2020 Fall System Programming

GCC & make

2020. 09. 25

권진세

Embedded System Lab.
Computer Engineering Dept.
Chungnam National University

CNU
Embedded System
Laboratory

실습 소개

- 과목 홈페이지
  - 충남대학교 사이버캠퍼스 (http://e-learn.cnu.ac.kr/)
- 실습 서버
  - Putty 활용
  - 133.186.221.214

목차

CNU
Embedded System
Laboratory

개요

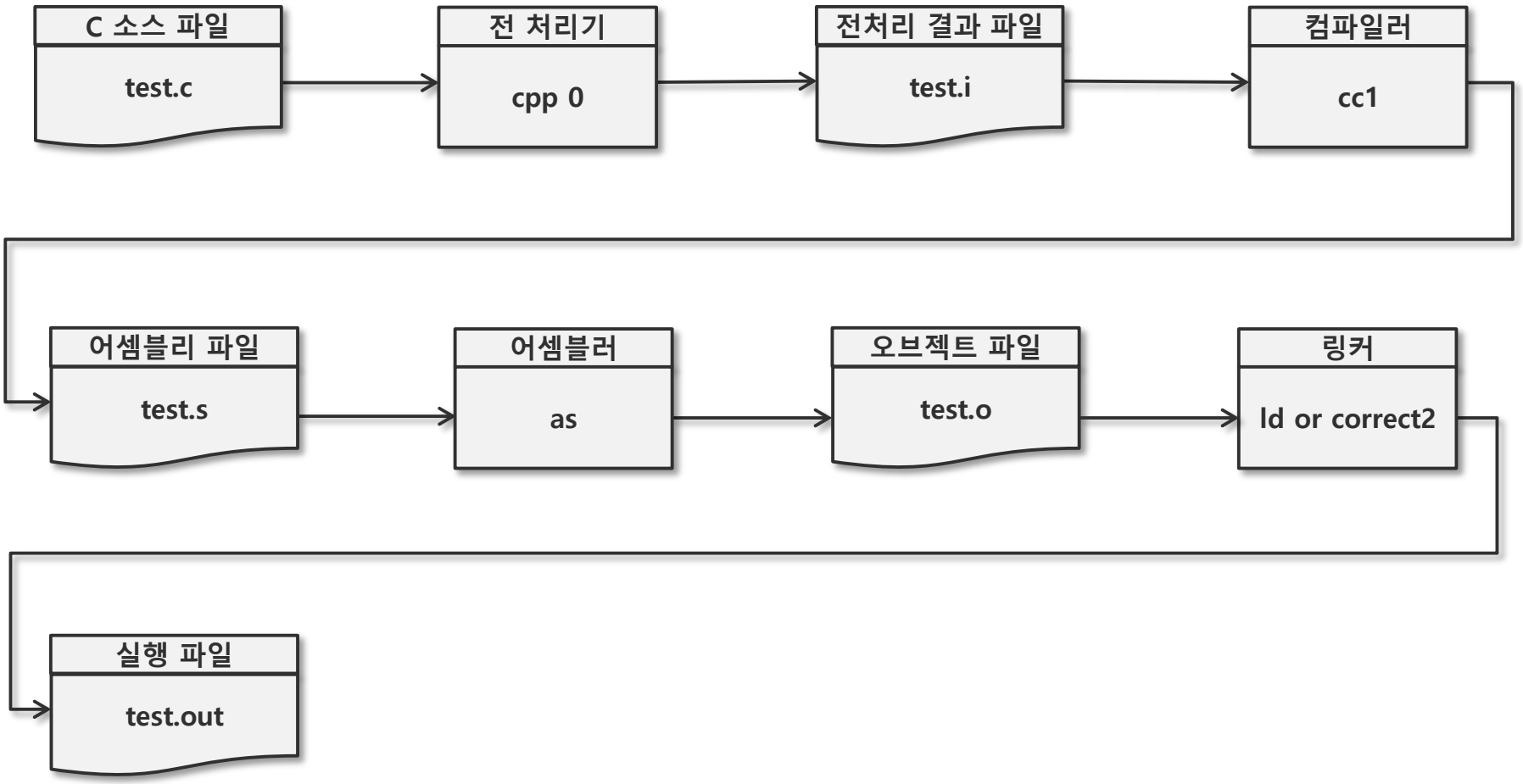
- 실습 명
  - GCC와 make를 통한 컴파일
- 목표
  - GCC를 통해 컴파일 할 수 있다.
  - Makefile을 작성 할 수 있다.
- 주제
  - GCC
  - make

