



2016 Fall System Programming

# GCC & make

2016. 09. 13

황슬아

seula.hwang@cnu.ac.kr

Embedded System Lab.  
Computer Engineering Dept.  
Chungnam National University



## 실습 소개

### ❖ 과목 홈페이지

- ◆ 충남대학교 사이버캠퍼스 ( <http://e-learn.cnu.ac.kr/> )

### ❖ 연락처

- ◆ 황슬아
- ◆ 공대 5호관 533호 임베디드 시스템 연구실
- ◆ [seula.hwang@cnu.ac.kr](mailto:seula.hwang@cnu.ac.kr)
  - ❖ Email 제목은 [sys00] 으로 시작하도록 작성



# 목차

## 1. 개요

## 2. GCC

- I. GCC 란?
- II. GCC 컴파일 과정
- III. 실습 1

## 3. make

- I. make 란?
- II. Makefile 작성법
- III. 실습 2



# 개요

## ❖ 실습 명

- ◆ GCC와 make를 통한 컴파일

## ❖ 목표

- ◆ GCC를 통해 컴파일 할 수 있다.
- ◆ Makefile을 작성 할 수 있다.

## ❖ 주제

- ◆ GCC
- ◆ make

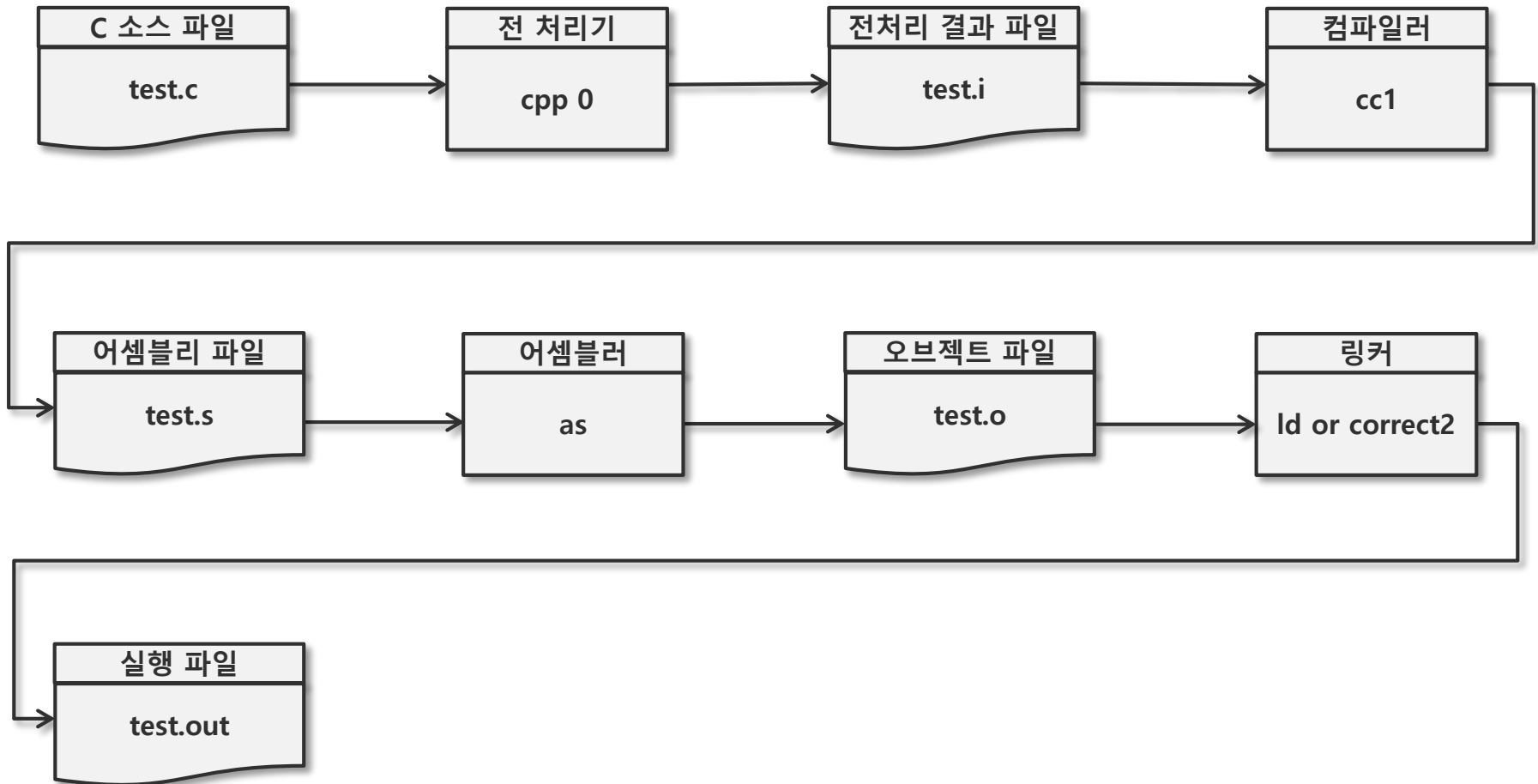


# GCC 란?

## ❖ GCC

- ◆ GNU Compiler Collection
- ◆ GNU(GNU is Not Unix) 프로젝트의 일환으로 개발되어 널리 쓰이고 있는 **컴파일러**
- ◆ 원래 C만을 지원했던 컴파일러로 “GNU C Compiler” 였지만, 현재는 C++, JAVA, FORTRAN 등의 프로그래밍 언어를 지원
- ◆ GCC는 실제 컴파일을 과정을 담당하는 것이 아니라 **전 처리기**와 **C 컴파일러**, **어셈블러**, **링커**를 각각 **호출하는 역할**

# GCC 컴파일 과정





# GCC 컴파일 과정

## 1. 소스 코드 작성

- I. 자신의 홈 디렉토리(~)에서 vi를 이용해 소스코드를 작성

- ◆ 소스 파일명: **like.c**

```
