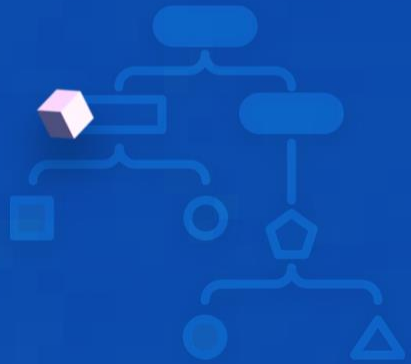
강원LRS공유대학

강원혁신플랫폼

리눅스프로그래밍

리눅스(Linux) 프로그래밍 도구



C언어 여러 모듈을 동시에 컴파일하고 목적파일을 관리하기 위해 사용하는 명령어는 무엇인가?

make

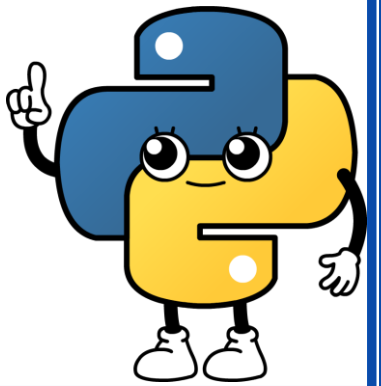
학습목표

학습 내용

1. GCC(GNU Compiler Collection)
2. Make

학습 목표

- GCC(GNU Compiler Collection)에 대해 파악할 수 있다.
- Make 유틸리티에 대해 설명할 수 있다.



GCC(GNU Compiler Collection)

GCC(GNU Compiler Collection)

리눅스의 프로그래밍 도구

- 1987년 리처드 스톨먼은 공개용 컴파일러인 GCC를 작성
- GCC 는 GNU 소프트웨어와 리눅스 같은 오픈 소스 운동의 시발점
- GCC는 다양한 플랫폼을 지원하며, C, C++, Objective-C, Fortran, Java 등의 다양한 프로그래밍 언어들을 지원

리눅스의 프로그래밍 도구

도구	내용	도구	내용
gcc	리눅스의 기본 컴파일러	objdump	목적 파일에 대한 정보 출력
ld	GNU 링커(Linker)	ranlib	아카이브 색인 작성
as	GNU 어셈블러(Assembler)	strip	목적 파일 등에서 심복 삭제
ar	어카이브 생성 및 수정	strings	출력 가능한 문자열 출력
nm	목적 파일에서 심볼 추출	size	섹션 크기의 리스트 출력
objcopy	목적 파일 복사	gdb	디버깅 지원
make	소스코드를 빌드하기 위한 자동화 지원		

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

소스코드 다운로드

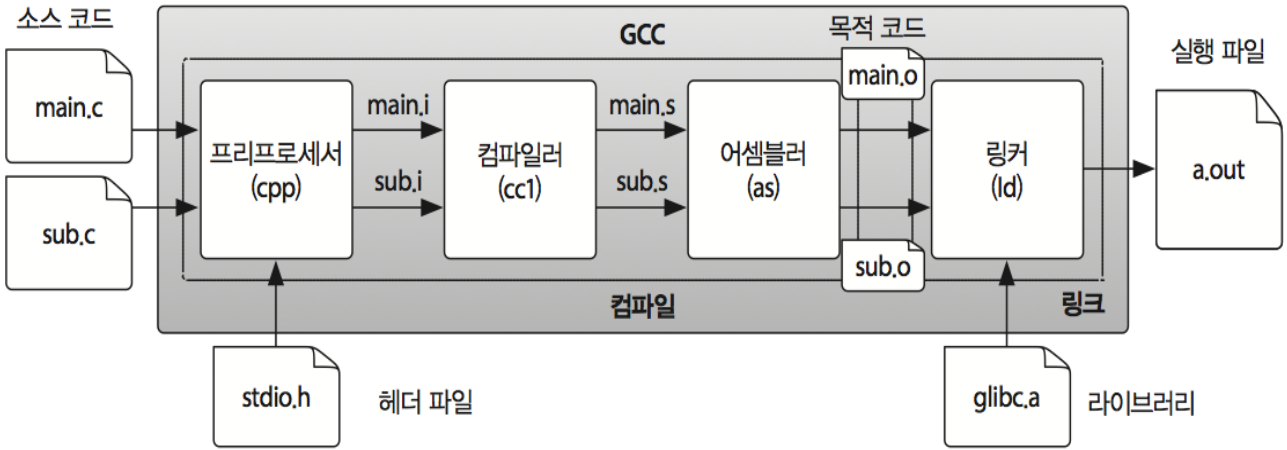

```
freetime@ATwin:~$ sudo apt-get update
/* 중간 생략 */
[sudo] password for freetime:
freetime@ATwin:~$ sudo apt-get install git-core
/* 중간 생략 */
freetime@ATwin:~$ git clone
https://github.com/valentis/LinuxProgrammingWithRaspberryPi.git
/* 중간 생략 */
```

