System Programming & OS 실습
4. Vi editor, GCC, Make

정지헌, 안석현, 김선재
Dankook University
{wlgjsjames7224, seokhyun, rlatjswo0824}@dankook.ac.kr

Index

❖ Vi editor

- 기본 조작
- 실습
- Vi editor setting
- 파일 생성: 김소월 '가늘 길' 작성

❖ GCC

- 설치
- 첫 번째 예제 – 단일 파일 컴파일
- 두 번째 예제 – 다수 파일 컴파일

❖ Make

- 예제
- 매크로

Vi editor

❖ Vi 시작

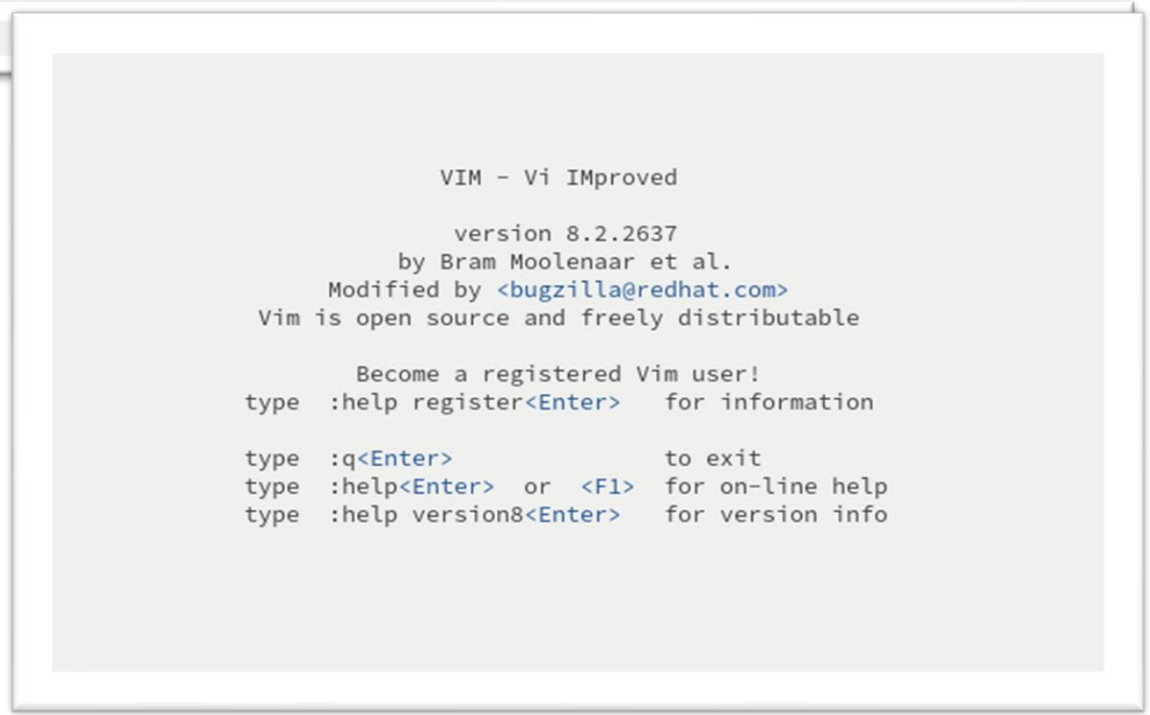
- 파일을 지정할 경우 : 해당 파일이 있으면 파일의 내용이 보이고, 없는 파일이면 빈 파일이 열린다.

```
sunjae@localhost ~]$ vi text.txt
```

- 파일을 지정하지 않을 경우 : 그냥 빈 파일이 열린다(파일명은 저장할 때 지정 가능)

```
sunjae@localhost ~]$ vi
```