#include <stdio.h>

int main() {
    printf("I like you!\n");
}
```

- ◆ vi편집기에서 소스코드 작성 후, 표준 모드에서 :wq 혹은 shift + zz 명령을 통해 저장하고 종료. (1주차 실습자료 참고)



## GCC 컴파일 과정

### 2. GCC를 이용하여 소스코드 컴파일

❖ 사용법: gcc [옵션] [소스파일명]

❖ 옵션을 따로 지정하지 않으면 default로 a.out이라는 이름의 실행파일이 생성됨.

```
esl03@localhost: /home/sys03/esl03/lab02
esl03@localhost:/home/sys03/esl03/lab02$ vi like.c
esl03@localhost:/home/sys03/esl03/lab02$ ls
like.c
esl03@localhost:/home/sys03/esl03/lab02$ gcc like.c
esl03@localhost:/home/sys03/esl03/lab02$ ls
a.out like.c
esl03@localhost:/home/sys03/esl03/lab02$
esl03@localhost:/home/sys03/esl03/lab02$ ./a.out
I like you!
esl03@localhost:/home/sys03/esl03/lab02$
```

1. gcc를 사용하여 컴파일

2. 컴파일된 파일의 실행

◆ ./[파일명]을 통해 실행





# GCC 컴파일 과정 - 컴파일 옵션

## ❖ -c 옵션

- 컴파일 과정 중, 링크를 하지 않고 오브젝트 파일(\*.o)만 생성
- 사용법: `gcc -c [소스파일명]`

```
[eslab@eslab like_ex]$ gcc -c like.c  
[eslab@eslab like_ex]$ ls  
like.c like.o
```

## ❖ -o 옵션

- 실행 파일(\*.out)의 이름을 지정
- 사용법: `gcc -o [실행파일명] [소스파일명]`

```
[eslab@eslab like_ex]$ gcc -o like like.c  
[eslab@eslab like_ex]$ ls  
like like.c
```



