KWANGWOON UNIVERSITY

2019년 1학기 **시스템프로그래밍실습** 3주차

Linux-based Programming

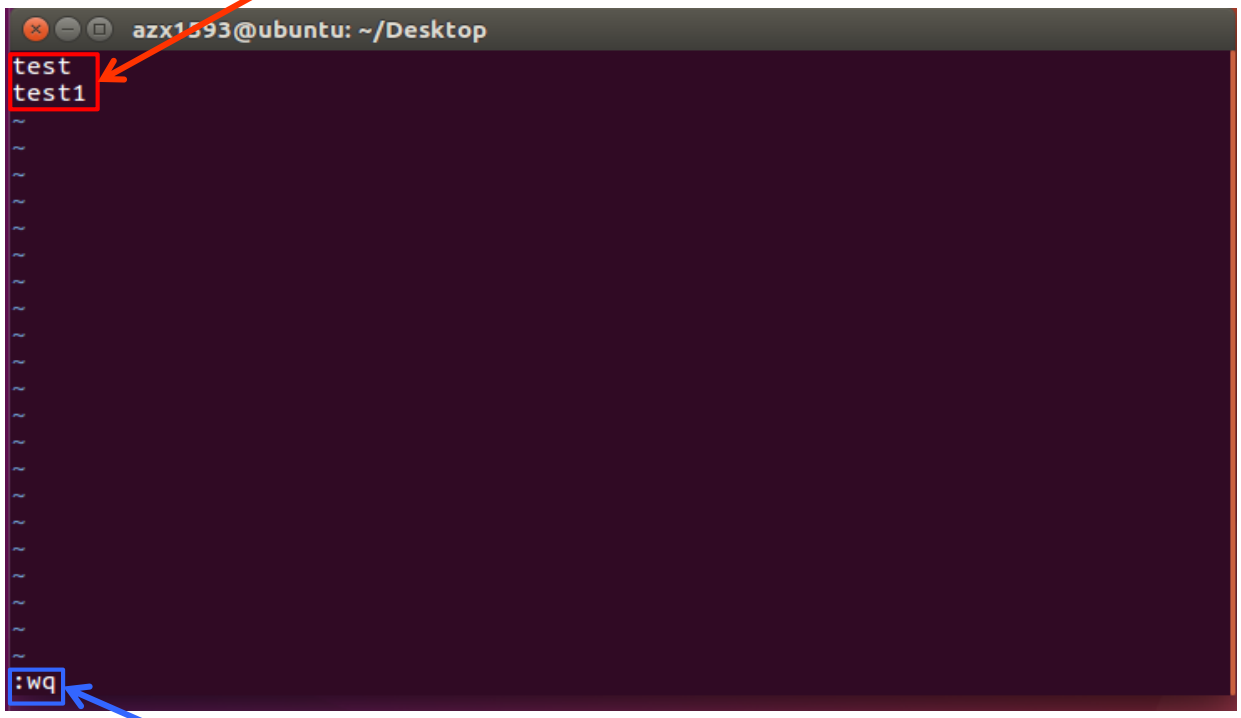
System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

Contents

- vi Editor
- gcc compiler
- make
- gdb
- 실습

vi Editor (1/2)

- **vi 에디터**
  - Linux 기본 편집기
  - 실행 화면



- **vim 패키지 설치**
  - 기본 설치된 vi는 기능이 제한적이므로 vim 패키지 설치 필요
  - **$ sudo apt-get install vim**

vi Editor (2/2)

- **vi 모드**
  - 명령모드
    - 한/두 문자로 구성된 vi 전용 명령어를 사용하는 모드
    - vi 진입 시 기본 모드
  - 입력모드에서 [esc] 키로 진입 가능
  - 입력모드
    - vi 편집 화면에서 문자를 입력 할 수 있는 상태를 의미
    - 명령 모드에서 [esc] 후 [i] 등의 키로 진입 가능

- **vi 시작과 종료**
  - 시작
  - vi file_name ➔ vi를 시작하여 지정한 파일 편집
  - 종료
    - :wq ➔ 데이터를 저장하고 종료
    - :q ➔ 데이터를 저장하지 않고 종료
    - :q! ➔ 데이터를 저장하지 않고 강제 종료

vi Editor Options (1/3)

