Compilers, linkers, and libraries

Source code

- Human readable? code that describes what we want the computer to do
- Common suffixes:
  - .c,.cpp,.c++,.f,.f77,.f90,.F,.f2003,.f66,java

Compiler

- Takes the source code and turns into a set of assembly instructions
- Three stages
  - Pre-process
  - Compile
  - Assemble

Example

```
#include <stdio.h>
#define STRING "Hello World"
int main(void)
{
/* Using a macro to print 'Hello
World'*/
printf(STRING);
return 0;
}
```

http://www.thegeekstuff.com/2011/10/c-program-to-an-executable/

Pre-process

```
......
......
......
......
# 846 "/usr/include/stdio.h" 3 4
extern FILE *popen (__const char *__command, __const char *__modes) ;
extern int pclose (FILE *__stream);
extern char *ctermid (char *__s) __attribute__ ((__nothrow__));

# 886 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__));
extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__));

# 916 "/usr/include/stdio.h" 3 4
# 2 "print.c" 2

int main(void)
{
printf("Hello World");
return 0;
}
```

http://www.thegeekstuff.com/2011/10/c-program-to-an-executable/

Pre-process

```
......
......
......
......
# 846 "/usr/include/stdio.h" 3 4
extern FILE *popen (__const char *__command, __const char *__modes) ;
extern int pclose (FILE *__stream);
extern char *ctermid (char *__s) __attribute__ ((__nothrow__));

# 886 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__));
extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__));

# 916 "/usr/include/stdio.h" 3 4
# 2 "print.c" 2

int main(void)
{
printf("Hello World");
return 0;
}
```

1. Macro expansion
2. Comments stripped
3 Expansion of included files

http://www.thegeekstuff.com/2011/10/c-program-to-an-executable/

Compiling

```
.file "print.c"
.section .rodata
.LC0:
.string "Hello World"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
movq %rsp, %rbp
.cfi_offset 6, -16
.cfi_def_cfa_register 6
movl $.LC0, %eax
movq %rax, %rdi
movl $0, %eax
call printf
movl $0, %eax
leave
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (Ubuntu 4.4.3-4ubuntu5) 4.4.3"
.section .note.GNU-stack,"",@progbits
```

Converts code into a set of assembly instructions. Along the way optimizes the code

http://www.thegeekstuff.com/2011/10/c-program-to-an-executable/

Assembly

```
^?
ELF^B^A^A^@^@^@^@^@^@^@^@^@^A^@>^@^A^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@0^
^@UH<89>å¸^@^@^@^@H<89>Ç¸Hello World^@^@GCC: (Ubuntu
4.4.3-4ubuntu5) 4.4.3^@^
T^@^@^@^@^@^@^AzR^@^Ax^P^A^[^L^G^H<90>^A^@^@^\^@^@]^@
^@^@^@A^N^PC<86>^B^M^F
^@^@^@^@^@^@^@^@.symtab^@.strtab^@.shstrtab^@.rela.text
^@.data^@.bss^@.rodata
^@.comment^@.note.GNU-
stack^@.rela.eh_frame^@^@^@^@^@^@^@^
...
...
…
```

Converts assembly into machine level instructions

http://www.thegeekstuff.com/2011/10/c-program-to-an-executable/

How to compile

```
gcc -c -I/my/path/ -I./includes -O3 file.c -o file.o
```

Compiler

```
ifc -c -I/my/path/ -I./includes -O3 file.f90 -o file.o
```

How to compile

```
gcc -c -I/my/path/ -I./includes -O3 file.c -o file.o
```

Create an object file

```
ifc -c -I/my/path/ -I./includes -O3 file.f90 -o file.o
```

How to compile

```
gcc -c -I/my/path/ -I./includes -O3 file.c -o file.o
```

Paths to search for included files, order matters

```
ifc -c -I/my/path/ -I./includes -O3 file.f90 -o file.o
```

How to compile

```
gcc -c -I/my/path/ -I./includes -O3 file.c -o file.o
```

Optimizations to attempt (more later)

```
ifc -c -I/my/path/ -I./includes -O3 file.f90 -o file.o
```

How to compile

```
gcc -c -I/my/path/ -I./includes -O3 file.c -o file.o
```

File we want to compile

```
ifc -c -I/my/path/ -I./includes -O3 file.f90 -o file.o
```

How to compile

```
gcc -c -I/my/path/ -I./includes -O3 file.c -o file.o
```

Next argument is the name of the output file we want to create

```
ifc -c -I/my/path/ -I./includes -O3 file.f90 -o file.o
```

How to compile

```
gcc -c -I/my/path/ -I./includes -O3 file.c -o file.o
```

Name of output file

```
ifc -c -I/my/path/ -I./includes -O3 file.f90 -o file.o
```

Object files

- Object files contain machine level instructions for all the functions contained within the source code
- They do not contain instructions for functions not defined in the source code