# GCC 컴파일 과정 - 컴파일 옵션

- ❖ -v 옵션
  - ◆ 컴파일러의 버전과 각 단계에서 실행하는 자세한 사항을 출력 (verbose)
- ❖ --save-temp 옵션
  - ◆ 컴파일 과정 중 발생하는 모든 중간 파일을 저장
- ❖ -W 옵션
  - ◆ 합법적이지만 모호한 코딩에 대해 부가적인 정보 제공
- ❖ -Wall 옵션
  - ◆ 모든 모호한 문법에 대한 경고 메시지 출력

```
[eslab@eslab test]$ gcc -Wall -W -o like like.c
      In function '':
      warning: control reaches end of non-void function [-Wreturn-type]
  }
  ^
[eslab@eslab test]$
```



## GCC 컴파일 과정

### 3. 여러 개의 파일을 함께 컴파일

- ◆ 사용법: `gcc -o [실행파일명] [소스코드1.c] [소스코드2.c] ...`

```
[eslab@eslab test]$ ls  
func.c  like.c  
[eslab@eslab test]$ gcc -o exefile like.c func.c  
[eslab@eslab test]$ ls  
exefile func.c  like.c  
[eslab@eslab test]$
```

### 4. 필요한 소스만 컴파일(1)

- 1) `gcc -c [소스파일명1]`
- 2) `gcc -c [소스파일명2]`
- 3) `gcc -o [실행파일명] [소스파일명1.o] [소스파일명2.o]`
  - ◆ `-c` 옵션은 컴파일은 하지만, 링크는 하지 않음.
  - ◆ 소스코드가 매우 많은 파일로 분리되어 있는 경우 효과적임.



## GCC 컴파일 과정

### 4. 필요한 소스만 컴파일 하기(2)

```
[eslab@eslab test]$ ls  
file1.c file2.c file3.c func.c like.c
```

```
[eslab@eslab test]$ gcc -c file1.c  
[eslab@eslab test]$ gcc -c file3.c  
[eslab@eslab test]$ gcc -c like.c  
[eslab@eslab test]$ ls  
file1.c file1.o file2.c file3.c file3.o func.c like.c like.o
```

❖ file1.c, file3.c, like.c를 -c 옵션을 사용해  
서 각각 오브젝트 파일로 변환

```
[eslab@eslab test]$ gcc -o file file1.o file3.o like.o  
[eslab@eslab test]$ ls  
file file1.c file1.o file2.c file3.c file3.o func.c like.c like.o  
[eslab@eslab test]$
```

❖ 생성된 \*.o 파일 들을 -o 옵션을  
사용해 실행 파일로 컴파일



# GCC 컴파일 과정 - 따라하기

❖ 아래의 소스를 작성하고 컴파일, 실행

## ❖ ex01.c

```
#include <stdio.h>

int main()
{
    int nResult = 0;
    int nAlpha = 5, nBeta = 3;

    nResult = funcAdd( nAlpha, nBeta );
    printf( " %d + %d = %d\n", nAlpha, nBeta, nResult);

    nResult = funcSub( nAlpha, nBeta );
    printf( " %d - %d = %d\n", nAlpha, nBeta, nResult);

    return 0;
}
```

## ❖ sub.c

```
int funcSub( int nAlpha, int nBeta )
{
    return nAlpha - nBeta;
}
```

## ❖ add.c

```
int funcAdd( int nAlpha, int nBeta )
{
    return nAlpha + nBeta;
}
```

## ❖ 컴파일

```
[a000000000@eslab lab02]$ ls
add.c ex01.c sub.c
[a000000000@eslab lab02]$ gcc -c add.c
[a000000000@eslab lab02]$ gcc -c sub.c
[a000000000@eslab lab02]$ gcc -c ex01.c
[a000000000@eslab lab02]$ gcc -o ex01.out sub.o add.o ex01.o
[a000000000@eslab lab02]$ ls
add.c add.o ex01.c ex01.o ex01.out sub.c sub.o
[a000000000@eslab lab02]$ ./ex01.out
5 + 3 = 8
5 - 3 = 2
[a000000000@eslab lab02]$
```