커서 이동

h	← 이동
j	↓ 이동
k	↑ 이동
l	→ 이동
Backspace	커서가 있는 행에서 커서를 왼쪽으로 옮김
Space	커서가 있는 행에서 커서를 오른쪽으로 옮김

입력 모드

i	커서 앞으로 문장 삽입
I	행의 시작 부분에서 문장 삽입
a	커서 뒤로 문장 삽입
A	행의 끝 부분에서 문장 삽입
o	커서가 위치한 행의 아래에 문장 삽입
O	커서가 위치한 행의 위에 문장 삽입

수정

r	커서가 있는 문자를 변경
R	커서가 있는 부분부터 글자 덮어서 씀
s	한 글자를 삭제 한 후 문장 삽입
S	커서가 있는 문장을 삭제하고 문장 삽입
C	커서가 있는 행에서 커서를 왼쪽으로 옮김

삭제

cc	현재 행 삭제 후 문장 입력
cw	커서가 있는 문자 삭제 후 문장 입력
x	커서가 있는 한 글자 삭제
X	커서 앞 한 글자 삭제
D	커서가 있는 부분의 뒷 부분의 행을 삭제
dd	커서가 있는 한 행 삭제

vi Editor Options (2/3)

커서 이동

nY	커서가 있는 행 부터 n행 만큼 복사
yy	커서가 있는 행 복사
p	커서 아래 복사된 문자열 붙임
P	커서 위에 복사된 문자열 붙임

입력 모드

:w	vi 파일을 저장
:w 파일 이름	파일이름으로 저장
:wq	저장 후 종료
:wq!	저장 후 강제 종료

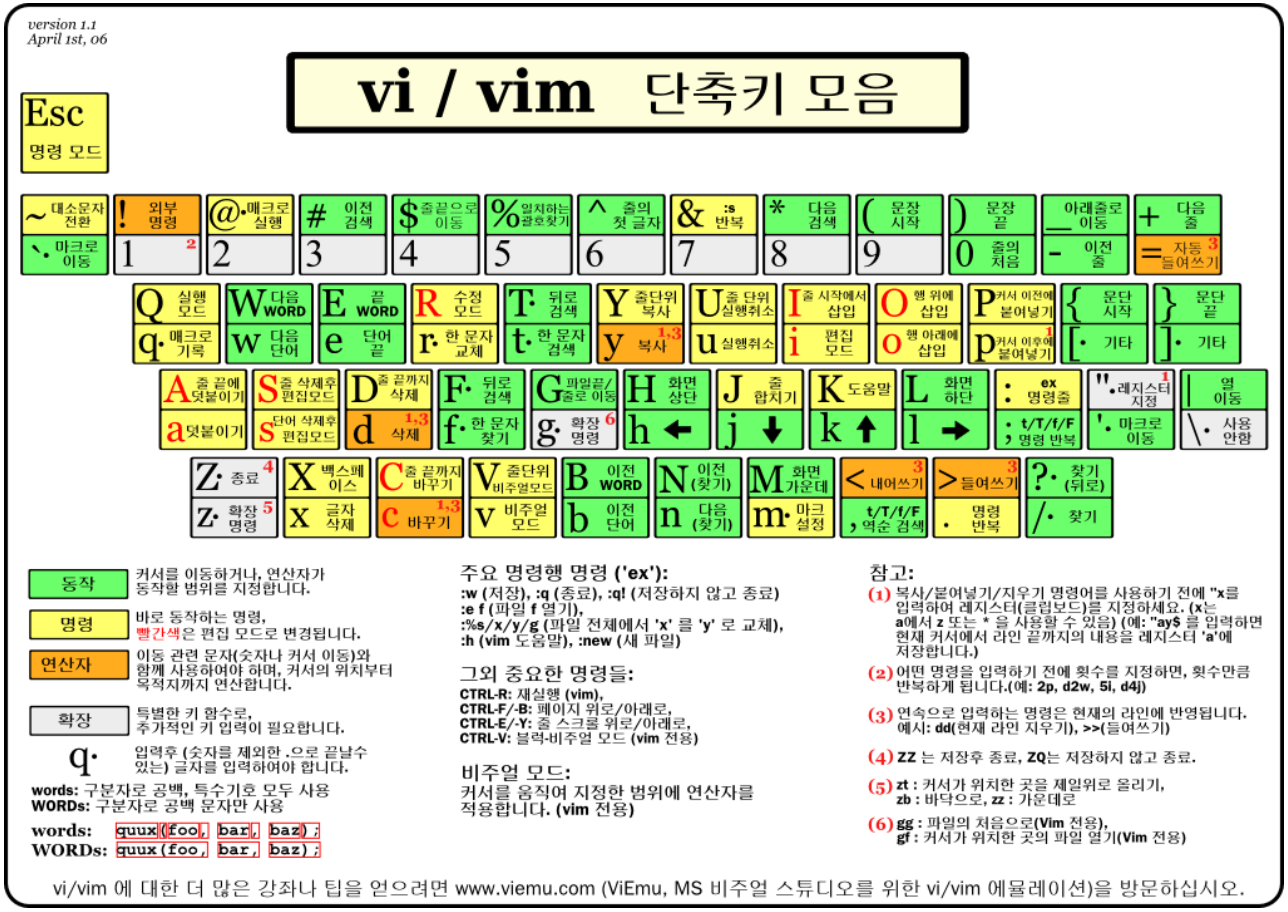
수정

:e 파일 이름	vi 를 종료하지 않고 해당 파일 편집
:e!	현재 편집하고 있는 파일 다시 부르기
:e#	한 글자를 삭제 한 후 문장 삽입

삭제

:set number	편집기의 라인 표시
:set nu	편집기의 라인 표시
:set non	편집기의 라인 표시 없애기

vi Editor 단축키



- Reference