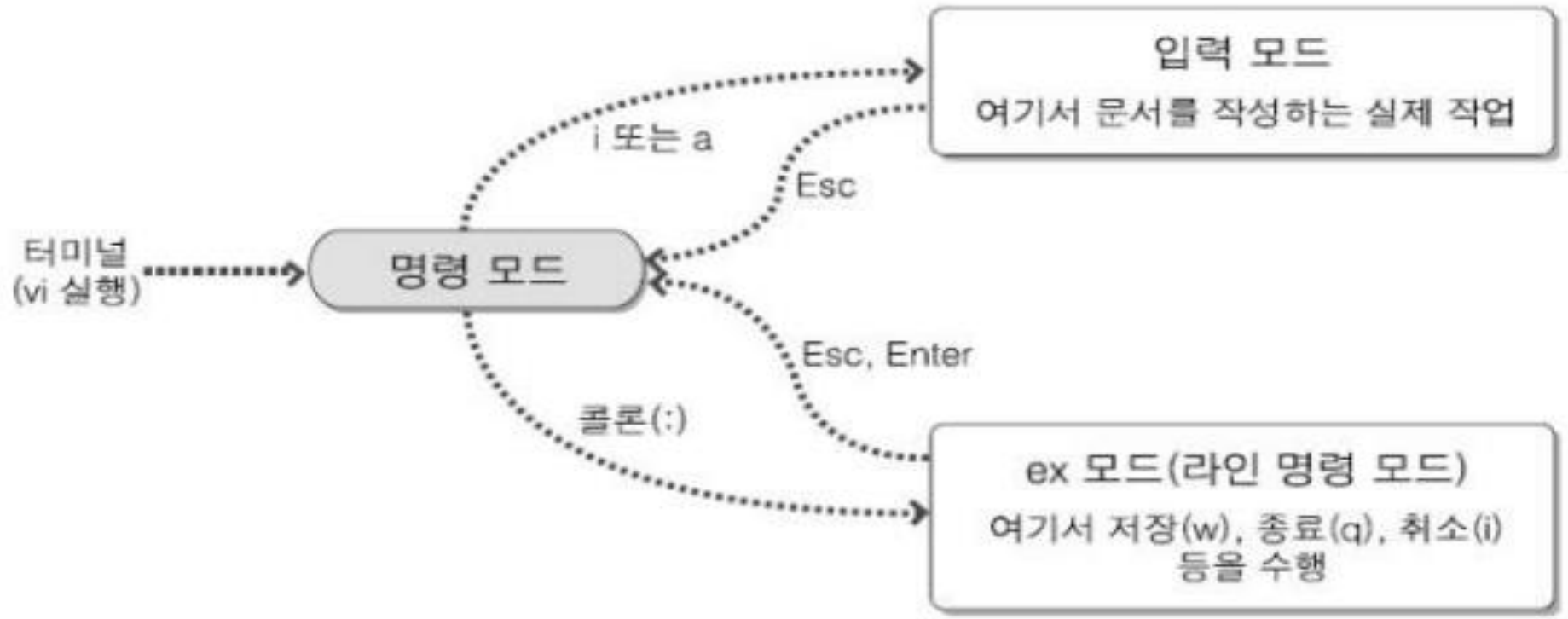
❖ 초기 화면



Vi editor

❖ 실습

- Vi의 기본적인 사용법을 익혀보자



Vi editor

❖ Vi 종료

- 명령모드나 마지막행 모드에서 저장하고 종료 가능

구분	명령키	기능
마지막 행 모드	:q	Vi에서 작업한 것이 없을 때 그냥 종료한다
	q!	작업한 내용을 저장하지 않고 종료한다.
	:w [파일명]	작업한 내용을 저장만 한다, 파일명을 지정하면 새 파일로 저장한다
	:wq, :wq!	작업한 내용을 저장하고 Vi를 종료한다
명령 모드	ZZ(shift + zz)	작업한 내용을 저장하고 vi를 종료한다

Vi editor

❖ 입력 모드로 전환

명령 키	기능
i	커서 앞에 입력한다(현재 커서 자리에 입력)
a	커서 뒤에 입력한다(현재 커서 다음 자리에 입력)
o	커서가 위치한 행의 다음 행에 입력한다
I	커서가 위치한 행의 첫 칼럼으로 이동하여 입력한다
A	커서가 위치한 행의 마지막 칼럼으로 이동하여 입력한다
O	커서가 위치한 행의 앞 행에 입력한다

Vi editor

❖ 실습

- 삽입
- 커서 이동

① 삽입 명령			
a	커서 뒤에 입력	A	라인 끝에 입력
i	커서 앞에 입력	I	라인 시작 부분에 입력
o	커서 있는 라인 밑에 입력	O	커서가 있는 라인 위에 입력

② 커서 이동 명령			
h	왼쪽으로 커서 한칸 이동	H	화면의 처음으로 이동
l	오른쪽으로 한칸 이동	L	화면 끝으로 이동
e	다음 단어의 마지막으로 이동	E	커서를 공백으로 구분된 다음 단어 끝으로 이동
b	한 단어 뒤로 이동	B	커서를 공백으로 구분된 이전 단어로 이동
w	커서를 한 단어 뒤로 이동	W	커서를 공백으로 구분된 다음 단어로 이동
k	커서를 한 라인 위로 이동	j	커서를 한 라인 아래로 이동
0	커서를 라인의 시작으로 이동	$	커서를 라인의 끝으로 이동
Enter	커서를 다음 라인 시작으로 이동	-	커서를 전 라인의 시작으로 이동
Ctrl + F	다음 화면으로 이동	Ctrl + D	화면의 반만 앞으로 이동
Ctrl + B	전 화면으로 이동	Ctrl + U	화면의 반만 뒤로 이동
G	커서를 텍스트의 마지막 라인으로 이동	숫자G	커서를 숫자 라인만큼 이동
M	커서를 화면 중간 라인으로 이동	''	커서를 전 위치로 이동
(	문단의 시작으로 이동	{	문단의 시작 위치로 이동
)	문장 끝으로 이동하여 다음 단어의 시작으로 커서 이동	}	문단 끝으로 이동

Vi editor

❖ 실습

- 삭제
- 바꾸기

③ 삭제 명령			
x	커서가 있는 문자 삭제	X	커서가 있는 문자 앞의 문자 삭제
dw	커서가 있는 단어 삭제	db	커서 앞에 있는 단어 삭제
dW	공백으로 구분된 뒷 단어 삭제	dB	공백으로 구분된 앞 단어 삭제
dd	커서가 있는 라인 삭제	D	커서가 있는 라인의 나머지 삭제
d)	문장의 나머지 삭제	d}	문단의 나머지 삭제
dG	파일의 나머지 삭제	dH	화면의 시작까지 삭제
dL	화면의 나머지 삭제	J	커서와 다음 단어의 공백을 모두 삭제

④ 바꾸기 명령			
r	커서가 있는 문자 대치	R	입력 모드로 한 문자씩 덮어씀
s	커서가 있는 문자 삭제 후 입력 모드로 전환	S	커서가 있는 줄을 삭제한 후 입력 모드로 전환
cb	커서가 있는 앞 문자 삭제 후 입력 모드	cW	공백으로 구분된 뒷 단어를 삭제한 후에 입력 모드
cB	공백으로 구분된 앞 단어 삭제 후 입력 모드	cc	커서가 있는 라인을 삭제하고 입력 모드
C	커서가 있는 라인의 나머지를 삭제하고 입력모드로 전환	c0	커서에서부터 라인의 시작까지 텍스트 바꾸기
c	특정 텍스트 바꾸기	c)	문장의 나머지 바꾸기
c}	문단의 나머지 바꾸기	cG	파일의 나머지 바꾸기
cm	표시까지 모든 것 바꾸기	cL	화면의 나머지 바꾸기
cH	화면의 시작까지 바꾸기		

