	0x00	0x80	0x10		
		}	0x00	0x80	0x11		

Pointer Example

Code			Address			Variables	Data
		#define right 0	0x00	0x80	0x00	wheel_off[0]	0x00
		#define left 1	0x00	0x80	0x01	wheel_off[1]	0x00
//Globals			0x00	0x80	0x02	wheel_on[0]	0xC8
		int left_speed, right_speed	0x00	0x80	0x03	wheel_on[1]	0xC8
		char wheel_off [2];	0x00	0x80	0x04	left_speed	0x2D
		char wheel_on [2];	0x00	0x80	0x05	left_speed	0x01
		int *left_sp, *right_sp;	0x00	0x80	0x06	right_speed	0x7A
// MAIN			0x00	0x80	0x07	right_speed	0x01
		void main (void) {	0x00	0x80	0x08	*left_sp	
	1	left_speed = 301;	0x00	0x80	0x09	*left_sp	
	2	right_speed = 378;	0x00	0x80	0x0A	*right_sp	
	3	Wheel_on[right] = wheel_on[left] = 200;	0x00	0x80	0x0B	*right_sp	
	4	Wheel_off[right] = wheel_off[left] = 0;	0x00	0x80	0x0C		
	5	left_sp = &left_speed;	0x00	0x80	0x0D		
	6	right_sp = &right_speed;	0x00	0x80	0x0E		
	7	*left_sp = *left_sp + *right_sp;	0x00	0x80	0x0F		
	8	right_sp = left_sp;	0x00	0x80	0x10		
		}	0x00	0x80	0x11		

Pointer Example

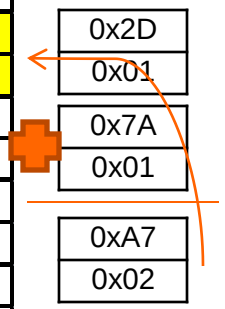
Code			Address			Variables	Data
		#define right 0	0x00	0x80	0x00	wheel_off[0]	0x00
		#define left 1	0x00	0x80	0x01	wheel_off[1]	0x00
//Globals			0x00	0x80	0x02	wheel_on[0]	0xC8
		int left_speed, right_speed	0x00	0x80	0x03	wheel_on[1]	0xC8
		char wheel_off [2];	0x00	0x80	0x04	left_speed	0x2D
		char wheel_on [2];	0x00	0x80	0x05	left_speed	0x01
		int *left_sp, *right_sp;	0x00	0x80	0x06	right_speed	0x7A
// MAIN			0x00	0x80	0x07	right_speed	0x01
		void main (void) {	0x00	0x80	0x08	*left_sp	0x04
	1	left_speed = 301;	0x00	0x80	0x09	*left_sp	0x80
	2	right_speed = 378;	0x00	0x80	0x0A	*right_sp	
	3	Wheel_on[right] = wheel_on[left] = 200;	0x00	0x80	0x0B	*right_sp	
	4	Wheel_off[right] = wheel_off[left] = 0;	0x00	0x80	0x0C		
	5	left_sp = &left_speed;	0x00	0x80	0x0D		
	6	right_sp = &right_speed;	0x00	0x80	0x0E		
	7	*left_sp = *left_sp + *right_sp;	0x00	0x80	0x0F		
	8	right_sp = left_sp;	0x00	0x80	0x10		
		}	0x00	0x80	0x11		

Pointer Example

Code			Address			Variables	Data
		#define right 0	0x00	0x80	0x00	wheel_off[0]	0x00
		#define left 1	0x00	0x80	0x01	wheel_off[1]	0x00
//Globals			0x00	0x80	0x02	wheel_on[0]	0xC8
		int left_speed, right_speed	0x00	0x80	0x03	wheel_on[1]	0xC8
		char wheel_off [2];	0x00	0x80	0x04	left_speed	0x2D
		char wheel_on [2];	0x00	0x80	0x05	left_speed	0x01
		int *left_sp, *right_sp;	0x00	0x80	0x06	right_speed	0x7A
// MAIN			0x00	0x80	0x07	right_speed	0x01
		void main (void) {	0x00	0x80	0x08	*left_sp	0x04
	1	left_speed = 301;	0x00	0x80	0x09	*left_sp	0x80
	2	right_speed = 378;	0x00	0x80	0x0A	*right_sp	0x06
	3	Wheel_on[right] = wheel_on[left] = 200;	0x00	0x80	0x0B	*right_sp	0x80
	4	Wheel_off[right] = wheel_off[left] = 0;	0x00	0x80	0x0C		
	5	left_sp = &left_speed;	0x00	0x80	0x0D		
	6	right_sp = &right_speed;	0x00	0x80	0x0E		
	7	*left_sp = *left_sp + *right_sp;	0x00	0x80	0x0F		
	8	right_sp = left_sp;	0x00	0x80	0x10		
		}	0x00	0x80	0x11		