  - 영문 버전: http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html
  - 한글 버전: https://kldp.org/node/102947

GCC Compiler

- **컴파일**
  - hello.c 파일 준비

```c
#include <stdio.h>

void main()
{
   printf("Hello World! \n");
}
```

  - 컴파일
    - $ gcc (파일명)
      - e.g. $ gcc hello.c

```
azx1593@ubuntu:~/Desktop$ ls
hello.c
azx1593@ubuntu:~/Desktop$ gcc hello.c
azx1593@ubuntu:~/Desktop$ ls
a.out  hello.c
azx1593@ubuntu:~/Desktop$
```

    - 실행파일 이름을 정해주지 않고 컴파일 한 경우에는 a.out으로 자동 생성됨

- **실행**
  - $ ./a.out

```
azx1593@ubuntu:~/Desktop$ ./a.out
Hello World
```

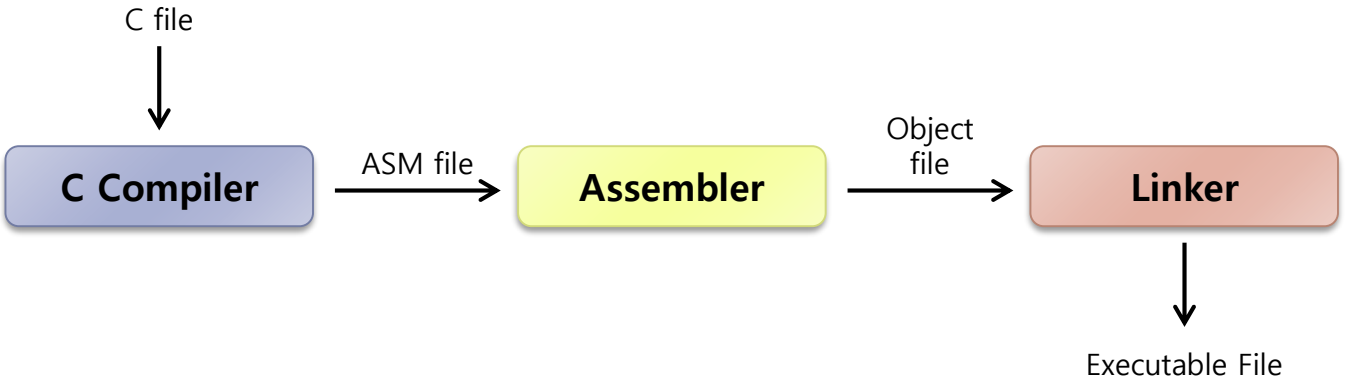
  - 실행 파일명 지정
    - $ gcc -o (실행파일명) (소스파일명)
      - e.g. $ gcc -o hell hello.c

```
azx1593@ubuntu:~/Desktop$ ls
a.out  hello.c
azx1593@ubuntu:~/Desktop$ gcc -o hell hello.c
azx1593@ubuntu:~/Desktop$ ls
a.out  hell  hello.c
azx1593@ubuntu:~/Desktop$
```

make (1/6)

