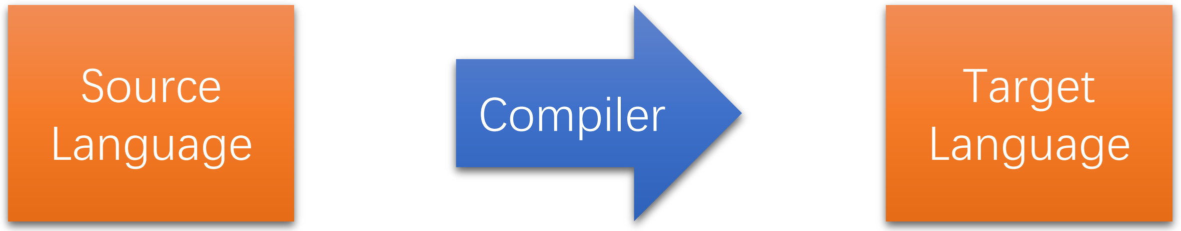
Compilation - Basics

Yuchen Gu

2022/01/08

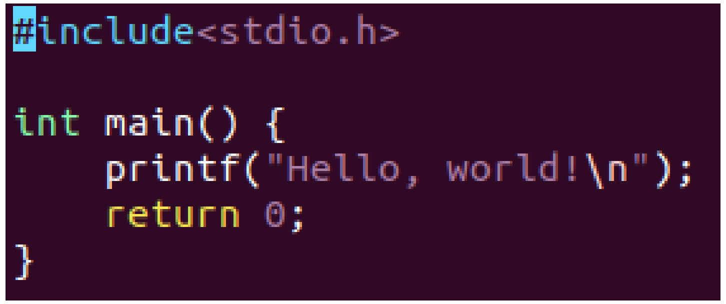
Compilation



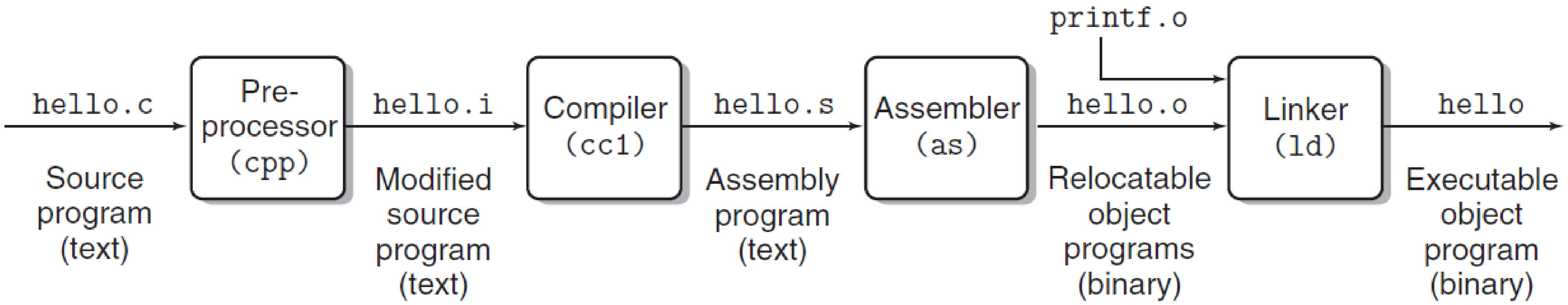
- C, C++, FORTRAN, Pascal, ……

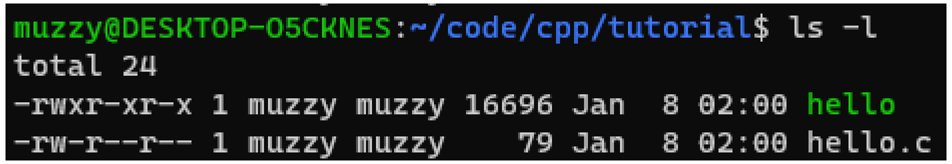
Compilation

- `gcc hello.c -o hello`

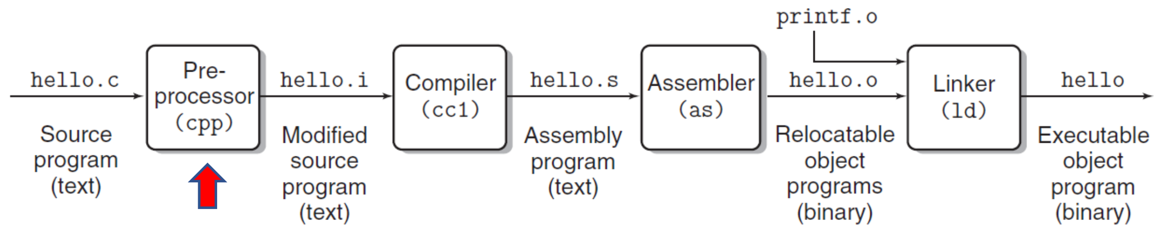


```
#include<stdio.h>

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



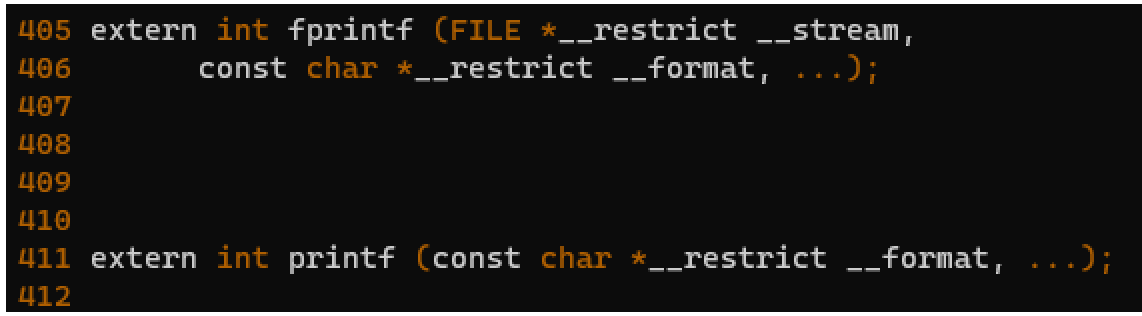


