# Data Structure and Algorithm

## Class 4

---

Seongjin Lee

Updated: 2017-03-06
DSA_2017_04

insight@gnu.ac.kr
http://resourceful.github.io
Systems Research Lab.
GNU

# Table of contents

# Makefile

# Make

- Stuart Feldman developed make in April in 1976 at Bell Labs
- Received 2003 ACM Software system Award for the tool
- It is utility that automatically builds executable programs and libraries from source code by reading file called makefile

# Problem of multiple source files and repetitive routines

Used to detect a change made to an image file and the transformation action might be converted the file to some specific format

- Also can be used to copy the result into a content management system, and then send e-mail to a predefined set of users to note the changes

```
/* main.c */
#include "foo.h''
...
```

```
/* sub.c*/
#include “foo.h”
#include “bar.h”
. . .
```

```
/* utmost.c*/
#include “bar.h”
#include “baz.h”
. . .
```

- If foo.h is changed – main.c and sub.c must be recompiled
- If foo.h is changed but forgot to compile sub.c, the program might not work correctly

It can be used not only for compiling programs, but also for to produce output files from several input files such as TeX

Comment starts with # and continues to the end of the line

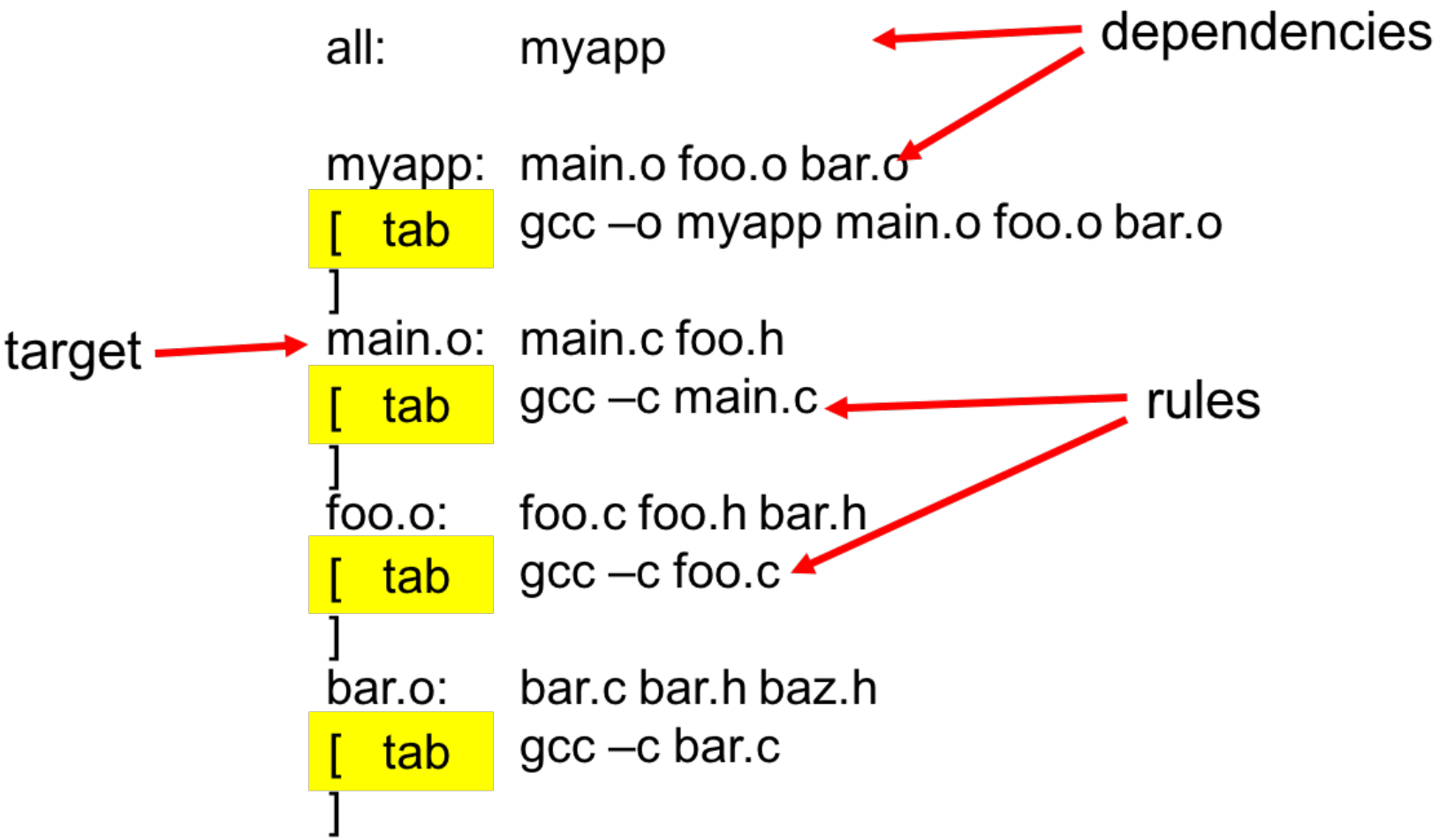
# Syntax of Makefile

Makefile consists of a set of dependencies and rules

- A dependency has a target (a file to be created) and set of source files upon which it is dependent
- The rules describe how to create the target from the dependent files

Typically target is a single executable file

# Make


all: myapp
dependencies
myapp: main.o foo.o bar.o
[ tab
gcc –o myapp main.o foo.o bar.o
]
target
main.o: main.c foo.h
[ tab
gcc –c main.c
rules
]
foo.o: foo.c foo.h bar.h
[ tab
gcc –c foo.c
]
bar.o: bar.c bar.h baz.h
[ tab
gcc –c bar.c
]

# Syntax of Makefile

```c
/* main.c */
#include <stdlib.h>
#include "foo.h"

extern void function_two
     ();
extern void function_
     three();

int main()
{
