1. Multiple File Programs
   1. Programs do not have to be defined in the same file.
      1. Unwieldy
      2. Concept of reuse
   2. C and C++ can distribute source code across several files
   3. Best for functions to be used elsewhere (and by others) in separate files that can be easily included.
      1. Libraries
      2. .h files
   4. Adjustments necessary to…
      1. variables
      2. function definitions
         1. All in one file
         2. Function prototypes must be included in files where they are called but not defined.
            1. static can be used to do the same with functions
   5. Inclusion
      1. Same directory: #include “filename”
      2. Not same directory: #include <filename>
2. Variables
   1. Local/automatic variables: cease to exist after the function in which they are defined exits.
      1. Do not use the same variable name for different local variables, even if it does not confuse compiler.
   2. Global/external variables: valid throughout the entire file.
      1. By convention: right after preprocessor directives.
      2. Identifiable by extern identifier.
      3. If defied outside any function, does not have to be declared anywhere else inside the same file.
      4. Recognized across all files, but…
         1. Must be declared in each file in which they are used, preceded by extern
         2. Must use the exact same name for variables
      5. Scope can be limited to some but not all files.
         1. Identifier static must be used In order to make them global, but only accessible for certain functions within the same file.
         2. Internal linkage: declaration in the function definitions where the global variable is to be recognized.
3. UNIX: provides a way for the compiler to compile just the file changed, and link it to the rest of the already-compiled program.
   1. Used to fix small mistake
   2. make directive used