C Programming
Lecture 11: make & Makefile

Lecturer: *Dr.* Wan-Lei Zhao

Autumn Semester 2022

Outline

Why make? (1)

```
1  #ifndef MYLIB_H
2  #define MYLIB_H
3  int isodd(int x);
4  float square(float x);
5  #endif
```
mylib.h

```
1  #include "mylib.h"
2  float square(float x){
3      return x*x;
4  }
5
6  int isodd(int x){
7      if(x%2 != 0)
8          return 1;
9      else
10         return 0;
11 }
```
mylib.c

```
1  #include "mylib.h"
2  #include <stdio.h>
3  int main(){
4      float x = 3.4;
5      int a = 5;
6      float y = square(x);
7      if(isodd(a))
8      {
9          printf("%d is odd\n", a);
10     }
11     return 0;
12 }
```
main.c

```
1  gcc myproj.c -o myproj.o -c
2  gcc mylib.c -o mylib.o -c
3
4  gcc -o myproj myproj.o mylib.o
```
Build the project

Why make? (2)

```
1 gcc myproj.c -o myproj.o -c
2 gcc mylib.c -o mylib.o -c
3
4 gcc -o myproj myproj.o mylib.o
```

Listing 1: "Build the project"

- In practice, we may have many libraries to compile and link

- gcc -o myproj myproj.o mylib.o

- If we do it manually, it is too laborious!!!

- This is where "Makefile" comes to fit in

Makefile

- A script file organize all the compilation things together

- It is responsible for

  1 Compiling the source files (compile from .c to .o)

  2 Linking the files into the final executable software

  3 Installing the software to target directory

- Command make will parse the script

- It fulfills the intructions in the script

Prepare Environment (1)

- Define the variables

```
1 WORK_DIR=.
2 CC=gcc
3 LD=gcc
4 OBJ_DIR=$(WORK_DIR)/obj
5 OBJ_RELEASE=$(OBJ_DIR)/mylib.o $(OBJ_DIR)/myproj.o
6 RELEASE=$(WORK_DIR)/bin/myproj
```

- command make supports environment variable definitions

$$VARIABLE\_NAME = value$$

- One can specify the file, directory, command, compilation parameters
- They may support the compilation of the project

```
1  WORK_DIR=.
2  CC=gcc
3  LD=gcc
```

Makefile

- Command make supports environment variable definitions

$$WORK\_DIR = .$$

- Specify the project directory where "Makefile" and the project is located
- The variable name is by convention CAPITALIZED

Prepare Environment (3)

```
1  WORK_DIR=.
2  CC=gcc
3  LD=gcc
```

Listing 2: Makefile

- Command make supports environment variable definitions

$$CC=gcc$$
$$LD=gcc$$

- "CC=gcc" specifies the compiler
- "LD=gcc" specifies the linker

Prepare Environment (4)

```
1  OBJ_DIR=$(WORK_DIR)/obj
2  OBJ_RELEASE=$(OBJ_DIR)/mylib.o $(OBJ_DIR)/myproj.o
3  RELEASE=$(WORK_DIR)/bin/myproj
```

<div align="center">Listing 3: Makefile</div>

- Command make supports environment variable definitions

$$\$(WORK\_DIR)/obj$$

- $(VARIABLE) cite the value of the VARIABLE
- Here "$(WORK_DIR)" is replaced by "./"

Prepare Environment (5)

```
1  OBJ_DIR=$(WORK_DIR)/obj
2  OBJ_RELEASE=$(OBJ_DIR)/mylib.o $(OBJ_DIR)/myproj.o
3  RELEASE=$(WORK_DIR)/bin/myproj
```

Listing 4: Makefile

- The above instructions indicate
  1. The object files will be put to ./obj/
  2. "OBJ_RELEASE" keeps the lists of all object files
  3. The final target binary software name is "myproj"
  4. It will be put to ./bin/

Prepare Environment (6)

```
1  WORK_DIR=.
2  CC=gcc
3  LD=gcc
4  OBJ_DIR=$(WORK_DIR)/obj
5  OBJ_RELEASE=$(OBJ_DIR)/mylib.o  $(OBJ_DIR)/myproj.o
6  RELEASE=$(WORK_DIR)/bin/myproj
```

1. We know the working directory
2. We have the compiler and linker
3. We know where we should put the object files
4. We know where we should put the target binary file

Prepare Environment (7)

```
1  WORK_DIR=.
2  CC=gcc
3  LD=gcc
4  OBJ_DIR=$(WORK_DIR)/obj
5  OBJ_RELEASE=$(OBJ_DIR)/mylib.o $(OBJ_DIR)/myproj.o
6  RELEASE=$(WORK_DIR)/bin/myproj
7
8  before_release:
9          test -d bin || mkdir -p bin
10         test -d $(OBJ_DIR) || mkdir -p $(OBJ_DIR)
```

