시스템 프로그래밍

- GCC & make -

2014. 09. 18.

박시형

sihyeong@cnu.ac.kr

Embedded System Lab. Computer Engineering Dept.

Chungnam National University

개요

실습 명

- GCC와 make를 통한 컴파일

목표

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

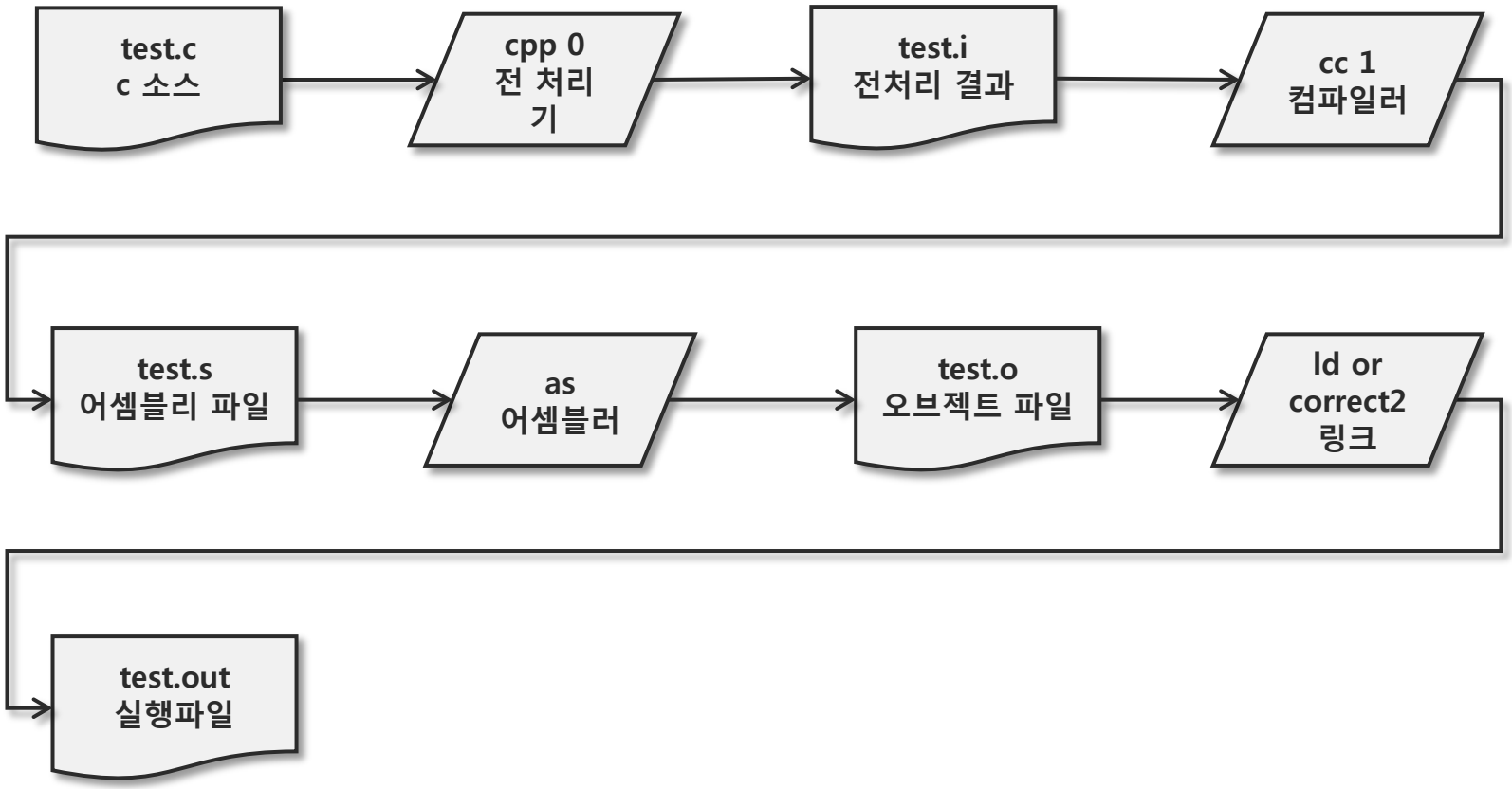
주제

- GCC
- make

GCC 란?

