ITP20004 – Open-Source Software Labs

Project Management

Charmgil Hong

charmgil@handong.edu

Spring, 2023

Handong Global University

Announcements

- Weekly schedule

Week	Mon		Week	Thur		
1	Course overview, motivation, administrivia		1	CPR: C Programming Reinforcement - Functions		
2	Computer organization and Linux environment (1)		2	CPR: C Programming Reinforcement - Strings		
3	Computer organization and Linux environment (2)		3	CPR: C Programming Reinforcement - User-defined types, and memory allocation		
4	Basic Linux commands + Writing code on Linux (vim)		4	Getting started with Linux / Hands-on Linux command-line tools		
5	More Linux commands		5	CPR: C Programming Reinforcement - Understanding compilation and build process		
6	Project management (1)	**Proj 1 출제**	6	Project management (2)		
7	-		7	Project: BASIC interpreter (2 periods)		Project 1
8	Midterm exam		8	Proj 1 due		
9	CPR: C Programming Reinforcement - Accessing files and		9	Debugging with GDB + Unit testing with gtest		
10	Code review GNU utilities		12	Writing an application in C		
11	Computer network basics		10	Linux network commands	**AWS 가입 - lightsail**	
12	Linux machine as a server + Web services		11	Service launching	**lab problem + AWS 가입해지**	
13	Project: Text-based Game		13	Github and open-source community		Project 2
14	Using Github		14	Socket programming		
15	Project: Multi-user game		15	Project: Multi-user game		Project 3
16	Final exam		16			

Announcements

- No tutor session for this week
- Due for the 3rd lab outcomes: 4pm, April 6 (submit to LMS)

Voogle

Announcements

- There will be new teams assigned on April 6
  - There will be a peer evaluation at the end of the first cycle

Announcements

- For each lab
  - Before a lab, every student submits a pre-lab report (worksheet-type assignment) – **individual work**
  - After a lab, each team sees and reports to the TA with the results – **team work**

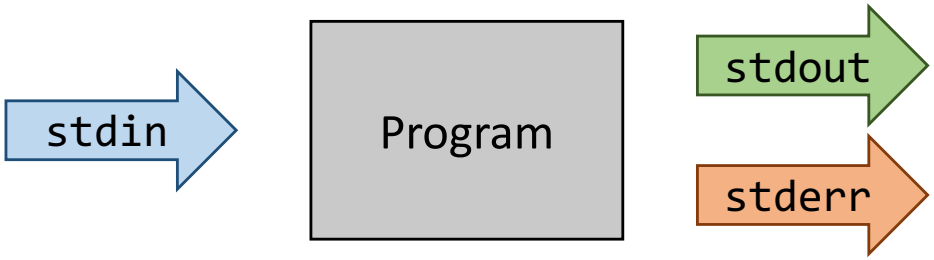
- Be prepared for pre-lab assignment #4
  - Read and fill-in-the-blank stuff
  - Reading: Ch. 23 of TLCL ("Compiling Programs")

Agenda

- Regular expression in *vim*

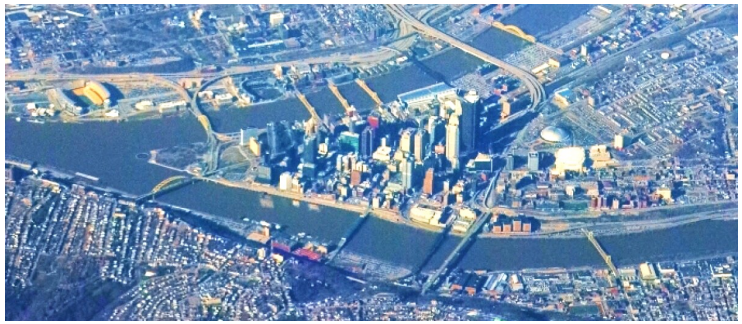
Streams

- In Linux (POSIX OS's), programs have the `stdin`, `stdout`, and `stderr` streams attached to them by default



