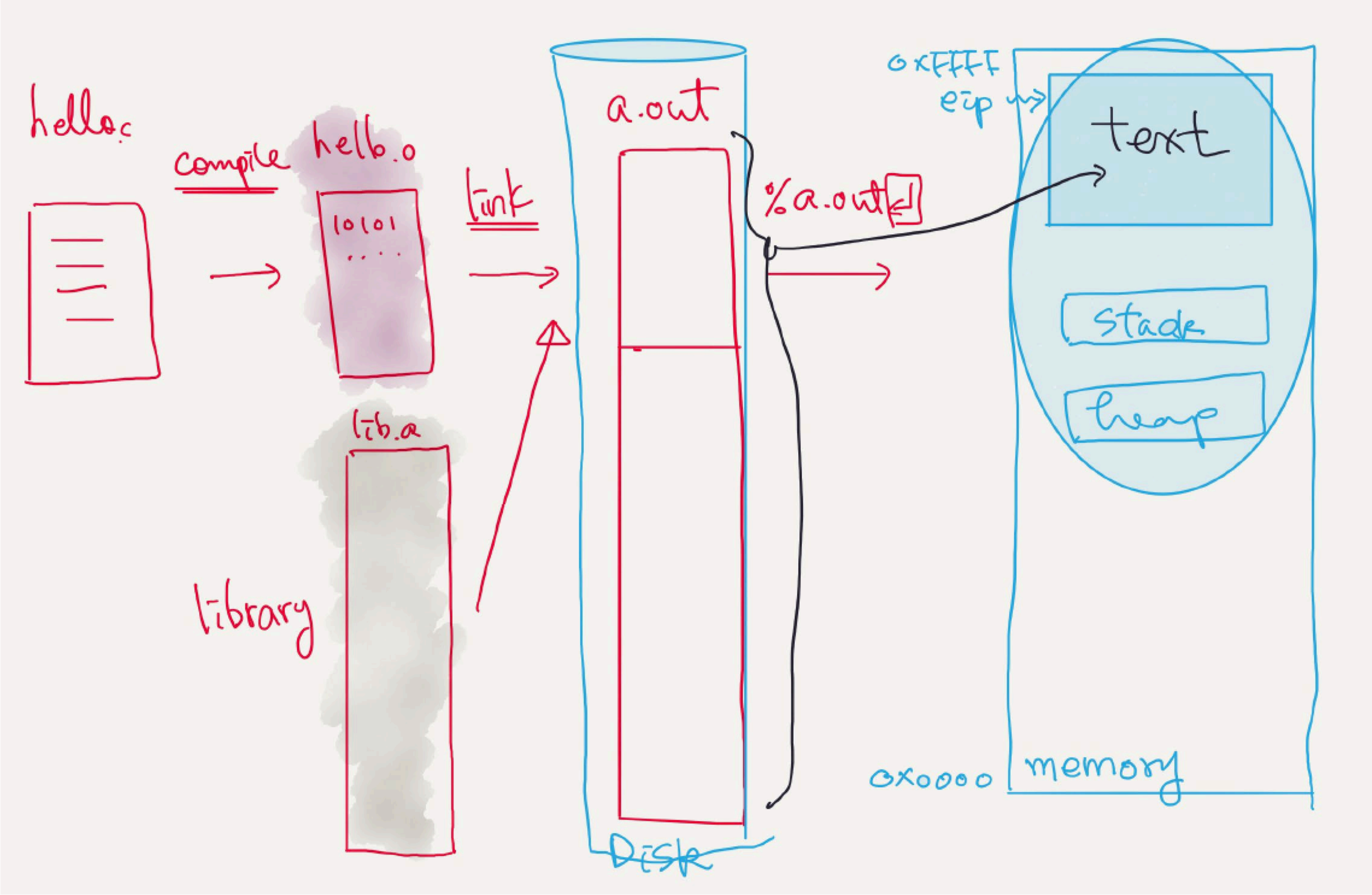
EE485: Introduction to Environment and Tools for Modern Software Development

