

보충 교재

make 사용법

make를 사용하는 목적

- ▶ 커다란 프로그램에서 다시 컴파일해야 하는 부분을 자동적으로 판단
- ▶ 그것들을 컴파일하는 명령을 실행

make 문법

- ▶ 일반적인 파일 이름 : Makefile, makefile
- ▶ target, dependency, command로 이루어진 기본적인 규칙(rule)들을 계속적으로 나열

targetList : dependencyList

commandList

- ▶ target 부분 : command가 수행이 되어서 나올 결과 파일(obj 파일 혹은 실행 파일)을 지정
- ▶ 명령이 실행되는 경우
 - ▶ dependency부분에 정의된 파일의 내용이 바뀌었을 때
 - ▶ 목표 부분에 해당하는 파일이 없을 때
- ▶ commandList
 - ▶ 셸에서 쓸 수 있는 모든 명령어들과 bash에 기반한 쉘 스크립트로 구성
 - ▶ 반드시 TAB 글자로 시작
- ▶ 주석은 #으로 시작

make 실행

% make [-f makeFileName]

- ▶ -f가 없는 경우 default file 이용
- ▶ Default file : Makefile, makefile
- ▶ Makefile과 makefile이 같이 있는 경우 Makefile이 우선

예제 #1 (1)

- ▶ 다음과 같은 4개의 파일(test.h, main.c, test1.c, test2.c) 생성

```
/* file – main.c */  
#include <stdio.h>  
#include "test.h"  
int main()  
{  
    test1();  
    test2();  
    printf("Hi !\n");  
    return 0;  
}
```

예제 #1 (2)

```
/* file - test.h */  
void test1(void);  
void test2(void);
```

```
/* file - test1.c */  
void test1(void)  
{  
}
```

```
/* file - test2.c */  
void test2(void)  
{  
}
```

예제 #1 (3)

▶ gcc로 컴파일

```
% gcc -c main.c
```

```
% gcc -c test1.c
```

```
% gcc -c test2.c
```

```
% gcc -o test main.o test1.o test2.o
```

예제 #1 (4)

- ▶ Makefile 작성

```
% vi Makefile
```

```
test : main.o test1.o test2.o
    gcc -o test main.o test1.o test2.o
main.o : test.h main.c
    gcc -c main.c
test1.o : test.h test1.c
    gcc -c test1.c
test2.o: test.h test2.c
    gcc -c test2.c
```

- ▶ make 실행 – 명령 자동 실행

```
% make
```

```
gcc -c main.c
gcc -c test1.c
gcc -c test2.c
gcc -o test main.o test1.o test2.o
```


예제 #1 (5)

- ▶ test2.c 수정

```
% vi test2.c
/* file - test2.c */
#include <stdio.h>
void test2(void)
{
    printf("test2\n");
}
```

- ▶ make 실행 (test2.c만 다시 컴파일되고 실행 파일 생성)

```
gcc -c test2.c
```

```
gcc -o test main.o test1.o test2.o
```

- ▶ [Tip] touch : 파일의 날짜 갱신

예제 #2 매크로 변수 이용

▶ Makefile 생성

```
% vi Makefile
```

```
OBJECTS = main.o test1.o test2.o
```

```
test : $(OBJECTS)
```

```
    gcc -o test $(OBJECTS)
```

```
main.o : test.h main.c
```

```
    gcc -c main.c
```

```
test1.o : test.h test1.c
```

```
    gcc -c test1.c
```

```
test2.o: test.h test2.c
```

```
    gcc -c test2.c
```

```
% make
```

예제 #3 레이블 이용

▶ Makefile 생성

```
% vi Makefile
```

```
OBJECTS = main.o test1.o test2.o
```

```
test : $(OBJECTS)
```

```
    gcc -o test $(OBJECTS)
```

```
main.o : test.h main.c
```

```
    gcc -c main.c
```

```
test1.o : test.h test1.c
```

```
    gcc -c test1.c
```

```
test2.o: test.h test2.c
```

```
    gcc -c test2.c
```

```
clean :
```

```
    rm $(OBJECTS)
```

```
% make clean
```

보통 이용하는 매크로 변수

- ▶ `CC = gcc` # C Compiler
- ▶ `CFLAGS`
플래그 # 컴파일시 이용하는 옵션
- ▶ `SRCS` # 프로그램 원본 파일
- ▶ `OBJS` # obj 파일들
- ▶ `LIBS` # library 파일들
- ▶ `LIBDIRS` # library 파일들의 폴더

예제 #4 매크로 변수 이용

▶ Makefile 생성

```
% vi Makefile
```

```
OBJECTS = main.o test1.o test2.o
```

```
SRCS = main.c test1.c test2.c
```

```
CC = gcc
```

```
CFLAGS = -g
```

```
TARGET = test
```

```
$(TARGET) : $(OBJECTS)
```

```
$(CC) -o $(TARGET) $(OBJECTS)
```

```
main.o : test.h main.c
```

```
test1.o : test.h test1.c
```

```
test2.o: test.h test2.c
```

```
% make
```

예제 #5 Implicit rule

▶ Makefile 생성

```
% vi Makefile
```

```
.c.o :
```

```
    gcc -o $* $<
```

```
clean :
```

```
    rm *.o $@
```

- ▶ \$< : 현재의 목표 파일보다 더 최근에 갱신된 파일 이름
- ▶ \$* : 확장자가 없는 현재의 목표 파일(Target)
- ▶ \$@ : 현재의 목표 파일(Target)
- ▶ % make main.o

예제 #6 Multiple Target

▶ Makefile 생성

```
% vi Makefile
```

```
.SUFFIXES : .c .o
```

```
CC = gcc
```

```
CFLAGS = -g
```

```
OBJS1 = main.o test1.o
```

```
OBJS2 = main.o test2.o
```

```
SRCS = $(OBJS1:.o=.c) $(OBJS2:.o=.c)
```

```
all : test1 test2
```

```
test1 : $(OBJS1)
```

```
$(CC) -o test1 $(OBJS1)
```

```
test2 : $(OBJS2)
```

```
$(CC) -o test2 $(OBJS2)
```

```
% make all
```

과제

- ▶ 예제 1 ~ 6까지 직접 실행
- ▶ `make_hw` 폴더 생성
- ▶ C 파일 4개 이상, h 파일 2개 이상 포함된 프로그램 작성
- ▶ Makefile 작성