Data Types

The basic data types in C are the following: `char, int, float,` and `double`. Each of the basic types has modifiers: `unsigned, long, short, *` and several other miscellaneous modifiers: `void, const, register, volatile, struct, typdef,` and `static`.

`char` is a character which is one byte long x86 and can be signed or unsigned. As always it is up to the programmer to figure out if the program requires a signed or unsigned variable. The default modifier is `signed` for most types. (The compiler for our robot default to unsigned.) An example declaration follows:

```
unsigned char c;
```

The syntax for a variable is <modifier> <type> <variable name>. In the above example the modifier is `unsigned`, the type is `char`, and the variable name is `c`. Each variable declared must have a type, name, and end in a semicolon. The variable may have no, one, or more modifiers.

`int` is typically a 32 bit signed number (for x86). `long` and `short` change the size of an `int`. A `short int` is typically 2 bytes and a `long int` can be 4 or 8 bytes. `float` is single precision floating point number, `double` is a double precision floating point number. The size of float and double vary from machine to machine. Use the macro `sizeof()` to find the size in bytes of a type. To create a pointer for a type just place a `*` after the type:

```
unsigned char * character_pointer;
```

printf() again

`printf()` is a confusing function that prints out just about any format you want. For now, just the basics will do. `printf()` has conversion types so that different data types can be printed. The syntax for `printf()` is as follows:

```
printf("%<CC>",<variable>);
```

where <CC> is the conversion character and <variable> is a variable in a program. The conversion characters are as follows:

CC	Input Argument	Format of Output
d	integer	singed decimal integer
i	integer	signed decimal integer
o	integer	unsigned octal integer
u	integer	unsigned decimal integer
x	integer	unsigned lowercase hexadecimal integer
X	integer	unsigned uppercase hexadecimal integer
f	floating point	[-]dddd.dddd
e	floating point	[-]d.ddde[+/-]ddd
g	floating point	[-]dddd.dddd
E	floating point	[-]d.ddddE[+/-]ddd
G	floating point	[-]dddd.dddd
c	character	single character
s	string pointer	print characters until a NULL character is encountered
%	none	prints a %
p	pointer	prints a pointer 0xYYYYYYYY

For example to print an integer variable:

```
printf("%d\n",my_int);
```

prints the value of `my_int`. The `\n` character prints the correct character(s) to move to the next line. On a MAC it will print a carriage return, on a PC it will print a carriage return and line feed and in Linux it will print a line feed. To print more than one variable just add the appropriate modifier prefixed with a `%`:

```
int my_int;
char my_char;

my_int=4;
my_char='D';

printf("%d$%c - HELLO!\n"my_int,my_char);
```

prints

```
# a.out
4$D -- HELLO!
```

Using `sizeof()` and `printf()` write a program on the supercomputer that will print the size of a `short int`, `int`, `long int`, `long long int`, `float`, `double`, and `char *`. Format the output so it is readable. Here's a little snippet to get you started:

```
#include <stdio.h>

// A program to print sizes of data types
int main(int argc, char * argv[])
{
        // Size of a char
        printf("The size of an char is:     %d\n",sizeof(char));

        /* more sizeof() and printf() */

        return 0;
}
```

Next time we'll look at a simpler way to write this using return values from functions and macros.