Vi editor

❖ 실습

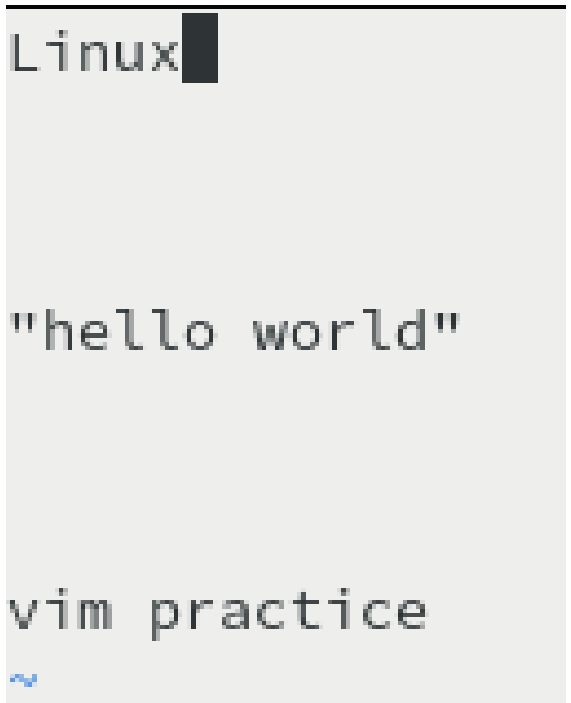
- 이동
- 복사

①텍스트 이동			
p	삭제나 복사된 텍스트를 커서가 있는 문자라 라인 뒤에 삽입	P	삭제나 복사된 텍스트를 커서가 있는 문자라 라인 앞에 삽입
dw p	커서가 있는 단어를 삭제한 후 이를 원하는 곳 커서 뒤로 삽입	dw P	커서가 있는 단어를 삭제한 후 이를 변경한 커서가 있는 곳 앞으로 삽입
d p	지정한 다음 텍스트로 삭제한 후 커서가 가리키는 곳으로 이동	d) P	문장의 나머지로 이동
d} p	문단의 나머지로 이동	dG p	파일의 나머지로 이동
dH p	화면 시작 부분으로 이동	dL p	화면의 나머지를 이동

복사			
yw	커서가 있는 단어를 복사	yb	커서가 있는 앞 단어를 복사
yW	공백으로 구분된 뒷 단어 복사	yB	공백으로 구분된 앞 단어를 복사
y	특정한 다음 텍스트 복사	yy	커서가 있는 라인을 복사, 커서가 가리키는 곳으로 라인을 이동
y)	문단의 나머지 복사	y}	문단의 나머지 복사
yG	파일의 나머지 복사	yH	화면 시작까지 복사
yL	화면의 나머지 복사		

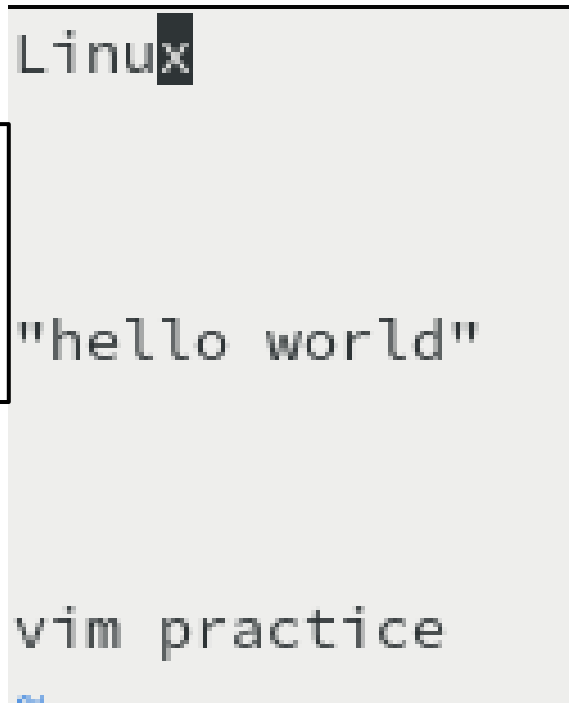
Vi editor

❖ 동작모드 예 – i, I, a, A, o, O



a 명령은 커서가 있는
x 문자 뒤에 커서가 입력
모드로 대기

➔ x문자 뒤에서부터 문자 입력

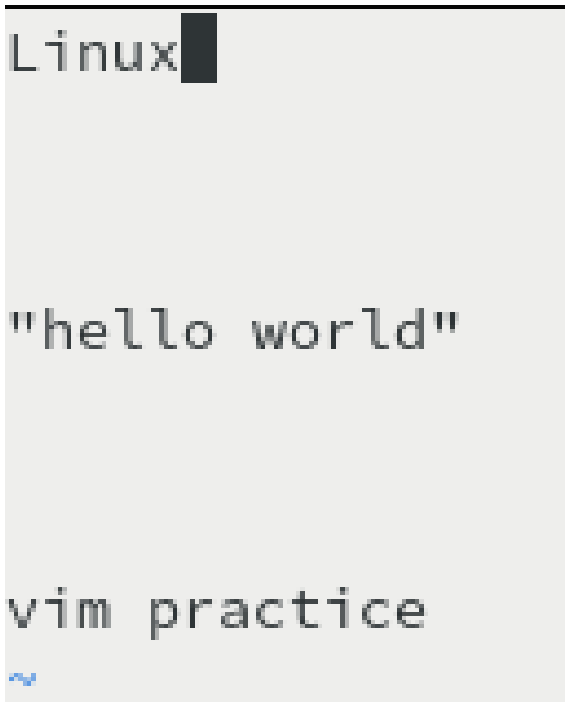


i 명령은 커서가 있는
x 문자에서 커서를 입력
모드로 대기

➔ x문자 앞에서부터 문자 입력

Vi editor

❖ 동작모드 예 – i, I, a, A, o, O



A 명령은 커서가 있는 줄의 마지막 칸에 커서가 입력모드로 대기

➔ 첫 번째 라인 아무 곳에서 A명령 시 x문자 뒤에서부터 입력

I 명령은 커서가 있는 줄의 처음 시작 칸에 커서가 입력 모드로 대기

➔ 첫 번째 라인 아무 곳에서 A명령 시 L문자 앞에서부터 입력

Vi editor

❖ 동작모드 예 – i, I, a, A, o, O

O 명령은 커서가 있는 줄의 첫 칸에 커서가 대기

o 명령은 커서가 있는 줄의 아래 줄 첫 칸에 커서가 위치

Vi editor

❖ Vi editor setting

- 색상 변경, 자동 들여쓰기...