```
muzzy@DESKTOP-O5CKNES:~/code/cpp/tutorial$ ls -l
total 24
-rwxr-xr-x 1 muzzy muzzy 16696 Jan  8 02:00 hello
-rw-r--r-- 1 muzzy muzzy    79 Jan  8 02:00 hello.c
```

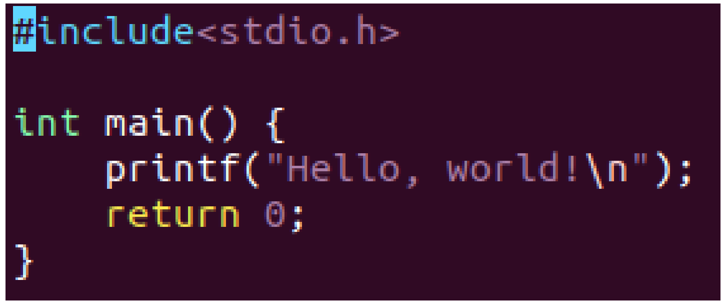


a. Pre-processing

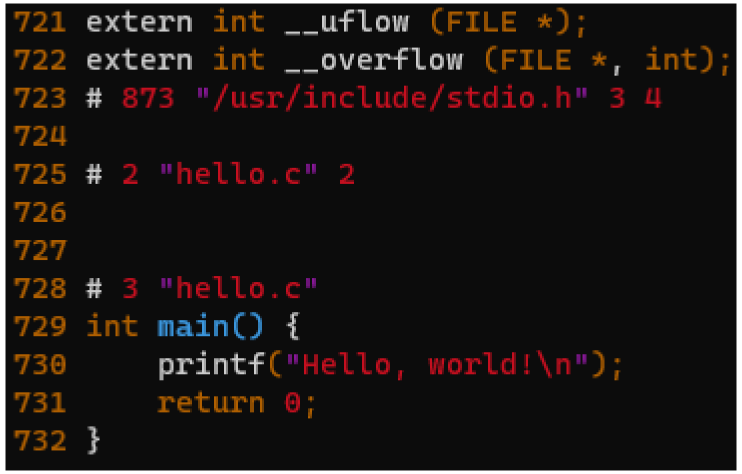
- Expands macros, defines, includes…
- `gcc -E hello.c -o hello.i`



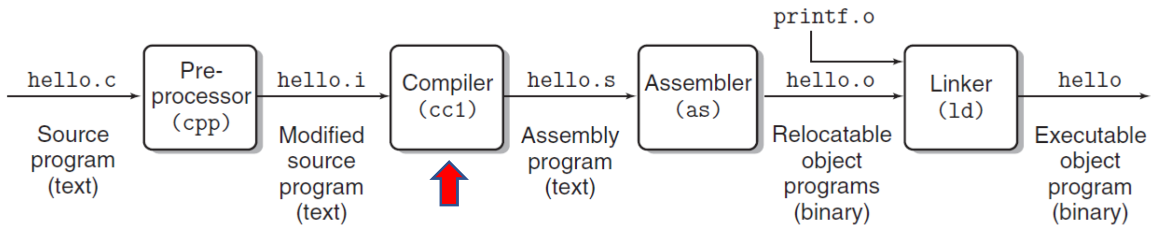
```
extern int fprintf (FILE *__restrict __stream,
        const char *__restrict __format, ...);