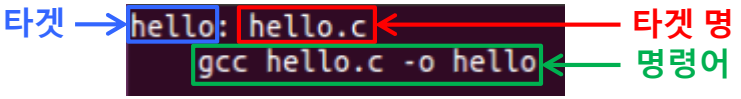
- **필요성**
  - 컴파일 과정을 자동화하기 위해 사용
  - gcc 컴파일러의 다양한 옵션들을 컴파일 할 때마다 입력한다면 많은 시간이 소요될 것임

- **Makefile**
  - 컴파일 할 소스 파일과 컴파일 옵션에 관해 정의해 놓은 스크립트 파일
  - 아래 그림과 같이 빌드는 여러 과정으로 구성되어 있는데, 이를 통해 한 번 만에 빌드를 수행할 수 있도록 함



make (2/6)

- **Makefile (cont'd)**
  - 작업할 대상 ➔ **타겟**
    - 타겟 명을 의미하기 위해 이름 뒤에 ':'
    - 타겟 명 다음 줄에는 실행할 명령을 명시



  - 여러 개 빌드
    - "all:" 이라는 타겟 뒤에 새로운 하위 타겟("hello1 hello2") 추가
    - $ make는 "all" 타겟에서 새로운 타겟 "hello1 hello2" 를 확인하고 실행
    - hello2만 빌드하고 싶다면, "$make hello2" 실행

```
all: hello1.c hello2.c

hello1: hello1.c
    gcc hello1.c -o hello1

hello2: hello2.c
    gcc hello2.c -o hello2
```

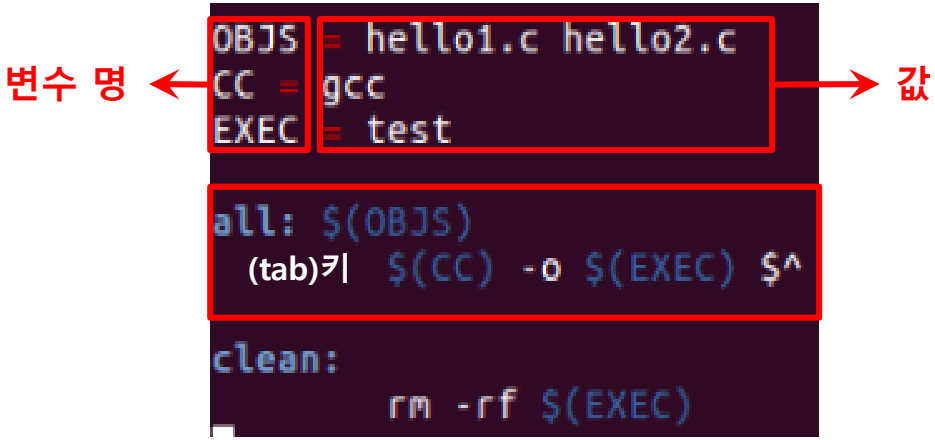
타겟(Target)	명령어가 수행되어 나온 결과를 저장한 파일
타겟 명(Target Name)	소스 파일 명
명령어(Command)	실행 명령어

make (3/6)

- **Makefile (cont'd)**
  - make 수행 시 타겟을 명시하지 않는 경우
    - i.e. $ make
    - Makefile의 제일 첫 번째 타겟에 해당하는 명령 수행
  - make 수행 시 타겟을 명시하는 경우
    - i.e. $ make hello1
    - 해당 타겟의 명령 수행

make (4/6)

- **변수사용과 대체**
  - 변수는 '이름 = 값'의 형태로 지정
  - 변수는 '$(이름)'형태로 사용
  - 타겟은 '$@'로 대체, 타겟 명은 '$^'로 대체
  - Example



make (5/6)