Streams

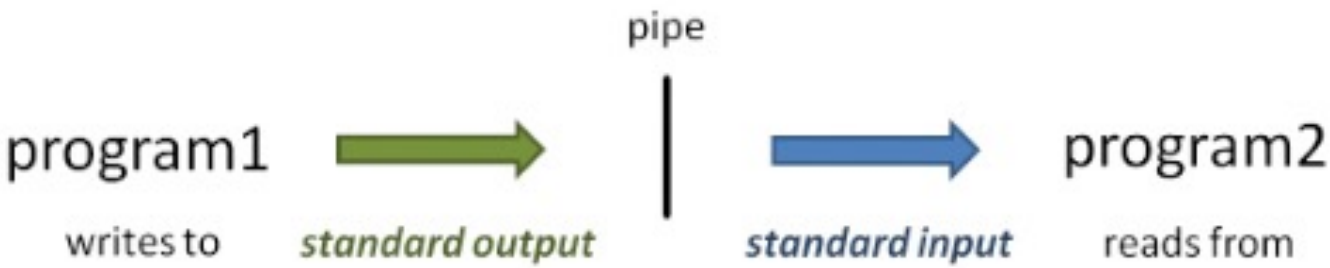
- Linux streams are like streams of water
  - One can redirect streams
  - One can pipe to carry water from one place to another



* Image src: https://wikis.utexas.edu/display/CoreNGSTools/Linux+fundamentals;
https://www.wesa.fm/environment-energy/2013-02-04/pollutants-continue-to-hamper-wildlife-fishing-recreation-in-pittsburghs-three-rivers

Streams & Pipes

- Let us feed `cowsay` through the `stdin` stream using a pipe (`|`)



* Image src: https://wikis.utexas.edu/display/CoreNGSTools/Linux+fundamentals

Streams

- Example: `fortune` and `cowsay`
  - `fortune` tells you some pieces of wisdom

```
$ fortune
Don't worry.  Life's too long.
-- Vincent Sardi, Jr.
```

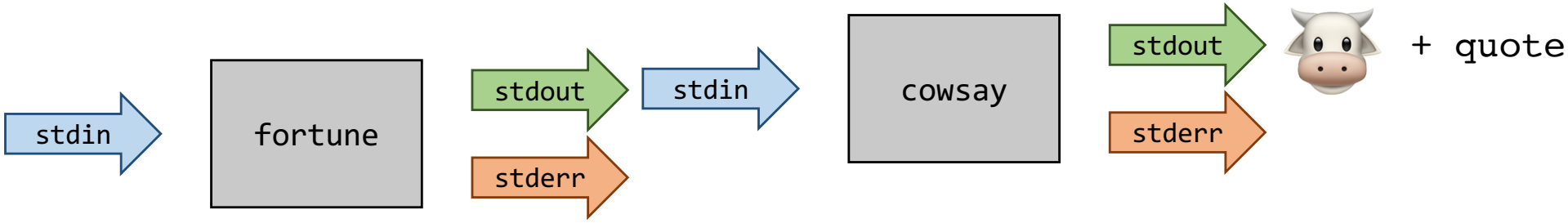
  - `cowsay` takes a string and shows a cow saying it

```
$ cowsay "OSSL rules!”

 _____________
< OSSL rules! >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Streams & Pipes

- Let us feed `cowsay` through the `stdin` stream using a pipe (`|`)

```
$ fortune | cowsay
 _____________________________________________
/ Ships are safe in harbor, but they were \
\ never meant to stay there.              /
 ---------------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```



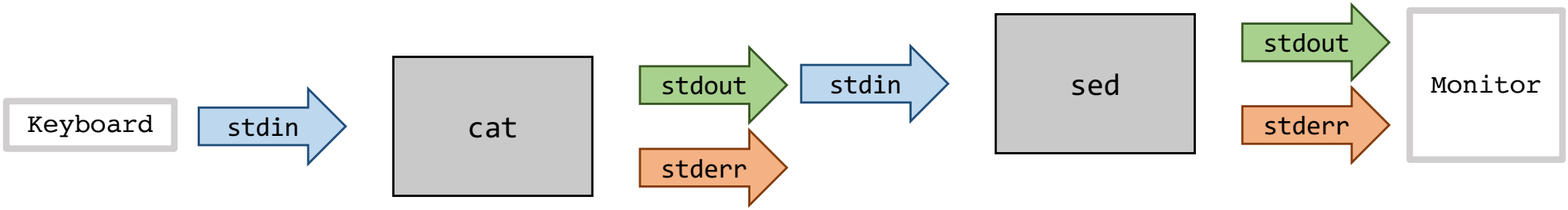
Streams & Pipes

- cat, sed, and a pipe (|)

```
$ cat
Everything I write is repeated by cat
Everything I write is repeated by cat
```