소스코드 다운로드


```
freetime@ATwin:~$ ls
LinuxProgrammingWithRaspberryPi
freetime@ATwin:~$ ls LinuxProgrammingWithRaspberryPi/
Chapter10  Chapter12  Chapter2  Chapter4  Chapter6  Chapter8  LICENSE   RPi
Chapter11  Chapter13  Chapter3  Chapter5  Chapter7  Chapter9  README.md
makefile
freetime@ATwin:~$
```

소스코드 빌드 과정

- 여러 단계로 나누어져 빌드



소스코드 빌드 과정

gcc 컴파일러의 기본 옵션

옵션	내용	옵션	내용
-o	실행 파일 이름 지정	-I(대문자i)	헤더 파일 위치 지정
-c	목적 파일 생성	-l(소문자L)	링크할 라이브러리 지정
-g	디버깅 정보 추가	-L	라이브러리 파일 위치 지정
-O, -O2 등	최적화 수행	-D 매크로	컴파일이 매크로 사용

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

따라하기 helloworld.c

소스코드 경로로 이동

```
freetime@ATwin:~$ cd LinuxProgrammingWithRaspberryPi/Chapter2
```

내용 확인

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ ls
a.txt  common.h  helloworld.c  input.c    main.c   makefile1  makefile3  makefile5  makefile7
c.txt  e.txt     helloworld.s  libmine.so  makefile  makefile2  makefile4  makefile6  print.c
```

따라하기 helloworld.c

소소코드 내용 확인

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ cat
helloworld.c
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello, World!\n");

    return 0;
}
```

따라하기 helloworld.c

소스코드 컴파일, gcc 컴파일러가 없음

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ gcc -o helloworld helloworld.c
Command 'gcc' not found, but can be installed with:
sudo apt install gcc
```

gcc 컴파일러 설치

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ sudo apt install gcc
[sudo] password for freetime:
```

따라하기 helloworld.c

소스코드 컴파일

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ gcc -o
helloworld helloworld.c
```

따라하기 helloworld.c

소스코드 단계적 컴파일

1. 전처리, 어셈블러코드 생성

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ gcc -S
helloworld.c
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ ls hello*
helloworld  helloworld.c  helloworld.s
```

2. 어셈블러 코드 컴파일, 오브젝트 생성

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ as -o
helloworld.o helloworld.s
```

3. 오브젝트로 부터 실행파일 생성

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ gcc -o helloworld helloworld.o
```

파일 유형 확인

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ file helloworld
helloworld: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=8f4be24ae973b5b42cca279057faae5d923726bd, for GNU/Linux 3.2.0, not stripped
```

따라하기 helloworld.c

실행, 결과 확인

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ ./helloworld
Hello, World!
```

Make

make 유틸리티(1)