extern int printf (const char *__restrict __format, ...);

```



```
#include<stdio.h>

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

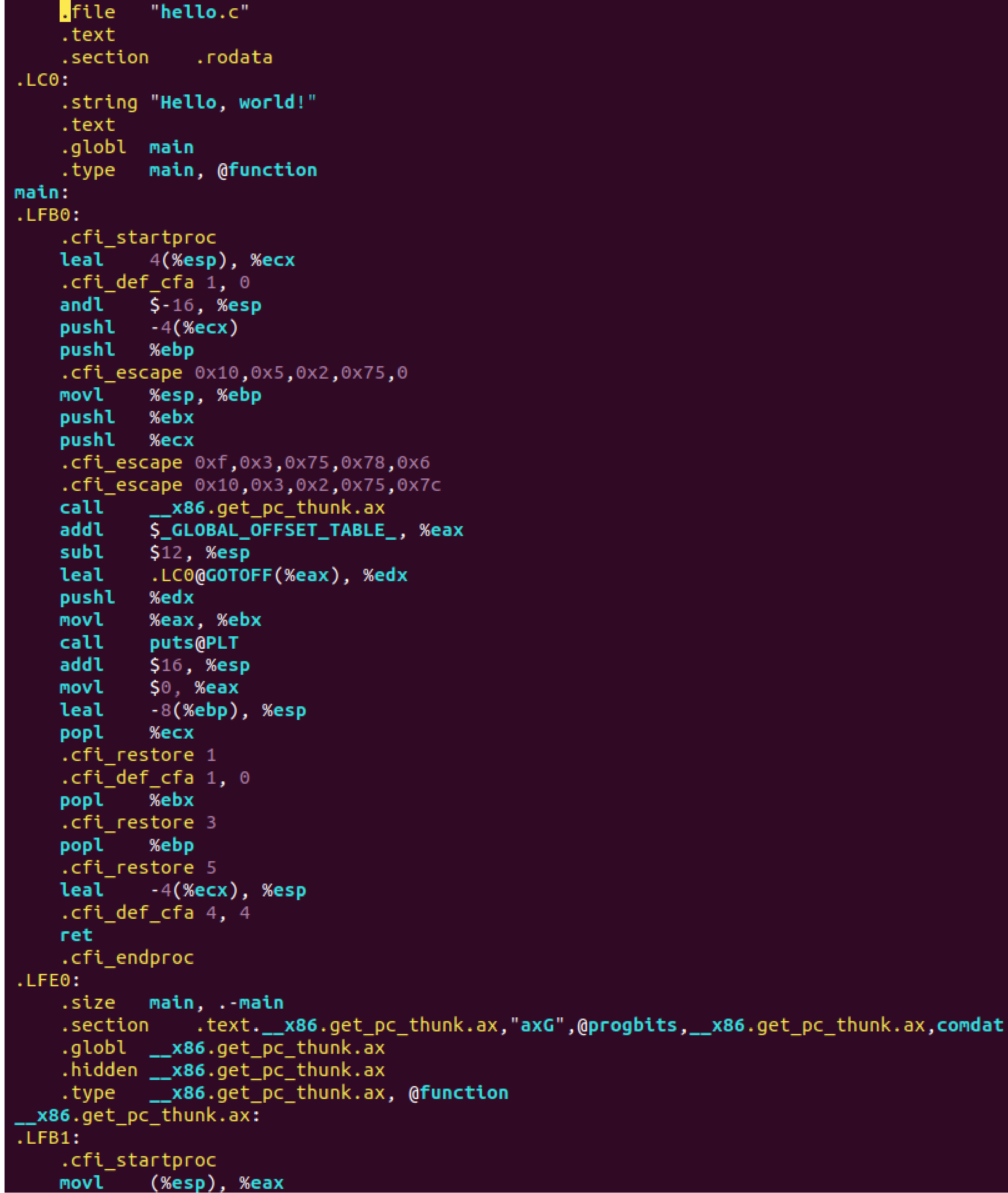


```
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 873 "/usr/include/stdio.h" 3 4

# 2 "hello.c" 2


# 3 "hello.c"
int main() {
    printf("Hello, world!\n");
    return 0;
}
```