# 실습 1

## ❖ 실습 1-1

- ◆ 곱셈과 나눗셈 연산을 하는 코드를 아래의 조건에 맞게 작성하고 컴파일 후 실행
  - ❖ 조건1: 곱셈의 기능을 하는 함수를 **mul.c**에 작성
  - ❖ 조건2: 나눗셈의 기능을 하는 함수를 **div.c**에 작성
  - ❖ 조건3: **main** 함수는 **ex01\_2.c**에 작성하고 내용은 아래의 실행 결과 처럼 나오도록 작성
  - ❖ 조건4: 컴파일 한 후 실행하여 다음과 같이 출력되어야 함.
- ◆ 실행 결과

```
[a0000000000@eslab lab02]$ ls
add.c  div.c  ex01.o  ex01_2.c  mul.c  sub.o
add.o  ex01.c  ex01.out  ex01_2.out  sub.c
[a0000000000@eslab lab02]$ ./ex01_2.out
6 * 2 = 12
6 / 2 = 3
```



# 실습 1

## ❖ 실습 1-2

- ◆ Gcc를 사용하여 다음 조건에 맞게 컴파일 후 생성된 파일을 실행
  - ❖ 조건1: /home/ubuntu/lab02/practice01\_2.tar.gz를 자신의 홈 디렉토리(~)에 복사 후 압축 해제
  - ❖ 조건2: practice01\_2 디렉토리에 포함된 calendar.c, diary.h, main.c, memo.c 파일들을 이용하여 아래의 순서대로 컴파일 후 실행
    - ◆ Memo.o 생성 -> calendar.o 생성 -> ex01\_3.out 생성 -> 실행
    - ◆ 위의 파일을 생성하는 과정을 필수적으로 포함
- ◆ 실행 결과

```
[a0000000000@eslab lab02]$ ./ex01_3.out  
function memo.  
function calendar.
```



# Make 란?

- ❖ 프로그램 빌드 자동화 도구
- ❖ 여러 파일들 간의 의존성과 각 파일에 필요한 명령을 정의함으로써 프로그램을 자동으로 컴파일 해주는 프로그램
- ❖ 의존성과 필요한 명령을 서술할 수 있는 표준적인 문법을 가지고 있음
- ❖ 위의 문법으로 기술된 파일(주로 Makefile)을 make프로그램이 해석하여 프로그램 빌드
- ❖ 복잡하고 방대한 프로그램을 개발할 때 단순 반복 작업과 재 작성을 최소화 시켜 생산성을 높이는데 도움을 주는 도구





# Make file 작성법

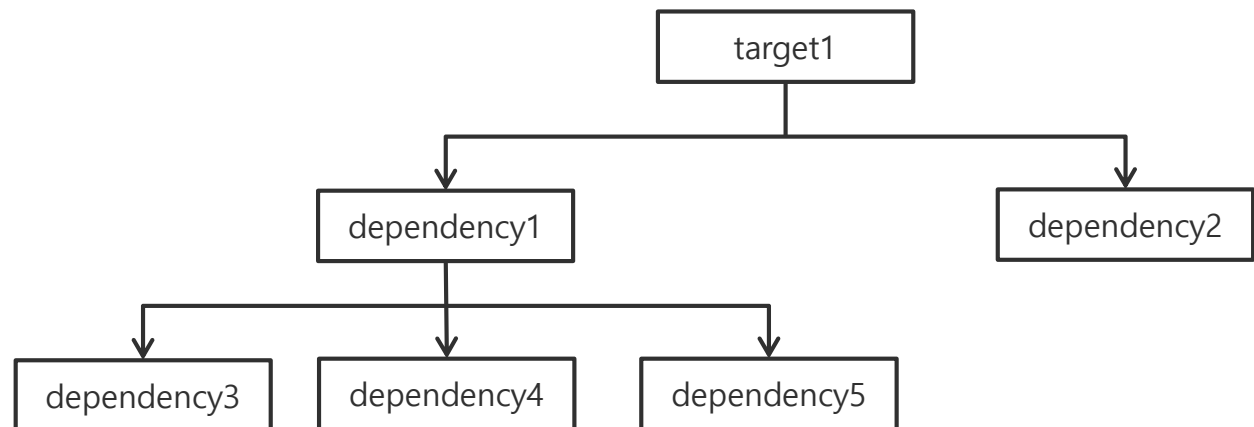
## ❖ 기본 구조

```
CC = gcc

target1 : dependency1 dependency2
        command1
        command2

dependency1 : dependency3 dependency4 dependency5
            command3
            ←→ command4
```