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, PORTRAN 등의 프로그래밍 언어를 지원
  - GCC는 실제 컴파일 과정을 담당하는 것이 아니라 전 처리기와 C 컴파일러, 어셈블러, 링커를 각각 호출하는 역할

CNU
Embedded System
Laboratory

GCC 컴파일 과정



CNU Embedded System Laboratory

GCC 컴파일 과정

1. 소스 코드 작성

I. 자신의 홈 디렉토리(~)에서 vi를 이용해 소스코드를 작성

- 소스 파일명: **like.c**

```c
#include <stdio.h>

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

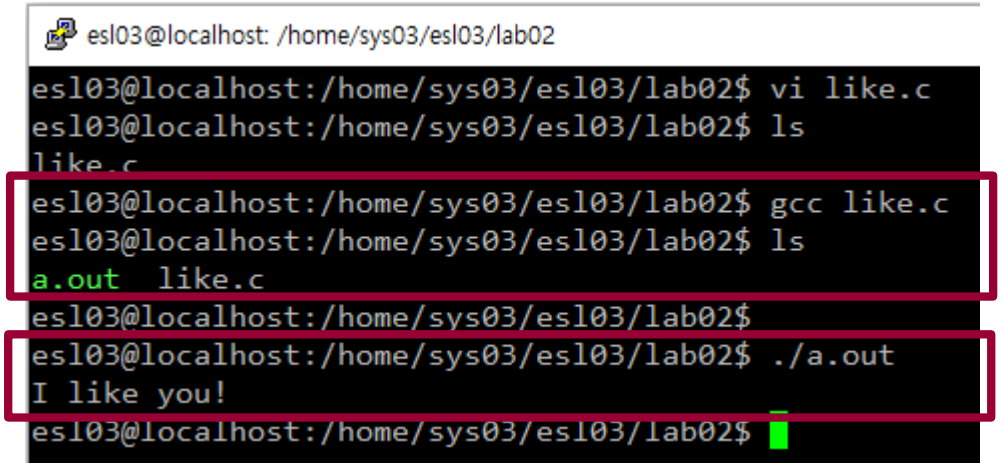
- vi편집기에서 소스코드 작성 후, 표준 모드에서 :wq 혹은 shift + zz 명령을 통해 저장하고 종료. (1주차 실습자료 참고)

CNU Embedded System Laboratory

GCC 컴파일 과정

2. GCC를 이용하여 소스코드 컴파일

- 사용법: gcc [옵션] [소스파일명]
  - 옵션을 따로 지정하지 않으면 defaul로 a.out이라는 이름의 실행파일이 생성됨.



1. gcc를 사용하여 컴파일
2. 컴파일된 파일의 실행
   - ./[파일명] 을 통해 실행

CNU Embedded System Laboratory

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
  - 결과 파일(output)의 이름을 지정
  - 사용법: gcc -o [결과파일] [소스파일명]
    - Ex) gcc -o like like.c
    - Ex) gcc -c -o memo.o memo.c

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

CNU Embedded System Laboratory

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, -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]$
```

CNU Embedded System Laboratory

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.

CNU Embedded System Laboratory

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

[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]$
```

- **file1.c, file3.c, like.c를 –c 옵션을 사용해서 각각 오브젝트 파일로 변환**
- **생성된 *.o 파일 들을 –o 옵션을 사용해 실행 파일로 컴파일**

CNU Embedded System Laboratory

GCC 컴파일 과정 - 따라하기

- 아래의 소스를 작성하고 컴파일, 실행

ex01.c

sys00@2019sp: ~/TestDir

```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);
}
~
```

sub.c

sys00@2019sp: ~/TestDir

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

}
~
```

add.c

sys00@2019sp: ~/TestDir

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

CNU Embedded System Laboratory

GCC 컴파일 과정 - 따라하기

- 아래의 소스를 작성하고 컴파일, 실행
  - 컴파일 방법 1)
    - 1번 방법을 사용하면 gcc -o ex01.c과 같은 실수를 한 경우, ex01.c 코드가 덮어쓰기 되어 날라감.

```
sys00@2019sp:~/TestDir$ gcc -o test1 ex01.c add.c sub.c
ex01.c: In function 'main':
ex01.c:7:12: warning: implicit declaration of function 'funcAdd' [-Wimplicit-function-declaration]
  nResult = funcAdd(nAlpha, nBeta);
            ^
ex01.c:9:13: warning: implicit declaration of function 'funcSub' [-Wimplicit-function-declaration]
  nResult   = funcSub(nAlpha, nBeta);
              ^
```

