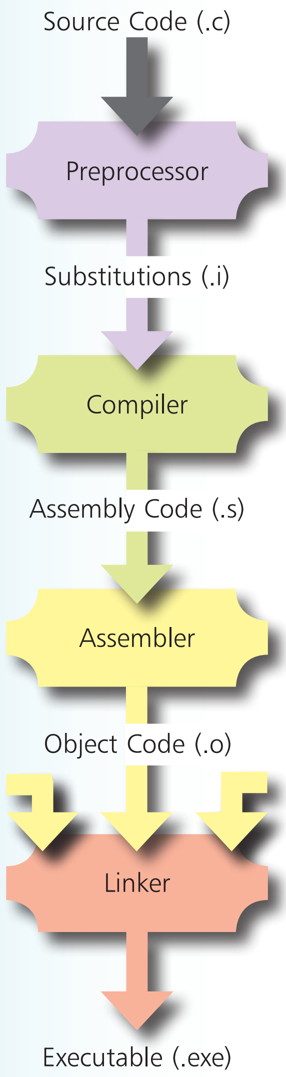
Understanding compilation

In producing an executable file from an original C source code file the compilation process actually undergoes four separate stages, which each generate a new file:



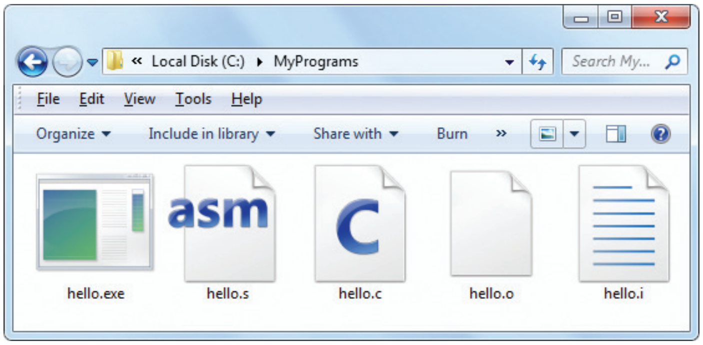
- Preprocessing – the preprocessor substitutes all preprocessor directives in the original source code **.c** file with actual library code that implements those directives. For instance, library code is substituted for **#include** directives. The generated file containing the substitutions is in text format and typically has a **.i** file extension
- Translating – the compiler translates the high-level instructions in the **.i** file into low-level Assembly language instructions. The generated file containing the translation is in text format and typically has a **.s** file extension
- Assembling – the assembler converts the Assembly language text instructions in the **.s** file into machine code. The generated object file containing the conversion is in binary format and typically has a **.o** file extension
- Linking – the linker combines one or more binary object **.o** files into a single executable file. The generated file is in binary format and typically has a .exe file extension

Strictly speaking "compilation" describes the first three stages above, which operate on a single source code text file and ultimately generate a single binary object file. Where the program source code contains syntax errors, such as a missing semi-colon statement terminator or a missing parenthesis, they will be reported by the compiler and compilation will fail.

The linker, on the other hand, can operate on multiple object files and ultimately generates a single executable file. This allows the creation of large programs from modular object files that may each contain re-usable functions. Where the linker finds a function of the same name defined in multiple object files it will report an error and the executable file will not be created.

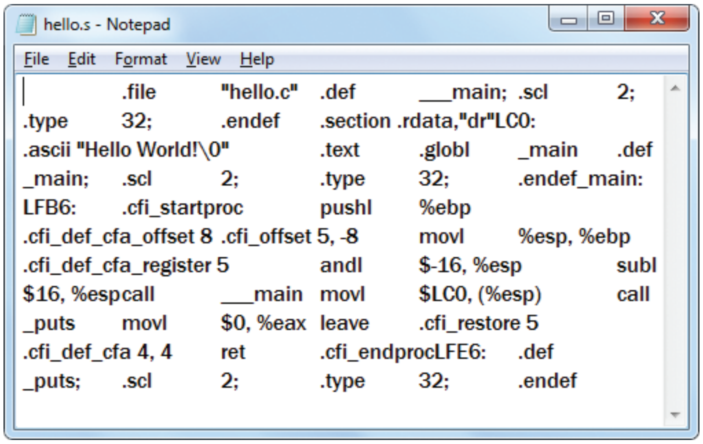
Normally the temporary files created during the intermediary stages of the compilation process are automatically deleted, but they can be retained for inspection by including a **-save-temps** option in the compiler command:

1. At a command prompt in the **MyPrograms** directory, type **gcc hello.c -save-temps -o hello.exe** then hit Return to re-compile the program and save the temporary files


Local Disk (C:) ▸ MyPrograms
Search My...
File Edit View Tools Help
Organize ▾ Include in library ▾ Share with ▾ Burn »
hello.exe
hello.s
hello.c
hello.o
hello.i

2. Open the **hello.i** file in a plain text editor, such as Windows' Notepad, to see your source code at the very end of the file preceded by substituted **stdio.h** library code

3. Now open the **hello.s** file in a plain text editor to see the translation into low-level Assembly code and note how unfriendly that appears in contrast to the C code version



```
          .file       "hello.c"   .def      ___main;  .scl      2;
.type     32;         .endef      .section .rdata,"dr"LC0:
.ascii "Hello World!\0"           .text     .globl    _main     .def
_main;    .scl        2;          .type     32;       .endef_main:
LFB6:     .cfi_startproc          pushl     %ebp
.cfi_def_cfa_offset 8 .cfi_offset 5, -8     movl      %esp, %ebp
.cfi_def_cfa_register 5           andl      $-16, %esp          subl
$16, %espcall         ___main     movl      $LC0, (%esp)        call
_puts     movl        $0, %eax    leave     .cfi_restore 5
.cfi_def_cfa 4, 4     ret         .cfi_endprocLFE6:   .def
_puts;    .scl        2;          .type     32;       .endef
```

Hot tip

Programs tediously written in Assembly language can run faster than those written in C but are more difficult to develop and maintain. For traditional computer programming C is almost always the first choice.