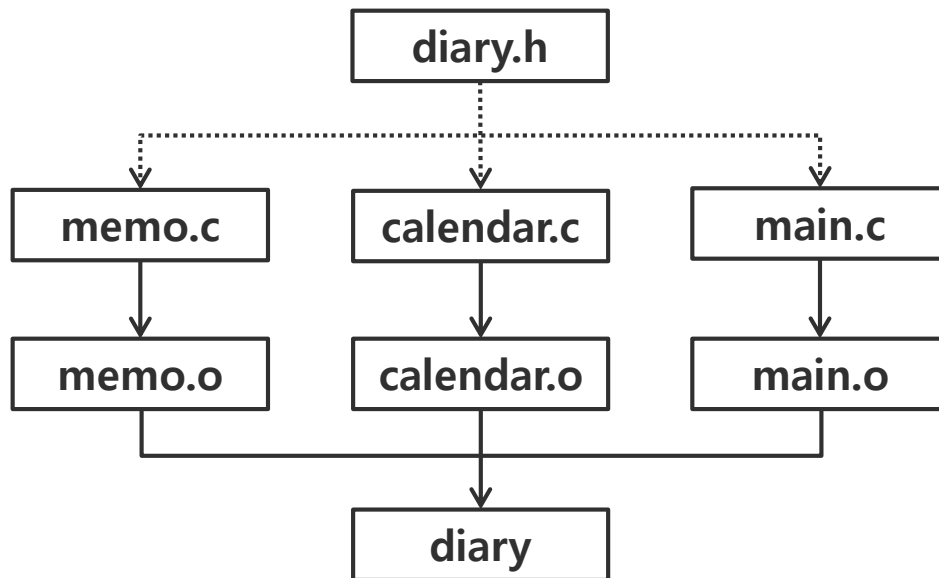
주의) command는 항상 tab키를 사용해서 작성해야 한다





## Make file 작성법 - 따라하기

- ❖ Makefile의 작성법 이해를 돕기 위한 예제
  - ♦ /home/ubuntu/lab02/make\_example.tar.gz를 자신의 홈 디렉토리에 복사 후 압축 해제
  - ♦ maketest 디렉토리에 들어있는 소스를 컴파일 하여 실행 파일 “diary”를 만든다고 할 때, 각 파일의 종속 구조는 다음과 같다.



- ❖ 다음 페이지의 Makefile 예는 해당 종속 관계에 맞추어 작성된 것.



# Make file 작성법 - 따라하기

## ❖ Makefile의 작성법 이해를 돕기 위한 예제

```
all : diary

diary : memo.o calendar.o main.o
        gcc -W -Wall -o diary memo.o calendar.o main.o

memo.o : memo.c
        gcc -W -Wall -c -o memo.o memo.c

calendar.o : calendar.c
        gcc -W -Wall -c -o calendar.o calendar.c

main.o : main.c
        gcc -W -Wall -c -o main.o main.c
```

- ♦ 위와 같이 Makefile을 작성한 후, “make” 명령을 입력하면 종속 관계를 만족하도록 각 오브젝트에 대한 컴파일 과정을 수행하여 diary 실행 파일을 생성한다.