- sed - stream editor for filtering and transforming text

```
$ cat | sed -E "s/write/type/"
Everything I write is repeated by cat
Everything I type is repeated by cat
```

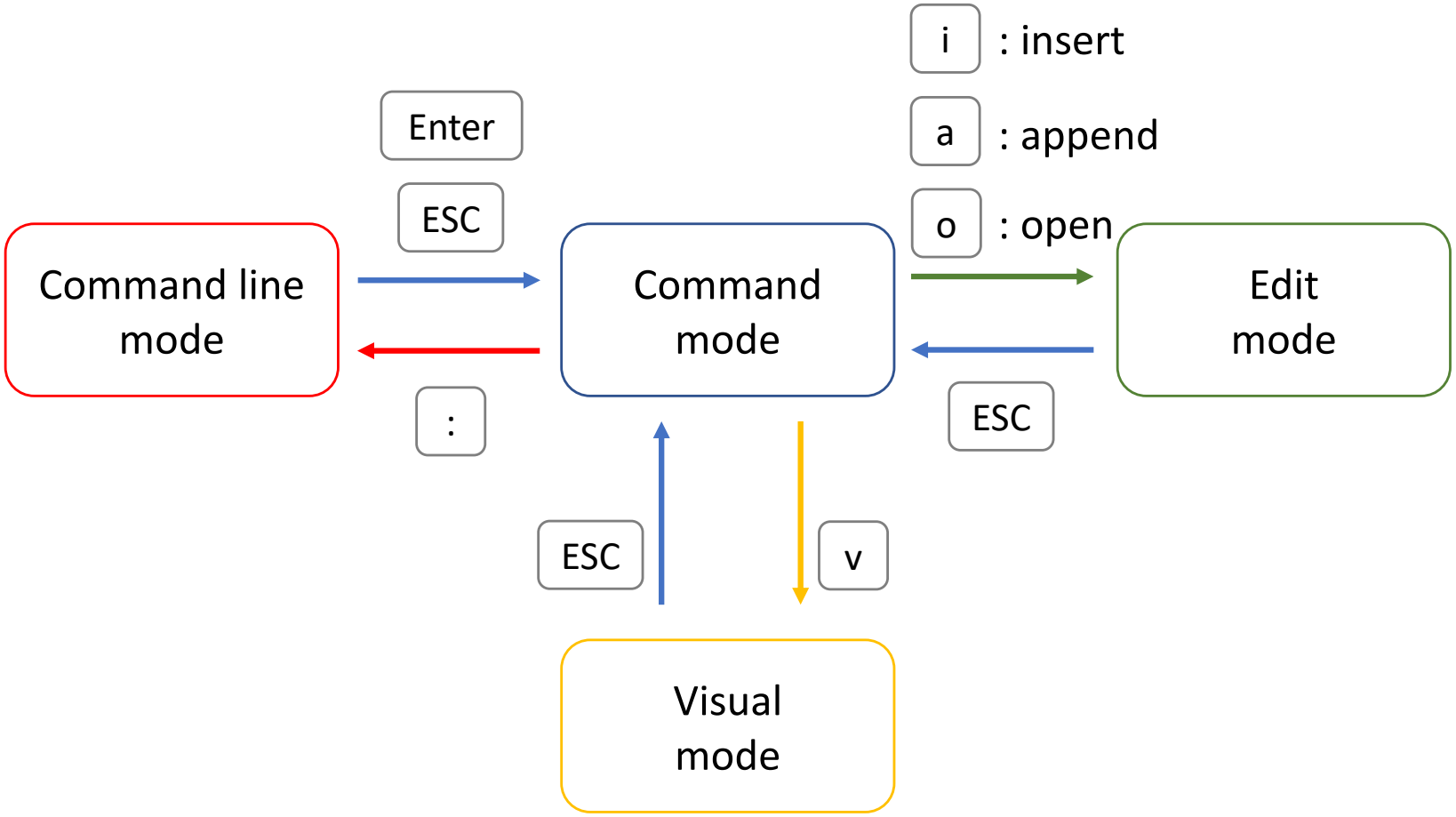


Redirection

- Redirecting output streams
  - To take the standard output of a program and save it to a file, you use the **>** operator
    - A single **>** overwrites any existing target; a double **>>** appends to it
    - Since standard output is stream #1, this is the same as **1>**
  - To redirect the standard error of a program you must specify its stream number using **2>**
  - To redirect standard output and standard error to the same place, use the syntax **2>&1**

vim – Modal Editor

- Switching between modes



Search

- Search in *vim*
  - Hit the **Esc** and **/** one after the other
  - Then enter the word you want to search for
  - Hit the **Enter** key will take you to the occurrence of the search word after the cursor
  - To move between occurrences, you can press the **n** key and to move backwards you press the **N** key

  - The **?** command is very similar to the **/** command, but it searches the whole file backwards
    - In this case, the **n** key searches backwards, and the **N** key searches forwards

* Source: https://www.linuxfordevices.com/tutorials/linux/vim-search-and-replace

Search for the Word at the Cursor

- Search for the word under the cursor
  - Press the ***** key in normal mode, the cursor will be placed to the nearest occurrence of the word under the cursor
    - Press the ***** key again to search for the next occurrence
    - Use the **n** and the **N** key to cycle through the search results forward or backwards
  - The **#** is the same as ***** but it searches backwards
- You can search ignoring the case by issuing the command
  - `:set ic`

* Source: https://www.linuxfordevices.com/tutorials/linux/vim-search-and-replace

Agenda

- Regular expression in *vim*
- Compiling programs

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - For each line in ***the range*** replace a match of ***the pattern*** with ***the string*** where:

c	Confirm each substitution
g	Replace all occurrences in the line (without **g** - only first)
i	Ignore case for the pattern
I	Don't ignore case for the pattern

  - Examples
    - `:s/old_word/new_word`