- **GCC (GNU Compiler Collection)**
  - GNU(GNU is Not UNIX) 프로젝트의 일환으로 개발되어 널리 쓰이고 있는 컴파일러
  - GCC는 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 역할만을 담당

GCC 컴파일 과정


test.c
c 소스
cpp 0
전 처리
기
test.i
전처리 결과
cc 1
컴파일러
test.s
어셈블리 파일
as
어셈블러
test.o
오브젝트 파일
ld or
correct2
링크
test.out
실행파일

GCC 컴파일 해보기

❖ 소스 코드 작성

- 자신의 홈 디렉터리(~)에서 vi를 이용해 소스코드를 작성한다.
  - 소스파일명 : **like.c**

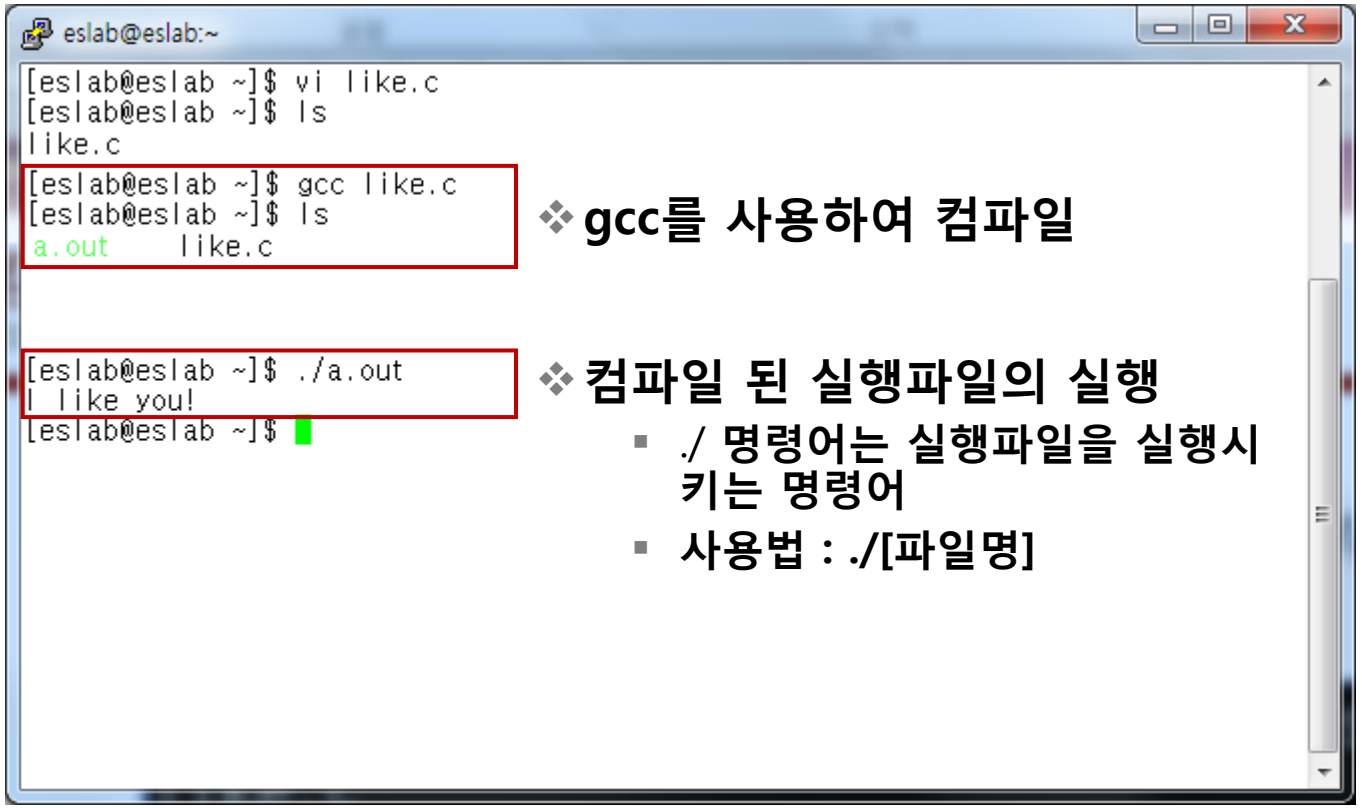
```
#include <stdio.h>

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

  - vi 편집기에서 소스코드 작성 후, 표준모드에서 :wq 또는 shift + zz 명령을 통해 저장하고 종료. (1주차 실습자료를 참고)

GCC 컴파일 해보기

❖ GCC를 이용하여 소스코드 컴파일

- 사용법 : gcc [옵션] [소스파일명]
  - 옵션을 지정하지 않으면 a.out 이라는 이름으로 실행파일이 생성됨



❖ gcc를 사용하여 컴파일

❖ 컴파일 된 실행파일의 실행

- ./ 명령어는 실행파일을 실행시키는 명령어
- 사용법 : ./[파일명]

GCC 컴파일 옵션

-c 옵션

- 컴파일 과정 중, 링크를 하지 않고 오브젝트 파일(*.o)만 생성
- 사용법 : gcc -c [소스파일명]
  - ※오브젝트파일 : 컴파일러나 어셈블러가 소스파일을 컴파일 또는 어셈블 한 결과 생성되는 파일

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

❖ 컴파일 한 결과 like.o가 생성

-o 옵션

- 실행파일(*.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
```

