Notes on Data Types in C

All variables must be *declared* before they can be used in a C program. For most of what we will do in ME 30, a declaration statement will consist of two parts[1]:

1. a type specifier for the *type of data* the variable will hold (i.e., integer, floating point, etc.)
2. an identifier specifying the *name* of the variable[2]

The format for declaring a variable is:

type_specifier   *name*;

where

type_specifier is a reserved keyword (or combination of keywords) that specifies the data type
name is the name or 'identifier' for the variable

(Note the semicolon (;) at the end of the declaration. You **MUST** include this or you will get an error in compilation).

Several variables of the same type can be declared in the statement by separating their names by commas.

Examples:

```
int my_var;             /* declares my_var to be the identifier for an integer */
long int count;         /* declares count to be the identifier for a long integer */
unsigned char uch1, uch2;   /* declares two unsigned character variables */
```

Hierarchy of Data Types

Figure 1[3] shows the hierarchy of data types.

Memory Allocation for the Arithmetic Data Types

It will depend on the compiler and target system as to the amount of memory that will be allocated for the arithmetic data types and therefore the range of values that they can represent. Table 1 summarizes the arithmetic types and gives the memory allocations that AVR-GCC (a compiler for a popular 8-bit microcontroller) and Microsoft Visual C++ (a popular C compiler for Windows PCs) use. Note that the integral types can be signed or unsigned. Unsigned means that the range includes only non-negative numbers (which includes zero). If you leave off the specifier for signed or unsigned when declaring integral types, most compilers will default to signed, but this is not guaranteed, and sometimes the default can be set by the programmer.

Recommendations

Think about what kind of data it is you need to work with, and declare the appropriate type of variable or constant. Keep in mind that integers will be handled much more quickly than floating point types.

[1] There is much more to be said on the subject of declarations. For a definitive treatment, see the Standard C Reference by P. J. Plauger and Jim Brodie (available at: http://www.mers.byu.edu/docs/standardC/index.html)

[2] See the handout, "Notes on Variable Names" for more information on names.

[3] Adapted from Darnell, P. A. & Margolis, P. E. (1996) C, a software engineering approach, 3rd ed., Springer, New York, p. 58. There is much more to be said on the subject of data types. See the reference from footnote 1 above and Cheng, Harry H. (2010). C for Engineers and Scientists: An Interpretive Approach, McGraw-Hill, New York. p. 47-65., and An Introduction to the C Programming Language
and Software Design by Tim Bailey, (available at: http://www-personal.acfr.usyd.edu.au/tbailey/ctext/index.html).

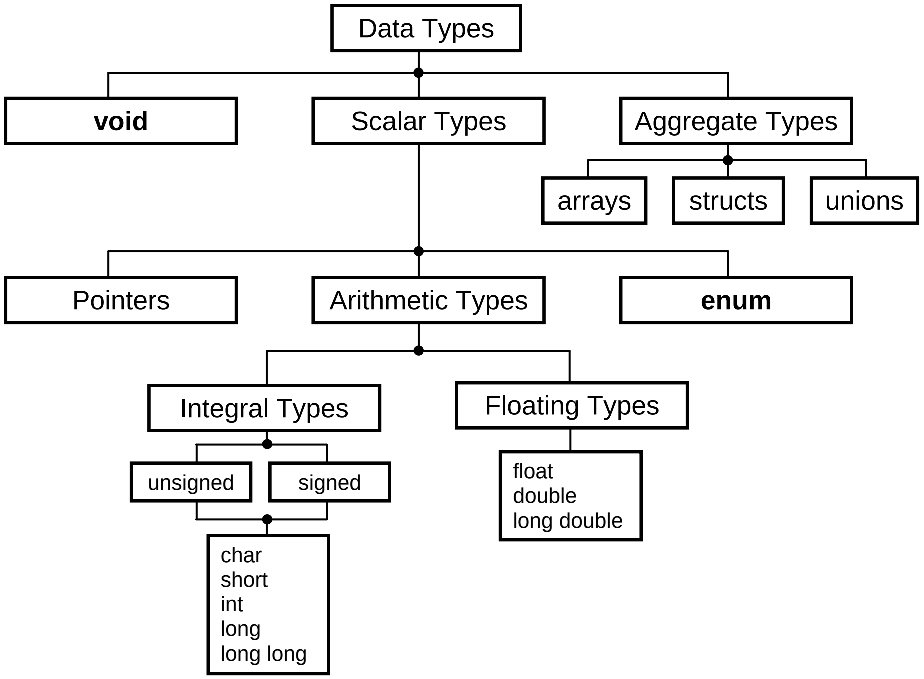

Figure 1. Hierarchy of data types[4].

Table 1. Memory Allocation for the Arithmetic Data Types[5]. AVR GCC is a compiler for Atmel microcontrollers. MSVC++ is Microsoft's Visual C++ compiler.

		Size (bytes)		**Range of Values**	
Name (alternate)	**Description**	AVR GCC	**MSVC++**	AVR GCC	**MSVC++**
char	Character or small integer	1	**1**	Signed: -128 to 127 ($-2^{(8-1)}$ to $2^{(8-1)}-1$) Unsigned: 0 to 255 (0 to 2^8-1)	**Signed: -128 to 127** **Unsigned: 0 to 255**
short int (short)	Short integer	2	**2**	Signed: -32768 to 32767 Unsigned: 0 to 65535	**Signed: -32,768 to 32,767** **Unsigned: 0 to 65535**
int	Integer	2	**4**	Signed: -32768 to 32767 Unsigned: 0 to 65535	**Signed: -2,147,483,648 to 2,147,483,647** **Unsigned: 0 to 4,294,967,295**
long int (long)	Long integer	4	**4**	Signed: -2,147,483,648 to 2,147,483,647 Unsigned: 0 to 4,294,967,295	**Signed: -2,147,483,648 to 2,147,483,647** **Unsigned: 0 to 4,294,967,295**
long long int (long long)	Really long integer	8	**8**	Signed: ≈ -9.2E+18 to ≈ 9.2E+18 Unsigned: 0 to ≈ 1.8E+19	**Signed: ≈ -9.2E+18 to ≈ 9.2E+18** **Unsigned: 0 to ≈ 1.8E+19**
float	'Single-precision' floating point number	4	**4**	≈ ± 1E ± 38 (7 decimal digits of precision)	**≈ ± 1E ± 38 (7 decimal digits of precision)**
double	'Double-precision' floating point number	4	**8**	≈ ± 1E ± 38 (7 decimal digits of precision)	**≈ ± 1E ±308 (15 decimal digits of precision)**
long double	Long double-precision floating point number		**8**		**≈ ± 1E ±308 (15 decimal digits of precision)**

[4] Ibid.

[5] Microsoft Visual C++ information found at http://msdn.microsoft.com/en-us/library/s3f49ktz(VS.80).aspx.