    - `:%s/old_word/new_word`
    - `:%s/old_word/new_word/g`

* Source: http://vimregex.com/

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - Some Vim commands can accept a line range in front of them
  - By specifying the line range, you restrict the command execution to this particular part of text only

Specifier	Description
number	an absolute line number
.	the current line
$	the last line in the file
%	the whole file. The same as **1,$**
't	position of mark "t"

  - `:%10,20s/old_word/new_word`

* Source: http://vimregex.com/

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - Anchors
    - `pattern`
    - `\<pattern\>`
    - `^pattern`
    - `pattern$`
    - `^pattern$`

* Source: http://vimregex.com/

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - "Escaped" characters (metacharacters)

#	Matching	#	Matching
.	any character except new line		
\s	whitespace character	**\S**	non-whitespace character
\d	digit	**\D**	non-digit
\x	hex digit	**\X**	non-hex digit
\o	octal digit	**\O**	non-octal digit
\h	head of word character (a,b,c...z,A,B,C...Z and _)	**\H**	non-head of word character
\p	printable character	**\P**	like **\p**, but excluding digits
\w	word character	**\W**	non-word character
\a	alphabetic character	**\A**	non-alphabetic character
\l	lowercase character	**\L**	non-lowercase character
\u	uppercase character	**\U**	non-uppercase character

* Source: http://vimregex.com/

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - *E.g.*, “Escaped” characters (metacharacters)
    - To match a date like 09/01/2000,
      `\d\d/\d\d/\d\d\d\d`
    - To match 6 letter word starting with a capital letter:
      `\u\w\w\w\w\w`

* Source: http://vimregex.com/

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - Quantifiers

Quantifier	Description
*	matches 0 or more of the preceding characters, ranges or metacharacters .* matches everything including empty line
\+	matches 1 or more of the preceding characters...
\=	matches 0 or 1 more of the preceding characters...
\{n,m}	matches from n to m of the preceding characters...
\{n}	matches exactly n times of the preceding characters...
\{,m}	matches at most m (from 0 to m) of the preceding characters...
\{n,}	matches at least n of of the preceding characters...
	where **n** and **m** are positive integers (>0)