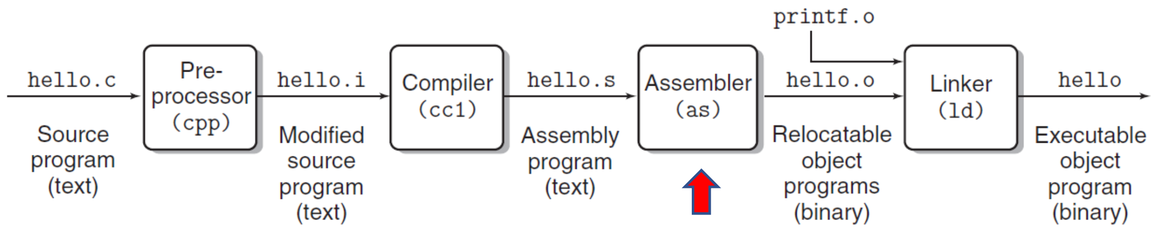


b. Compilation

- Modified C code to assembly code
- `gcc -S hello.c`

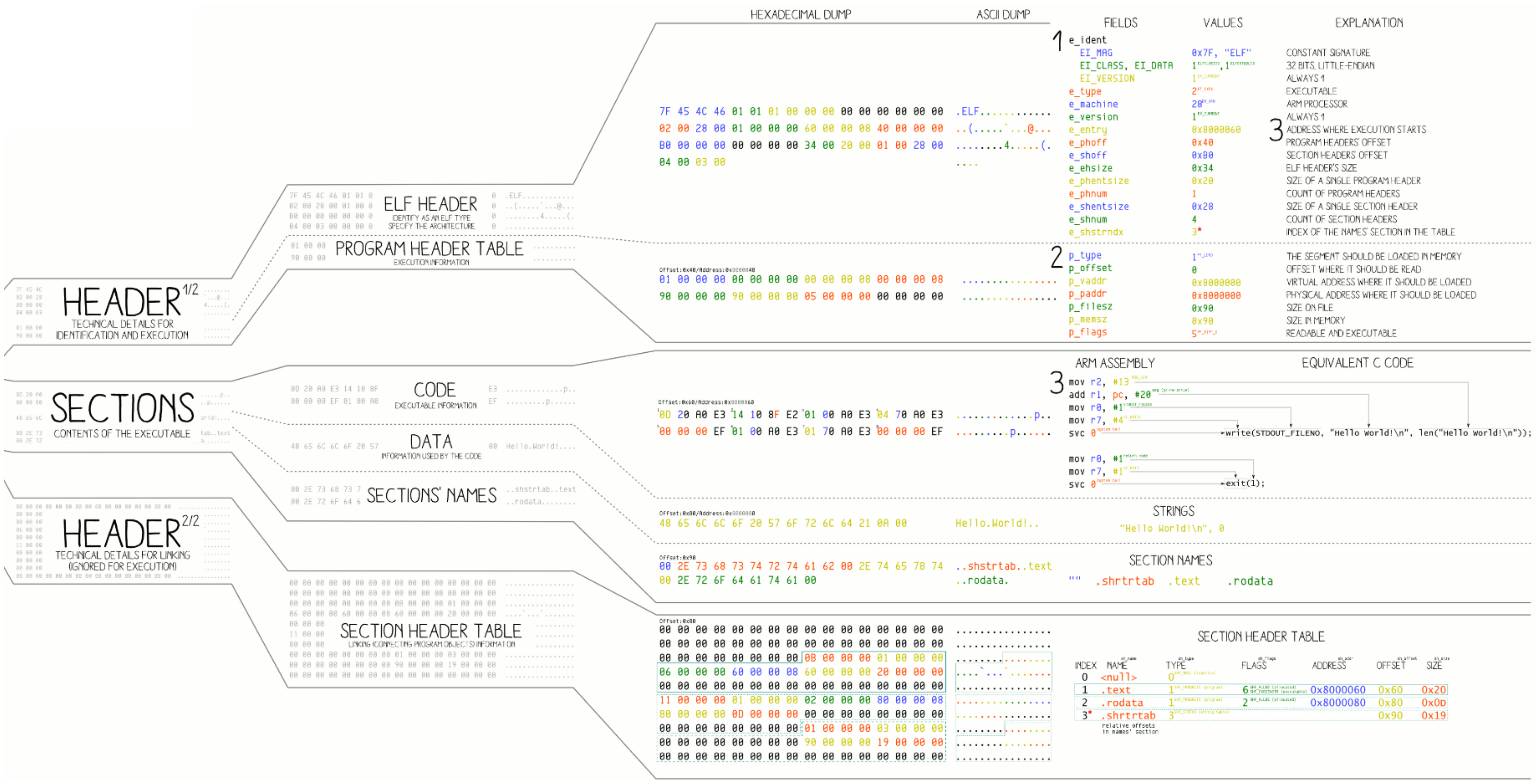


```
    .file   "hello.c"
    .text
    .section    .rodata
.LC0:
    .string "Hello, world!"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    leal    4(%esp), %ecx
    .cfi_def_cfa 1, 0
    andl    $-16, %esp
    pushl   -4(%ecx)
    pushl   %ebp
    .cfi_escape 0x10,0x5,0x2,0x75,0
    movl    %esp, %ebp
    pushl   %ebx
    pushl   %ecx
    .cfi_escape 0xf,0x3,0x75,0x78,0x6
    .cfi_escape 0x10,0x3,0x2,0x75,0x7c
    call    __x86.get_pc_thunk.ax
    addl    $_GLOBAL_OFFSET_TABLE_, %eax
    subl    $12, %esp
    leal    .LC0@GOTOFF(%eax), %edx
    pushl   %edx
    movl    %eax, %ebx
    call    puts@PLT
    addl    $16, %esp
    movl    $0, %eax
    leal    -8(%ebp), %esp
    popl    %ecx
    .cfi_restore 1
    .cfi_def_cfa 1, 0
    popl    %ebx
    .cfi_restore 3
    popl    %ebp
    .cfi_restore 5
    leal    -4(%ecx), %esp
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .section    .text.__x86.get_pc_thunk.ax,"axG",@progbits,__x86.get_pc_thunk.ax,comdat
    .globl  __x86.get_pc_thunk.ax
    .hidden __x86.get_pc_thunk.ax
    .type   __x86.get_pc_thunk.ax, @function
__x86.get_pc_thunk.ax:
.LFB1:
    .cfi_startproc
    movl    (%esp), %eax
```



c. Assembly

- Assembly code to relocatable object file (ELF format)
- `gcc -c hello.c`



```
7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00   .ELF............
02 00 28 00 01 00 00 00 60 00 00 08 40 00 00 00   ..(.....`...@...
B0 00 00 00 00 00 00 00 34 00 20 00 01 00 28 00   ........4. ...(.
04 00 03 00                                       ....
```

FIELDS	VALUES	EXPLANATION
1 e_ident		
EI_MAG	0x7F, "ELF"	CONSTANT SIGNATURE
EI_CLASS, EI_DATA	1, 1	32 BITS, LITTLE-ENDIAN
EI_VERSION	1	ALWAYS 1
e_type	2	EXECUTABLE
e_machine	28	ARM PROCESSOR
e_version	1	ALWAYS 1
e_entry	0x8000060	3 ADDRESS WHERE EXECUTION STARTS
e_phoff	0x40	PROGRAM HEADERS' OFFSET
e_shoff	0xB0	SECTION HEADERS' OFFSET
e_ehsize	0x34	ELF HEADER'S SIZE
e_phentsize	0x20	SIZE OF A SINGLE PROGRAM HEADER
e_phnum	1	COUNT OF PROGRAM HEADERS
e_shentsize	0x28	SIZE OF A SINGLE SECTION HEADER
e_shnum	4	COUNT OF SECTION HEADERS
e_shstrndx	3*	INDEX OF THE NAMES' SECTION IN THE TABLE

```
Offset:0x40/Address:0x8000040
01 00 00 00 00 00 00 00 00 00 00 08 00 00 00 08   ................
90 00 00 00 90 00 00 00 05 00 00 00 00 00 00 00   ................
```

