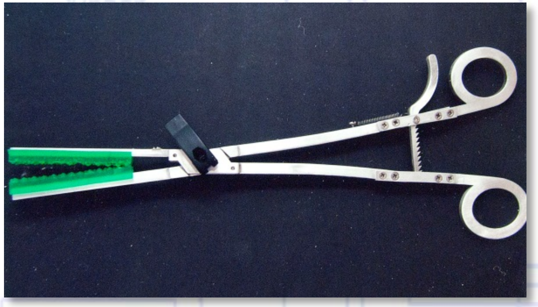
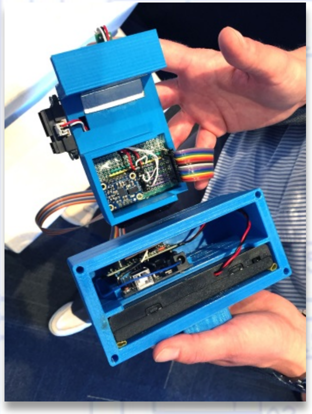
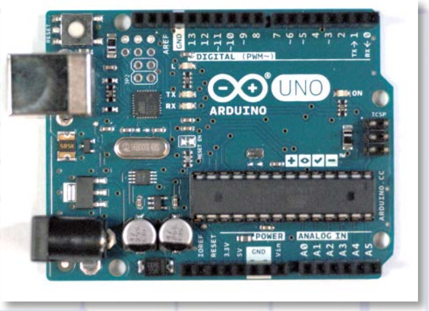
Introductory Medical Device Prototyping

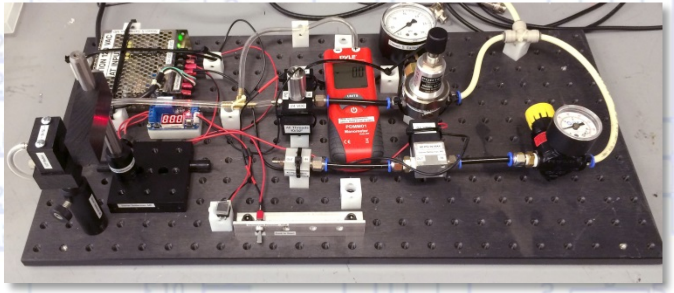
Advanced C Programming Topics

Prof. Steven S. Saliterman, http://saliterman.umn.edu/
Department of Biomedical Engineering, University of Minnesota









Operations on Bits

1. Recall there are 8 bits in a byte, and that each bit can be “1” or “0”.
2. A computer works with binary equivalents of numbers. An unsigned byte “11111111” would be equal to 255 (or 2^8-1).
3. The rightmost bit is the lowest order or least significant bit. The leftmost is the most significant or high-order bit.
4. A negative number is handled as “two’s compliment” (take the compliment of each bit), reserving the most significant bit to indicate the sign. A “1” meaning it is a negative number. Allowing for this, a signed integer of one byte can have a range of -128 to 127 (-2^{n-1} to $2^{n-1}-1$, where n = 8).
5. Operators include bitwise AND, inclusive-OR, and exclusive-OR; ones compliment; left shift and right shift.

Visualizing Bits and Byte

Bit	8	7	6	5	4	3	2	1
Power	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Binary	1	1	1	1	1	1	1	1
Decimal	128	64	32	16	8	4	2	1

As shown the binary value is 255. If you reserve the 8th bit for the sign, and bits 7 to 1 were all “1”, then the largest decimal value would be 127. Why is the largest negative -128?

Input & Output to a Terminal

1. All I/O operations are carried by calling functions in the standard C library.
2. Recall: #include <stdio.h> - this file contains function declarations and macro definitions.
3. Character I/O:
   - *getchar* and *putchar*
4. Formatted I/O:
   - *printf* and *scanf*

Character I/O

getchar (a);

Read a single character of data and assign it to variable “a”.

putchar (b);

Display the character contained in the variable “b”.