Libraries

- Libraries can be thought of as a collection of like minded object files
- Created using
  - ar -c mylib.a file1.o file2.o file3.o file4.o

Static libraries

- suffix .a
- Code from library is included in executable
- When anything in the library is changed the executable needs to be relinked

Dynamic libraries

- suffix .so
- Executable stores only a request for a function from a specific library
- Functions of library are loaded at runtime
- At runtime a user's LD_LIBRARY_PATH variable (env |grep LD_LIBRARY_PATH) is searched for the requested dynamic library)

Dynamic vs. Static

- Static libraries
  - More portable code
- Dynamic libraries
  - Easier to introduce bug fixes/ improvements in underlying libraries
- Almost all system level libraries are dynamic

How to link

```
gcc file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

Linker

```
ifc file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

How to link

```
gcc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

Object file(s) to link

```
ifc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

How to link

```
gcc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

A directory to
search for libraries
(order matters)

```
ifc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

How to link

```
gcc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

A specific library I
want to link with

```
ifc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

How to link

```
gcc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

Search all of the
directories listed
with -L/a/path for a
file called libgp257.a
or libgp257.so

```
ifc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

How to link

```
gcc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

Name of executable
I want to create

```
ifc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

Debugging linking: The dreaded undefined symbol

```
gcc  file.o -L/my/path/ ./libmy.a -lgp257 -o file.x
```

- The linker attempt to find the code for every function called by the main
- Order matters
  - All functions undefined in file.o can be defined in libmy.a, libgp257.a, or system libs
  - All functions defined in libgp257.a must be defined in libgp257.a or the system libs

nm your friend

- When you are having problems figuring out a undefined symbol problem use nm
- nm file.o or nm file.a

```
0000000000000550 T aux_unlink
0000000000000460 T auxin
00000000000002c0 T auxinout
0000000000000390 T auxout
0000000000000150 T auxscr
0000000000000210 T auxsockout
00000000000005f0 T auxtmp
0000000000000100 T copy_history
0000000000000530 T fauxin
                 U fclose
                 U fopen
                 U free
                 U getch
00000000000000a0 T grab_history
```

nm your friend

- When you are having problems figuring out a undefined symbol problem use nm
- nm file.o or nm file.a

```
0000000000000550 T aux_unlink
0000000000000460 T auxin
00000000000002c0 T auxinout
0000000000000390 T auxout
0000000000000150 T auxscr
0000000000000210 T auxsockout
00000000000005f0 T auxtmp
0000000000000100 T copy_history
0000000000000530 T fauxin
                 U fclose
                 U fopen
                 U free
                 U getch
00000000000000a0 T grab_history
```

A symbol defined in this file

nm your friend

- When you are having problems figuring out a undefined symbol problem use nm
- nm file.o or nm file.a

```
0000000000000550 T aux_unlink
0000000000000460 T auxin
00000000000002c0 T auxinout
0000000000000390 T auxout
0000000000000150 T auxscr
0000000000000210 T auxsockout
00000000000005f0 T auxtmp
0000000000000100 T copy_history
0000000000000530 T fauxin
                 U fclose
                 U fopen
                 U free
                 U getch
00000000000000a0 T grab_history
```

A symbol I am looking for

nm and fortran

- In c function name to symbol is directly comparable (function adj_null becomes symbol adj_null)
- In fortran the name to symbol is undefined and varies from compiler to compiler
  - ADJ_NULL,ADJ_NULL_,ADJ_NULL__,adj_null_,adj_null__ are all possible
- This makes calling C from Fortran (and visa versa problematic) and often compiler dependent

nm and fortran90

- To make sure that the correct modules is linked with fortran90 introduced a further level of mangling

```
0000000000000000 N .debug_info_seg
                 U _intel_fast_memcpy
                 U box_mp_boxn_
                 U cartesian_mp_line2cart_
                 U for_alloc_allocatable
                 U for_check_mult_overflow64
                 U for_dealloc_allocatable
                 U for_deallocate
                 U for_write_seq_fmt
                 U for_write_seq_lis
0000000000000000 T print._
0000000000000010 T print_mp_printn_
0000000000000048 d print_mp_printn_$BLK$format_pack.0.2
0000000000000000 d var$144.0.2
```

Module   Function

nm and C++

- A similar story in C++

Class Function

```
00000000000001b0 T _ZN8clip_bar10paintEventEP11QPaintEvent
0000000000000d90 T _ZN8clip_bar10to_bar_ptsE7QString
0000000000001c20 T _ZN8clip_bar11clear_picksEv
0000000000000cc0 T _ZN8clip_bar11resizeEventEP12QResizeEvent
0000000000000000 T _ZN8clip_bar11to_pt_smallEffPiS0_
00000000000000a0 T _ZN8clip_bar12to_pct_smallEiiPfS0_
                 U _ZN8clip_bar14actionDetectedESt6vectorI7QStringSaIS1_EE
```