Makefile 기초 작성법

목차

파일 코드 내용

foo.c

```
#include <stdio.h>

void foo() { printf("foo\n"); }
```

foo.h

```
void foo();
```

bar.c

```
#include <stdio.h>

void bar() { printf("bar\n"); }
```

bar.h

```
void bar();
```

main.c

```
#include "foo.h"
#include "bar.h"

int main() {
  foo();
  bar();

  return 0;
}
```

터미널에서 컴파일 하기 및 실행

```
$ gcc -c foo.c bar.c main.c
$ gcc foo.o bar.o main.o -o main
$ ./main
```

지금은 파일이 다섯 개 정도이니 터미널에다가 하나씩 명령어를 입력하는 게 할만하다고 느껴지겠지만, *파일이 많아질 때는 이 또한도 고통이다.*
그래서 필요한 게 Makefile이다!

Makefile의 구조

```
<target> : <dependency>
(tab)<Recipe>
```

- **Target** : 빌드 대상. 명령어가 수행된 최종 결과를 저장하는 파일
- **Dependencies** : 주어진 target을 make할 때 필요한 파일 목록
- **Recipe** : 실행할 명령어. 명령어를 쓸 때 **반드시** tab키로 띄워줘야 한다.

위 내용을 바탕으로 *Makefile*을 작성해보면

```
main : foo.o bar.o main.o
	gcc foo.o bar.o main.o -o main

foo.o : foo.c
	gcc -c foo.c

bar.o : bar.c
	gcc -c bar.c

main.o : main.c
	gcc -c main.c

clean :
	rm -f *.o main
```

...이 된다.

Makefile실행

다음과 같이 터미널에 입력하면 make할 수 있다.

```
$ make
```

목적파일과 실행파일 삭제

Makefile에 보면 "clean"이라는 Target에 "rm -f"라는 Recipe를 설정해 두었다.
다음과 같은 명령어를 실행하면 목적파일과 실행파일이 삭제 시킬수 있다.

```
$ make clean
```

변수 사용

지금까지 한 것만 보았을 때는 Makefile의 이점을 잘 느끼지 못했을 것이다.
하지만 앞으로 Makefile의 변수를 사용하면 Makefile의 사용 이점을 느끼게 될 것이다.

즉 전의 Makefile을 변수를 사용하여 작성하면 다음과 같이 사용 할 수 있다.

```
CC=gcc
TARGET=a.out
OBJS=a.o b.o main.o

$(TARGET): $(OBJS)
	$(CC) -o $@ $(OBJS)

a.o : a.c
	$(CC) -c -o a.o a.c

b.o : b.c
	$(CC) -c -o b.o b.c

main.o : main.c
	$(CC) -c -o main.o main.c

clean:
	rm $(OBJECT) $(TARGET)
```

Makefile의 변수는 세 가지로 나눌수가 있다.

변수	변수이름
암시적 변수	"CC", "TARGET", "OBJS" 등
자동 변수	"<", "^", "%", "@", "*" 등

암시적 변수

- 예제 파일에 있는 암시적 변수

변수	변수설명
CC	C 컴파일러
OBJS	중간 산물(목적파일 목록)
TARGET	빌드 대상(실행 파일) 이름

- 예제 파일에 없는 암시적 변수

변수	변수설명
CXX	C++ 컴파일러
CFLAGS	C 컴파일러 옵션
CXXFLAGS	C++ 컴파일러 옵션
LDFLAGS	링커 옵션
LDLIBS	링크 라이브러리

... 등이 있다.
더 많은 변수들은 Implicit Variables (GNU make)를 참조하세요.

자동 변수

자동변수는 따로 다루지는 않지만 작성은 하겠습니다.

변수	변수설명
<	첫번째 입력 파일을 의미
^	모든 입력 파일을 의미
%	타겟의 입력 인자
@	출력 파일을 의미
*	입력 파일에서 꼬리말을 제외한 파일명

마무리

지금까지 Makefile을 다루면서 필요한 아주 최소한의 작성법을 익혔습니다.
개인적으로 궁금하거나 더 필요한 내용이 있으시면 맨 밑의 참조링크를 통해 직접 알아보세요.

여담

- **CMake**
  프로젝트 규모가 커지면 Makefile 사용이 불편해지는 날이 언젠가는 올것입니다.
  그런날이 올 때는 CMake를 알아보세요.

* **Java 빌드 도구** 다음과 같은 빌드 도구를 이용해서 Java 컴파일 보조를 할 수 있다고 하는데, 관심있으신 분은 직접 알아보세요. * Apache Ant * Gradle * Maven

참조

GNU make 공식 메뉴얼 문서
쉽게먹는 C++ - <19 - 1. Make 사용 가이드 (Makefile 만들기)>
Makefile 만들기