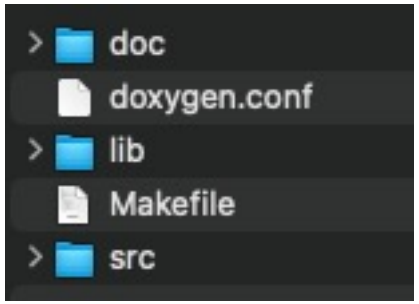
Makefile 기초 및 활용 방법

https://youtu.be/XzDwpJ3WuxQ

Makefile

- Make 파일은 소프트웨어 프로젝트에서 빌드 프로세스를 자동화하기 위해 사용되는 파일
  - Makefile 이라는 이름으로 프로젝트 폴더에 저장되어 있음



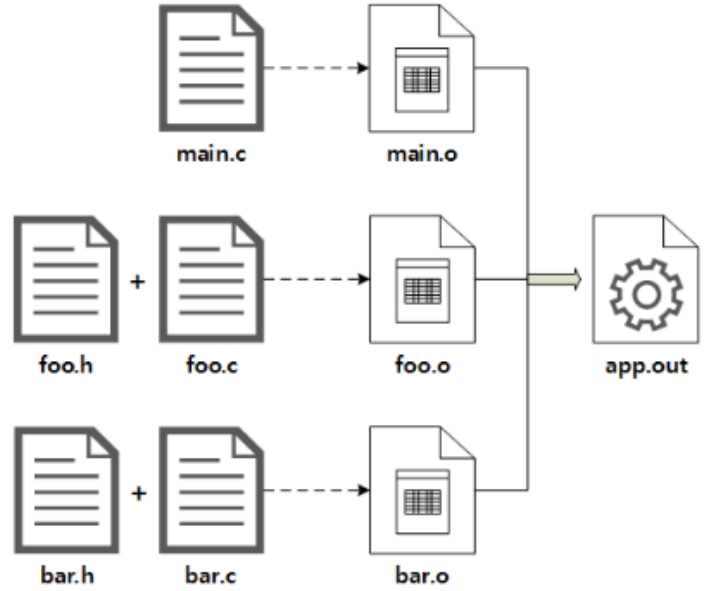
- 반복적인 빌드 작업을 자동화하여 시간과 비용을 절약
  - IDE를 사용할 때는 IDE에서 자동으로 해주지만 터미널에서 작업을 할 때는 명령어를 모두 작성해서 빌드해야함

빌드 작업

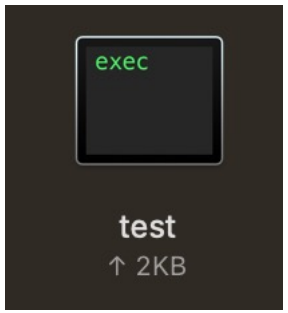
- 그림과 같이 여러 소스파일을 빌드하기 위해서는 여러번 과정을 통해서 app.out 실행 파일을 생성할 수 있음

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

```
gcc -o app.out main.o foo.o bar.o
```



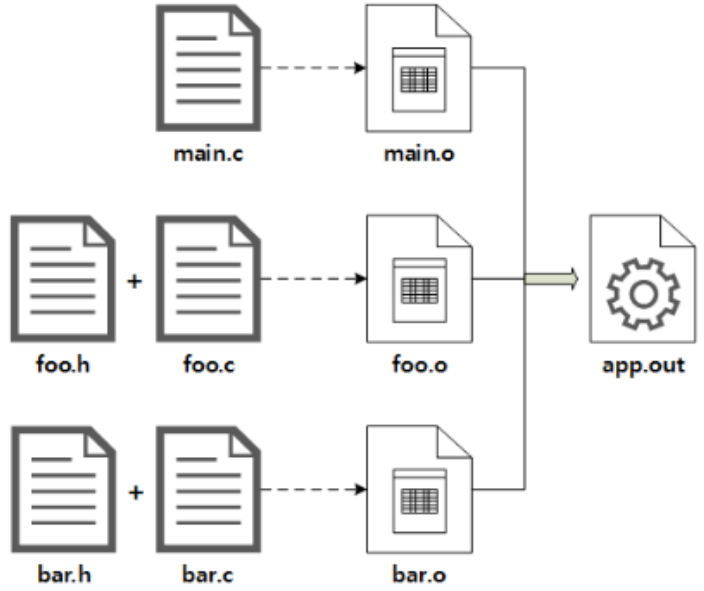

```
siwooeum@SiWooui-13inchi-MacPro Makefile_test % gcc -c -o makefile_test.o makefile_test.c
siwooeum@SiWooui-13inchi-MacPro Makefile_test % gcc -o test makefile_test.o
siwooeum@SiWooui-13inchi-MacPro Makefile_test % ./test
hello world
siwooeum@SiWooui-13inchi-MacPro Makefile_test %
```