1. However, "./obj/" and "./bin/" are not ready
2. We can test and make them if necessary

Compile the source file

```
1  $(OBJ_DIR)/mylib.o: mylib.c
2          $(CC) -c mylib.c -o $(OBJ_DIR)/mylib.o
```

- Instruction "$(OBJ_DIR)/mylib.o" compiles "mylib.c"
- The compilation relies on file "mylib.c"
- The indentation should be by "Tab"
- We can do so for all the source files
- The resulting file is put to "./obj/mylib.o"

```
1  $(OBJ_DIR)/mylib.o: mylib.c
2          $(CC) -c mylib.c -o $(OBJ_DIR)/mylib.o
3
4  $(OBJ_DIR)/myproj.o: myproj.c
5          $(CC) -c myproj.c -o $(OBJ_DIR)/myproj.o
```

Link the source file

```
1  release : $(OBJ_RELEASE)
2          $(LD) −o $(RELEASE) $(OBJ_RELEASE)
```

- The project will be linked with mylib.o and myproj.o
- The list of object files are kept in "$(OBJ_RELEASE)"
- $(LD) calls "gcc"
- The target is specified by "$(RELEASE)"

Build the whole project

```
1  release : before_release $(OBJ_RELEASE)
2          $(LD) −o $(RELEASE) $(OBJ_RELEASE)
```

- The instruction "release" relies on another two intructions
- "before_release" and "$(OBJ_RELEASE)"
- "$(OBJ_RELEASE)" are a list of instructions
    1. Run instruction "before_release"
    2. Run list of instructions in "$(OBJ_RELEASE)"
    3. Run $(LD) -o $(RELEASE) $(OBJ_RELEASE)

Clean the object files

- In some cases, we may want to clean the object files

```
1  clean :
2          rm −rf  $( OBJ_DIR ) /∗. o
3          rm −rf  $( RELEASE )
```

- We call command "rm"
- We label the instruction as "clean"

A Complete Makefile

```
 4  WORK_DIR=.
 5  CC=gcc
 6  LD=gcc
 7  OBJ_DIR=$(WORK_DIR)/obj
 8  OBJ_RELEASE=$(OBJ_DIR)/mylib.o $(OBJ_DIR)/myproj.o
 9  RELEASE=$(WORK_DIR)/bin/myproj
10
11  $(OBJ_DIR)/mylib.o: mylib.c
12          $(CC) -c mylib.c -o $(OBJ_DIR)/mylib.o
13
14  $(OBJ_DIR)/myproj.o: myproj.c
15          $(CC) -c myproj.c -o $(OBJ_DIR)/myproj.o
16
17  before_release:
18          test -d bin || mkdir -p bin
19          test -d $(OBJ_DIR) || mkdir -p $(OBJ_DIR)
20
21  release: before_release $(OBJ_RELEASE)
22          $(LD) -o $(RELEASE) $(OBJ_RELEASE)
```
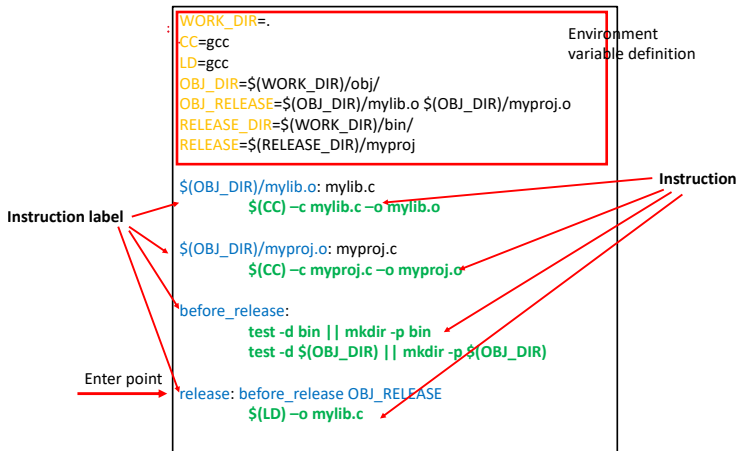
Makefile