* Source: http://vimregex.com/

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - *E.g.*, with Quantifiers
    - To match a date like 09/01/2000,
      `\d\{2}/\d\{2}/\d\{4}`
    - To match 6 letter word starting with a capital letter:
      `\u\w\{5}`
    - To match a word starting with a capital letter and in any length longer than 2
      `\u\w\+`

* Source: http://vimregex.com/

Agenda

- Regular expression in *vim*
- Compiling programs

Compiling and Linking in C

- **Coding**: Write programs (source code)
- **Preprocessing**: Modify the source code according to the preprocessor directives (#include, #define, …)
- **Compilation**: Translate the program into a low-level assembly code
- **Assembly**: Translate the assembly code into machine instructions; pack them in a form of a relocatable object
- **Linking**: Merge the relocatable object files and create an executable

A C Program

- Fibonacci in C

```
#include <stdio.h>

int main(void) {
    int x, y, z;

    while (1) {
        x = 0;
        y = 1;
        do {
            printf("$d\n", x);

            z = x + y;
            x = y;
            y = z;
        } while (x < 255);
    }
}
```

```
% gcc fib.c -o fib
% ./fib
0
1
1
2
3
5
8
13
21
34
55
89
144
233
0
1
1
2
3
5
8
13
21
34
55
89
144
233
```

C → ASM

- C to assembly

```c
#include <stdio.h>

int main(void) {
    int x, y, z;

    while (1) {
        x = 0;
        y = 1;
        do {
            printf("$d\n", x);

            z = x + y;
            x = y;
            y = z;
        } while (x < 255);
    }
}
```

```
% gcc fib.c -o fib
% otool -tv fib
Fib:
(__TEXT,__text) section
_main:
0000000100000f20 pushq   %rbp
0000000100000f21 movq    %rsp, %rbp
0000000100000f24 subq    $0x20, %rsp
0000000100000f28 movl    $0x0, -0x4(%rbp)
0000000100000f2f movl    $0x0, -0x8(%rbp)
0000000100000f36 movl    $0x1, -0xc(%rbp)
0000000100000f3d leaq    0x56(%rip), %rdi
0000000100000f44 movl    -0x8(%rbp), %esi
0000000100000f47 movb    $0x0, %al
0000000100000f49 callq   0x100000f78
0000000100000f4e movl    -0x8(%rbp), %esi
0000000100000f51 addl    -0xc(%rbp), %esi
0000000100000f54 movl    %esi, -0x10(%rbp)
0000000100000f57 movl    -0xc(%rbp), %esi
0000000100000f5a movl    %esi, -0x8(%rbp)
0000000100000f5d movl    -0x10(%rbp), %esi
0000000100000f60 movl    %esi, -0xc(%rbp)
0000000100000f63 movl    %eax, -0x14(%rbp)
0000000100000f66 cmpl    $0xff, -0x8 (%rbp)
0000000100000f6d jl      0x100000f3d
0000000100000f73 jmp     0x100000f2f
```

C → ASM

- C to assembly

```c
#include <stdio.h>

int main(void) {
    int x, y, z;

    while (1) {
        x = 0;
        y = 1;
        do {
            printf("$d\n", x);

            z = x + y;
            x = y;
            y = z;
        } while (x < 255);
    }
}
```

```
% gcc fib.c -o fib
% otool -tv fib
Fib:
(__TEXT,__text) section
_main:
0000000100000f20 pushq   %rbp
0000000100000f21 movq    %rsp, %rbp
0000000100000f24 subq    $0x20, %rsp
0000000100000f28 movl    $0x0, -0x4(%rbp)
0000000100000f2f movl    $0x0, -0x8(%rbp)
0000000100000f36 movl    $0x1, -0xc(%rbp)
0000000100000f3d leaq    0x56(%rip), %rdi
0000000100000f44 movl    -0x8(%rbp), %esi
0000000100000f47 movb    $0x0, %al
0000000100000f49 callq   0x100000f78
0000000100000f4e movl    -0x8(%rbp), %esi
0000000100000f51 addl    -0xc(%rbp), %esi
0000000100000f54 movl    %esi, -0x10(%rbp)
0000000100000f57 movl    -0xc(%rbp), %esi
0000000100000f5a movl    %esi, -0x8(%rbp)
0000000100000f5d movl    -0x10(%rbp), %esi
0000000100000f60 movl    %esi, -0xc(%rbp)
0000000100000f63 movl    %eax, -0x14(%rbp)
0000000100000f66 cmpl    $0xff, -0x8 (%rbp)
0000000100000f6d jl      0x100000f3d
0000000100000f73 jmp     0x100000f2f
```