Pointer Example

Code			Address			Variables	Data
		#define right 0	0x00	0x80	0x00	wheel_off[0]	0x00
		#define left 1	0x00	0x80	0x01	wheel_off[1]	0x00
//Globals			0x00	0x80	0x02	wheel_on[0]	0xC8
		int left_speed, right_speed	0x00	0x80	0x03	wheel_on[1]	0xC8
		char wheel_off [2];	0x00	0x80	0x04	left_speed	0xA7
		char wheel_on [2];	0x00	0x80	0x05	left_speed	0x02
		int *left_sp, *right_sp;	0x00	0x80	0x06	right_speed	0x7A
// MAIN			0x00	0x80	0x07	right_speed	0x01
		void main (void) {	0x00	0x80	0x08	*left_sp	0x04
	1	left_speed = 301;	0x00	0x80	0x09	*left_sp	0x80
	2	right_speed = 378;	0x00	0x80	0x0A	*right_sp	0x06
	3	Wheel_on[right] = wheel_on[left] = 200;	0x00	0x80	0x0B	*right_sp	0x80
	4	Wheel_off[right] = wheel_off[left] = 0;	0x00	0x80	0x0C		
	5	left_sp = &left_speed;	0x00	0x80	0x0D		
	6	right_sp = &right_speed;	0x00	0x80	0x0E		
	7	*left_sp = *left_sp + *right_sp;	0x00	0x80	0x0F		
	8	right_sp = left_sp;	0x00	0x80	0x10		
		}	0x00	0x80	0x11		



Pointer Example

Code			Address			Variables	Data
		#define right 0	0x00	0x80	0x00	wheel_off[0]	0x00
		#define left 1	0x00	0x80	0x01	wheel_off[1]	0x00
//Globals			0x00	0x80	0x02	wheel_on[0]	0xC8
		int left_speed, right_speed	0x00	0x80	0x03	wheel_on[1]	0xC8
		char wheel_off [2];	0x00	0x80	0x04	left_speed	0xA7
		char wheel_on [2];	0x00	0x80	0x05	left_speed	0x02
		int *left_sp, *right_sp;	0x00	0x80	0x06	right_speed	0x7A
// MAIN			0x00	0x80	0x07	right_speed	0x01
		void main (void) {	0x00	0x80	0x08	*left_sp	0x04
	1	left_speed = 301;	0x00	0x80	0x09	*left_sp	0x80
	2	right_speed = 378;	0x00	0x80	0x0A	*right_sp	0x04
	3	Wheel_on[right] = wheel_on[left] = 200;	0x00	0x80	0x0B	*right_sp	0x80
	4	Wheel_off[right] = wheel_off[left] = 0;	0x00	0x80	0x0C		
	5	left_sp = &left_speed;	0x00	0x80	0x0D		
	6	right_sp = &right_speed;	0x00	0x80	0x0E		
	7	*left_sp = *left_sp + *right_sp;	0x00	0x80	0x0F		
	8	right_sp = left_sp;	0x00	0x80	0x10		
		}	0x00	0x80	0x11		

What else are pointers used for?

- Data structures which reference each other
  - lists
  - trees
  - etc.
- Exchanging information between procedures
  - Passing arguments (e.g. a structure) quickly – just pass a pointer
  - Returning a structure
- Accessing elements within arrays (e.g. string)

Pointers and the MSP430 ISA

- Address space of MSP430 is 1 megabyte
  - Need 20 bits to address entire space
- This space is divided into two areas
  - Near: 64 kilobytes from 0x00000 to 0x0FFFF can be addressed with a 16-bit pointer (top 4 bits of 20-bit address are 0)
    - Pointer is shorter: 2 bytes
    - Pointer operations are faster
    - Note: internal RAM and SFRs are in this space
  - Far: Entire 1 megabyte area from 0x00000 to 0xFFFFF can be addressed with a 20-bit pointer
    - Pointer is longer: 4 bytes used
      - 1.5 bytes wasted, but easier to operate on 32 bits than 24
    - Pointer operations are slower, since ALU operates on 16 bits at a time