For example:

```
#include <stdio.h>
char a, b;
…
int main (void)
program
{ …
getchar (a);   //read “a” from terminal
putchar (b);  //write “b” to the terminal
…
}
```

Printf(…)

- *printf (“%[flags] [width] [.prec] [hlL]”, type);*
  - It’s all about *formatting* the output – defining what you are outputting and what it should look like on the display.
  - Optional fields are in brackets.
  - Order is important.
  - We will first look at some tables that summarize what can be between the *%* and *type* (also called the conversion factor), and then some examples.
  - *\n* means move to the beginning of the next line.

Some Printf(…) Examples

printf (“%[flags] [width] [.prec] [hlL]”, type);

Example:	Displayed Result:
printf (“Hello world! \n”);	Hello world! (\n - begin new line)
int i = 425;	
printf (“%i %o %x %u\n”, i, i, i, i);	425 651 1a9 425
float f = 12.978F;	
printf (“%f %e %g\n”, f, f, f);	12.978000 1.297800e+01 12.978

Kochan, S.G. *Programming in C*, 3rd ed., Developer’s Library, Indianapolis, Indiana (2005).

Character Examples

printf (“%[flags] [width] [.prec] [hlL]”, type);

Example:	Displayed Result:
char c = ‘X”;	
printf (“%c\n”, c);	X
printf (“%3c%3c\n”, c, c);	X X (field width of 3)
char s[ ] = “abcdefg”;	
printf (“%s\n”, s);	abcdefg (display the string)
printf (“%.5s\n”, s);	abcde (display 5 characters)
printf (“%10s\n”, s);	abcdefg (field width of 10, right justified)

Flags

printf ("%[flags] [width] [.prec] [hlL]", type);

Flag	Meaning
-	Left justify value
+	Precede value with + or -
(space)	Precede positive value with space
character)	Zero fill numbers
#	Precede octal value with 0, hexadecimal value with 0x; display decimal point for floats; leave trailing zeros for g or G format

Kochan, S.G. *Programming in C*, 3rd ed., Developer's Library, Indianapolis, Indiana (2005).

Width and .Precision Modifiers

printf (“%[flags] [width] [.prec] [hlL]”, type);

<u>Specifier</u>	<u>Meaning</u>
number	Maximum size of field
*	Take next argument to printf as size of field
.number	Minimum number of digits to display for integers; number of decimal places for e or f formats. maximum number of significant digits to display for g; maximum number of characters for s format.
.*	Take next argument to printf as precision (and interpret as indicated in the proceeding row)

Kochan, S.G. *Programming in C*, 3rd ed., Developer’s Library, Indianapolis, Indiana (2005).

Type Modifiers

printf (“%[flags] [width] [.prec] [hlL]”, type);

Type	Meaning
hh	Display integer argument as a character
h*	Display short integer
l*	Display long integer
ll*	Display long long integer
L	Display long double
j*	Display intmax_t or unimax_t value
t*	Display ptrdiff_t value
z*	Display size_t value

(* Can be placed in front of the n conversion character to indicate the corresponding pointer argument is of the specified type.)

Kochan, S.G. *Programming in C*, 3rd ed., Developer’s Library, Indianapolis, Indiana (2005).

Conversion Characters

printf (“%[flags] [width] [.prec] [hlL]”, type);

Char	Use to Display
i or d	Integer
u	Unsigned integer
o	Octal number
x	Hexadecimal integer; using a-f
X	Hexadecimal integer; using A-F
f or F	Floating point number, to six decimal places by default
e or E	Floating point number in exponential format (e places lower and E upper case)
g	Floating point number in f or e format
a or A	Floating point number in hexadecimal format 0xd.dddp+/-d
c	Single character
s	Null-terminated string
p	Pointer
n	Doesn’t print – stores the number of characters written so far by this call inside the int pointed to by the corresponding argument.
%	Percent

Kochan, S.G. *Programming in C*, 3rd ed., Developer’s Library, Indianapolis, Indiana (2005).

Scanf

1. Method for reading data into your program.
2. Like printf, it takes optional modifiers between the % and the modifier.
3. Usually, when searching the input stream for a value to read, it bypasses whitespace characters – blank space, tabs, carriage return, new line and from feed.
4. A *%c* will read the next character no matter what it is, or if it is a string within brackets.
5. When reading the value is terminated when the field width has been reached or until an invalid character is read.