ASM → Machine Code

- Asm to machine code

```
0x0: ldi    0x1
0x1: sta    [0xe]
0x2: ldi    0x0
0x3: out
0x4: add    [0xe]
0x5: sta    [0xf]
0x6: lda    [0xe]
0x7: sta    [0xd]
0x8: lda    [0xf]
0x9: sta    [0xe]
0xa: lda    [0xd]
0xb: jc     0x0
0xc: jmp    0x3
0xd:
0xe:
0xf:
```

```
0000:  0111 0001
0001:  0100 1110
0010:  0111 0000
0011:  0101 0000
0100:  0010 1110
0101:  0100 1111
0110:  0001 1110
0111:  0100 1101
1000:  0001 1111
1001:  0100 1110
1010:  0001 1101
1011:  1000 0000
1100:  0110 0011
```

ASM → Machine Code

- Asm to machine code

```
0x0: ldi    0x1
0x1: sta    [0xe]
0x2: ldi    0x0
0x3: out
0x4: add    [0xe]
0x5: sta    [0xf]
0x6: lda    [0xe]
0x7: sta    [0xd]
0x8: lda    [0xf]
0x9: sta    [0xe]
0xa: lda    [0xd]
0xb: jc     0x0
0xc: jmp    0x3
0xd:
0xe:
0xf:
```

```
0000:  0111 0001
0001:  0100 1110
0010:  0111 0000
0011:  0101 0000
0100:  0010 1110
0101:  0100 1111
0110:  0001 1110
0111:  0100 1101
1000:  0001 1111
1001:  0100 1110
1010:  0001 1101
1011:  1000 0000
1100:  0110 0011
```

References

- Ben Eater. Comparing C to machine language. URL: https://www.youtube.com/watch?v=yOyaJXpAYZQ

Working with Multiple Source Files

- A small program → a single file
- A not-so-small program
  - 10,000+ lines of code
  - Multiple components
  - Multiple collaborators

  - Issues
    - Long files are harder to manage
    - Every change requires a long compilation process
    - A source file cannot be opened and modified simultaneously

Example: Working with Multiple Source Files

- One header file (.h) – mylib.h
- Two source files (.c) – main.c, mylib.c

main.c

```c
#include <stdio.h>
#include "mylib.h"

int main(void){

  int a = 3, b = 5;
  printf("(initial) a=%d, b=%d\n", a, b);

  swap(&a, &b);
  printf("(swapped) a=%d, b=%d\n", a, b);

  return 0;
}
```