- **프로그램 실행 하기**
  - 실행 결과

```
azx1593@ubuntu:~/Desktop$ ls
hello1.c  hello2.c  hello2.h  Makefile  TT
azx1593@ubuntu:~/Desktop$ make
gcc -o test hello1.c hello2.c
azx1593@ubuntu:~/Desktop$ ls
hello1.c  hello2.c  hello2.h  Makefile  test  TT
azx1593@ubuntu:~/Desktop$ ./test
Hello World! 1
Hello World! 2
```

  - 소스

```c
#include <stdio.h>
#include "hello2.h"

int main(int argc, char **argv)
{
   printf("Hello World! 1 \n");
   hello2_func();
   return 0;
}
```

hello1.c

```
OBJS = hello1.c hello2.c
CC = gcc
EXEC = test

all: $(OBJS)
	$(CC) -o $(EXEC) $^

clean:
	rm -rf $(EXEC)
```

Makefile

```c
#include <stdio.h>

void hello2_func()
{
   printf("Hello World! 2 \n");
}
```

hello2.c

```c
#ifndef __HELLO2_H__
#define __HELLO2_H__
int hello2_func(void);
#endif
```

hello2.h

make (6/6)

- **매개변수**
  - Integer형 변수
    - 입력 인자의 개수를 저장
  - char* 배열 형 변수
    - 입력 인자들을 배열 형으로 저장

example.c

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i;
    printf("The Number Of Inputted Variable is %d \n", argc);
    printf("and they are ");

    for(i=0; i<argc; i++)
    {
      printf("%s ", argv[i]);
    }

    printf("\n");

    return 0;
}
```

수행 결과

```
azx1593@ubuntu:~/Desktop/TT$ ls
example.c
azx1593@ubuntu:~/Desktop/TT$ gcc example.c
azx1593@ubuntu:~/Desktop/TT$ ls
a.out  example.c
azx1593@ubuntu:~/Desktop/TT$ ./a.out  hello! system Proqramming lab. Class!
The Number Of Inputted Variable is 6
and they are ./a.out hello! system Proqramming lab. Class!
azx1593@ubuntu:~/Desktop/TT$
```

gdb (1/10)

- **GDB(GNU DeBugger) 란?**
  - 어떤 프로그램이 수행되는 도중 그 프로그램 내에서 어떤 일이 일어나는지를 볼 수 있게 해 주는 툴.

- **GDB 기능**
  - 프로그램을 수행시킨다.
  - 어떤 특별한 조건에서 프로그램의 수행을 stop 시킨다.
  - 프로그램이 stop 된 상태에서 그 프로그램의 내부를 볼 수 있다.
  - 프로그램의 일부분을 수정한다.
  - Stop 된 프로그램을 continue 시킨다.

gdb (2/10)

- **gdb options**

list	현재 위치에서 소스 파일의 내용을 10줄 보여준다 ex) list 2, 15 : 소스 파일의2 ~ 15 까지를 보여준다.
run	프로그램을 시작한다.(break가 있다면 break까지 실행)
break	특정 라인이나 함수에 정지점을 설정한다. break function : 현재 파일 안의 함수 function에 정지점을 설정한다. break file:function : 파일 file안의 function에 정지점을 설정한다. watch : 감시점 설정(감시점은 어떤 사건이 일어날 때에만 작동한다) until : 실행 중 line까지 실행
clear	특정 라인이나 함수에 있던 정지점을 삭제한다.
delete	몇몇 정지점이나 자동으로 출력되는 표현을 삭제한다.
next	다음 행을 수행한다. 서브루틴을 호출하면서 계속 수행한다.
step	한 줄씩 실행 시킨다.
print	print expr : 수식의 값을 보여준다.
display	현재 display된 명령의 목록을 보여준다.
bt	프로그램 스택을 보여준다. (backtrace)
kill	디버깅 중인 프로그램의 실행을 취소한다.
file	file program : 디버깅할 프로그램으로서 파일을 사용한다.
count	continue : 현재 위치에서 프로그램을 계속 실행한다.
help	명령에 관한 정보를 보여주거나 일반적인 정보를 보여준다.
quit	gdb에서 빠져나간다.

gdb (3/10)

- **디버깅하기 위한 컴파일 옵션**
  - $ gcc –o 실행파일 –g 소스파일

- **GDB 시작 – GDB 프롬프트"(gdb)" 가 나옴**
  - $ gdb 실행파일
  - 현재 수행중인 프로그램(프로세스 번호)를 디버깅
    - 매개변수의 사용 여부에 따라 run(r) 혹은 run arg1 arg2 …

- **GDB 디버깅 진행**
  - step(s) 혹은 next(n)

- **GDB 종료**
  - quit(q) 혹은 ^D (ctrl + D)

gdb (4/10)

- **Breakpoint**
  - Breakpoint란?
    - 프로그램의 수행을 정지시키는 지점
  - 특정 함수에 breakpoint 설정
    - break function 혹은 b function
  - 프로그램 소스 줄에 breakpoint 설정
    - break linenum 혹은 b linenum

gdb (5/10)

- **Breakpoint (cont'd)**
  - 현재 위치에서 상대적 위치에 breakpoint 설정
    - break +offset 혹은 break –offset
  - 조건 breakpoint 설정
    - break ... if cond
    - e.g. **(gdb) break 10 if i == 5**
      - 변수 i의 값이 5일 경우 10번 행에 breakpoint 설정
  - Breakpoint 설정지점 보기
    - info breakpoints

gdb (6/10)

- **Watchpoint**
  - Watchpoint 란?
    - 특정 식의 값이 변경되거나 읽혀질 때 프로그램의 수행이 stop 하는 특별한 breakpoint
  - 프로그램에 의하여 특정 변수가 쓰여지면(write) breakpoint 형성
    - watch 변수
  - 프로그램에 의하여 특정 변수가 읽혀지면(read) breakpoint 형성
    - rwatch 변수
  - 특정 변수가 써지거나(write) 혹은 읽혀지면(read) breakpoint 형성
    - awatch 변수
  - 설정된 watchpoint 보기
    - info watchpoints

실행 예시