# Make file 작성법 - 따라하기

## ❖ make 실행 결과

```
[eslab@eslab Maketest]$ ls
Makefile  calendar.c  diary.h  main.c  memo.c
[eslab@eslab Maketest]$ make
gcc -W -Wall -c memo.c
gcc -W -Wall -c calendar.c
gcc -W -Wall -c main.c
gcc -W -Wall -o diary memo.o calendar.o main.o
[eslab@eslab Maketest]$ ls
Makefile  calendar.o  diary.h  main.o  memo.o
calendar.c  diary      main.c  memo.c
```

- 다시 make 명령을 수행해보면, 동작하지 않는다.

```
[eslab@eslab Maketest]$ make
make: 'all'를 위해 할 일이 없습니다
```

❖ make 명령은 소스코드의 변경이 있을 때만 실행 할 수 있기 때문.

- main.c의 코드에서 memo() 함수의 호출을 두 개로 수정하면 make가 정상 동작함을 볼 수 있다.



# Make file 작성법 - 따라하기

## ❖ 매크로를 이용한 작성법

- 매크로를 사용하여 Makefile을 작성하는 방법. 아래와 같이 수정후 make 해본다.
- 매크로는 ‘사용자 정의 변수’에 특정한 문자열을 정의하고, 치환하여 사용하는 것
  - ❖ 매크로를 참조할 때는 아래와 같이 사용한다.
  - ❖ CC -> \$(CC)
  - ❖ CFLAGS -> \$(CFLAGS)

```
CC = gcc
CFLAGS = -W -Wall
TARGET = diary

all : $(TARGET)

$(TARGET) : memo.o calendar.o main.o
    $(CC) $(CFLAGS) -o diary memo.o calendar.o main.o

memo.o : memo.c
    $(CC) $(CFLAGS) -c -o memo.o memo.c

calendar.o : calendar.c
    $(CC) $(CFLAGS) -c -o calendar.o calendar.c

main.o : main.c
    $(CC) $(CFLAGS) -c -o main.o main.c
```



# Make file 작성법 - 따라하기

- ❖ 자동 매크로 리스트를 이용한 작성법
  - ♦ 자동 매크로 리스트를 사용하여 Makefile을 작성하는 방법. 아래와 같이 수정후 make 해본다.

```
CC = gcc
CFLAGS = -W -Wall
TARGET = diary

all : $(TARGET)

$(TARGET) : memo.o calendar.o main.o
    $(CC) $(CFLAGS) -o $@ $^

memo.o : memo.c
    $(CC) $(CFLAGS) -c -o $@ $^

calendar.o : calendar.c
    $(CC) $(CFLAGS) -c -o $@ $^

main.o : main.c
    $(CC) $(CFLAGS) -c -o $@ $^
```



# Make file 작성법

- ❖ 자동 매크로 리스트를 이용한 작성법
  - ♦ 자동 매크로는 내부적으로 정의되어 있는 매크로이다.
  - ♦ 자동 매크로들은 아래의 표와 같다.

매크로	설명
\$?	현재의 타겟보다 최근에 변경된 종속 항목 리스트
\$^	현재 타겟의 종속 항목 리스트
\$@	현재 타겟의 이름
\$<	현재 타겟보다 최근에 변경된 종속 항목 리스트
\$*	현재 타겟보다 최근에 변경된 현재 종속 항목의 이름
\$%	현재의 타겟이 라이브러리 모듈일 때, *.o 파일에 대응되는 이름



# Make file 작성법 - 따라하기

## ❖ .SUFFIXES 매크로를 이용한 작성법

- ◆ Make가 중요하게 여길 확장자 리스트를 등록해 준다.
- ◆ 사용법: .SUFFIXES: .o .c
  - ❖ 미리 정의된 .c (소스파일)를 컴파일 해서, .o (목적파일)을 만들어내는 루틴이 자동적으로 동작하도록 되어 있다.

```
CC = gcc
CFLAGS = -W -Wall
TARGET = diary

OBJECTS = memo.o calendar.o main.o

.SUFFIXES : .o .c

%.o : %.c
    $(CC) $(CFLAGS) -c $^

all : $(TARGET)

$(TARGET) : $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^
```