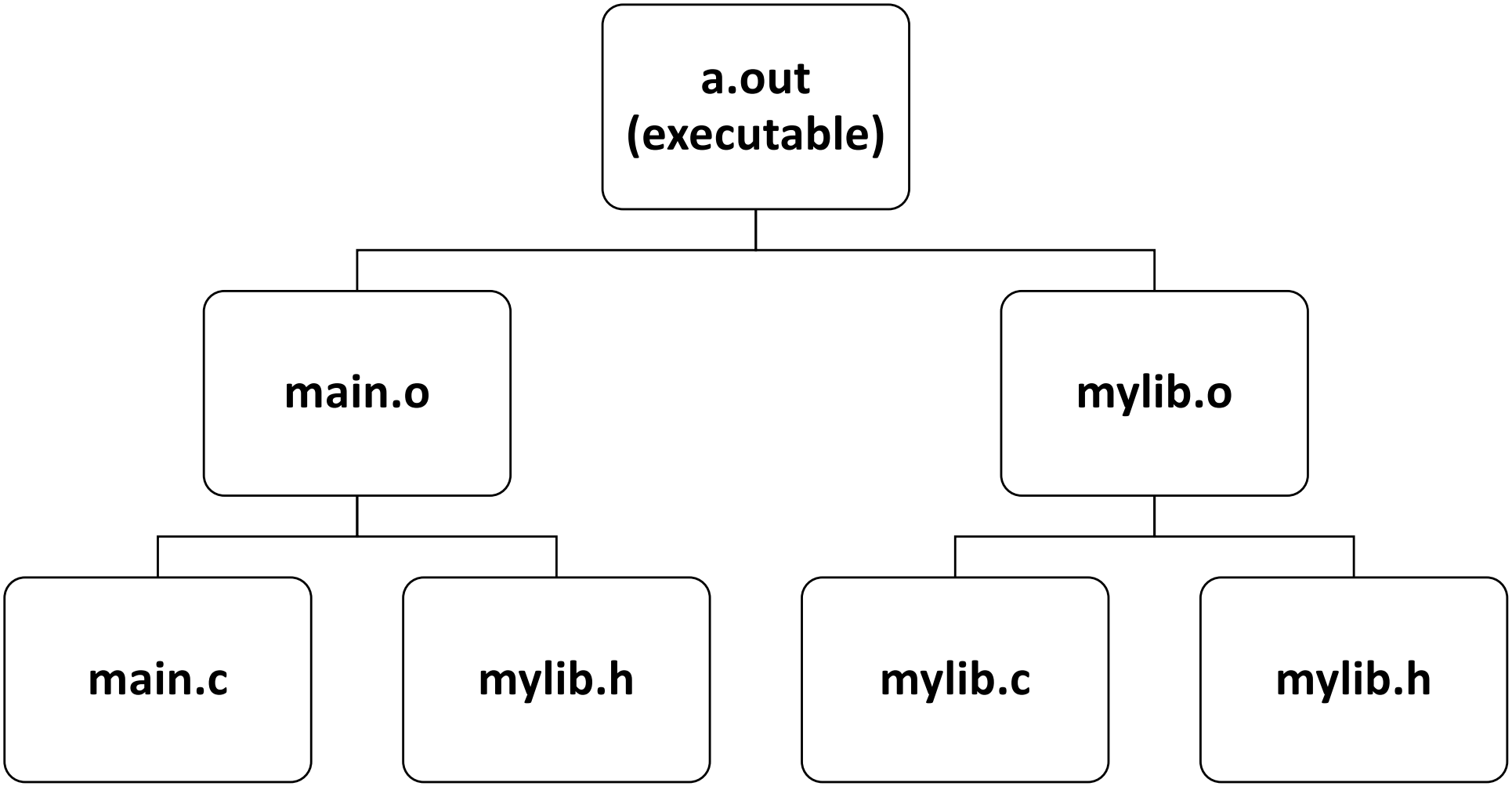
mylib.h

```c
#ifndef  _MYLIB_H_
#define  _MYLIB_H_

void swap(int*, int*);

#endif
```

mylib.c

```c
#include "mylib.h"

void swap(int* a, int* b){
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

Example: Working with Multiple Source Files



Example: Working with Multiple Source Files

- How to compile?

```
Charmgils-MacBook-Pro:OSS charmgil$ gcc -c -o mylib.o mylib.c
Charmgils-MacBook-Pro:OSS charmgil$ gcc -c -o main.o main.c
Charmgils-MacBook-Pro:OSS charmgil$ gcc mylib.o main.o
Charmgils-MacBook-Pro:OSS charmgil$
Charmgils-MacBook-Pro:OSS charmgil$ ./a.out
(initial) a=3, b=5
(swapped) a=5, b=3
```

Example: Working with Multiple Source Files

- Suppose you wanted to modify a line in the main() function
  - Do you want to retype and recompile everything?

main.c