  - 컴파일 방법 2)
    - 코드마다 목적파일을 만들어줘야 해서 불편함.

```
sys00@2019sp:~/TestDir$ gcc -c add.c
sys00@2019sp:~/TestDir$ gcc -c sub.c
sys00@2019sp:~/TestDir$ gcc -c ex01.c
ex01.c: In function 'main':
ex01.c:7:12: warning: implicit declaration of function 'funcAdd' [-Wimplicit-function-declaration]
  nResult = funcAdd(nAlpha, nBeta);
            ^
ex01.c:9:13: warning: implicit declaration of function 'funcSub' [-Wimplicit-function-declaration]
  nResult   = funcSub(nAlpha, nBeta);
              ^
sys00@2019sp:~/TestDir$ gcc -o test2 add.o sub.o ex01.o
```

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

CNU Embedded System Laboratory

Make file 작성법

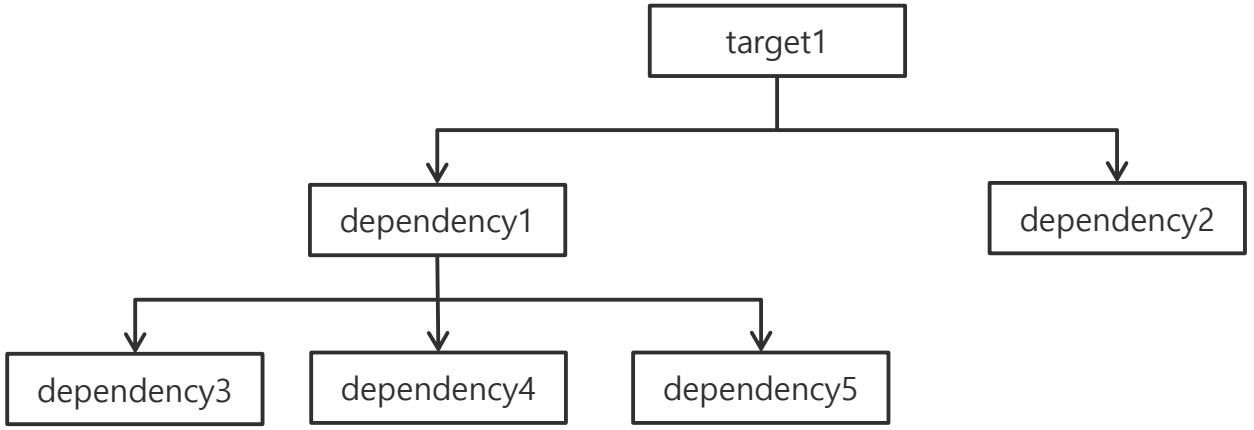
❖ 기본 구조

```
CC = gcc

target1 : dependency1 dependenty2
        command1
        command2

dependency1 : dependency3 dependency4 dependency5
        command3
        command4
```

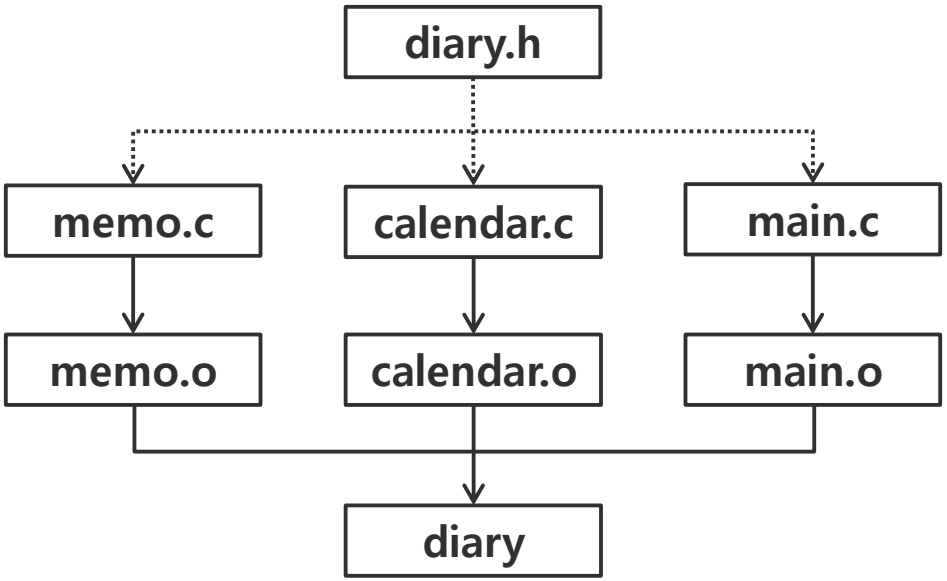
주의) command는 항상 tab키를 사용해서 작성해야 한다



CNU Embedded System Laboratory

Make file 작성법 - 따라하기

- Makefile의 작성법 이해를 돕기 위한 예제
  - /home/sys02/sys02/lab02.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.

