Declaration and Definition

A C/C++ program file begins with a comment section, the include statements, and the define statements. Next are the function declarations. Declarations are patterns which the compiler uses to recognize each function, and each variation on a function (polymorphism). The compiler uses these declarations to build its lookup table. They must be present in a program before the function is called within main(), or by any other function. If a function is not declared before it is called the compiler will throw an error because that function is not present in its lookup table.

The definition of a function is where it is fully written in code. Since a definition of a function also declares it, you can be sure the compiler has it in its lookup table by fully defining a function before the main() function is entered. You can either place main() before the function definitions it calls, or after them. If you define your functions after main() then you must declare each function near the beginning of the file.

In a windowing application I find myself using the same WinMain() function and window procedures, WinProc(), so I put them at the end of the source file. Since I won't be modifying them much, or at all in most cases, it is easier to have them near the end of the file. Thus, the functions which are getting added, edited, debugged, and modified are present before the main function and there is no need for a declaration section. I create the declaration section as my program grows larger, when I find myself factoring my code into smaller, more closely related functions. I clip the declarations and related functions, and place them in their own file, later in code development. Place a #include statement in the main file referencing the new file you have created. This enables reuse of those functions in other applications.

I like to have all the parts available in what I call a "flat" file format as I am building my application from my plans. The flat file has all the parts readily available in one place. While the application is in a state of flux I page forward, or backward, to find them. After each each function has been debugged and edited, I create a declaration from it near the beginning of the file, and move the function toward the end. Later in the coding cycle, I collect all of the functions and declarations which fall into a similar category, and create a new file from them. This cleans up the main file, ultimately to where there are only function calls in it, with a series of #include statements necessary to build them.

As you read through my code, you can determine how mature any file is by how few functions are defined within it, and how many are called from the included files. Each time I reuse code from a .h file I scan through the associated .c or .cpp file, adding comments where necessary. Over time the included files become more familiar to me, and more trustworthy.

> K&R page 9 - "A declaration announces the properties of variables; it consists of a type name and a list of variables."

Functions can be declared either implicitly or explicitly.

The explicit function declaration

```
void swap(int *, int *);
```

tells the compiler to put the name swap into its list of functions noting it requires two integer pointers as inputs, and produces no (void) output.

The function is defined later in the file as (from K&R page 96)

```
void swap(int *px, int *py)
   {
   int temp;

   temp = *px;
   *px = *py;
   *py = temp;
   }
```

Placing the function definition of swap() before it is called creates an implicit declaration. You must either declare, or define, the function before it is called in a program, otherwise the compiler will complain by giving you an error message. You can do this in two ways. The way I have shown, where you declare the function right after the comment, #include, and #define statements. Or, you can define the function before it is used. It all depends on whether you want your main() function at the end of the file, or near the beginning. There is less typing if you define the function first but there is less clutter if you declare the function then define it later in the file.

After you declare your functions before you define them, you are ready to move the functions to their own file. Then you need to #include them so your main() function (and the compiler) can find them. I think of it as a process of winnowing your code. First you test the function with your code. Once you are assured it is working you move the definition to much later in the file and declare it early on. When that is working, you move the code into a .c or a .cpp file, and put the declarations into a .h or .hpp file. Add #include “yourFile.h” in your main file and all should work properly. This last step prepares you to reuse the new files in other applications. You should have generalized the code enough to use it in many more places without rewriting it.

Remember to use the flag -Wall in your makefile so every warning is reported, as well as every error generated during the compilation process. When you can type **make clean** followed by **make** and no warnings are returned, your code is clean. However, that does not mean it will run, there may still be logical errors in the code which the compiler cannot see. Use the compiler to locate the errors it can find, but use your brain to determine any logical errors. A computer cannot think, that is where you come in handy.