2. 간단한 Makefile

2.1 Makefile 의 내부 구조

*Makefile은 기본적으로 아래와 같이 목표(target), 의존 관계(dependency), 명령(command)의 세개로 이루어진 기분적인 규칙(rule)들이 계속적으로 나열되어 있다*고 봐도 무방하다. make가 지능적으로 파일을 갱신하는 것도 모두 이 간단한 규칙에 의하기 때문이다.

```
target ... : dependency ...
             command
             ...
             ...
```

여기서 목표(target) 부분은 명령(command)이 수행이 되어서 나온 결과 파일을 지정한다. 당연히 목적 파일(object file)이나 실행 파일이 될 것이다.

명령(command)부분에 정의된 명령들은 의존 관계(depenency)부분에 정의된 파일의 내용이 바뀌었거나, 목표 부분에 해당하는 파일이 없을 때 이곳에 정의된 것들이 차례대로 실행이 된다. 일반적으로 쉘에서 쓸 수 있는 모든 명령어들을 사용할 수가 있으며 bash에 기반한 쉘 스크립트도 지원한다.

=> 참고: 참고로 목표 부분에는 결과 파일만 올 수 있는 것이 아니고, 보통 make clean 에서와 같이 간단한 레이블(label) 기능을 제공하기도 한다.

=> 명령 부분은 꼭 *TAB* 글자로 시작해야 한다. 그냥 빈칸 등을 사용하면 make 실행 중에 에러가 난다. 명심하세요. make가 명령어인지 아닌지를 *TAB* 가지고 구별하기 때문이죠.

2.2 Makefile 예제

간단한 Makefile을 만들어 본다. 우리가 만들려고 하는 프로그램은 main.c read.c write.c로 구성되어 있고 모두 io.h라는 헤더 파일을 사용한다고 가정한다. (흐... 구성을 간단하게 합시다.) 이들을 각각 컴파일해서 test 라는 실행 파일을 생성시킨다.

```
% gcc -c main.c
% gcc -c read.c
% gcc -c write.c

% gcc -o test main.o read.o write.o
```

위의 방식은 make를 쓰지 않고 그냥 명령어를 주는 방식이다. 파일의 수가 작아서 오히려 더 간단하게 보일 수 있으나, 파일이 100개정도 된다고 가정하면... 아찔...

그리고, 아래는 위와 똑같은 일을 수행하는 Makefile의 내용이다.

Makefile예제 1

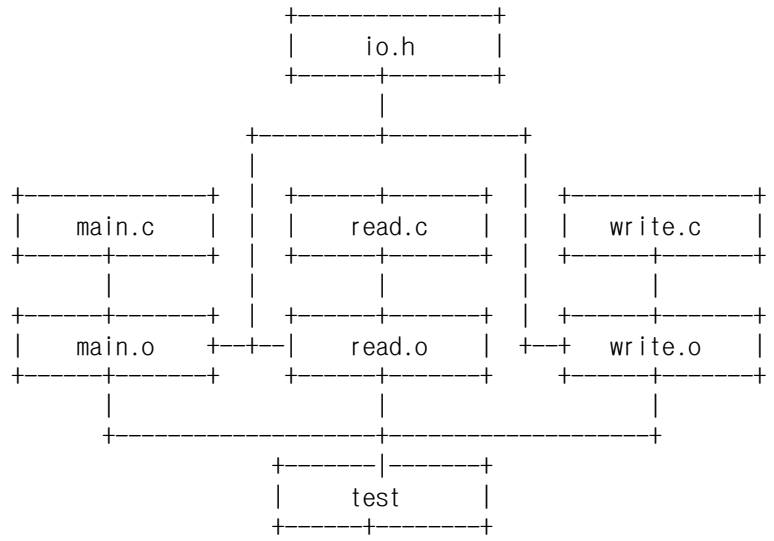
```
test : main.o read.o write.o
             gcc -o test main.o read.o write.o

main.o : io.h main.c
             gcc -c main.c
read.o : io.h read.c
```

```
                gcc -c read.c
write.o: io.h write.c
                gcc -c write.c
```

(대충 알아보시겠어요? 참 *TAB*문자 쓰는 것 있지 마세요)

make는 Makefile의 내용을 보고, 내부적으로 어떻게 파일들이 의존하고 있는지 조사한다. 위의 Makefile을 바탕으로 의존 관계를 그림으로 나타내 보면 아래와 같다.