소스코드 빌드 자동화 도구

Make 유틸리티

[makefile이라는 파일에 특수한 형태의 의존 규칙(dependency rule)들을 표시]

makefile

```
targetList:     dependencyList
                commandList
```

부분	내용
목표(target)	명령(command list)이 수행되어 생성될 결과 파일 (목적 파일이나 실행 파일)을 지정한다.
의존관계(dependency)	목표를 수행하기 위해 필요한 의존 관계를 설정한다.
명령(command)	의존관계(depenency list)에 정의된 파일의 내용이 바뀌었거나, 목표(target list)에 해당하는 파일이 없을 때 여기에 정의된 내용이 차례대로 실행된다.

make 유틸리티(2)

소스코드 빌드 자동화 도구

매크로(Macro)

- 매크로는 makefile에서 편리하게 사용하기 위해 미리 정의된 변수
- '${매크로}', '$(매크로)', '$매크로' 같은 형태를 사용 가능하나 일반적으로 '$(매크로)'를 사용
- 매크로 치환 : $(매크로이름:이전내용=새로운 내용)

확장자 규칙

- 파일의 확장자를 보고 그에 따라 적절한 연산을 수행

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

소스코드 빌드 자동화 도구

makefile의 주요 자동 변수

변수	내용
$*	확장자가 없는 현재 목표 파일(Target)의 이름을 지칭한다.
$@	현재 목표 파일(Target)의 이름을 지칭한다.
$<	현재 목표 파일(Target)보다 더 최근에 갱신된 파일명으로, 첫 번째 종속물의 이름이다.
$?	$<' 과 같다.

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

따라하기 make없이 여러 모듈 컴파일

파일 내용 확인

```
$ cat main.c
#include "common.h"

int main( )
{
    char* str = input( );
    print(str);
    print("\n");

    return 0;
}
```

따라하기 make없이 여러 모듈 컴파일


```
$ cat common.h
#ifndef __COMMON_H__
#define __COMMON_H__

extern void print(char* str);
extern char* input( );

#endif /* __COMMON_H__ */
```

따라하기 make없이 여러 모듈 컴파일

파일 내용 확인

```
$ cat print.c
#include <stdio.h>
#include "common.h"

void print(char* str)
{
    printf("%s", str);
}
```

따라하기 make없이 여러 모듈 컴파일


```
$ cat input.c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

char* input( )
{
    char *str;
    str = (char*)malloc(BUFSIZ);
    scanf("%s", str);

    return str;
}
```

따라하기 make없이 여러 모듈 컴파일

컴파일, 실행

```
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ gcc -o
test main.c print.c input.c
freetime@ATwin:~/LinuxProgrammingWithRaspberryPi/Chapter2$ ./test
Hello
Hello
```

따라하기 makefile 1, 2


```
$ cat makefile1
test :
        gcc -o test main.c print.c
input.c


$ make -f makefile1 clean
make: *** No rule to make target
'clean'.  Stop.
$ make -f makefile1
make: 'test' is up to date.
$ ./test
maketest1
maketest1
```

따라하기 makefile 1, 2


```
$ cat makefile2
test : main.o print.o input.o
	gcc -o test main.o print.o
input.o

main.o : main.c common.h
	gcc -c main.c

print.o : print.c common.h
	gcc -c print.c

input.o : input.c common.h
	gcc -c input.c
```

따라하기 makefile 1, 2


```
$ make -f makefile2 clean
make: *** No rule to make target
'clean'.  Stop.
$ make -f makefile2
gcc -c main.c
gcc -c print.c
gcc -c input.c
gcc -o test main.o print.o input.o
```

따라하기 makefile 3, 4


```
$ cat makefile3
OBJECTS = main.o print.o input.o

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

main.o : main.c common.h
	gcc -c main.c

print.o : print.c common.h
	gcc -c print.c

input.o : input.c common.h
	gcc -c input.c
```

따라하기 makefile 3, 4


```
$ make -f makefile3 clean
make: *** No rule to make target 'clean'.  Stop.
$ make -f makefile3
gcc -c main.c
gcc -c print.c
gcc -c input.c
gcc -o test main.o print.o input.o
```

