- Make

파일간의 종속적인 관계를 파악하여, Makefile에 기술된 대로 컴파일러에 순차적으로 명령을 실행하는 것

Make 사용의 장점

1) 각 파일에 대한 반복적 명령의 자동화
2) 프로그램 간의 종속관계를 쉽게 파악할 수 있고, 관리가 용이하다.
3) 단순 반복과 작성을 최소화한다.

- Makefile

Makefile의 기본 구조

1) 목적파일(Target): 명령어가 수행되어 나온 결과를 저장하는 파일
2) 의존성(Dependency): 목적파일을 만들기 위해 필요한 재료
3) 명령어(Command): 실행되어야 할 명령들
4) 매크로(macro): 코드를 단순화하기 위한 macro

Makefile의 코드는 다음과 같은 형태로 나타난다. Command는 반드시 tap으로 indent되어 있어야 한다.

```
매크로 설정
target1 : dependency1 dependency2
        command1
        command2
target2 : dependency3 dependency 4
        command3
        command4
```

예를 들어, Hello.c파일과 World.c파일을 compile해, HelloWorld 실행파일을 만들고 싶다면,

```
HelloWorld : Hello.o World.o
        gcc -o HelloWorld Hello.o World.o
Hello.o : Hello.c
        gcc -c -o Hello.o Hello.c
World.o : World.c
        gcc -c -o World.o World.c
```

의 형태로 나타낼 수 있다.

코드를 단순화하기 위해 매크로를 사용하면,

```
CC = gcc
TARGET = HelloWorld

$(TARGET): Hello.o World.o
	$(CC) -o $(TARGET) Hello.o World.o
Hello.o : Hello.c
	$(CC) -c -o Hello.o Hello.c
World.o : World.c
	$(CC) -c -o World.o World.c
```

의 형태로 나타낼 수 있다.

내부 매크로를 사용하여, 조금 더 코드를 단순화하면,

```
CC = gcc
TARGET = HelloWorld
OBJECT = Hello.o World.o

$(TARGET): $(OBJECT)
	$(CC) -o $@ $^
```

$@는 현재 target파일을 의미하고, $^는 현재 dependency 항목의 리스트를 의미한다.

object파일이 없다면, object파일과 이름이 같은 c파일을 찾아 compile을 하여, object파일을 생성한다.

또한 makefile은 incremental build를 지원하는데, 빌드를 할 때, 변경된 대상들이 있는 부분만 추려서 build를 수행하는 것이다.

이렇게 makefile이 위치한 directory에서

```
make
```

명령어를 실행하면, makefile에 기술된 내용에 따라 실행파일을 만들 수 있고,

```
make World.o
```

위와 같이 target을 명시해주면, makefile의 해당 target만 선별적으로 build를 실행하게 된다.

- Clean

Makefile은 빌드의 대상뿐만 아니라, 프로젝트에서 요긴하게 사용할 수 있는 매크로를 지정해서 사용할 수도 있다. 이렇게 쓰는 대표적이고, 또 널리 사용하는 매크로가 바로 clean이다. Clean 매크로는 빌드 결과물과 중간 부산물들을 모두 삭제하여 깨끗한 상태에서 다시 빌드할 수 있는 환경을 만들어 준다.

Clean build는

```
clean:
        rm *.o
        rm $(TARGET)
```

을 makefile에 추가해주면 된다.

clean build의 실행은

```
make clean
```

처럼 make 뒤에 명시해주면 된다.

- Dummy target

빌드하기 위한 규칙을 정의하지 않고, 단지 임의의 명령어를 실행시키기 위한 목적으로 허울뿐인 가상의 파일 이름을 빌려와서 타겟으로 한 것이다.

위의 clean도 dummy target에 속한다.

하지만 dummy target은 만약 같은 이름의 파일이 directory내에 존재한다면, make가 이미 build가 끝난 파일이라고 생각해, dummy를 실행하지 않는 문제점이 있다.

따라서 .PHONY라는 것을 사용하는데 .PHONY로 정의해두면, dummy라는 것을 make에서 인식해, 실제 폴더 내에서 build 되었는지 여부를 생각하지않고 dummy 명령을 수행하게 된다.

자주 쓰이는 dummy target은

```
.PHONY: all build install run clean
```

등이 있다.