Lecture 4: Compile

Youjip Won and Kyungsoo Park

KAIST EE

Life of a program



Building a C Program

- hello.c

```
#include <stdio.h>
int main(void)
{
    /* Write "hello, world\n" to stdout. */
    printf("hello, world\n");
    return 0;
}
```

- Compile and execute hello.c

```
ee209@ubuntu:~$ gcc209 hello.c -o hello
ee209@ubuntu:~$ ./hello
hello, world
```

`gcc209` is a script that executes
`gcc -Wall -Werror -ansi -pedantic -std=c99`

Preprocess C Code

```
gcc209 -E hello.c > hello.i
```

- Preprocessing
- Remove comments
- Processing Macros
- Substitute files in the #include

```
gcc209 -S hello.i
```

- assemble

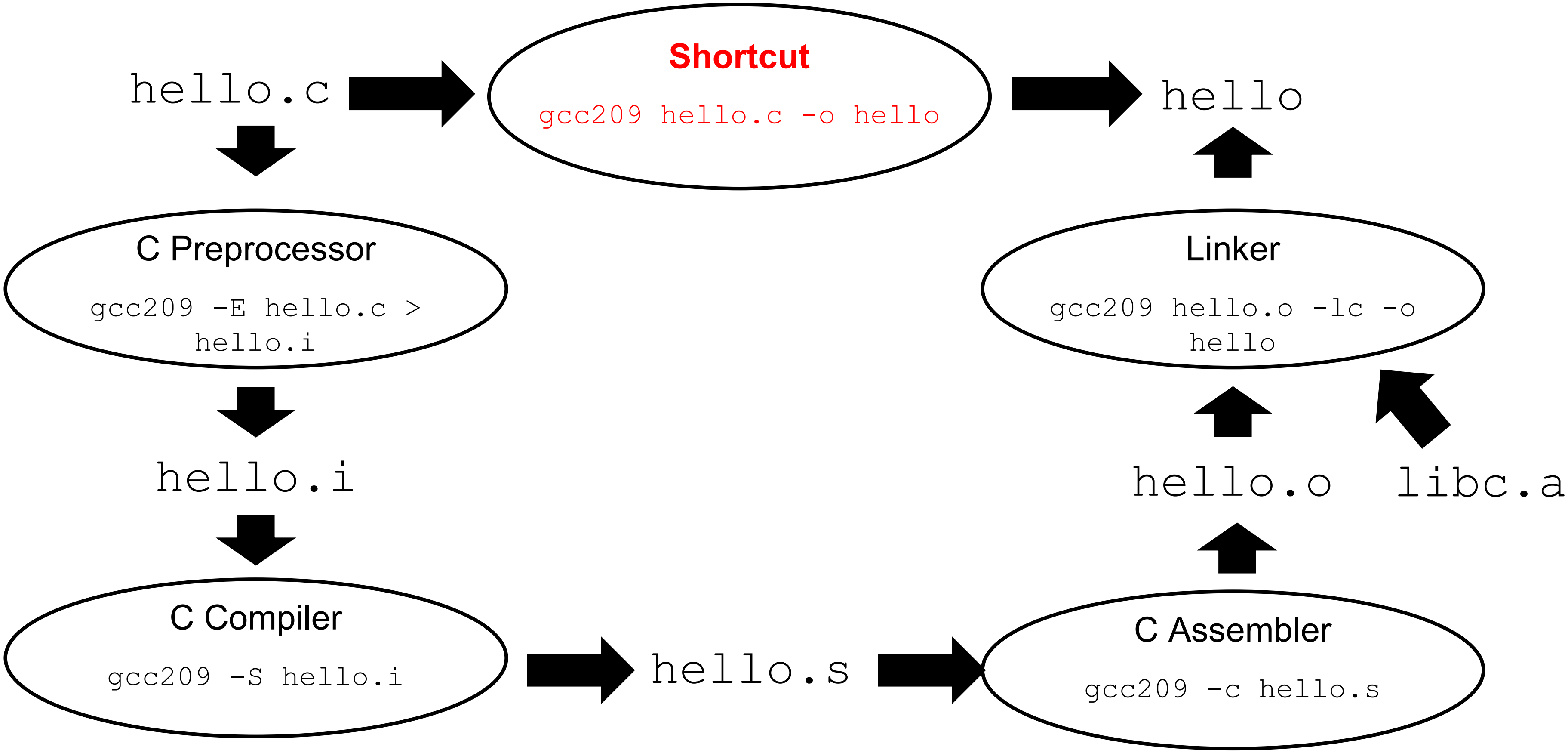
```
gcc209 -c hello.s
```

- compile

```
gcc209 hello.o -lc -o hello
```

- link

Shortcut of All Processes



hello.c

Shortcut

```
gcc209 hello.c -o hello
```

hello

C Preprocessor

```
gcc209 -E hello.c > hello.i
```

hello.i

C Compiler

```
gcc209 -S hello.i
```

hello.s

C Assembler

```
gcc209 -c hello.s
```

hello.o

libc.a

Linker

```
gcc209 hello.o -lc -o hello
```

Basics

```
% gcc -help
```

Will get all the options.

```
%gcc [compile options] [input files] [list of libraries]
-o [outputfile]
```

```
%gcc hello.c

%gcc hello.c -o hello
```

library

```
#include <stdio.h>

int main(void)
{
    /* Write "hello, world\n" to stdout. */
    printf("hello, world\n");
    return 0;
}
```

Who wrote `printf`?

Where is the definition of `printf()`?

- It is declared in <stdio.h>.
- It is defined in standard C library.
- The name of the standard C library is libc.a. (or glibc.a for gnu C library)

Using library

-llibrary option

- Searches the library of name liblibrary.a or liblibrary.so.
- Example: Using standard C library libc.a

```
%gcc hello.c -lc -o hello
```

