# Assembling | Coursera



## Assembling

## Assembling

The next step is to take the assembly that the compiler generated and *assemble* it into an *object file*. *gcc* invokes the assembler to translate the assembly instructions from the textual/human readable format into their numerical encodings that the processor can understand and execute. This translation is another example of the rule that "everything is a number"—even the instructions that the computer executes are numbers.

While it is possible to get error messages at this stage, it should not happen for anything you will try to do in this specialization. Generally errors here are limited to cases in which you explicitly write the specific assembly level instructions that you want into your program (which is possible in C, but limited to very advanced situations) and make errors in those.

The important thing to understand about this step is that it results in an *object file*. The object file contains the machine-executable instructions for the source file that you compiled, but is not yet a complete program. The object file may reference functions that it does not define (such as those in the C library, or those written in other files). You can request that gcc stop after it assembles an object file by specifying the *-c* option. By default, the name of the object file will be the name of the *.c* file with the *.c* replaced by *.o*. For example, *gcc -c xyz.c* will compile *xyz.c* into *xyz.o*. If you wish to provide a different name for the object file, use the -o option followed by the name you want. For example, *gcc -c xyz.c -o awesomeName.o* will produce an object file called *awesomeName.o*.

This ability to stop is important for large programs, where you will split the code into multiple different C files. Splitting the program into large files is primarily useful to help you keep the code split into manageable pieces. However, each source file can be individually compiled to an object file, and those object files can then be linked together (as we will discuss in the next section). If you change code in one file, you can recompile only that file (to generate a new object file for it), and re-link the program without recompiling any of the other source files. For programs that you write in this specialization, this will not make a difference. However, when you write real programs, you may have tens or hundreds of thousands of lines of code split across dozens or hundreds of files. In this case, the difference between recompiling one file and recompiling all files (especially if optimizations are enabled) may be the difference between tens of seconds and tens of minutes.