❖ 컴파일 한 결과 사용자가 지정한 이름의 실행파일(like)이 생성

GCC 컴파일 옵션

-v --save-temp 옵션

- 컴파일에 사용되는 모든 파일을 지우지 않고 컴파일
- 사용법 : gcc -v --save-temp [소스파일명]
  - -v : 컴파일 전 과정을 화면에 출력
  - --save-temp : 컴파일 과정에서 생성되는 모든 파일을 지우지 않음(전처리 결과, 어셈블리 파일, 오브젝트 파일, 실행 파일)

-W -Wall 옵션

- 컴파일 과정에서 발생하는 에러정보 표시
- 사용법 : gcc -W -Wall [소스파일명]
  - -W : 합법적이지만 모호한 코딩에 대한 부가적 정보제공
  - -Wall : 모든 모호한 문법에 대한 경고 메시지를 출력
    - 두 옵션을 함께 사용하면 아주 사소한 모호성에서도 경고가 발생

```
[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 해보기

❖ 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]$
```

❖ 필요한 소스파일만 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 (.o 파일만 만듦)
  - 소스코드가 매우 많은 파일로 분리되어 있는 경우 효과적임.

GCC 컴파일 해보기

❖ 필요한 소스파일만 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**

GCC 컴파일 실습 - 따라 하기

❖ 아래의 소스를 작성하고 컴파일, 실행

❖ ex01.c

```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

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

❖ add.c

```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 (이전 슬라이드와 유사하게 진행)
  - 조건 1 : 곱셈의 기능을 하는 함수를 mul.c 에 작성
  - 조건 2 : 나눗셈의 기능을 하는 함수를 div.c 에 작성
  - 조건 3 : main 함수는 ex01_2.c 에 작성하고 내용은 아래의 실행 결과처럼 나오도록 작성
  - 조건 4 : 컴파일 한 다음 실행하여 아래와 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 실행 파일을 실행
  - 조건 1 : /home/syspro/gcclab.tar.gz 를 자신의 홈 디렉터리(~)에 복사한 다음 압축을 해제
  - 조건 2 : gcclab 디렉터리에 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 ?

