Makefile Tutorials

서강대학교 컴퓨터공학과

2022. 03. 25

What is Makefile?

- 일반적으로 리눅스에서는 두가지의 컴파일 방법을 이용한다.
  1. gcc : gcc 등의 직접 command를 사용해 컴파일 하거나,
  2. make : make를 이용한 Makefile 컴파일

- make는 여러 파일을 한번에 컴파일 해줄 때 유용하다.
- 따라서, 오픈소스 (컴파일할 파일이 많음) 에서는 gcc로 직접 하나하나 컴파일 하기 보다는 make를 이용하여 한번에 모든 파일을 컴파일 할 수 있도록 한다.

- Makefile은 text file로 make 라는 프로그램을 이용할 수 있도록 만들어준다. (슬라이드 4 예시 참조)
- Makefile은 따로 확장자를 지정하지 않아도 된다.

1. 기본 구성 (이번 과제)

Source Folder
|
|_ file1.c
|_ file2.c
|_ Makefile

명령어 make
------>

Source Folder
|
|_ file1.c
|_ file2.c
|_ file1
|_ file2
|_ Makefile

2. 헤더파일 포함 구성

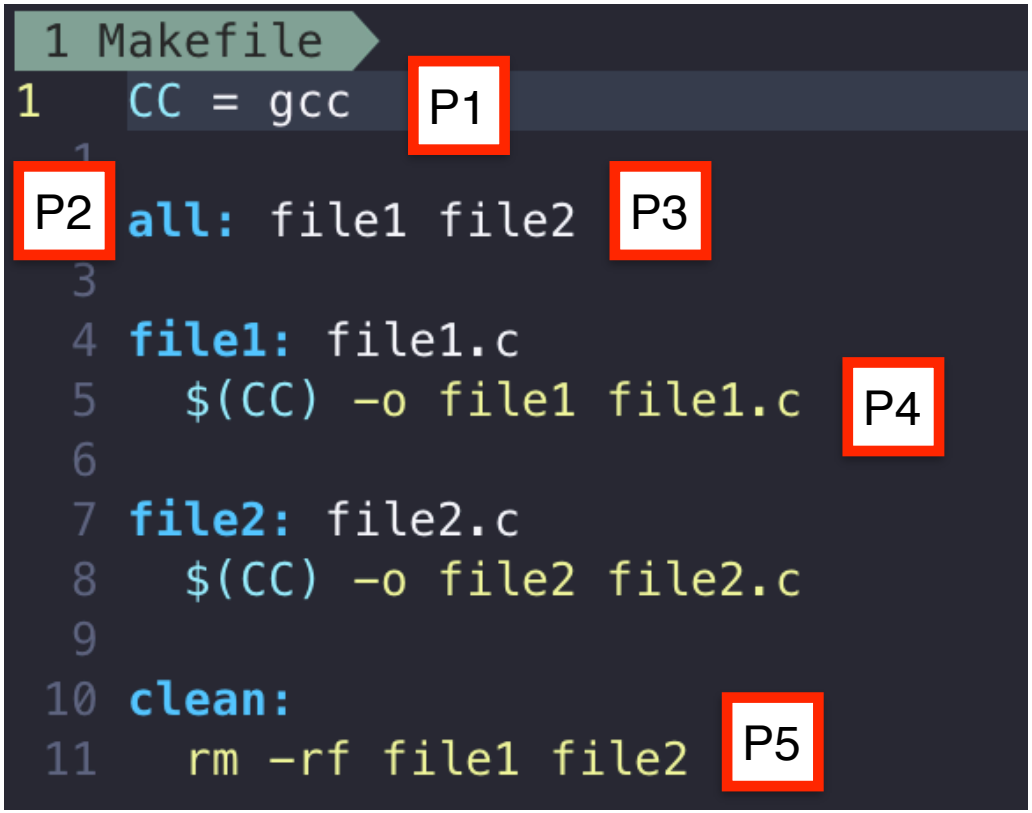
Source Folder
|
|_ header.h
|_ main.c
|_ file1.c
|_ Makefile