FIELDS	VALUES	EXPLANATION
2 p_type	1	THE SEGMENT SHOULD BE LOADED IN MEMORY
p_offset	0	OFFSET WHERE IT SHOULD BE READ
p_vaddr	0x8000000	VIRTUAL ADDRESS WHERE IT SHOULD BE LOADED
p_paddr	0x8000000	PHYSICAL ADDRESS WHERE IT SHOULD BE LOADED
p_filesz	0x90	SIZE ON FILE
p_memsz	0x90	SIZE IN MEMORY
p_flags	5	READABLE AND EXECUTABLE

```
Offset:0x60/Address:0x8000060
0D 20 A0 E3 14 10 8F E2 01 00 A0 E3 04 70 A0 E3   .............p..
00 00 00 EF 01 00 A0 E3 01 70 A0 E3 00 00 00 EF   .........p......
```

ARM ASSEMBLY / EQUIVALENT C CODE

```
3 mov r2, #13
  add r1, pc, #20
  mov r0, #1
  mov r7, #4
  svc 0          write(STDOUT_FILENO, "Hello World!\n", len("Hello World!\n"));

  mov r0, #1
  mov r7, #1
  svc 0          exit(1);
```

```
Offset:0x80/Address:0x8000080
48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0A 00         Hello.World!..
```

STRINGS

"Hello World!\n", 0

```
Offset:0x90
00 2E 73 68 73 74 72 74 61 62 00 2E 74 65 78 74   ..shstrtab..text
00 2E 72 6F 64 61 74 61 00                        ..rodata.
```

