Introducing your best friend: Mr. Compiler

In this hands-on you we will briefly explore the role of your compiler that should become your best friend when programming your scientific problem.

I am gcc, solving problems

Let us start with a very simple compilation of a simple code using the always available compiler on almost all the machine: the GNU suite and namely `gcc`

GCC is a key component of so-called "*GNU Toolchain*", for developing applications and writing operating systems. The GNU Toolchain includes:

1. GNU Compiler Collection (GCC): a compiler suite that supports many languages, such as C/C++ and Fortran as well.
2. GNU Make: an automation tool for compiling and building applications.
3. GNU Binutils: a suite of binary utility tools, including linker and assembler.
4. GNU Debugger (GDB).
5. GNU Autotools: A build system including Autoconf, Autoheader, Automake and Libtool.
6. GNU Bison: a parser generator (similar to lex and yacc).

GCC is *portable* and run in many operating platforms. GCC (and GNU Toolchain) is currently available on all Unixes. They are also ported to Windows (by Cygwin, MinGW and MinGW-W64). GCC is also a *cross-compiler*, for producing executables on different platform.

So let's first check if the compiler is available and which version we have on our system:

on my mac:

```
>gcc
clang: error: no input files
```

on my supercomputer:

```
$ gcc
gcc: fatal error: no input files
compilation terminated.
```

and which version we have on our system:

on my mac:

```
>gcc --version
Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --with-gxx-include-
dir=/usr/include/c++/4.2.1
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin15.6.0
Thread model: posix
InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin
```

and on my supercomputer:

```
>gcc --version
gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-36)
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Getting help

Help You can get the help manual via the --help option. For example,

```
>gcc --help
OVERVIEW: clang LLVM compiler

USAGE: clang [options] <inputs>

OPTIONS:
  -###                    Print (but do not run) the commands to run for this compilation
  --analyze               Run the static analyzer
  -arcmt-migrate-emit-errors
                          Emit ARC errors even if the migrator can fix them
  -arcmt-migrate-report-output <value>
                          Output path for the plist report
  --cuda-device-only      Do device-side CUDA compilation only
  --cuda-host-only        Do host-side CUDA compilation only
  --cuda-path=<value>     CUDA installation path
  -cxx-isystem <directory>
                          Add directory to the C++ SYSTEM include search path
  -c                      Only run preprocess, compile, and assemble steps
```

You can also read the GCC manual pages (or man pages) via the man utility:

```
> man gcc
GCC(1)                                                          GNU
GCC(1)

NAME
       gcc - GNU project C and C++ compiler

SYNOPSIS
       gcc [-c|-S|-E] [-std=standard]
           [-g] [-pg] [-Olevel]
           [-Wwarn...] [-Wpedantic]
           [-Idir...] [-Ldir...]
           [-Dmacro[=defn]...] [-Umacro]
           [-foption...] [-mmachine-option...]
           [-o outfile] [@file] infile...

       Only the most useful options are listed here; see below for the remainder.  g++ accepts
mostly the same options as gcc.

DESCRIPTION
       When you invoke GCC, it normally does preprocessing, compilation, assembly and linking.  The
"overall options" allow you to
       stop this process at an intermediate stage.  For example, the -c option says not to run the
linker.  Then the output consists
       of object files output by the assembler.
...
```

Alternatively, you could look for an online man pages, e.g., http://linux.die.net/man/1/gcc.

We finally check where the actual files are...

```
>whereis gcc
gcc: /usr/bin/gcc /usr/lib/gcc /usr/libexec/gcc /usr/share/man/man1/gcc.1.gz
```

Compiling and Linking a simple C program

Given the following hello_world.c file:

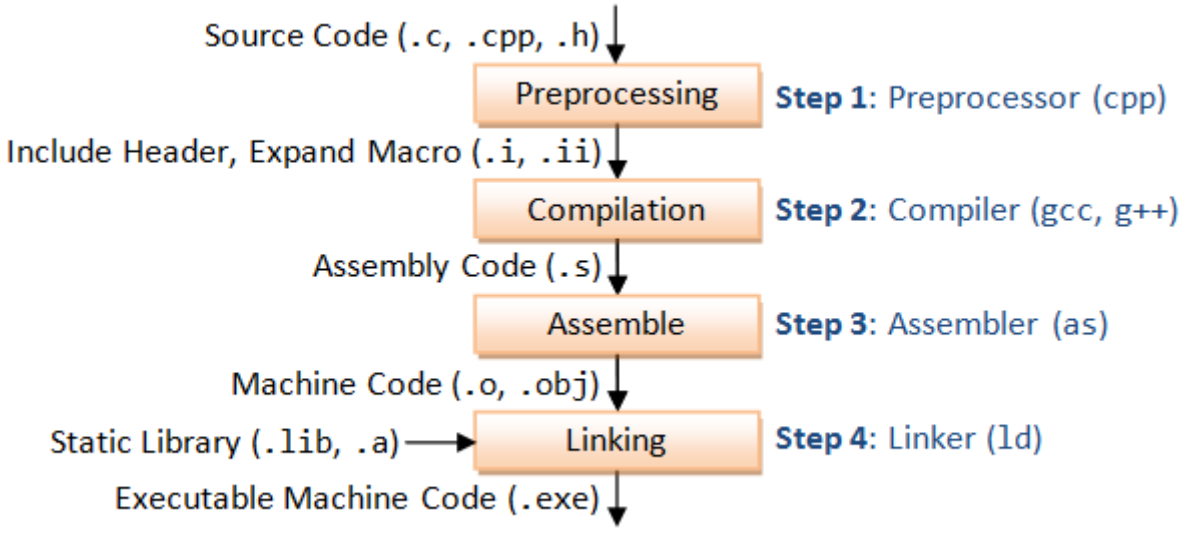
```c
// hello.c
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