따라하기 makefile 3, 4


```
$ cat makefile4
OBJECTS = main.o print.o input.o
SRCS = main.c print.c input.c
CFLAGS = -g
TARGET = test

$(TARGET) : $(OBJECTS)
	$(CC) -o $(TARGET) $(OBJECTS)

clean :
	rm -f $(OBJECTS) $(TARGET) core
```

따라하기 makefile 3, 4


```
main.o : main.c common.h
print.o : print.c common.h
input.o : input.c common.h

$ make -f makefile4 clean
rm -f main.o print.o input.o test core
$ make -f makefile4
cc -g   -c -o main.o main.c
cc -g   -c -o print.o print.c
cc -g   -c -o input.o input.c
cc -o test main.o print.o input.o
```

따라하기 makefile 5

.c .o 확장자의 경우 지정한 규칙을
따르도록 설정

```
$ cat makefile5
# makefile
.SUFFIXES : .c .o

OBJECTS = main.o print.o input.o
SRCS = main.c print.c input.c

CC = gcc
CFLAGS = -g
TARGET = test
```

따라하기 makefile 5


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

.c.o :
	$(CC) $(CFLAGS) -c $< -o
$@

clean :
	$(RM) $(OBJECTS)
$(TARGET) core

main.o : main.c common.h
print.o : print.c common.h
input.o : input.c common.h
```

따라하기 makefile 5


```
$ make -f makefile5 clean
rm -f main.o print.o input.o test
core
$ make -f makefile5
gcc -g -c main.c -o main.o
gcc -g -c print.c -o print.o
gcc -g -c input.c -o input.o
gcc -o test main.o print.o input.o
```

따라하기 makefile 6

.c 소스 파일 이름 자동 지정

```
$ cat makefile6
# makefile
.SUFFIXES : .c .o

OBJECTS = main.o \
          print.o \
          input.o
SRCS = $(OBJECTS:.o=.c)

CC = gcc
CFLAGS = -g
TARGET = test
```

따라하기 makefile 6


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

.c.o :
	$(CC) $(CFLAGS) -c $< -o
$@

clean :
	$(RM) $(OBJECTS)
$(TARGET) core

main.o : main.c common.h
print.o : print.c common.h
input.o : input.c common.h
```

따라하기 makefile 6


```
$ make -f makefile6 clean
rm -f main.o print.o input.o test
core
$ make -f makefile6
gcc -g -c main.c -o main.o
gcc -g -c print.c -o print.o
gcc -g -c input.c -o input.o
gcc -o test main.o print.o input.o
```

따라하기 makefile 7

다중 타킷

```
$ cat makefile7
# makefile
.SUFFIXES : .c .o

OBJECTS = main.o \
          print.o \
          input.o
SRCS = $(OBJECTS:.o=.c)

CC = gcc
CFLAGS = -g
TARGET = test helloworld
```

따라하기 makefile 7


```
all : $(TARGET)
helloworld : helloworld.c
	$(CC) -o $@ $<

test : $(OBJECTS)
	$(CC) -o $@ $(OBJECTS)

clean :
	$(RM) $(OBJECTS)
$(TARGET) helloworld.o core

main.o : main.c common.h
print.o : print.c common.h
input.o : input.c common.h
```

따라하기 makefile 7


```
$ make -f makefile7 clean
rm -f main.o print.o input.o test
helloworld helloworld.o core
freetime@ATwin:~/LinuxProgram
mingWithRaspberryPi/Chapter2$
make -f makefile7
gcc -g   -c -o main.o main.c
gcc -g   -c -o print.o print.c
gcc -g   -c -o input.o input.c
gcc -o test main.o print.o input.o
gcc -o helloworld helloworld.c
```

요약정리

- 리눅스의 프로그래밍 도구
  - GCC(GNU Compiler Collection)
  - Make