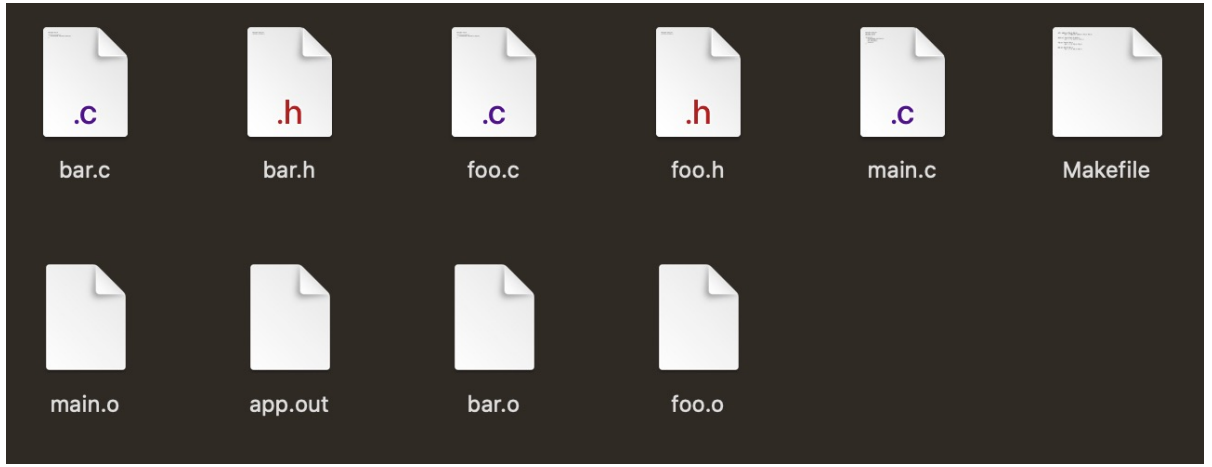
Makefile을 활용한 빌드

Makefile

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

main.o: foo.h bar.h main.c
    gcc -c -o main.o main.c

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

bar.o: bar.h bar.c
    gcc -c -o bar.o bar.c
```




```
siwooeum@SiWooui-13inchi-MacPro footest % make
gcc -c -o main.o main.c
gcc -c -o foo.o foo.c
gcc -c -o bar.o bar.c
gcc -o app.out main.o foo.o bar.o
```



Makefile의 형식

- Makefile의 구성은 target, dependencies, commands 로 구성되어 있음
  - Target : 빌드 과정에서 생성되는 파일의 이름
    - 일반적으로 실행 파일, 라이브러리, 오브젝트 파일이 될 수 있음
  - Dependencies : target을 생성하기 위해 필요한 파일의 목록
    - 일반적으로 소스 코드 파일, 헤더 파일이 될 수 있음
  - Commands : dependencies에서 지정한 파일을 사용하여 target을 생성하기 위해 실행되야 하는 명령어
    - 보통 컴파일러, 링커 등의 명령어가 될 수 있음

makefile

```
target: dependencies
    commands
```

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

main.o: foo.h bar.h main.c
    gcc -c -o main.o main.c

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

bar.o: bar.h bar.c
    gcc -c -o bar.o bar.c
```