```c
#include <stdio.h>
#include "mylib.h"

int main(void){

  int a = 3, b = 5;
  printf("(initial) a=%d, b=%d\n", a, b);

  swap(&a, &b);
  printf("(swapped) a=%d, b=%d\n", a, b);

  return 0;
}
```

8

```
$ gcc -c -o mylib.o mylib.c
$ gcc -c -o main.o main.c
$ gcc mylib.o main.o
$
$ ./a.out
```

GNU Make

- A build automation tool by GNU

- *makefile*: a text file governing the Make tool
  - Tells how to compile and link a program
  - Specifies the project structure

Example: Working with Multiple Source Files

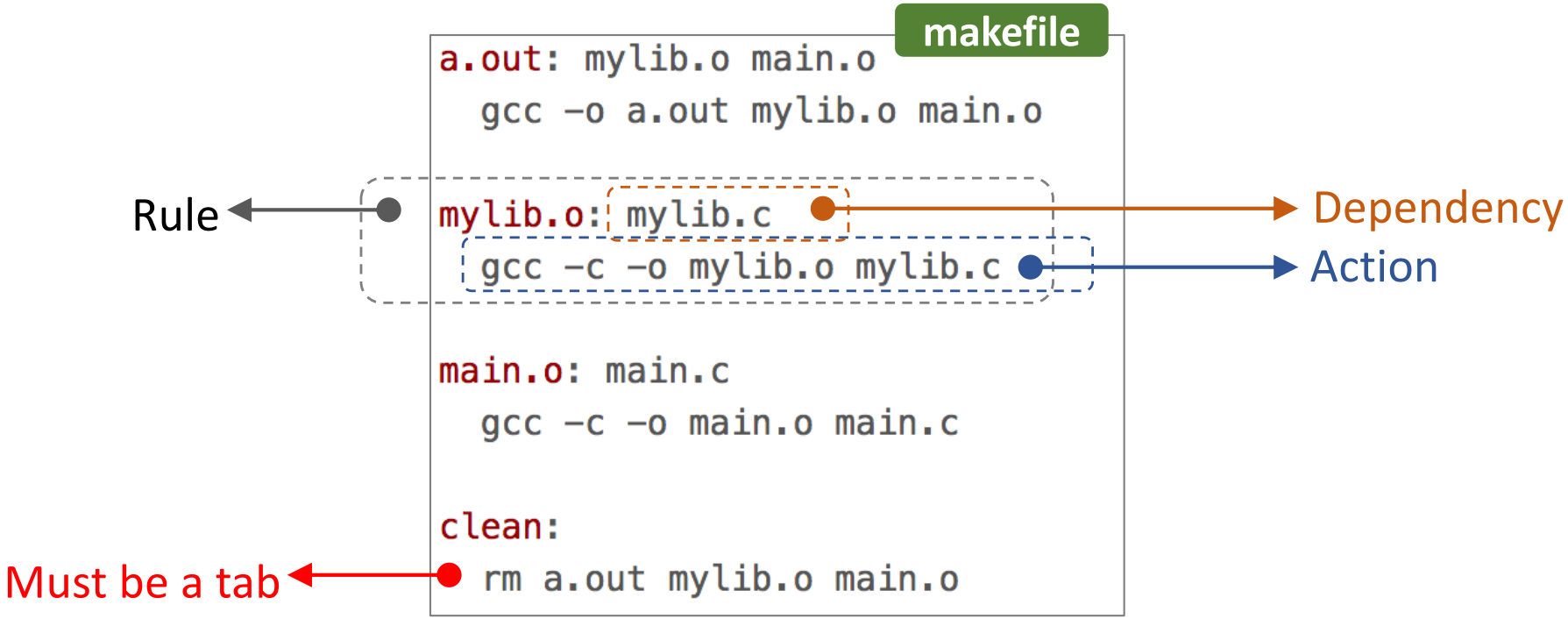
- An example *makefile*

makefile

```
a.out: mylib.o main.o
  gcc -o a.out mylib.o main.o

mylib.o: mylib.c
  gcc -c -o mylib.o mylib.c

main.o: main.c
  gcc -c -o main.o main.c

clean:
  rm a.out mylib.o main.o
```

Example: Working with Multiple Source Files

- An example *makefile*



Example: Working with Multiple Source Files

- How to "make"?

```
$ make
gcc –c –o mylib.o mylib.c
gcc –c –o main.o main.c
gcc -o a.out mylib.o main.o
$
$ ./a.out
(initial) a=3, b=5
(initial) a=5, b=3
```