.vimrc 파일

```
syntax enable                                ## 하이라이트
syntax on                                    ## 파일종류 자동인식
filetype on                                  ## 자동 들여쓰기
set autoindent                               ## 배경 컬러
set background=dark                          ## C언어 자동들여쓰기
set cindent                                  ## 명령어 기록
set history=100                              ## 검색어 강조
set hlsearch                                 ## 행 번호 표시
set number                                   ## 계단현상 제거 (붙여넣기)
set paste!                                   ## 들여쓰기 설정
set shiftwidth=4                             ## 괄호의 짝을 표시해주는 기능
set showmatch                                ##상태정보라인 구성
set statusline=%h%F%m%r%=[%l:%c(%p%%)]       ## 스마트한 들여쓰기
set smartindent                              ## 탭(tab) 간격
set tabstop=4                                ## 자동줄 바꿈 길이
set textwidth=80                             ## 현재 수정중인 파일명 표시
set title                                    ## 좌표 표시
set ruler                                    ## 색상 테마출처
colo koehler
```

```
[root@ip-172-31-15-105 ec2-user]# yum install vim
```

1. Vim 패키지 설치

```
[root@ip-172-31-15-105 ec2-user]# vi ~/.vimrc
```

2. .vimrc 파일 생성 및 내용 추가

Vi editor

❖ Vi editor setting

- 색상 변경, 자동 들여쓰기...

```
[root@ip-172-31-15-105 ec2-user]# vi ~/.bashrc
```

3. vi ~/.bashrc 파일 수정

```
1  # .bashrc
2
3  # Source global definitions
4  if [ -f /etc/bashrc ]; then
5      . /etc/bashrc
6  fi
7
8  # User specific environment
9  if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
10 then
11     PATH="$HOME/.local/bin:$HOME/bin:$PATH"
12 fi
13 export PATH
14
15 # Uncomment the following line if you don't like systemctl's auto-paging feature:
16 # export SYSTEMD_PAGER=
17
18 # User specific aliases and functions
19
20 alias rm='rm -i'
21 alias cp='cp -i'
22 alias mv='mv -i'
23 alias vi='vim'
```

4. alias vi='vim' 내용 추가

```
[root@ip-172-31-15-105 ec2-user]# source ~/.bashrc
```

5. 저장 후 설정한 부분 적용

Vi editor

❖ 파일 생성

- 김소월 '가늘 길' 작성

```
[ec2-user@ip-172-31-15-105 taba1]$ vi poem
```

```
[ec2-user@ip-172-31-15-105 taba1]$ cat poem
```

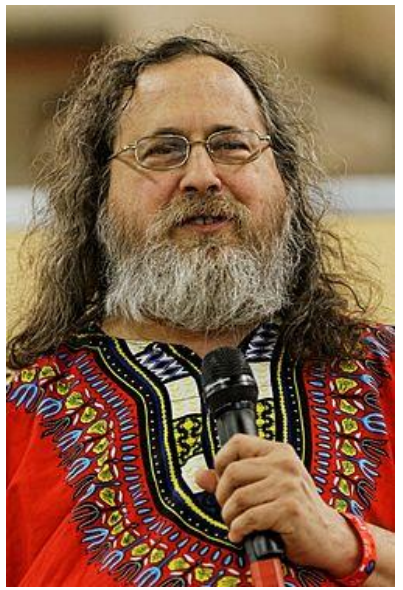
```
 1 가 늘   길
 2                          김 소 월
 3 그 립 다
 4 말 을   할 까
 5 하 니   그 리 워 .
 6
 7 그 냥   갈 까
 8 그 래 도
 9 다 시   더   한   번 …….
10
11 저   산 에 도   까 마 귀 , 들 에   까 마 귀 ,
12 서 산 에 는   해   진 다 고
13 지 저 귑 니 다 .
14
15 앞 강 물 , 뒷 강 물 ,
16 흐 르 는   물 은
17 어 서   따 라 오 라 고   따 라 가 자 고
18 흘 러 도   연 달 아   흐 릅 디 다 려 .
```

GCC

❖ GCC란?