- 프로그램의 일부가 수정되었을 때 의존 관계에 따라 필요한 부분만 다시 컴파일, 링크하는 작업을 자동으로 수행해주는 프로그램

- 하나의 프로그램을 여러 부분으로 나누어서 개발할 때, 각 부분들 사이의 의존 관계를 나타냄

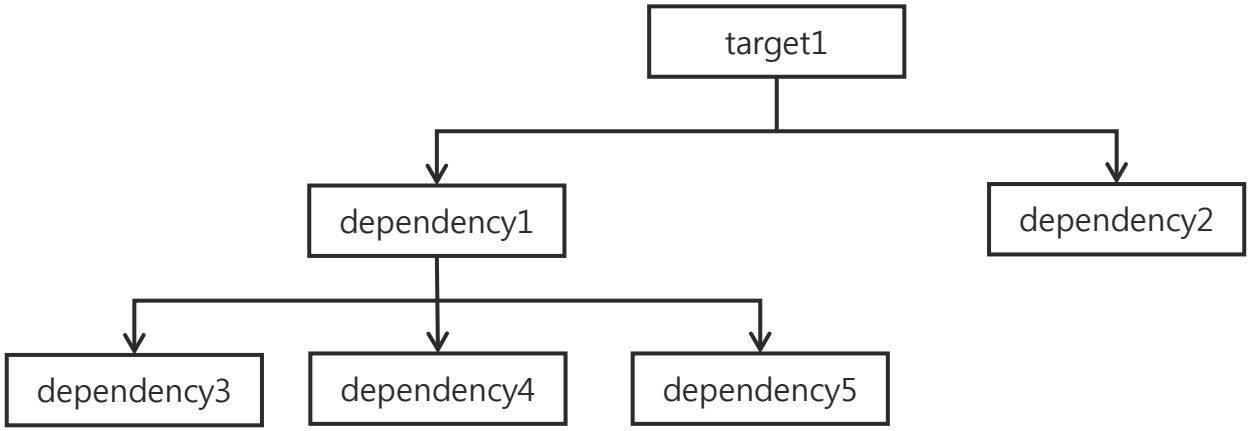
- 복잡하고 방대한 프로그램을 개발할 때 생산성을 높이는데 도움을 주는 도구
  - 단순 반복 잡업과 재 작성을 최소화 할 수 있다.

Makefile 작성법

❖ 기본 구조

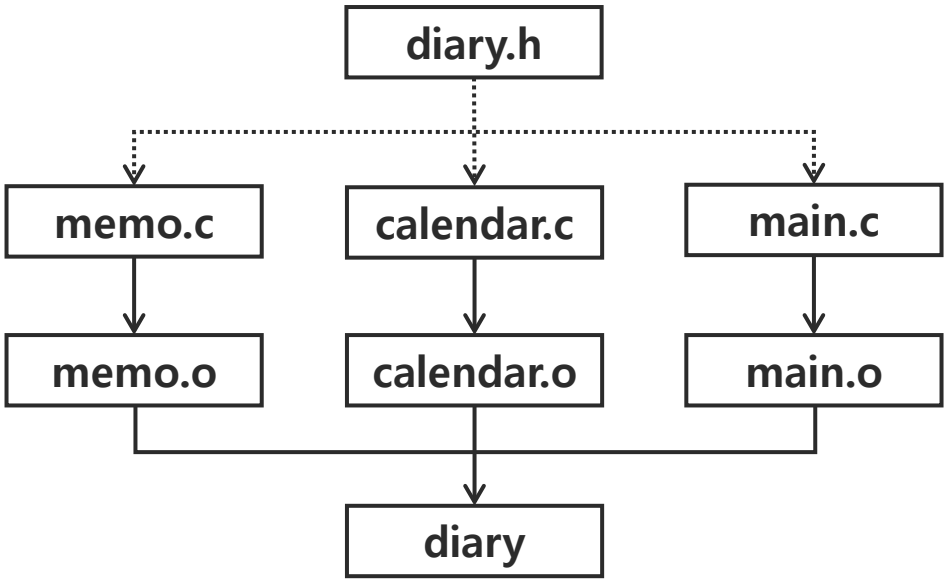
```
CC = gcc

target1 : dependency1 dependenty2
        command1
        command2

dependency1 : dependency3 dependency4 dependency5
        command3
        command4
```

주의) command는 항상 tab키를 사용해서 작성해야 한다



Makefile 작성법 - 따라 하기

- Makefile의 작성법의 이해를 돕기 위한 예제
  - /home/syspro/maketest1.tar.gz 를 자신의 홈 디렉터리로 복사 한 후, 압축 해제
  - Maketest 디렉터리에 들어있는 소스를 컴파일 하여 실행 파일 "diary"를 만든다고 할 때, 각 파일들의 종속 구조는 아래와 같다.



  - 다음 페이지의 Makefile은 해당 종속 구조에 맞추어 작성된 것이다.

Makefile 작성법 - 따라 하기

❖ Makefile (기본 작성법)

```
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 작성 한 후, "make" 명령을 입력하면 종속 관계를 만족하도록, 각 오브젝트에 대한 컴파일 과정을 수행한 후, 최종적으로 diary 실행 파일을 생성한다.

Makefile 작성법 - 따라 하기

❖ 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 않음을 볼 수 있다.