# Make file 작성법 - 따라하기

## ❖ Clean 명령어

- Clean 명령어를 이용하여 불필요한 파일을 삭제할 수 있다.
- 아래와 같이 입력하고 “make clean” 명령어를 치면 불필요한 파일을 삭제하도록 할 수 있다.

```
CC = gcc
CFLAGS = -W -Wall
TARGET = diary

OBJECTS = memo.o calendar.o main.o

.SUFFIXES : .o .c

%.o : %.c
    $(CC) $(CFLAGS) -c $^

all : $(TARGET)

$(TARGET) : $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^

clean :
```

```
    rm -rf *.o
    rm -rf *.out
```

**make clean** 명령어를 수행하면  
모든 .o 와 .out 파일을 삭제한다.



## 실습 2

### ❖ 실습 2-1

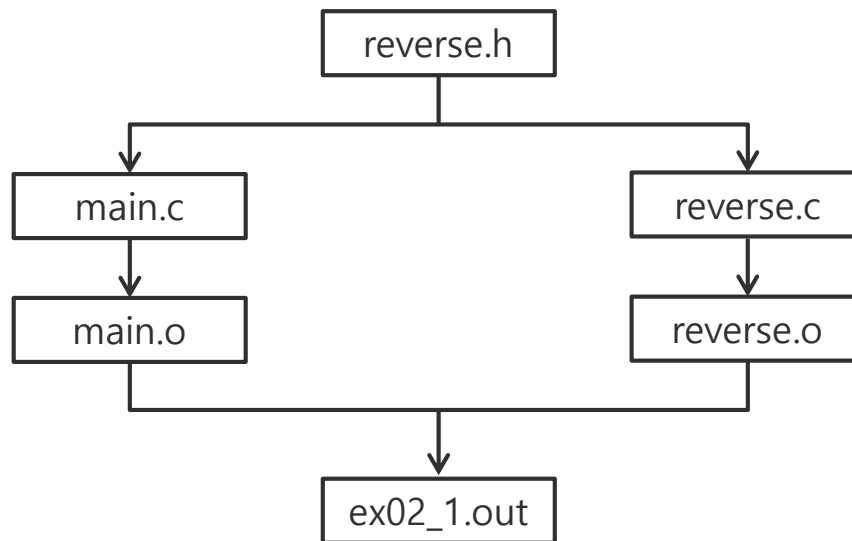
- ◆ 다음 조건들을 만족하는 코드를 작성하고 Makefile을 만들어서 컴파일 후 실행 파일을 실행시켜 결과를 확인
  - ❖ 조건 1 : 자신의 학번을 입력 받아 거꾸로 출력하는 프로그램을 작성
  - ❖ 조건 2 : reverse.c 에는 입력 받은 학번을 거꾸로 출력하는 함수 작성
  - ❖ 조건 3 : reverse 함수를 main에서 호출하여 아래의 실행 결과가 나오도록 작성.
  - ❖ 조건 4 : Makefile의 네 가지 작성법(기본, 매크로, 자동매크로, SUFFIXES) 을 모두 이용하여 컴파일 (각각의 Makefile 캡처)
    - ◆ 컴파일 과정은 동일하므로 한번만 캡처
  - ❖ 조건 5 : 파일 종속 구조를 다음 장의 그림처럼 만들고 그 파일 구조대로 Makefile을 작성



## 실습 2

### ❖ 실습 2-1

#### ◆ 파일의 종속 구조



#### ◆ 실행 결과

❖ 학번을 입력하면 거꾸로 출력

```
[a000000000@eslab lab02]$ ./ex02_1.out  
Student Number : 201312345  
Reverse : 543213102
```