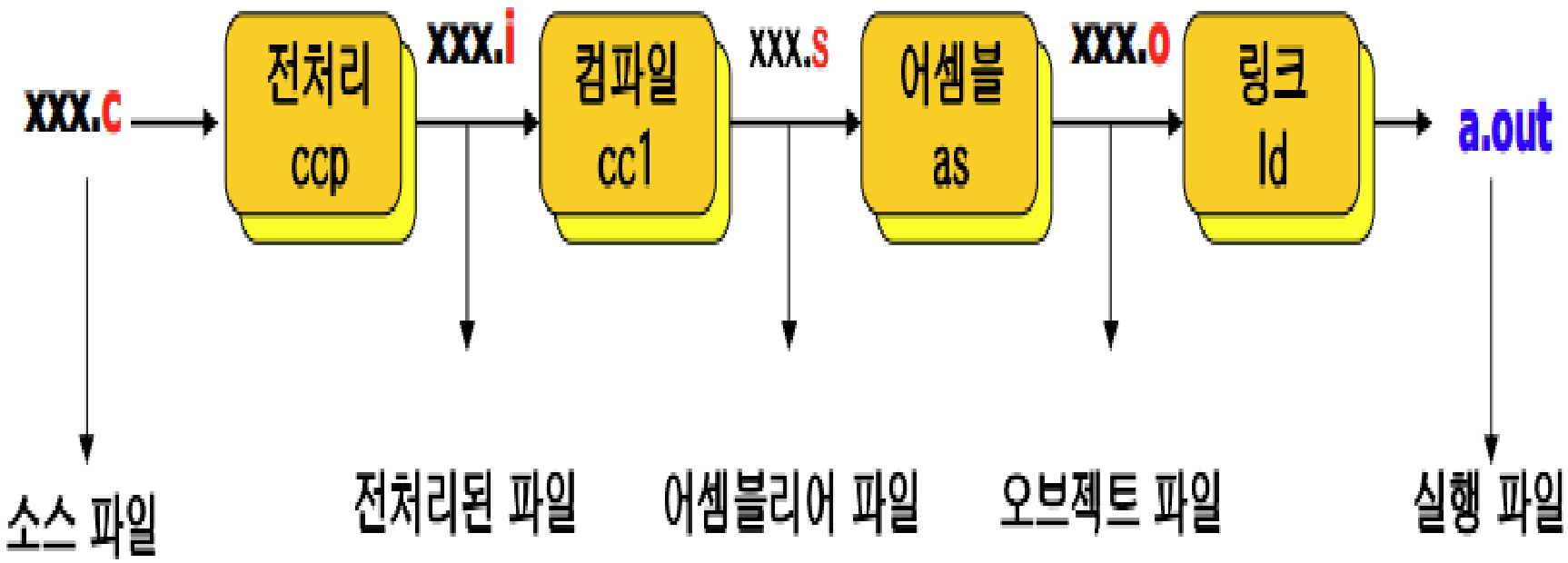
- GNU Compiler Collection
- C , C++, go 등 라이브러리가 포함
- 리처드 스톨먼이 1987년에 만듬

https://gcc.gnu.org/



❖ GCC란?

- 일반적으로 GCC를 컴파일러라고 하지만 정확히 말하면 GCC는 소스 파일을 이용해 실행 파일을 만들 때까지 필요한 프로그램을 차례로 실행시키는 툴



❖ 어셈블리 코드

- objdump -d main.o #디스어셈블

```
[ec2-user@ip-172-31-15-105 day4]$ cat main.s
        .file   "main.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello, GCC"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $.LC0, %edi
        call    puts
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 11.5.0 20240719 (Red Hat 11.5.0-2)"
        .section        .note.GNU-stack,"",@progbits
```

```
Dump of assembler code for function main:
   0x0000000000401126 <+0>:     push   %rbp
   0x0000000000401127 <+1>:     mov    %rsp,%rbp
   0x000000000040112a <+4>:     mov    $0x402010,%edi
   0x000000000040112f <+9>:     call   0x401030 <puts@plt>
   0x0000000000401134 <+14>:    mov    $0x0,%eax
   0x0000000000401139 <+19>:    pop    %rbp
   0x000000000040113a <+20>:    ret
```

```
[ec2-user@ip-172-31-15-105 day4]$ gdb ./a.out
GNU gdb (CentOS Stream) 14.2-3.el9
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
ne at:
d"...
owing URLs:
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for /home/ec2-user/day4/a.out
--Type <RET> for more, q to quit, c to continue without paging--c
(No debugging symbols found in ./a.out)
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000401126 <+0>:     push   %rbp
   0x0000000000401127 <+1>:     mov    %rsp,%rbp
   0x000000000040112a <+4>:     mov    $0x402010,%edi
   0x000000000040112f <+9>:     call   0x401030 <puts@plt>
   0x0000000000401134 <+14>:    mov    $0x0,%eax
   0x0000000000401139 <+19>:    pop    %rbp
   0x000000000040113a <+20>:    ret
End of assembler dump.
(gdb) quit
```

❖ GCC 설치

- sudo yum install gcc

```
[sunjae@localhost ~]$ sudo yum install gcc
```

```
================================================================================
 Package                 Architecture Version                 Repository       Size
================================================================================
Installing:
 gcc                     x86_64       11.5.0-2.el9            appstream        32 M
Upgrading:
 cpp                     x86_64       11.5.0-2.el9            appstream        11 M
 libgcc                  x86_64       11.5.0-2.el9            baseos           88 k
 libgomp                 x86_64       11.5.0-2.el9            baseos          264 k
Installing dependencies:
 glibc-devel             x86_64       2.34-114.el9            appstream        34 k
 glibc-headers           x86_64       2.34-114.el9            appstream       540 k
 kernel-headers          x86_64       5.14.0-496.el9          appstream       3.5 M
 libxcrypt-devel         x86_64       4.4.18-3.el9            appstream        29 k
 make                    x86_64       1:4.3-8.el9             baseos          536 k

Transaction Summary
================================================================================
Install  6 Packages
Upgrade  3 Packages

Total download size: 48 M
Is this ok [y/N]: y
```

❖ 첫 번째 예제 파일

```
#include <stdio.h>

int main() {

        printf("Hello, GCC\n");

        return 0;
}
```

1) cd ~ (Home으로 이동)
2) mkdir gcc_practice (dir 생성)
3) Cd gcc_practice (dir 이동)
4) Vim main.c (main.c 작성)
5) 코드 작성
6) 저장 후 종료

