g++, Makefile

Learning about how to compile

using Makefile

g++

- g++
  - GNU Compiler
  - C 또는 C++ 로 작성된 소스파일(.c, .cc, .cpp)을 C++ 컴파일러로 컴파일해준다.
  - C++ 라이브러리와 링크를 해준다.
  - 대부분의 입출력 형식 및 옵션은 기본 C compiler (cc) 와 동일함

- 참고) gcc는 .c는 C 컴파일러로, .cc나 .cpp는 C++ 컴파일러로 컴파일을 해주지만, 링크 과정은 C 라이브러리와 해주기 때문에 .cc로 작성된 파일을 gcc로 컴파일하면 링크과정에서 에러가 날 수 있다.

g++ install

- $ sudo apt-get update
- $ sudo apt-get upgrade
- $ sudo apt-get install g++

```
hyungwon@hyungwon:~$ g++ --version
g++ (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

g++ options

- g++ [options] <infile> ...
  - -c : compile only, 실행파일을 만들지 않고 오브젝트 파일(.o)까지 생성.
  - -g : debugging, 디버깅에 필요한 정보를 포함해서 컴파일.
  - -o <output file> : 컴파일, 링크한 output file(대부분 실행파일)의 이름을 지정.
  - -I<dir> : include directory, 컴파일할 때 헤더파일을 찾아볼 디렉토리 지정.
  - -L<dir> : library directory, 링크할 때 라이브러리 파일을 찾아볼 디렉토리 지정.
  - -D<symbol>[=def] : define a macro. 컴파일 시 사용할 매크로 정의
  - …

Practice

- 다음 두 소스파일을 g++을 이용하여 컴파일한다.

$ vi main.cc

```
void print_hello();

int main(void) {
   print_hello();
   return 0;
}
```

$ vi print.cc

```
#include <iostream>

void print_hello() {
   std::cout << “Hello World” << std::endl;
}
```

1. 컴파일 + 링크 : **$ g++ -o hello_world main.cc print.cc**
2. 컴파일 후 링크 :
   $ g++ -c –o print.o print.cc
   $ g++ -c –o main.o main.cc
   $ g++ -o hello_world main.o print.o
3. 실행 : **$ ./hello_world**

Makefile

- Make
  - UNIX 계열에서 오랫동안 사용된 프로그램 빌드 툴.
  - 파일의 종속관계를 파악하여 컴파일러에 명령하고 SHELL 명령을 순차적으로 실행한다.
- Makefile
  - 소스파일을 어떻게 컴파일하고 링크해서 실행파일을 만들지에 대한 규칙이 기술되어 있는 파일.
  - make 가 실행되면 해당 디렉토리에서 Makefile을 찾아서 해석하고 실행.
- Makefile 의 장점
  - 반복적인 컴파일 명령을 간편하게 수행 가능
  - 프로그램의 종속 구조를 빠르게 파악 가능

Makefile 문법

- Makefile 작성 방법

```
target: prerequisites
	command1
	command2
	...
```

  - target : 만들고자 하는 파일 (.o 또는 실행파일 등)
  - prerequisites : target을 만드는 데 필요한 파일 목록
  - command(s) : target을 만드는 각 단계별 명령어.
  - 주의) command(s) 앞에는 반드시 tab이 있어야함 (space 안됨)
    - vimrc에서 set expandtab을 한 경우, 실제 tab이 space로 치환되므로
    - 입력모드에서 <Ctrl> + <v> 한 후 [tab] 을 누르면 실제 tab이 입력된다.

Practice

- 두 소스파일 main.cc/print.cc 을 컴파일, 링크하는 Makefile 작성

$ vi Makefile

```
hello_world:  main.o print.o
        g++ -o hello_world  main.o print.o
main.o:  main.cc
        g++ -c main.cc
print.o:  print.cc
        g++ -c print.cc
clean:
        rm hello_world  main.o  print.o
```

- 실행파일 생성 : make
- 실행파일 삭제 : make clean

Makefile 고급 문법

- Macro
  - 반복되는 문자열을 Macro를 통해 변수처럼 정의할 수 있다.

$ vi Makefile

```
TARGET  = hello_world
$(TARGET):  main.o  print.o
        g++ -o $(TARGET)  main.o  print.o
...
```

  - TARGET이라는 Macro 명을 먼저 정의하였다.
  - Macro를 참조할 경우 괄호 안에 Macro 명을 넣고 앞에 $를 붙여서 사용한다.
  - 반드시 정의부가 사용할 항목보다 먼저 작성되어야 한다.
  - 정의 시 Macro 앞에 tab이 시작할 수 없으며, ‘’,“”, =, :는 Macro명에 사용할 수 없다.
  - CC나 CPPFLAG, LD 등 미리 내부정의된 Macro도 있다.

Makefile 고급 문법

- Macro 치환

$ vi Makefile

```
...
SRCS = abc.cc def.cc ghi.cc
OBJS = $(SRCS: .cc=.o)
...
```

- Macro의 정의에는 여러 문자열이 공백을 기준으로 들어갈 수 있다.
- 앞에서 정의된 Macro의 일부를 치환해서 정의할 수 있다.
  - OBJS = abc.o def.o ghi.o 라고 정의된 것과 같다.

Project Structure

- C/C++ 프로젝트는 대게 src, bin, include, lib 등의 디렉토리로 구성된다.
- Project/
  - src : 소스 파일로 구성
  - include : 헤더 파일로 구성
  - lib : 링크에 필요한 유저 라이브러리 파일로 구성
  - bin : 컴파일 후 나오는 실행 파일로 구성

```
hyungwon@hyungwon:~/Project-template$ ls
bin  include  lib  Makefile  README  src
```

Practice

- Project Structure 과 Makefile 예제
  - Project 디렉토리 생성한다.
  - Project 내부에 src, bin, include, lib 디렉토리를 생성한다.
  - 이전에 작성한 main.cc와 print.cc를 Project/src디렉토리에 넣는다.

```
hyungwon@hyungwon:~/Project-template$ ls
bin  include  lib  Makefile  README  src
hyungwon@hyungwon:~/Project-template$ ls src
main.cc  print.cc
```

  - Makefile을 작성한다.

Practice

```
# Compiler and Compile options.
CC = g++
CXXFLAGS = -g -Wall -std=c++11

# Macros specifying path for compile.
SRCS := $(wildcard src/*.cc)
OBJS := $(SRCS:.cc=.o)
BIN = ./bin/
INC = ./include/
LIB = ./lib/ -lpthread

# Pre-Processor.
CPPFLAGS += -I$(INC)

# Compile command.
TARGET = print_hello
$(TARGET): $(OBJS)
    $(CC) $(CXXFLAGS) $(CPPFLAGS) -o $(BIN)$(TARGET) $(OBJS) -L$(LIB)

# Delete binary & object files.
clean:
    rm $(BIN)$(TARGET) $(OBJS)
```

Practice

- 실행 결과
  - $ make

```
hyungwon@hyungwon:~/Project-template$ make
g++ -g -Wall -std=c++11 -I./include/  -c -o src/main.o src/main.cc
g++ -g -Wall -std=c++11 -I./include/  -c -o src/print.o src/print.cc
g++ -g -Wall -std=c++11 -I./include/ -o ./bin/print_hello src/main.o
print.o -L./lib/ -lpthread
```

  - $ make clean

```
hyungwon@hyungwon:~/Project-template$ make clean
rm ./bin/print_hello src/main.o src/print.o
```

수고하셨습니다.