```
[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 수정해본다.
- 수정 후, make 명령을 수행해보면 컴파일이 진행됨을 볼 수 있다.

Makefile 작성법 - 따라 하기

- **Makefile (매크로를 이용한 작성법)**
  - 매크로를 사용하여 Makefile을 작성하는 방법이다. 아래와 같이 수정하여 다시 make 해본다.
    - 매크로는 '사용자 정의 변수'에 특정한 문자열을 정의하고 표현하는 것
    - 매크로는 Makefile 내에 정의된 문자열로 치환되어 사용될 수 있다.
      - 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
```

Makefile 작성법 - 따라 하기

- **Makefile (자동 매크로 리스트를 이용한 작성법)**
  - **자동 매크로 리스트를 사용하여 작성하는 방법이다. 아래와 같이 작성 후 다시 컴파일을 실행 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 $@ $^
```

  - **자동 매크로 리스트에 대한 내용은 다음 장을 참조한다.**

참조 - Makefile 자동 매크로

- 내부적으로 정의되어 있는 매크로
- Makefile 의 작성을 더 간단하고 편리하게 만들기 위한 매크로

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

Makefile 작성법 - 따라 하기

- **Makefile (.SUFFIXES 매크로를 이용한 작성법)**
  - **.SUFFIXES 라는 매크로를 이용하여 작성하는 방법이다. 아래와 같이 다시 작성 후 실행**
    - **make 가 중요하게 여길 확장자 리스트를 등록해 준다.**
    - **사용법 : .SUFFIXES : .o .c**
      - **미리 정의된 .c (소스파일)를 컴파일 해서, .o (목적파일)을 만들어내는 루틴이 자동적으로 동작하도록 되어있다.**

```
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 $@ $^
```

Makefile 작성법 - 따라 하기

Makefile (clean 명령 부분)