SECTION NAMES

"" .shrtrtab .text .rodata

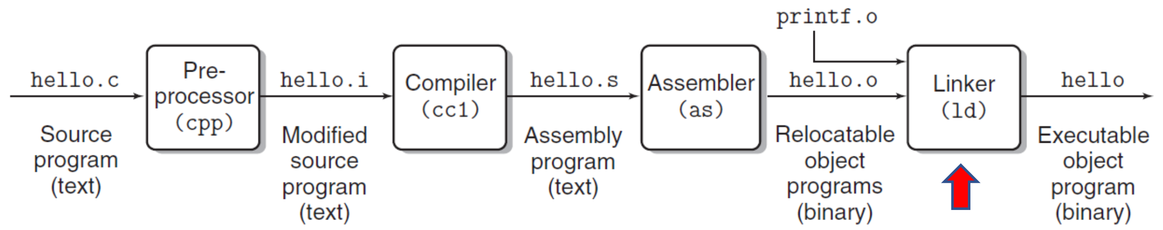
```
Offset:0xB0
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 0B 00 00 00 01 00 00 00   ................
06 00 00 00 60 00 00 08 60 00 00 00 20 00 00 00   ....`...`... ...
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
11 00 00 00 01 00 00 00 02 00 00 00 80 00 00 08   ................
80 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 01 00 00 00 03 00 00 00   ................
00 00 00 00 00 00 00 00 90 00 00 00 19 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

SECTION HEADER TABLE

INDEX	NAME	TYPE	FLAGS	ADDRESS	OFFSET	SIZE
0	<null>	0				
1	.text	1	6	0x8000060	0x60	0x20
2	.rodata	1	2	0x8000080	0x80	0x0D
3*	.shrtrtab	3			0x90	0x19

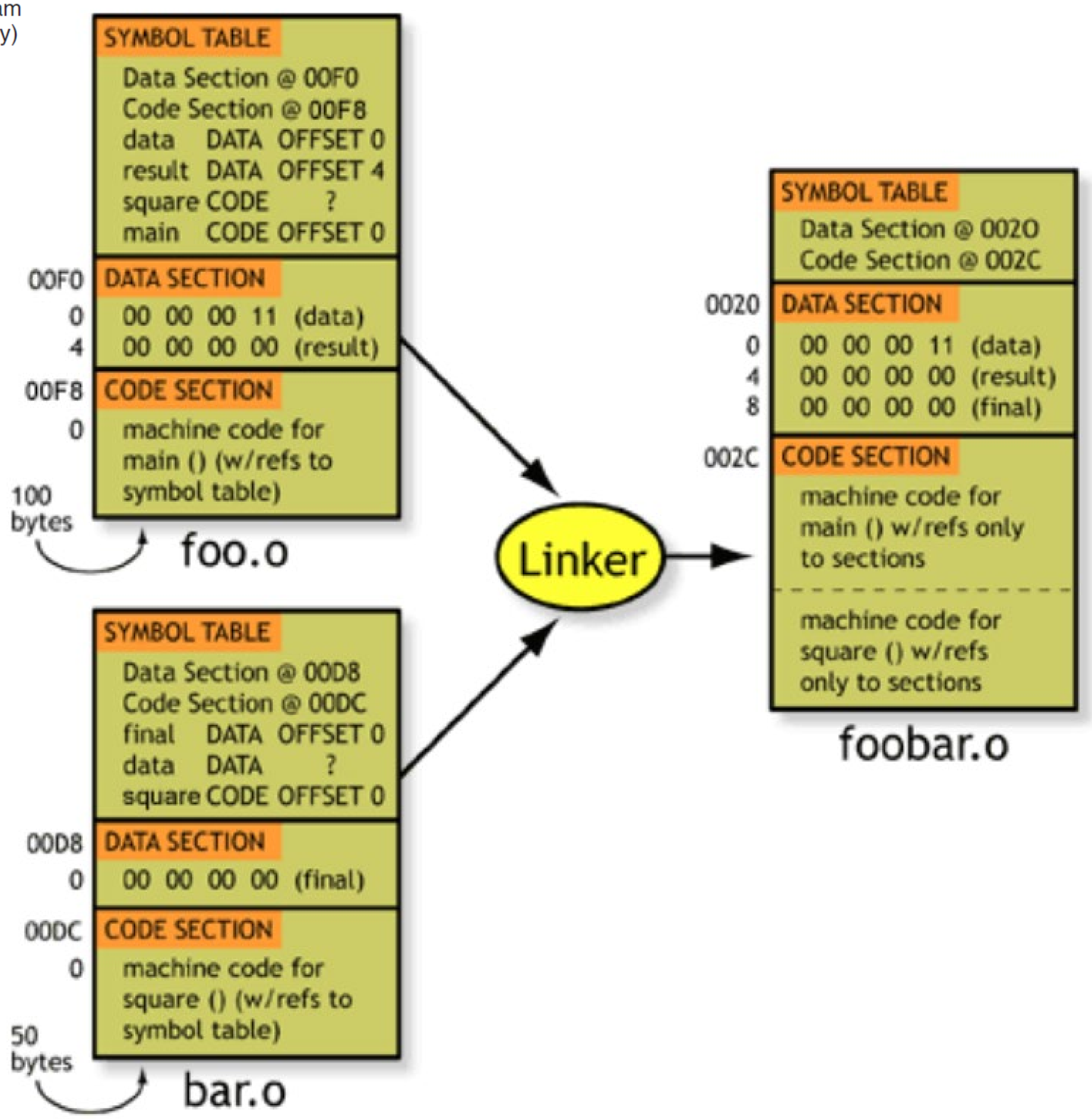
relative offsets in names' section

https://upload.wikimedia.org/wikipedia/commons/e/e4/ELF_Executable_and_Linkable_Format_diagram_by_Ange_Albertini.png



d. Linking

- Map multi relocatable object files to a single executable object file (also ELF format)
- `gcc hello.c -o hello`



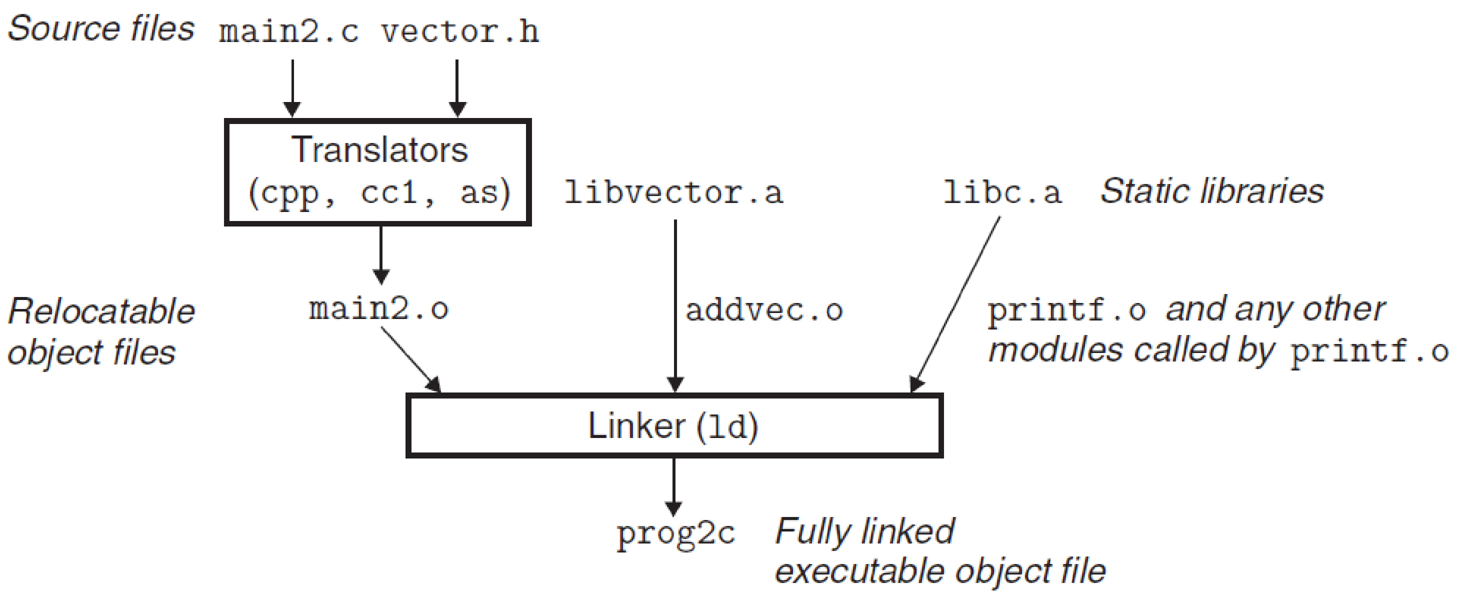
More on linking – libraries

- Motivation: How to use things like standard functions (e.g. printf)?

- [BAD] Get their *.o files and link them one by one (Difficult)
- [BAD] Get a single integrated *.o file (Too large)
- [GOOD] Pack them into a library and link the library to get necessary parts
- [GOOD] Don't pack code and data into the executable while linking but import when running

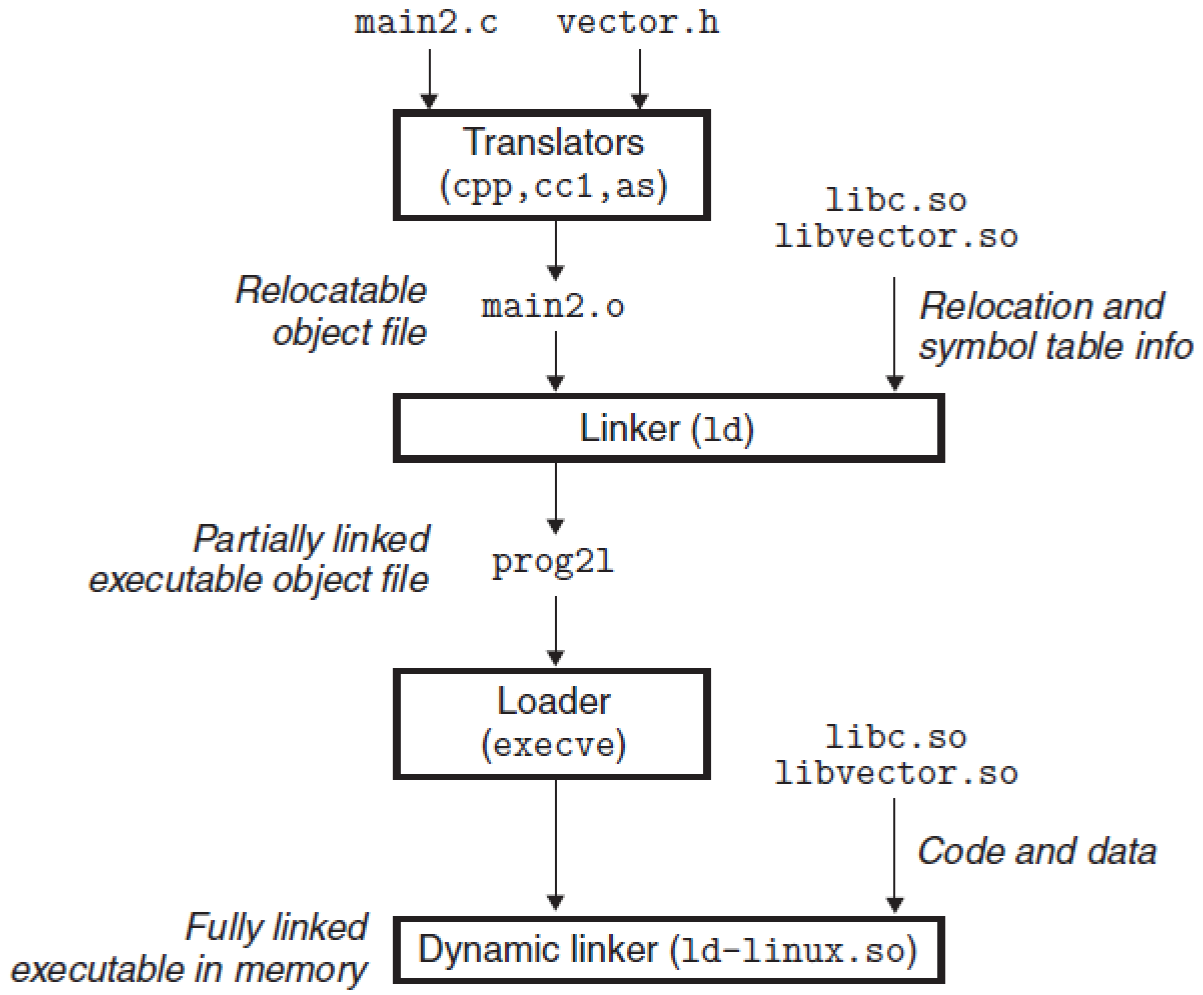
More on linking – libraries

- Static library (*.a files)
  - Link statically, add to the executable on demand
  - Generate library: `ar rcs libxxx.a yyy.o zzz.o`
  - Link library: `gcc -static -o main main.o ./libxxx.a`
    `(gcc -static -o main main.o -L. -lxxx)`



More on linking – libraries

- Dynamic library (*.so files)
  - Link at runtime
  - Generate library: `gcc -shared -fpic -o libxxx.so yyy.c zzz.c`
  - Link library: `gcc -o main main.c ./libxxx.so`

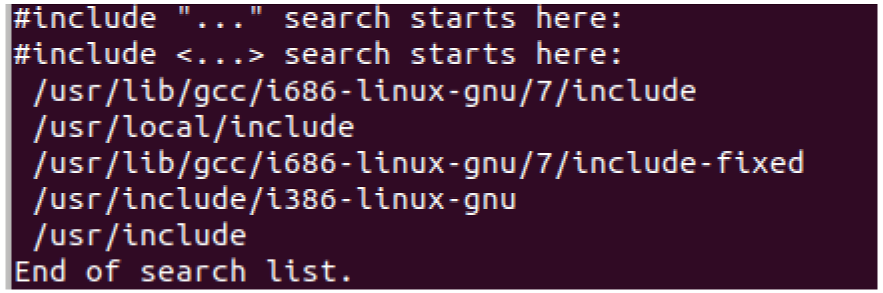


More on linking – libraries

- Static library
    - Faster
    - Large executable files
    - Need to relink against the updated version
    - Maybe multiple occurrence in memory
- Dynamic library
    - Small executable file
    - Easy upgrade
    - Only be loaded once in memory
    - Link before running → slower

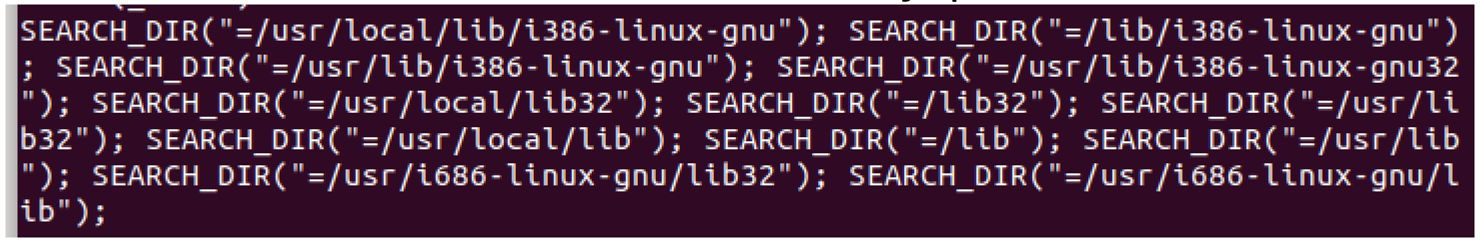
Useful flags & options

- Include path `-I`*dir*
  - **`cpp -v`** shows current include path