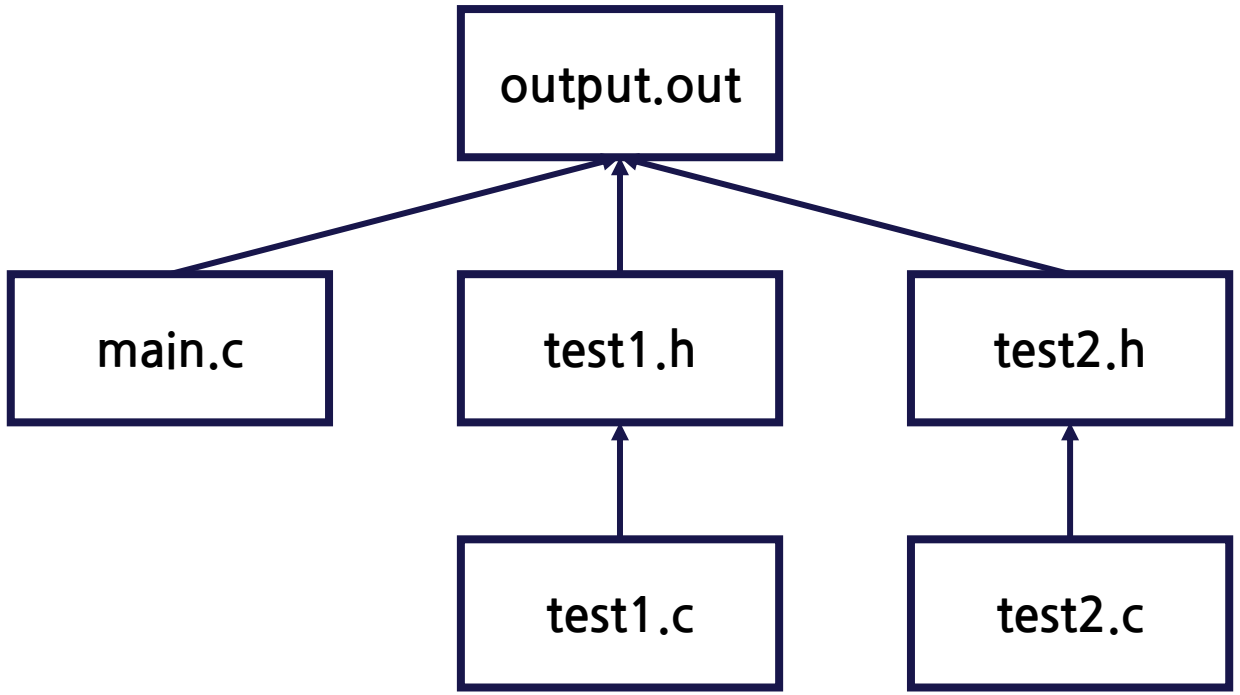
❖ 첫 번째 예제 실행 – 단일 파일 컴파일

```
[sunjae@localhost gcc_practice]$ ls
main.c
[sunjae@localhost gcc_practice]$ gcc main.c
[sunjae@localhost gcc_practice]$ ls
a.out  main.c
[sunjae@localhost gcc_practice]$ ./a.out
Hi, GCC
```

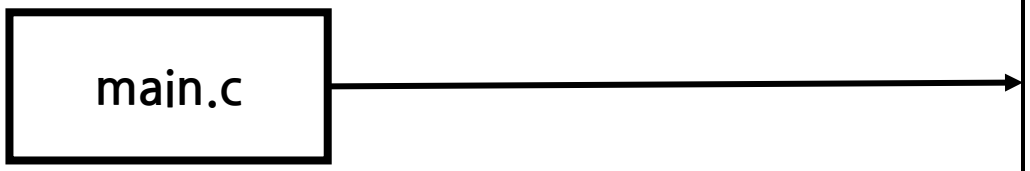
❖ GCC 옵션

옵션	의미
-o	지정된 이름으로 실행파일 생성(지정 안 할 시, a.out으로 생성)
-c	오브젝트 파일 (.o) 생성
-l	같이 링크할 라이브러리 지정
-v	컴파일 수행 메시지 표시
-S	어셈블리 파일 생성
-g	디버깅 옵션, gdb에서 제공하는 정보를 삽입

❖두 번째 예제 – 다수 파일 컴파일



❖두 번째 예제 – 다수 파일 컴파일




```
#include <stdio.h>
#include "test1.h"
#include "test2.h"

int main() {
        printf("Hello, I am GCC\n”);

        print_function1();
        print_function2();

        printf(“Bye!\n");
        return 0;
}
```

❖두 번째 예제 – 다수 파일 컴파일

test1.h > vim test1.h

```
#include <stdio.h>

void print_function1();
```

test1.c > vim test1.c

```
#include "test1.h"

void print_function1() {
        printf("Hello, I am Function1 in test1.c\n");
}
```

❖두 번째 예제 – 다수 파일 컴파일

test2.h > vim test2.h

```
#include <stdio.h>

void print_function2();
```

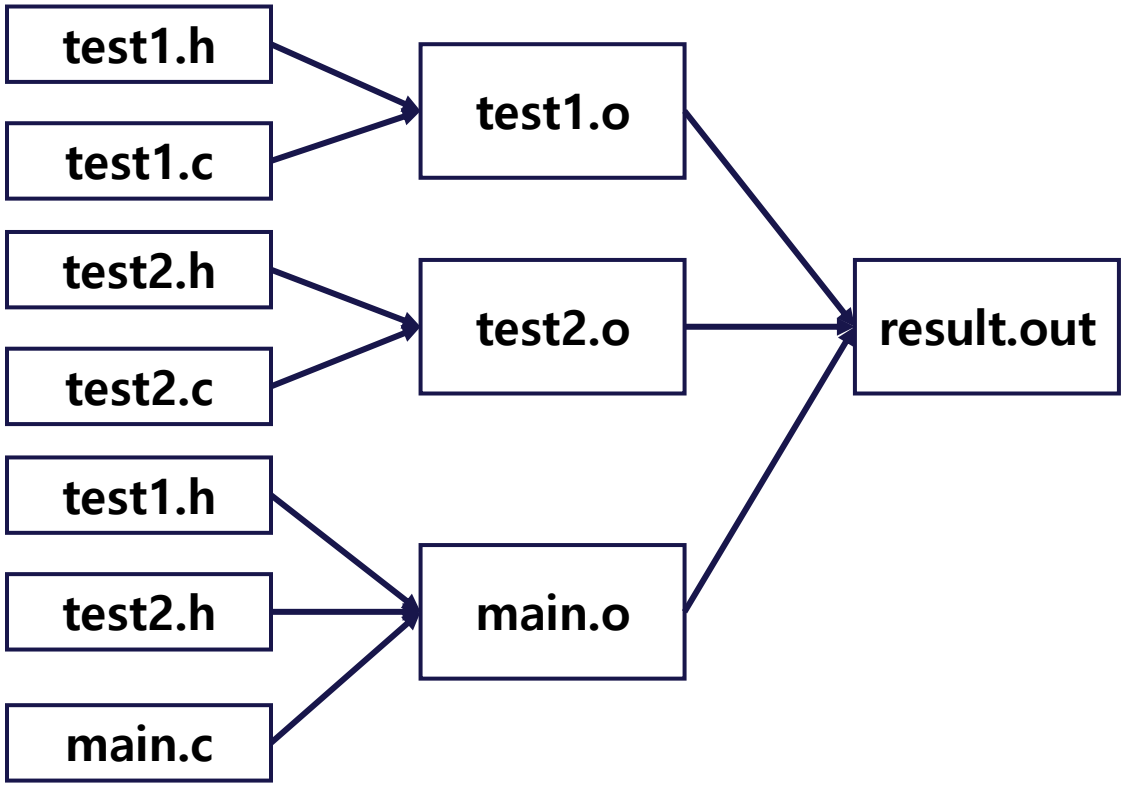
test2.c > vim test2.c

```
#include "test2.h"

void print_function2() {
        printf("Hello, I am Function2 in test2.c\n");
}
```