- Example: Using math libarary libm.a

```
%gcc hello.c -lm -o hello
```

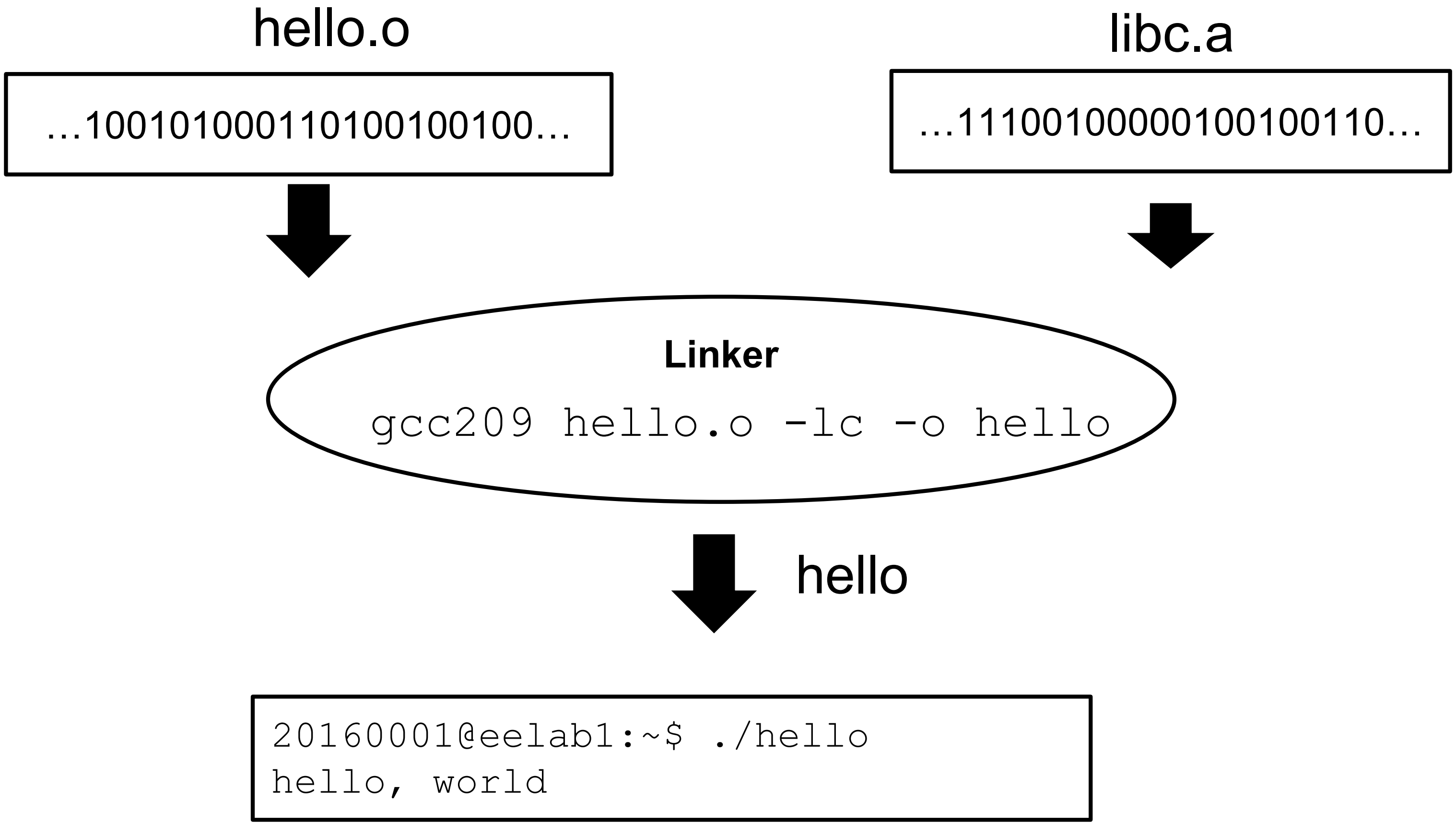
- Example: Using pthread libarary libpthread.a

```
%gcc hello.c -lpthread -o hello
```

- LD_LIBRARY_PATH: Directory to search for the library in Linux.

```
%echo $LD_LIBRARY_PATH
```

Generate Executable Binary



hello.o

…100101000110100100100…

libc.a

…11100100000100100110…

Linker

```
gcc209 hello.o -lc -o hello
```

hello

```
20160001@eelab1:~$ ./hello
hello, world
```

Optimization

`-On`

- `-O0`: reduce the cost of compilation and make debugging
- `-O1`: reduce code size and execution time
- `-O2`: performs nearly all supported optimizations that do not involve a spac e-speed tradeoff
- `-O3`: optimize more the O2
- `-Os`: optimize for size

```
gcc -O3 count.c -o count
```

Optimization

```c
/* count.c */
#include <stdio.h>

#define COUNT  10000000000

int main(void)
{
    for ( long int i = 0 ; i < COUNT ; ++i) ;
    return 0;
}
```

```
%gcc count.c -o count

%time ./count



%gcc -O1 count.c -o count

%time ./count
```

Directly assign the last value of iteration to `i`.

Optimization

```
/* count.c */
#include <stdio.h>

#define COUNT  10000000000

int main(void)
{
    for ( volatile long int i = 0 ; i < COUNT ; ++i) ;
    return 0;
}
```

```
%gcc count.c -o count

%time ./count
```

```
%gcc -O1 count.c -o count

%time ./count
```

Enforce that the `i` is read from memory in every iteration.

```
$ gcc -v -I/usr/local/include -DDEBUG -Wall -W -O2 -L/usr/local/l
ib -o hello hello.c -lm
```

-v : output the messages to the screen.