## 실습 2

### ❖ 실습 2-2

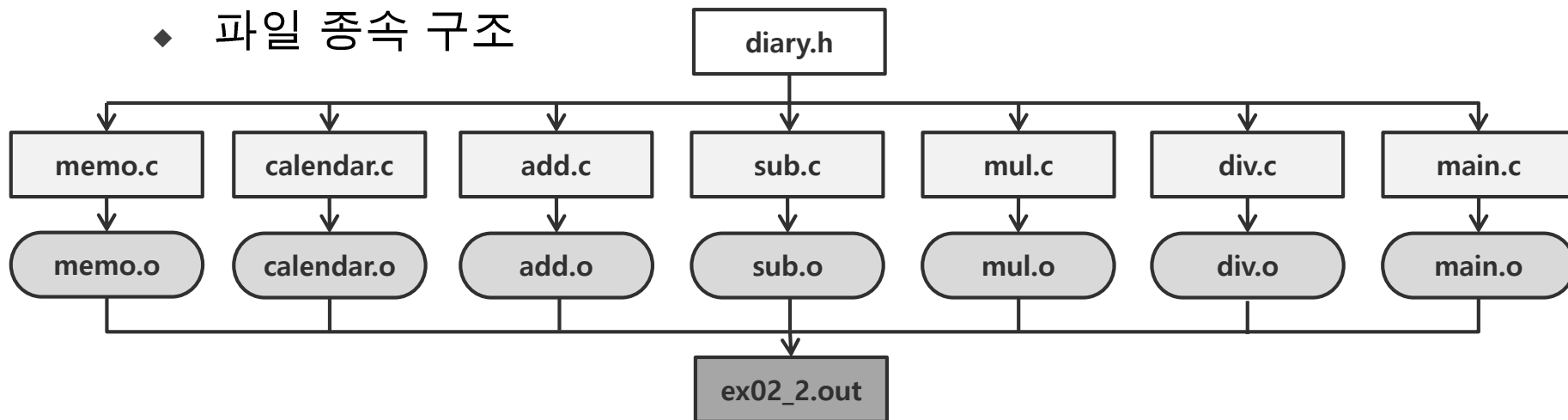
- ◆ 다음 조건들을 만족하는 코드를 작성하고 Makefile을 만들어서 컴파일 후 실행 파일을 실행시켜 결과를 확인
  - ❖ 조건 1: /home/ubuntu/lab02/practice02\_2.tar.gz 를 자신의 홈 디렉토리로 복사
  - ❖ 조건 2: main.c, add.c, sub.c, div.c를 수정하여, 사칙연산과 memo, calendar 코드를 수행
    - ◆ practice02\_2.tar.gz 에 add.c, sub.c, div.c, memo.c, calendar.c가 포함되어 있음.
  - ❖ 조건 3: Makefile을 네 가지 작성법(기본, 매크로, 자동매크로, SUFFIXES)를 모두 이용하여 컴파일 (각각의 Makefile 캡처)
    - ◆ 컴파일 과정은 동일하므로 한번만 캡처



## 실습 2

### ❖ 실습 2-2

#### ◆ 파일 종속 구조



#### ◆ 실행 결과

```
[a000000000@eslab lab02]$ ./ex02_02.out  
function memo.  
function calendar.  
5 + 2 = 7  
5 - 2 = 3  
5 * 2 = 10  
5 / 2 = 2
```



## 제출 사항

- ❖ 오늘 수업에서 진행된 **실습1과 실습2**를 문서로 작성하여 사이버캠퍼스와 서면으로 제출
  - ◆ **PDF** 파일로 제출
    - ❖ 한글과 MS word의 다른이름으로 저장 기능 활용
  - ◆ 파일 제목: [sys00]HW02\_학번\_이름
  - ◆ 반드시 위의 양식을 지켜야 함. (**위반 시 감점**)
  - ◆ **보고서는 제공된 양식 사용**
- ❖ 자신이 실습한 내용을 증명할 것 (자신의 학번이 항상 보이도록)
- ❖ **제출일자**
  - ◆ **사이버 캠퍼스: 2016년 9월 20일 화요일 08시 59분 59초까지**
  - ◆ **서면 제출 : 2016년 9월 20일 실습시간까지**