❖두 번째 예제 – 다수 파일 컴파일

- GCC 순서
  - gcc –c test1.c test1.h
  - gcc –c test2.c test2.h
  - gcc –c main.c test1.h test2.h
  - gcc –o result.out main.o test1.o test2.0



***.gch 파일은 미리 컴파일된 헤더 파일로, GCC가 헤더 파일을 빠르게 처리하기 위해 사용**

❖두 번째 예제 – 다수 파일 컴파일

- GCC 순서
  - 오류 발생 시 vim 편집기를 열고 파일 수정! (오타 조심)

```
[sunjae@localhost complete]$ gcc -c test1.c test1.h
[sunjae@localhost complete]$ ls
main.c  test1.c  test1.h  test1.h.gch  test1.o  test2.c  test2.h
[sunjae@localhost complete]$ gcc -c test2.c test2.h
[sunjae@localhost complete]$ ls
main.c   test1.h      test1.o  test2.h      test2.o
test1.c  test1.h.gch  test2.c  test2.h.gch
[sunjae@localhost complete]$ gcc -c main.c test1.h test2.h
main.c: In function ‘main’:
main.c:8:9: warning: implicit declaration of function ‘printf_function1’; did yo
u mean ‘print_function1’? [-Wimplicit-function-declaration]
    8 |         printf_function1();
      |         ^~~~~~~~~~~~~~~~
      |         print_function1
main.c:9:9: warning: implicit declaration of function ‘printf_function2’; did yo
u mean ‘print_function2’? [-Wimplicit-function-declaration]
    9 |         printf_function2();
      |         ^~~~~~~~~~~~~~~~
      |         print_function2
[sunjae@localhost complete]$ vim main.c
[sunjae@localhost complete]$ gcc -c main.c test1.h test2.h
[sunjae@localhost complete]$ ls
main.c  test1.c  test1.h.gch  test2.c  test2.h.gch
main.o  test1.h  test1.o      test2.h  test2.o
[sunjae@localhost complete]$ gcc -o result.out main.o test1.o test2.o
[sunjae@localhost complete]$ ls
main.c  result.out  test1.h      test1.o  test2.h      test2.o
main.o  test1.c     test1.h.gch  test2.c  test2.h.gch
[sunjae@localhost complete]$ ./result.out
HI, I am main
Hello, I am Fuction1 in test1.c
Hello, I am Function2 in test2.c
bye~!
```

Make

❖ Make

- 소스 코드를 빌드(컴파일)하기 위한 자동화 도구
- 소스 파일 간의 의존성을 관리하고, 변경된 파일만 다시 컴파일하여 효율적인 빌드 가능
- 의존성 관리 : 소스 파일 간의 의존 관계를 정의하고, 필요한 경우에만 빌드를 수행
- 자동화 : 복잡한 빌드 과정(예: 컴파일, 링크)을 자동으로 처리하여 개발자의 수고를 덜어준다
- Makefile 사용 : 빌드 규칙과 의존성을 정의한 텍스트 파일(Makefile)을 바탕으로 작동