-o : output filename

-I : location of the header file

-D : Define the macro. It is the same as writing the #define statement.

-Wall : Warning all. Shows all warning

-W : shows all rest of the warnings that cannot be shown with –Wall option.

-O2 : optimization level

-lm : link `libm.a` (math library).

-L : location of the library files

-c : generate `*.o` file

library

- A set of functions
- Two types of library
  - Static library
    - Library is included in the binary program.
    - Advantage: no dependency in the libraries installed in the system.
    - Disadvantage: binary size becomes large. Same library can be loaded on to memory multiple times (wastage of memory)
  - Shared library
    - Library is linked when the program is executed
    - Advantage: small binary size
    - Disadvantage: dependency in the installed library. Relatively long execution time (to dynamically load the library in on-demand basis)

main.c

```
#include <stdio.h>
#include "swapper.h"

#define MAX_STR 20

int main(int argc, char *argv[])
{
        int a, b;
        char name[MAX_STR];

        printf("Pleas enter two numbers: ");
        scanf("%d %d", &a, &b);

        swapper_v1(&a,&b);

        printf("Swapping is completed. What's your name? ");
        scanf("%19s", name);

        printf("Ok, %s. Good job!\n", name);

        return 0;
}
```

swapper.h

```
/*
                              swapper.h
*/

void swapper_v1(int *a, int *b);
```

```
/*
                              swapper.c
*/

#include "swapper.h"

int buf[1024] = {1};

void swapper_v1(int *a, int *b)
{
        int local_a, local_b;

        local_a = *a;
        local_b = *b;

        *a = local_b;
        *b = local_a;
}
```

build static library

```
$ gcc -c swapper.c
$ ar -cr libswapper.a swapper.o
```

- Create the static library named libswapper.a with swapper.o

```
$ gcc -o simple main.c -L. -lswapper
```

- Create the binary with the static library
- `-L`: location of the library
- `-l`: name of library, `libswapper.a`

Build shared library

```
$ gcc -c swapper.c

$ gcc -shared -o libswapper.so swapper.o
```

- Build shared library `libswapper.so`

```
$ sudo ln -s [path to libswapper.so] /usr/lib/x86_64-linux/gnu/libswapper.so
```

- Install `libswapper.so` to the system.

```
$ gcc -o simple main.c -L. -lswapper
```

- Build binary with the shared library.

Build shared library (without sudo)

```
$ gcc -c swapper.c
$ gcc -shared -o libswapper.so swapper.o
```

- Build shared library `libswapper.so`

```
$ USER_LD_PATH=$(dirname $(realpath [path to libswapper.so]))
$ ex) USER_LD_PATH=$(dirname $(realpath ./libswapper.so))
$ export LD_LIBRARY_PATH="$USER_LD_PATH:$LD_LIBRARY_PATH“
```

- Add library path for ld.

```
$ echo $USER_LD_PATH
$ echo $LD_LIBRARY_PATH
```

- You can check the changes.

Build shared library (without sudo) Cont.

Please, replace the variable in command, If you use other OS.

LD_LIBRARY_PATH → ???

Windows	PATH
Linux	LD_LIBRARY_PATH
Mac OS X	DYLD_LIBRARY_PATH

```
$ gcc -o simple main.c -L. -lswapper
```

- Build binary with the shared library.

Homework

- Build the static library `libswapper.a` using the code provided in the page 16.
- Print the files that are in the libswapper.a library. Use `ar -t` command.
- Compile the `main.c` with `libswapper.a`.
- Print the size of the compiled binary.
- Run the program and provides the screen shot that shows the results of the program execution.

- Build the shared library libswapper.so using the code provided in the page 16.
- Compile the `main.c` with `libswapper.so`.
- Print the size of the compiled binary.
- Run the program and provide the screen shot that shows the result of the program execution.

- Upload the screen shot of the result to KLMS!!