   function_two();
   function_three();
   exit (EXIT_SUCCESS);
}
```

```c
/* foo.c */
#include "foo.h"
#include "bar.h"

void function_two(){
}
```

```c
/* bar.c */
#include "bar.h"
#include "baz.h"

void function_three(){
}
```

```
> make
> make
```

```
> touch bar.h
> make
> rm bar.o
> make
```

# Macros in a Makefile

Define a macro in a makefile by writing

- `MACRONAME=value`

Accessing the value of MACRONAME by writing

- `$(MACRONAME) or $MACRONAME`

Usually macros are defined inside the makefile

- But can be specified by calling make with macro definition
- `make CC=c89`

# Makefile with macros

```
1  all: myapp
2
3  # Which compiler to use
4  CC = gcc
5
6  # Where are included files are kept
7  INCLUDE = .
8
9  # Options for development
10 CFLAGS = -g Wall ansi
11
12 # Options for release
13 # CFLAGS = -O Wall ansi
```

```
1  myapp: main.o foo.o bar.o
2     $(CC) o myapp main.o foo.o bar.o
3
4  main.o: main.c foo.h
5      $(CC) I$(INCLUDE) $(CFLAGS) c main
           .c
6
7  foo.o: foo.c foo.h bar.h
8      $(CC) I$(INCLUDE) $(CFLAGS) c foo.
           c
9
10 bar.o: bar.c bar.h baz.h
11     $(CC) I$(INCLUDE) $(CFLAGS) c bar.
           c
```

# Special Internal Macros

- `$?` - List of prerequisites (files the target depends on) changed more recently than the current target
- `$@` - Name of the current target
- `$<` - Name of the Current prerequisite
- `$*` - Name of the current prerequisite, without any suffix
- `@` - (applies to rules) Tell make not to print the command to standard output before executing it
- `-` - (applies to rules) Tell make to ignore any errors

# Multiple Targets

```
1  # Where to install
2  INSTDIR = /home/james/class/instdir
3  . . .
4
5  clean:
6     -rm main.o foo.o bar.o
7
8  install: myapp
9     @if [ -d $(INSTDIR) ]; \
10       then \
11       cp myapp $(INSTDIR);\
12       chmod a+x $(INSTDIR)/myapp;\
13       chmod og-w $(INSTDIR)/myapp;\
14       echo “Installed in $(INSTDIR)”;\
15    else \
16       echo “Sorry, $(INSTDIR) does not exist”;\
17    fi
```