Makefile에서의 변수 사용

- 변수를 사용하여서 좀 더 깔끔하고 확장성 있게 작성 가능
  - CC : 컴파일러 (gcc, g++ 등)
  - CFLAGS : 컴파일 옵션 (-O0, -O2 등)
  - OBJS : dependencies에 해당하는 Object 파일 목록
  - TARGET : 빌드 대상(실행 파일) 이름

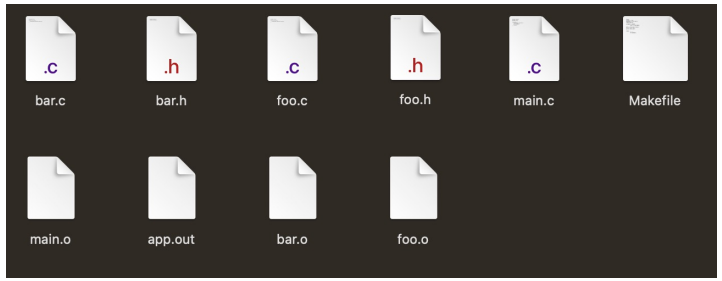
```
CC=gcc
CFLAGS=-g -Wall
OBJS=main.o foo.o bar.o
TARGET=app.out

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

main.o: foo.h bar.h main.c
foo.o: foo.h foo.c
bar.o: bar.h bar.c
```

- 추가적으로 LDFLAGS(링크 옵션), LDLIBS(링크 라이브러리) 같은 내장 변수도 있음.

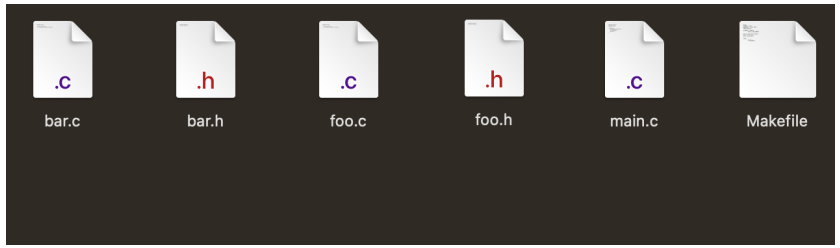
```
CC=<컴파일러>
CFLAGS=<컴파일 옵션>
LDFLAGS=<링크 옵션>
LDLIBS=<링크 라이브러리 목록>
OBJS=<Object 파일 목록>
TARGET=<빌드 대상 이름>

all: $(TARGET)

clean:
    rm -f *.o
    rm -f $(TARGET)

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

Makefile

- all 과 clean
  - all 을 통해서 여러개의 실행 파일을 만들 수 있음.
  - Clean 을 통해서 생성된 오브젝트 파일과 실행파일을 한번에 정리할 수 있음




```
siwooeum@SiWooui-13inchi-MacPro footest % make clean
rm *.o
rm app.out
```




```
CC=<컴파일러>
CFLAGS=<컴파일 옵션>
LDFLAGS=<링크 옵션>
LDLIBS=<링크 라이브러리 목록>
OBJS=<Object 파일 목록>
TARGET=<빌드 대상 이름>

all: $(TARGET)

clean:
    rm -f *.o
    rm -f $(TARGET)

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

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

KPQC에서 배포된 소스 코드의 Makefile 분석

```
CC      = gcc
LDFLAGS=-lcrypto
LDFLAGS+="-L/usr/local/opt/openssl@3/lib"
CPPFLAGS="-I/usr/local/opt/openssl@3/include"

SOURCES= PQCgenKAT_pke.c pke.c Keygen.c Encryption.c MultiEnc.c Decryption.c rng.c verify.c
    fips202.c
HEADERS= api.h Keygen.h Encryption.h MultiEnc.h Decryption.h params.h rng.h verify.h
    fips202.h


PQCgenKAT_pke: $(HEADERS) $(SOURCES)
    $(CC) -o $@ $(SOURCES) $(LDFLAGS) $(CPPFLAGS)

clean:
    @rm -rf *.o PQCgenKAT_pke
```

감 사 합 니 다