```
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc/i686-linux-gnu/7/include
 /usr/local/include
 /usr/lib/gcc/i686-linux-gnu/7/include-fixed
 /usr/include/i386-linux-gnu
 /usr/include
End of search list.
```

- Library path `-L`*dir* `-l`*libname*
  - **`ld --verbose`** shows default library path

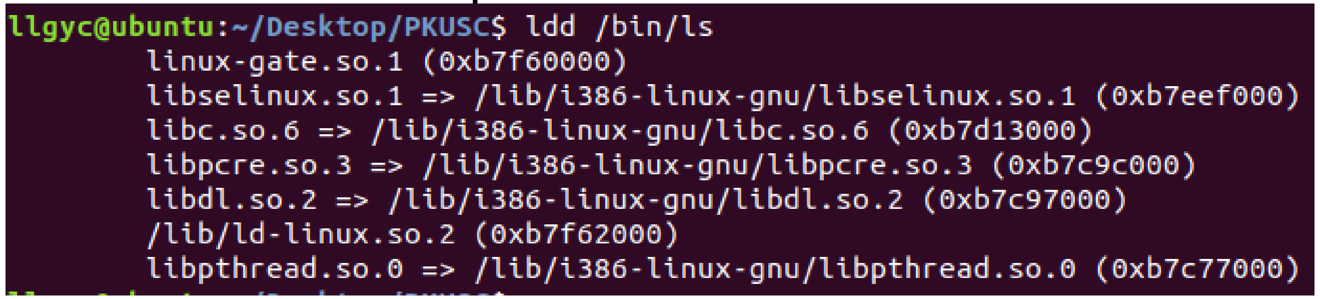