CNU Embedded System Laboratory

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.

CNU Embedded System Laboratory

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
```

CNU Embedded System Laboratory

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 $@ $^
```

CNU Embedded System Laboratory

Make file 작성법

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

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

CNU Embedded System Laboratory

Make file 작성법 - 따라하기

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

```
1  .SUFFIXES : .c .o
2
3  CC = gcc
4  CFLAGS = -W -Wall
5  TARGET = diary
6
7  OBJ = add.o calendar.o div.o  main.o mul.o sub.o memo.o
8
9  all : $(TARGET)
10
11 $(TARGET) : $(OBJ)
12     $(CC) $(CFLAGS) -o $@ $^
```

CNU Embedded System Laboratory

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.

```
1  .SUFFIXES : .c .o
2
3  CC = gcc
4  CFLAGS = -W -Wall
5  TARGET = diary
6
7  OBJ = add.o calendar.o div.o  main.o mul.o sub.o memo.o
8
9  all : $(TARGET)
10
11 $(TARGET) : $(OBJ)
12     $(CC) $(CFLAGS) -o $@ $^
13
14 clean :
15     rm -rf *.o
16     rm -rf *.out
```

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

CNU Embedded System Laboratory

실습 1 : gcc 컴파일 실습

- 곱셈과 나눗셈 연산을 하는 코드를 아래의 조건에 맞게 작성하고 컴파일 후 실행
- 함수 위에 학번 이름을 주석으로 표기 (//학번 이름)
  - 조건1: 곱셈의 기능을 하는 함수를 mul.c에 작성
  - 조건2: 나눗셈의 기능을 하는 함수를 div.c에 작성
  - 조건3: main 함수는 ex01.c에 작성하고 내용은 아래의 실행 결과 처럼 나오도록 작성
  - 조건4: 컴파일 한 후 실행하여 다음과 같이 출력되어야 함.
    (hint : gcc -o ex01.out mul.c div.c ex01.c)

```
1
2 // 20191111 홍 길 동
3
4 int funcMul(int a,
```

- 실행 결과

```
sys01@systemprogramming17:~/test/codes/ex01_1$ ls
div.c  ex01.c  ex01.out  mul.c
sys01@systemprogramming17:~/test/codes/ex01_1$ ./ex01.out
6 * 2 = 12
6 / 2 = 3
sys01@systemprogramming17:~/test/codes/ex01_1$
```

CNU Embedded System Laboratory

실습 2 : make 파일 만들기!

- 다음 조건들을 만족하는 코드를 작성하고 Makefile을 만들어서 컴파일 후 실행 파일을 실행시켜 결과를 확인
  - 조건 1: /home/sys02/sys02/lab02.tar.gz 를 자신의 홈 디렉토리로 복사
    - lab02.tar.gz 에 main.c, add.c, sub.c, mul.c div.c, memo.c, calendar.c diary.h 이 포함되어 있음
  - 조건 2: main.c / add.c / sub.c / mul.c / div.c 소스코드 작성
  - 조건 3: Makefile을 이용하여 컴파일 (※ 21, 23 slide 참고 - 두가지 중 하나 이용)

    (hint : gcc -o diary main.c add.c sub.c mul.c div.c memo.c calendar.c 와 같은 기능)
  - 조건 4: ./ 명령어 이용하여 실행
  - 조건 5: Makefile에 clean 명령어를 작성하여 make clean입력으로 *.o 파일과 *.out 파일을 삭제

조건 4
결과화면 예시=>

```
sys00@2018-sp: ~/weak02/practice02_2
sys00@2018-sp:~/weak02/practice02_2$ ls
add.c  calendar.c  diary.h    div.c  main.c  Makefile  memo.o  mul.o  sub.o
add.o  calendar.o  diary.out  div.o  main.o  memo.c    mul.c   sub.c
sys00@2018-sp:~/weak02/practice02_2$ ./diary.out
 5 + 3 = 8
 5 - 3 = 2
 5 * 3 = 15
 5 / 3 = 1
function memo.
function calendar.
sys00@2018-sp:~/weak02/practice02_2$
```

CNU Embedded System Laboratory

실습 2

- 파일 종속 구조