A Complete Makefile

```
23  clean :
24          rm −r f  $(OBJ_DIR)/*.o
25          rm −r f  $(RELEASE)
```
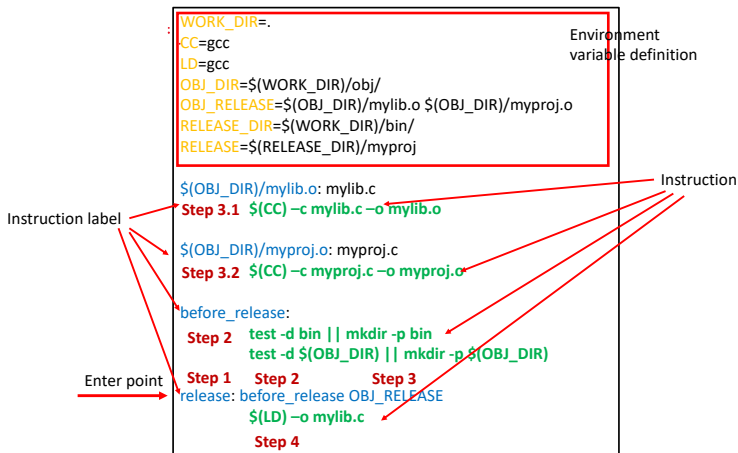
Makefile

- Five major sections
  1. Define the environment variables
  2. Prepare directories
  3. Instructions of compiling source files to object files
  4. Link object files to target binary executable or library
  5. Instructions to clean the object files

Running Flow inside Makefile (1)


```
WORK_DIR=.
CC=gcc
LD=gcc
OBJ_DIR=$(WORK_DIR)/obj/
OBJ_RELEASE=$(OBJ_DIR)/mylib.o $(OBJ_DIR)/myproj.o
RELEASE_DIR=$(WORK_DIR)/bin/
RELEASE=$(RELEASE_DIR)/myproj
```
Environment variable definition

```
$(OBJ_DIR)/mylib.o: mylib.c
        $(CC) –c mylib.c –o mylib.o

$(OBJ_DIR)/myproj.o: myproj.c
        $(CC) –c myproj.c –o myproj.o

before_release:
        test -d bin || mkdir -p bin
        test -d $(OBJ_DIR) || mkdir -p $(OBJ_DIR)

release: before_release OBJ_RELEASE
        $(LD) –o mylib.c
```

Instruction label

Instruction

Enter point

- Run command "make release"

Running Flow inside Makefile (2)

WORK_DIR=.
CC=gcc
LD=gcc
OBJ_DIR=$(WORK_DIR)/obj/
OBJ_RELEASE=$(OBJ_DIR)/mylib.o $(OBJ_DIR)/myproj.o
RELEASE_DIR=$(WORK_DIR)/bin/
RELEASE=$(RELEASE_DIR)/myproj

Environment variable definition

$(OBJ_DIR)/mylib.o: mylib.c
Step 3.1 **$(CC) –c mylib.c –o mylib.o**

$(OBJ_DIR)/myproj.o: myproj.c
Step 3.2 **$(CC) –c myproj.c –o myproj.o**

Instruction label

Instruction

before_release:
Step 2 **test -d bin || mkdir -p bin**
test -d $(OBJ_DIR) || mkdir -p $(OBJ_DIR)

Enter point

Step 1 **Step 2** **Step 3**
release: before_release OBJ_RELEASE
$(LD) –o mylib.c
Step 4

- Run command "make release"

Add libraries in Makefile (1)

- We may need either static or dynamic libraries or both

```c
1  #include <math.h>
2  #include "mylib.h"
3  #include <stdio.h>
4  int main(){
5      float x = 3.4;
6      int a = 5;
7      float y = square(x);
8      float z = sqrt(x);
9      return 0;
10 }
```

<div align="center">myproj.c</div>

- For this code, we should compile it by

```
1  gcc -o myproj myproj.o mylib.o -lm
```

Add libraries in Makefile (2)

```
1  WORK_DIR=.
2  CC=gcc
3  LD=gcc
4  LDFLAGS= −lm
5  OBJ_DIR=$(WORK_DIR)/obj
6  OBJ_RELEASE=$(OBJ_DIR)/mylib.o $(OBJ_DIR)/myproj.o
7  RELEASE=$(WORK_DIR)/bin/myproj
8
9  $(OBJ_DIR)/mylib.o: mylib.c
10         $(CC) −c mylib.c −o $(OBJ_DIR)/mylib.o
11
12 $(OBJ_DIR)/myproj.o: myproj.c
13         $(CC) −c myproj.c −o $(OBJ_DIR)/myproj.o
14
15 before_release:
16         test −d bin || mkdir −p bin
17         test −d $(OBJ_DIR) || mkdir −p $(OBJ_DIR)
18
19 release: before_release $(OBJ_RELEASE)
20         $(LD) $(LDFLAGS) −o $(RELEASE) $(OBJ_RELEASE)
```

Makefile

Outline

Why cmake?

- However, writing a Makefile line-by-line is still too sweaty
- There are several convenient ways
  1. "cbp2make"
  2. cmake

About cmake

- It is another useful tool

- It helps to build the Makefile

- The cmake requires another simpler script