```
(gdb) watch j
Hardware watchpoint 2: j
(gdb) c
Continuing.
j is 0.000000

Hardware watchpoint 2: j

Old value = 0
New value = 1
main () at example1.c:10
10                  printf("j is %f \n" , j);
```

gdb (7/10)

- 스택 전체 보기
  - backtrace 혹은 bt

```c
#include <stdio.h>

int kk(void)
{
  int k = 5;
  printf("%d", k);
}

void main()
{
    int i;
    double j;

    kk();
    for( i=0; i<5 ; i++)
    {
        j= i / 2 + i;
        printf("j is %f \n" , j);
    }
}
```

example1.c

```
(gdb) list 10
5           int k = 5;
6           printf("%d", k);
7       }
8
9       void main()
10      {
11          int i;
12          double j;
13
14          kk();
(gdb) b 14
Breakpoint 1 at 0x400554: file example1.c, line 14.
(gdb) r
Starting program: /home/azx1593/Desktop/TT/gg/anything

Breakpoint 1, main () at example1.c:14
14          kk();
(gdb) n
15          for( i=0; i<5 ; i++)
(gdb) bt
#0  main () at example1.c:15
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/azx1593/Desktop/TT/gg/anything

Breakpoint 1, main () at example1.c:14
14          kk();
(gdb) s
kk () at example1.c:5
5           int k = 5;
(gdb) bt
#0  kk () at example1.c:5
#1  0x0000000000400559 in main () at example1.c:14
```

실행 결과

gdb (8/10)

- **소스 라인 프린트**
  - list linenum
    - 해당 줄을 기준으로 출력
  - list function
    - 해당 함수의 정의를 출력
  - list
    - 현재 줄의 아래쪽 소스를 출력
  - list –
    - 현재 줄의 위쪽 소스를 출력

gdb (9/10)

- **변수 값 보기**
  - print or p 변수
  - print /f expr // f는 format (예: /d /o /x)
  - info locals // 지역 변수들 정보 출력
  - info variables // 전역 변수들 정보 출력

- **메모리 값 보기**
  - x 명령어는 메모리 특정 범위의 값들을 확인하는데 사용하는 명령어
  - 사용 방법
    - x/[범위][출력format][단위]
    - x/[범위][단위][출력format]
    - x /nfu 주소값
      - n : 개수
      - f : format
      - u :단위 (e.g. b(1), h(2), w(4))

B(1byte) h(2byte) w(4byte)	
/t	2진수
/o	8진수
/d	10진수
/u	Unsigned int
/x	16진수
/f	부동소수점 값
/s	문자열

gdb (10/10)

- 변수 값 수정(대입)
  - print x = 4

- 다른 곳으로 jump
  - jump linespec

KWANGWOON UNIVERSITY

실습

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

make 실습 (1/2)

- **Makefile 작성**
  - Makefile을 생성하고 컴파일 후 test 실행파일을 실행하여 “this is A, this is B” 를 출력하시오.

- **결과**