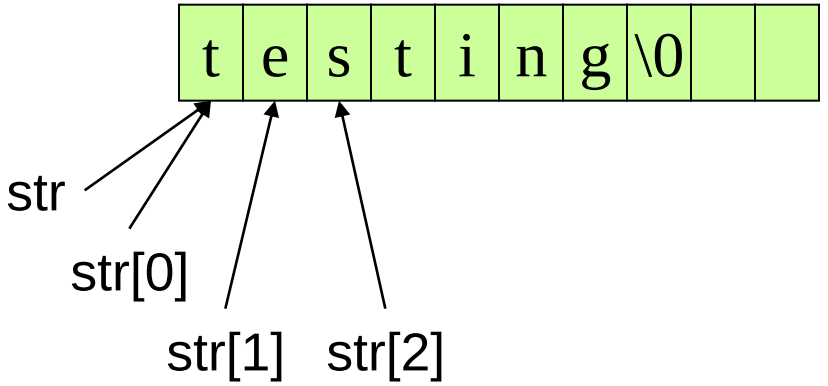
Specifying Data Areas and Pointer Sizes

- Default locations for data
  - Near area: RAM data
    - data, bss sections
  - Far area: ROM data
    - rom, program sections
    - const data section
- Pointer sizes chosen by compiler based on area holding type of data
  - Near pointer (16 bits) used for near data
  - Far pointer (32 bits) used for far data
- Using the _near and _far keywords
  - specify location for static variables
    - int _near near_data; // located in near area, &near_data is 16 bits
    - int _far far_data; // located in far area, &far_data is 20 bits
  - specify pointers to data
    - int _near * near_data; // points to *near* data. 16 bit pointer is in *near* section
    - int _far * far_data; // points to *far* data. 32 bit pointer is in *near* section

    - int * _near near_data; // points to *near* data. 16 bit pointer is in *near* section
    - int * _far far_data; // points to *far* data. 32 bit pointer is in *far* section

Strings

- There is no “string” type in C.
- Instead an **array of characters** is used - *char a[44]*
- The string is terminated by a NULL character (value of 0, represented in C by \0).
  - Need an extra array element to store this null
- Example
  - char str[10] = “testing”;



Displaying Text on the Control Board LCD

- Control Board contains a 4 line by 10 character LCD
- Sample Code provides an interface (device driver) code to simplify LCD use

- Application Programmer's Interface (API)
  - Init_LCD(): sets up LCD
  - lcd_out(char *s, char line, char position);
    - string: pointer to null-terminated array of characters. [10 characters]
    - line : LCD_HOME_L1, LCD_HOME_L2, LCD_HOME_L3, LCD_HOME_L4
    - position: 0 to 9
  - display_1 = "Embedded";
  - display_2 = "Systems";
  - display_3 = "Rock!";
  - display_4 = "Go Pack!";

What if there are more than 10 characters?

More on LCDs Later

- Learning how the interface with the LCD works
- Enhancing the API to support bit-mapped graphics
- Interfacing with larger graphics LCD panels

Formatted String Creation

- Common family of functions defined in **stdio.h**
  - printf: print to standard output
  - sprintf: print to a string
  - fprintf: print to a file
- Syntax: sprintf(char *str, char * frmt, arg1, arg2, arg3 ..);
  - str: destination
  - fmt: format specifying what to print and how to interpret arguments
    - %d: signed decimal integer
    - %f: floating point
    - %x: unsigned hexadecimal integer
    - %c: one character
    - %s: null-terminated string
  - arg1, etc: arguments to be converted according to format string

Memory Requirements for String Functions

- sprintf, strcat, etc. all require memory (code (program/text), data (bss, data))
- Examine linker output map file for details (file.map)
  - Shows each section's start address, length and source module (file)
- printf, sprintf, fprintf all call **print** function

```
########################################
# (2) SECTION INFORMATION              #
########################################
# SECTION     ATR TYPE    START  LENGTH ALIGN MODULENAME
data_SE       ABS DATA    000400 000000       NCRT0_UART
bss_SE        REL DATA    000400 000000 2     NCRT0_UART
data_SO       REL DATA    000400 000000       NCRT0_UART
bss_SO        REL DATA    000400 000000       NCRT0_UART
data_NE       REL DATA    000400 000000 2     NCRT0_UART
              REL DATA    000400 000014       GLOBALS
              REL DATA    000414 000002       ERRNO
              REL DATA    000416 00000C       INFINITY
bss_NE        REL DATA    000422 000000 2     NCRT0_UART
              REL DATA    000422 000218       GLOBALS
              REL DATA    00063A 000004       SPRINTF
              REL DATA    00063E 000108       PRINT
data_NO       REL DATA    000746 000000       NCRT0_UART
bss_NO        REL DATA    000746 000000       NCRT0_UART
              REL DATA    000746 00026A       PRINT
                          0009B0 000200       NCRT0_UART
                          000BB0 000000       NCRT0_UART
```

	Memory Section Size (bytes)			
Function	**program**	**rom**	**bss**	**data**
strncmp	90	0	0	0
strcat	73	0	0	0
strchr	63	0	0	0
sprintf	218	0	4	0
print	7050	10	882	0