(텍스트 기반이라서 그림 그리기가 꽤 어렵네요. =)

위의 그림에서 보면 test 가 만들어지기 위해서는 main.o read.o write.o가 필요하게 각각의 목적 파일들은 모두 자신의 소스 파일과 io.h 에 의존함을 알 수가 있다.

가령 main.c를 고쳤다고 생각한다면 main.o가 컴파일되어 다시 생기고, test 도 다시 링크되어 갱신된다. 만약 io.h가 바뀌었다고 가정하면 모든 파일들이 컴파일되어서 목적 파일이 생기고, 그것들이 링크가 되어 test가 생긴다.

위와 같이 파일들을 구성한 다음 Makefile을 실행시켜 보자. Makefile의 실행은 그냥 make라고만 치면 된다.

```
% make
gcc -c main.c
gcc -c read.c
gcc -c write.c
gcc -o test main.o read.o write.o <- OK
```

=> 참고: 그냥 테스트에 불과하기 때문에 read.c writec io.h 는 모두 내용 없이 파일만 만들어 두기로 하고, main.c 에 간단히 printf 함수만 적어 봅시다. 정말 위와 같이 됨을 실감할꺼예요... 신기하게...

2.3 매크로의 사용

간단한 매크로 기능을 사용해 보자. main.o read.o write.o라는 것을 OBJECTS 라는 매크로로 바꾸는 것이 아래의 예제 2에 나와 있다.

Makefile예제 2

```
OBJECTS = main.o read.o write.o

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

main.o : io.h main.c
		gcc -c main.c
read.o : io.h read.c
		gcc -c read.c
write.o: io.h write.c
		gcc -c write.c
```

위에서 보다시피 매크로는 그냥 프로그램 짤 때와 같이 사용해서 값을 대입한다. 대신 사용할 때는 반드시 $(..) 안에 넣어서 사용한다. 매크로 치환을 위한 특수한 방법이 아닐까... 히... 매크로의 사용법은 위와 같이 간단하므로 다양하게 정의해서 사용할 수 있다. 매크로에 대한 자세한 설명은 다음 장에서 언급하기로 한다.

2.4 레이블의 사용

목표 부분에 해당하는 부분이 그냥 레이블과 같이 사용될 수도 있다고 이미 설명하였다. 예제 2 에다가 목적 파일들을 모두 삭제하는 명령어를 추가하기로 한다.

Makefile예제 3

```
OBJECTS = main.o read.o write.o

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

main.o : io.h main.c
		gcc -c main.c
read.o : io.h read.c
		gcc -c read.c
write.o: io.h write.c
		gcc -c write.c

clean :
		rm $(OBECTS)
```

레이블로 사용될 때는 당연히 의존 관계 부분은 없어도 된다. 그리고 clean을 실행시키려면 아래와 같이 한다.

```
% make clean
rm main.o read.o write.o <- OK
```

다음 장에서는...

지금까지는 Makefile의 간단한 예제를 가지고 무엇을 할 수 있는지 대충 알아보았습니다. 예제를 많이 쓰다 보니까 내용이 불어나게 됐군요. RCS와 달리 make는 한번 알고 있으면 정말 유용한 유틸리티입니다.

다음 장에서는 본격적으로 Makefile의 구성및 그 사용법을 자세히 알아 보고자 합니다. 그냥 일반적으로 Makefile 사용하시려면 오늘 한 것에 몇 가지만 더 알고 계시면 됩니다. 계속 예제 중심으로 이해가 잘되도록... 그럼 계속 봐주시면 감사.