Compilation and Linking is simple enough:

```
gcc hello.c -o hello.x
```

GCC however compiles a C/Fortran program into executable in 4 steps as shown in the diagram below:



For example, a " `gcc -o hello.x hello.c` " is carried out as follows:

1. Pre-processing: via the GNU C Preprocessor (`cpp`) which includes the headers (`#include`) and expand the macros (`define`)

```
> cpp hello.c > hello.i
```

The resultant intermediate file `hello.i` contains the expanded source code.

2. Compilation: The compiler compiles the pre-processed source code into assembly code for a specific processor.

```
> gcc -S hello.i
```

The `-S` option specifies to produce assembly code, instead of object code. The resultant assembly file is `hello.s`

3. Assembly: The assembler (`as`) converts the assembly code into machine code in the object file `hello.o`

```
> as -o hello.o hello.s
```

4. Linker: Finally, the linker (`ld`)links the object code with the library code to produce an executable file `hello.x`

```
> ld -o hello.x hello.o ...libraries...
```

Verbose Mode (-v)

You can see the detailed compilation process by enabling `-v` (verbose) option. For example,

```
> gcc -v -o hello.x hello.c
```

Defining Macro (-D)

You can use the `-D*name*` option to define a macro, or `-D*name*=*value*` to define a macro with a value. The `*value*` should be enclosed in double quotes if it contains spaces.

```
>gcc -DDEBUG demo.c - o demo_debug.x
>./demo_debug.x
sin(x) is -0.756802
```

Useful utilities to check your executable

"nm" Utility - List Symbol Table of Object Files

The utility " `nm` " lists symbol table of object files. For example,

```
$ nm hello.o
0000000000000000 b .bss
0000000000000000 d .data
0000000000000000 p .pdata
0000000000000000 r .rdata
0000000000000000 r .rdata$zzz
0000000000000000 t .text
0000000000000000 r .xdata
                 U __main
0000000000000000 T main
                 U puts

$ nm hello.exe | grep main
00000001004080cc I __imp___main
0000000100401120 T __main
00000001004010e0 T main
......
```

"nm" is commonly-used to check if a particular function is defined in an object file. A `'T'` in the second column indicates a function that is *defined*, while a `'U'` indicates a function which is *undefined* and should be resolved by the linker.

"ldd" Utility - List Dynamic-Link Libraries

The utility " `ldd` " examines an executable and displays a list of the shared libraries that it needs. For example,

```
> ldd hello.x
ntdll.dll => /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll (0x7ff9ba3c0000)
KERNEL32.DLL => /cygdrive/c/WINDOWS/System32/KERNEL32.DLL (0x7ff9b9880000)
KERNELBASE.dll => /cygdrive/c/WINDOWS/System32/KERNELBASE.dll (0x7ff9b6a60000)
SYSFER.DLL => /cygdrive/c/WINDOWS/System32/SYSFER.DLL (0x6ec90000)
ADVAPI32.dll => /cygdrive/c/WINDOWS/System32/ADVAPI32.dll (0x7ff9b79a0000)
msvcrt.dll => /cygdrive/c/WINDOWS/System32/msvcrt.dll (0x7ff9b9100000)
sechost.dll => /cygdrive/c/WINDOWS/System32/sechost.dll (0x7ff9b9000000)
RPCRT4.dll => /cygdrive/c/WINDOWS/System32/RPCRT4.dll (0x7ff9b9700000)
cygwin1.dll => /usr/bin/cygwin1.dll (0x180040000)
```

Useful flag/tips to help writing clean code

- Enable `-Wall` to show all warnings:

```
>gcc -Wall   demo.c
demo.c: In function 'main':
demo.c:13:8: warning: unused variable 'f' [-Wunused-variable]
   13 |  float f=sin(x);
      |        ^
```

- Promote warnings to errors to be force to fix them:

```
>gcc -Wall -Werror=unused-variable   demo.c
demo.c: In function 'main':
demo.c:13:8: error: unused variable 'f' [-Werror=unused-variable]
   13 |  float f=sin(x);
      |        ^
cc1: some warnings being treated as errors
```