```
SEARCH_DIR("=/usr/local/lib/i386-linux-gnu"); SEARCH_DIR("=/lib/i386-linux-gnu")
; SEARCH_DIR("=/usr/lib/i386-linux-gnu"); SEARCH_DIR("=/usr/lib/i386-linux-gnu32
"); SEARCH_DIR("=/usr/local/lib32"); SEARCH_DIR("=/lib32"); SEARCH_DIR("=/usr/li
b32"); SEARCH_DIR("=/usr/local/lib"); SEARCH_DIR("=/lib"); SEARCH_DIR("=/usr/lib
"); SEARCH_DIR("=/usr/i686-linux-gnu/lib32"); SEARCH_DIR("=/usr/i686-linux-gnu/l
ib");
```

- Optimization *-Olevel -march=native*
  - **`gcc -O0/-O1/-O2/-O3/-Ofast/-Og`**

Dynamic linker related

- Where to find library at runtime?
- `LD_PRELOAD > LD_LIBRARY_PATH > /etc/ld.so.cache > /lib > /usr/lib`
  - `ldconfig` build `/etc/ld.so.cache` from `/etc/ld.so.conf`
  - More details on `man ld.so`
- `ldd`: show required libraries with their location



```
llgyc@ubuntu:~/Desktop/PKUSC$ ldd /bin/ls
        linux-gate.so.1 (0xb7f60000)
        libselinux.so.1 => /lib/i386-linux-gnu/libselinux.so.1 (0xb7eef000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7d13000)
        libpcre.so.3 => /lib/i386-linux-gnu/libpcre.so.3 (0xb7c9c000)
        libdl.so.2 => /lib/i386-linux-gnu/libdl.so.2 (0xb7c97000)
        /lib/ld-linux.so.2 (0xb7f62000)
        libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0 (0xb7c77000)
```

Useful binary utilities

- `readelf`: show ELF file
- `objdump`: support other binary formats, disassembly
- `nm`: show symbols
- `strings`: show text strings

- `gdb`: debugger

Makefile

- Motivation: How to compile 10000 files?

```
$ gcc –o prog file1.c file2.c … file10000.c
```

- Or

```
$ gcc –c –o file1.o file1.c
$ gcc –c –o file2.o file2.c
$ gcc –c –o file3.o file3.c
$ gcc –o prog file1.o file2.o file3.o
```

Makefile

- A list of *rules*

```
target files: source files
              action1
              action2
```

Makefile

- Example:

```
prog: file1.o file2.o file3.o
      gcc -o prog file1.o file2.o file3.o
file1.o: file1.c
      gcc -c -o file1.o file1.c
file2.o: file2.c lib2.h
      gcc -c -o file2.o file2.c
file3.o: file3.c
      gcc -c -o file3.o file3.c
```

Makefile

- Example:

```
prog: file1.o file2.o file3.o
      gcc -o $@ $^
%.o: %.c
     gcc -c -o $@ $^
```

$@	目标文件名
$%	档案文件(库) 的成员
$<	第一个依赖文件的文件名
$?	所有比目标文件新的倚赖文件名列表, 以空格分隔
$^	所有依赖文件名列表, 以空格分隔
$+	和$^ 类似, 包含重复文件名
$*	目标文件名去除后缀后的部分

Additional Readings

- CSAPP Chapter 7
- Wikipedia & Google
- http://make.mad-scientist.net/papers/advanced-auto-dependency-generation/