```
azx1593@ubuntu:~/sslab/one$ ls
a.c  a.h  b.c  b.h  main.c  Makefile
azx1593@ubuntu:~/sslab/one$ make
gcc -o test main.c b.c a.c
azx1593@ubuntu:~/sslab/one$ ls
a.c  a.h  b.c  b.h  main.c  Makefile  test
azx1593@ubuntu:~/sslab/one$ ./test
----------
this is A
this is B
----------
azx1593@ubuntu:~/sslab/one$
```

- **코드**

a.h

```
#ifndef __A_H__
#define __A_H__
#include <stdio.h>

void a_func();

#endif
```

b.h

```
#ifndef __B_H__
#define __B_H__
#include <stdio.h>
void b_func();

#endif
```

a.c

```
#include "a.h"

void a_func()
{
 printf("this is A\n");
}
```

b.c

```
#include "b.h"

void b_func()
{
  printf("this is B\n");
}
```

main.c

```
#include "a.h"
#include "b.h"

int main()
{
  printf("----------\n");
  a_func();
  b_func();
  printf("----------\n");
  return 0;
}
```

Makefile

?

make 실습 (2/2)

- **Makefile 작성**
  - Makefile을 생성하고 컴파일 후 “client, server”를 출력하시오.

- **결과**

```
azx1593@ubuntu:~/sslab/second$ ls
cli.c  Makefile  srv.c
azx1593@ubuntu:~/sslab/second$ make
gcc -o srv srv.c
gcc -o cli cli.c
azx1593@ubuntu:~/sslab/second$ ls
cli  cli.c  Makefile  srv  srv.c
azx1593@ubuntu:~/sslab/second$ ./cli
client
azx1593@ubuntu:~/sslab/second$ ./srv
server
```

- **코드**

```c
#include <stdio.h>

int main()
{
  printf("client\n");
  return 0;
}
```

cli.c

```c
#include <stdio.h>

int main()
{
  printf("server\n");
  return 0;
}
```

srv.c

Makefile

gdb 실습

```c
#include <stdio.h>
void main()
{
    int i;
    double j;

    for( i=0; i<5 ; i++)
    {
        j= i / 2 + i;
        printf("j is %f \n" , j);
    }
}
```

example1.c

```
azx1593@ubuntu:~/Desktop/TT/gg$ ls
example1.c
azx1593@ubuntu:~/Desktop/TT/gg$ gcc -o example -g example1.c
azx1593@ubuntu:~/Desktop/TT/gg$ ls
example  example1.c
azx1593@ubuntu:~/Desktop/TT/gg$ ./example
j is 0.000000
j is 1.000000
j is 3.000000
j is 4.000000
j is 6.000000
```

실행 결과

```
azx1593@ubuntu:~/Desktop/TT/gg$ ls
example  example1.c
azx1593@ubuntu:~/Desktop/TT/gg$ gdb example
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
```

⋮

```
Type "apropos word" to search for commands related to "
Reading symbols from example...done.
(gdb) list
1       #include <stdio.h>
2       void main()
3       {
4           int i;
5           double j;
6
7           for( i=0; i<5 ; i++)
8           {
9               j= i / 2 + i;
10              printf("j is %f \n" , j);
(gdb) b 9
Breakpoint 1 at 0x400537: file example1.c, line 9.
(gdb) r
Starting program: /home/azx1593/Desktop/TT/gg/example

Breakpoint 1, main () at example1.c:9
9               j= i / 2 + i;
(gdb) s
10              printf("j is %f \n" , j);
(gdb) n
j is 0.000000
7           for( i=0; i<5 ; i++)
(gdb) quit
A debugging session is active.

        Inferior 1 [process 9115] will be killed.

Quit anyway? (y or n) y
azx1593@ubuntu:~/Desktop/TT/gg$
```

소스 내용 10줄 씩 출력

Line 9에 breakpoint

프로그램 시작

한 줄 씩 실행

다음 행 출력

gdb 종료