* Makefile: 목적파일, 의존성, 명령어, 매크로 등을 활용하여 컴파일을 쉽게 하기 위해 사용하는 make파일의 설정 파일

Make

❖ Make 설치

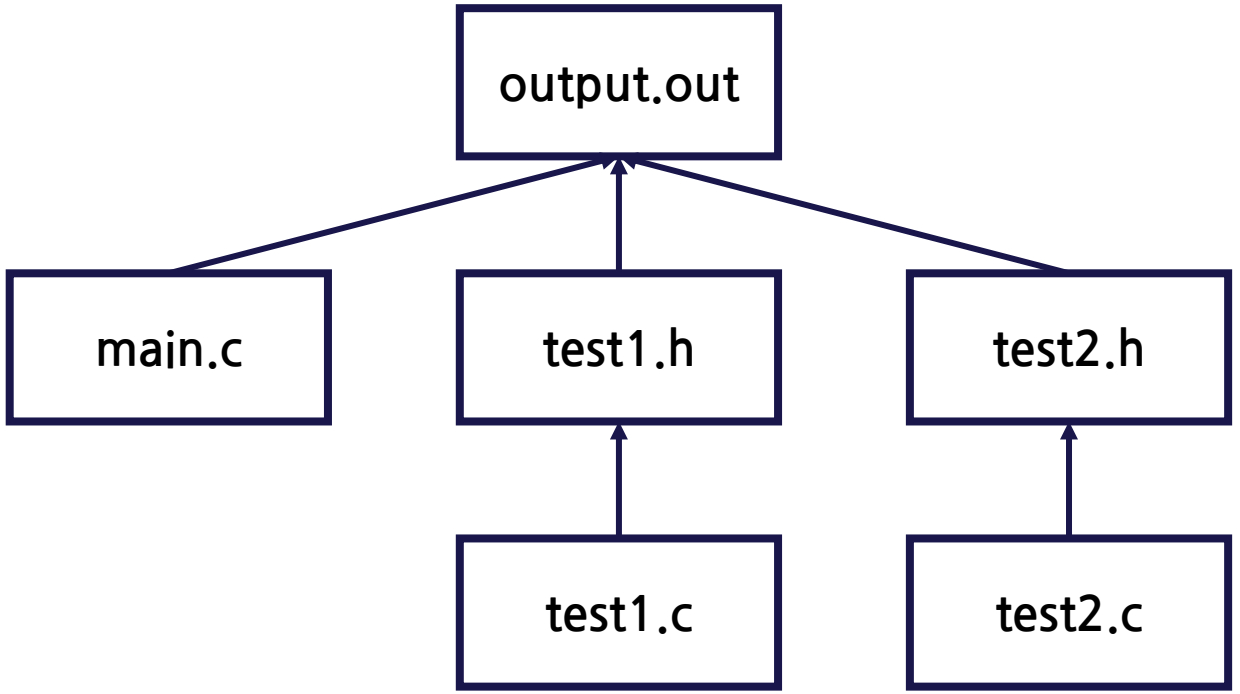
- sudo yum install make 명령어로 설치가 가능
- 하지만, CentOS의 경우 기본적으로 설치 되어 있음

```
[sunjae@localhost ~]$ sudo yum install make
[sudo] password for sunjae:
CentOS Stream 9 - BaseOS                        8.9 kB/s | 8.2 kB     00:00
CentOS Stream 9 - AppStream                     9.5 kB/s | 8.3 kB     00:00
CentOS Stream 9 - Extras packages               8.2 kB/s | 8.9 kB     00:01
Package make-1:4.3-8.el9.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

Make

❖ Make 예제

- gcc –c test1.c test1.h
- gcc –c test2.c test2.h
- gcc –c main.c test1.h test2.h
- gcc –o result.out main.o test1.o test2.o



Make

❖ Make 예제

- gcc –c test1.c test1.h
- gcc –c test2.c test2.h
- gcc –c main.c test1.h test2.h
- gcc –o result.out main.o test1.o test2.o

Makefile > vim Makefile

```
result.out : main.o test1.o test2.o
        gcc -o result.out main.o test1.o test2.o

main.o : main.c test1.h test2.h
        gcc -c main.c test1.h test2.h
test1.o : test1.c test1.h
        gcc -c test1.c test1.h
test2.o : test2.c test2.h
        gcc -c test2.c test2.h
```

Make

❖ Make 예제

- > rm *.o
- > rm *.gch
- > rm *.out
- > make

```
[sunjae@localhost complete]$ ls
main.c  Makefile  test1.c  test1.h  test2.c  test2.h
[sunjae@localhost complete]$ make
gcc -c main.c test1.h test2.h
gcc  -c test1.c test1.h
gcc -c test2.c test2.h
gcc -o result.out main.o test1.o test2.o
[sunjae@localhost complete]$ ls
main.c  Makefile    test1.c  test1.h.gch  test2.c  test2.h.gch
main.o  result.out  test1.h  test1.o      test2.h  test2.o
[sunjae@localhost complete]$ ./result.out
HI, I am main
Hello, I am Fuction1 in test1.c
Hello, I am Function2 in test2.c
bye~!
```

Make

❖ Make 매크로

- 변수 이름들은 $사용
  - $(변수) : 변수
  - $@ : 현재 목표 파일(target)
  - $<: 현재 목표 파일보다 더 최근에 갱신된 파일

```
1  CC=gcc
2  CFLAGS=-c -g -Wall
3  OBJS=main.o test1.o test2.o
4  SRCS=test1.c test2.c
5  HEARS=test1.h test2.h
6  LIBS = -lpthread
7  TARGET=result.out
8  $(TARGET): $(OBJS)
9      $(CC) -o $(TARGET) $(OBJS) $(LIBS)
10 %.o: %.c %.h
11     $(CC) $(CFLAGS) -c $(SRCS) $(HEARS)
12 clean :
13     rm -f *.o
14     rm -f *.gch
15     rm -f $(TARGET)
```

Make

❖ Make 매크로

```
[ec2-user@ip-172-31-15-105 make]$ ls
main.c  Makefile  test1.c  test1.h  test2.c  test2.h
[ec2-user@ip-172-31-15-105 make]$ make
gcc -c -g -Wall   -c -o main.o main.c
gcc -c -g -Wall -c test1.c test2.c test1.h test2.h
gcc -o result.out main.o test1.o test2.o -lpthread
[ec2-user@ip-172-31-15-105 make]$ ./result.out
Hello, I am Main
Hello, I am Function1 in test1.c
Hello, I am Function2 in test2.c
Bye!
```

Make

```
CC=gcc
CFLAGS=-c -g -Wall
OBJS=main.o test1.o test2.o
SRCS=test1.c test2.c
HEARS=test1.h test2.h
LIBS = -lpthread
TARGET=result.out

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

%.o: %.c %.h
	$(CC) $(CFLAGS) -c $(SRCS) $(HEARS)

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

```
CC=gcc
CFLAGS=-g -Wall
OBJS=main.o test1.o test2.o
LIBS = -lpthread
TARGET=result.out

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

main.o : main.c test1.h test2.h
	gcc -c main.c
test1.o : test1.c test1.h
	gcc -c test1.c
test2.o : test2.c test2.h
	gcc -c test2.c

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

마지막 실습

❖ 계산기 프로그램 만들기

- 더하기, 뺄셈, 곱하기, 나누기
- Main.c Add.c substract.c, multiply.c, divide.c
- 다한사람 실습 조교님한테 검사받기