명령어 make
------>

Source Folder
|
|_ header.h
|_ main.c
|_ file1.c
|_ main
|_ file1.o
|_ Makefile

1. 기본 구성

- 기본 구성의 경우 두 파일의 이진 파일을 각각 만들어 내는 것을 목표로 한다.
- 각각의 이진파일은 gcc로 컴파일 한 것과 마찬가지로, ./file을 통해서 실행이 가능하다.
- 두 개의 이진 파일은 다음과 같은 Makefile을 만들고, 해당 폴더에서 make를 입력하여 만들 수 있다.



1. 기본 구성

P1

Macro : Makefile의 변수 역할을 해준다. 해당 예시에서는 CC=gcc로 되어있고 이를 아래에서 $(CC) 로 불러올 수 있다.
따라서, $(CC) -o (execution file name) (c file) 은 gcc로 c파일을 컴파일 해준다는 것을 의미한다.

P2

Target : 실행파일, Object 파일, 라이브러리 등 목적 규칙을 정의한다. 기본 구성에서는 해당 부분을 한꺼번에 컴파일 하기위한 수단으로 이용하는데, 즉 현재 all을 예시에서 제거하면 file1만 컴파일이 된 후 끝난다.
이는 Makefile이 첫번째 target을 실행하고 이것과 종속되는 다른 target이 없으면 make를 종료하기 때문이다.

P3

Dependency:

Target을 만들 때, 의존성을 규정한다. 컴파일을 이미 한 후 수정한 파일이 있을 때, 의존성을 규정한 파일들 중에 수정한 파일이 있으면 Target을 다시 만든다.

P4 Recipe :

Target을 만들기 위한 실행 파일이다. 해당 실행 규칙에 의해 Target이 생성되는데, 이 부분은 리눅스 명령어를 이용해서 작성한다.
예시에서는 gcc로 c파일을 컴파일한다.

P5 Dummy Target :

"make clean"처럼 실행할 수 있는 target으로, 명령을 실행 시 생성되는 Target 파일들을 제거해주는 용도로 사용된다. make를 수행 전 안전하게 파일들을 제거하고 다시 새롭게 하기 위한 용도로 사용된다.

주의사항

Makefile에서 target 아래의 Recipe (리눅스 명령어)를 적을 때 indentation(tab)을 꼭 넣어서 라인을 구별해주어야 한다. (들여쓰기)

2. 헤더파일 포함 구성

- 헤더파일 포함 구성의 경우 코드 진행은 main.c가 header를 통해서 file1.c 에 있는 함수를 불러올 때 주로 이용된다.
- 이 경우 file1은 object file로 (확장자 .o)로 컴파일 되며, main은 해당 object를 이용하는 실행 파일(이진 파일)로 만들어지게 된다. 실행할 때는 ./main을 통해서 실행한다.
- 이 또한 make 명령어를 이용해서 만든다. 예시는 아래와 같다.

Makefile

```
CC = gcc

main: header.h main.c file1.o
	$(CC) -o main main.c file1.o

file1.o: header.h file1.c
	$(CC) -c file1.c

clean:
	rm -rf main file1.o
```

2. 헤더파일 포함 구성

- 기본 구성과 구조는 일치하나, all이 들어가지 않았다.
- 이는 이진 파일을 두 개를 만드는 것이 아닌 main 하나만 만들면 되기 때문이다. 또한, main에 file1.o라는 object file이 종속되어 있으므로 바로 아래 target인 file1.o의 recipe를 실행시키게 된다.
- Dummy Target 또한 "make clean" 명령어를 이용하여 파일들을 없앨 수 있도록 한다.
- header는 해당 파일에서 이용하게 된다면 target에 추가할 수 있도록 한다.