- clean 명령어를 이용하여 불필요한 파일을 삭제할 수 있다. 아래와 같이 입력하고 "make clean" 명령어를 치면 삭제됨.

```
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

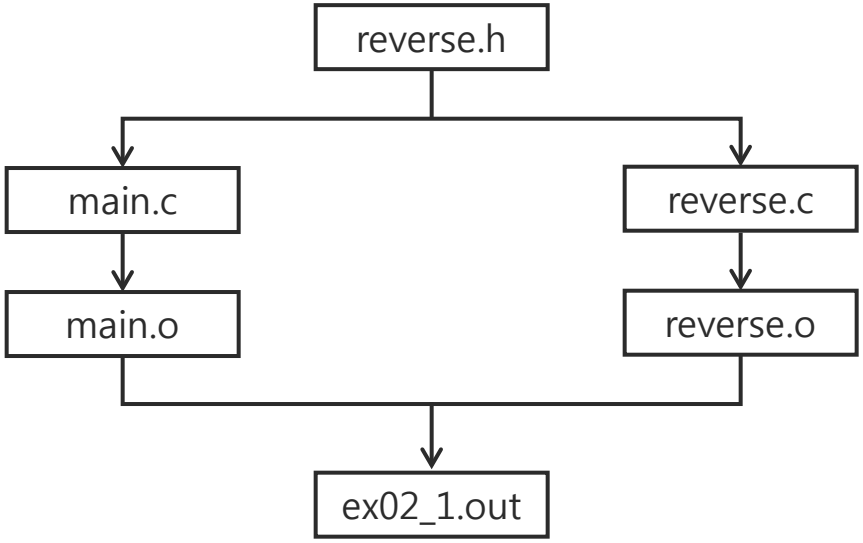
  - 조건 1 : 자신의 학번을 입력 받아 거꾸로 출력하는 프로그램을 작성
  - 조건 2 : reverse.c 에는 입력 받은 학번을 거꾸로 출력하는 함수 작성
  - 조건 3 : reverse 함수를 main에서 호출하여 아래의 실행 결과가 나오도록 작성.
  - 조건 4 : Makefile의 네 가지 작성법(기본, 매크로, 자동매크로, SUFFIXES) 을 모두 이용하여 컴파일 (각각의 Makefile 캡쳐)
  - 조건 5 : 컴파일 과정은 동일하므로 한번만 캡쳐
  - 조건 6 : 파일 종속 구조를 다음 장의 그림처럼 만들고 그 파일 구조대로 Makefile을 작성

  - 파일 종속 구조 및 실행 결과는 다음 페이지를 참고

실습 2

실습 2-1

- 파일 종속 구조



- 실행 결과

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

  - 학번을 입력하면, 학번을 거꾸로 출력

실습 2

실습 2-2

- 다음 조건들을 만족하는 코드를 작성하고 Makefile을 만들어서 컴파일 한 다음 실행파일을 실행시켜 결과를 확인

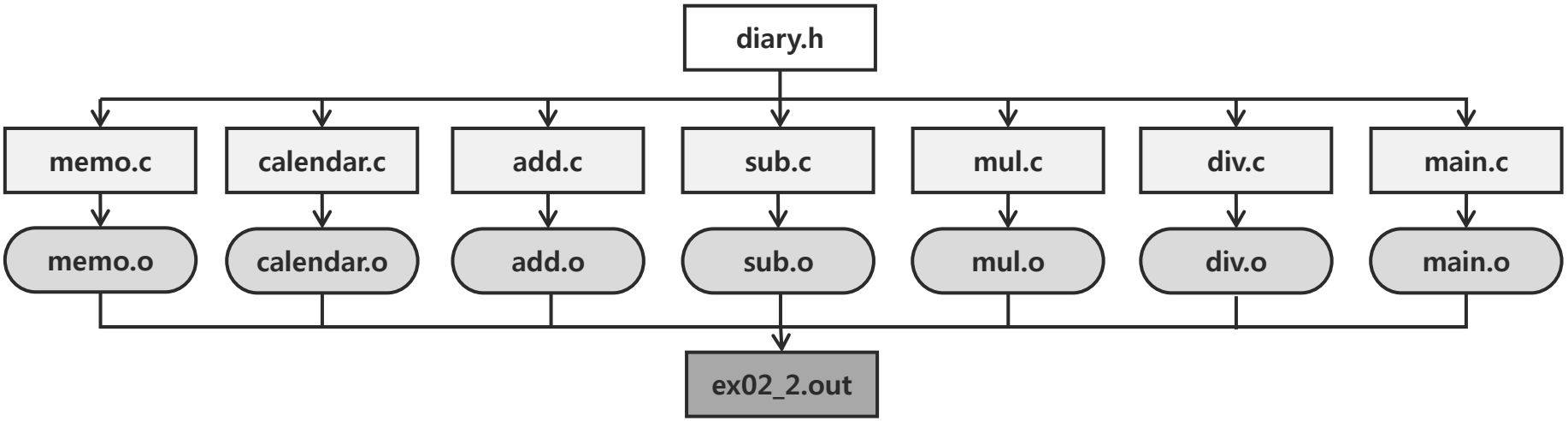
  - 조건 1 : /home/syspro/example02_2.tar.gz 를 자신의 홈 디렉터리로 복사
  - 조건 2 : main을 수정하여, 사칙연산과 memo, calendar 코드를 수행
    - example02_2.tar.gz 에 add.c, sub.c, div.c, memo.c, calendar.c가 포함되어 있음.
  - 조건 3 : Makefile을 네 가지 작성법(기본, 매크로, 자동매크로, SUFFIXES)을 모두 이용하여 컴파일 (각각의 Makefile 캡쳐)
  - 조건 4 : 컴파일 과정은 동일하므로 한번만 캡쳐

  - 파일 종속 구조와 실행 결과는 다음 페이지를 참고

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

- 따라 하기와 실습을 진행한 내용을 모두 보고서로 작성
- 이메일과 서면으로(프린트해서) 제출
  - sihyeong@cnu.ac.kr
  - 제목 양식 : [sys02]HW02_학번_이름
  - 파일 제목 : [sys02]HW02_학번_이름
  - 반드시 메일 제목과 파일 양식을 지켜야 함. (위반 시 감점)
  - 보고서는 제공된 양식 사용
- 자신이 실습한 내용을 증명할 것 (자신의 학번이 항상 보이도록 캡쳐)
- 제출 일자
  - 이메일 제출 : 2014년 09월 24일 23시 59분 59초
  - 서면 제출 : 2014년 09월 25일 수업시간