Scanf Conversion Modifiers

Modifier	Meaning
*	Field is to be skipped and not assigned
size	Maximum size of the input field
hh	Value is to be stored in a *signed* or *unsigned* char
h	Value is to be stored in a *short int*
l	Value is to be stored in a long int, double or wchar_t
j, z, or t	Value is to be stored in a size_t (%j), ptrdiff_t (%z), intmax_t, or unimax_t (%t)
ll	Value is to be stored in a *long int*
L	Value is to be stored in a *long double*
type	Conversion character

Scanf Conversion Characters

Character	Action
d	Value to be read is in decimal notation, argument is a pointer to an *int*, unless h, l, or ll modifier is used, in which case the argument is a pointer to a *short, long, or long long*.
i	Like d, except numbers expressed in *octal* (leading 0) or *hexadecimal* (leading 0x or 0X) also can be read.
u	Value is an *integer*, and the argument is a point to an *unsigned int*.
o	The value to be read is in *octal* notation, and the argument is a pointer to an *int*, unless h, l, or ll modifier used.
x	The value to be read is expressed in *hexadecimal* notation
a, e, f, g	The value to be read is expressed in *floating-point* notation. The corresponding argument is a pointer to *float*, unless an l or L modifier is used.

More…

Character	Action
c	The value to be read is a single character. The argument is a *pointer* to a *character array*.
s	The value to be read is a *sequence of characters.*
[…]	A character string is to be read.
n	Nothing gets read.
p	The value to be read is a *pointer*, and the argument is a *pointer to a pointer to void*.
%	The next non-whitespace character on input must be a %.

Scanf Examples

Example…	*Text Entered…*	*Reads…*
scanf (“%i%c”, &i, &c);	29 w	29 stored in *i*, *space* in *c*
scanf(“%i %c”, &i, &c);	29 w	29 stored in *i*, w in *c*
scanf (“%i %5c %*f %s”, &il, text, string);	144abcde 736.55 (wine & cheese)	144 stored *il*, abcde to character array *text*, 735.55 is matched but not assigned, “(wine” to *string*
The next call to scanf picks up where the last one left off…		
scanf(“%s %s %i”, string 2, string 3, &i2);		& to *string2* cheese) to *string3* Waits for an integer to be typed.

Kochan, S.G. *Programming in C*, 3rd ed., Developer’s Library, Indianapolis, Indiana (2005).

Special Functions for Files

1) *fopen* - opens the file and creates a pointer for reading, writing or appending to the file;
2) *getc* and *putc* - reading and writing characters to the file.
3) *fclose* – closes file.
4) *feof* - test for end of file.
5) *fprintf* and *fscanf* – reading or writing data from a file.
6) *fgets* and *fputs* – reading and writing lines of data.
7) *stdin*, *stout* and *stderr* – defined in <stdio.h>

Preprocessor Command: #define

- #define - assigns symbolic names to a constant
  - e.g. *#define CARD 6* – defines the name card and assigns a value of 6. (Capitalized is optional)
  - Anywhere (except in a character string) that ‘card’ is used, it will be substituted by the value 6.
  - May appear anywhere in the program.
  - Examples: *#define PI 3.1415926*, *#define TWO_PI 2.0 * 3.1415926, #define AND && , #define OR ||, or #define EQUALS ==.*

- #define is also known as a *macro* because it can take an argument like a function.

  e.g. #define SQUARE(x) x*x

  y = SQUARE (v); //v^2 is assigned to y

  - The type of the argument is unimportant.
  - Becomes resident in the program (more memory but faster execution).

#include

- A method of grouping all of your macros together into a separate file, then including them into your program. Typically placed at the beginning. Examples: <*stdio.h*>, <*float.h*>, <*limit.h*>
- These files end with *.h*
- May be contained in a *libraries folder* when working with Arduino and other microcontrollers.
- Placing in < > tells the compiler to look for the file in a specific location.
- Once created, they can be used in any program.

Working with Large Programs

- Large programs, i.e. > 100 statements, might benefit from entering some of the code in separate *modules*.
  - A module is a function or number of related functions that you choose to group.
  - Allows for easier editing and a team approach.
  - These multiple source files are brought together at the time of compilation (a command line).

Summary

- Input/output to a terminal, and printf/scanf formatting
- File management
- Preprocessor commands - #define, #include
- Working with large files