

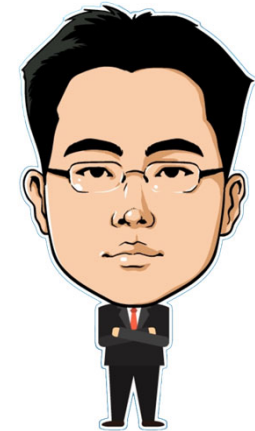
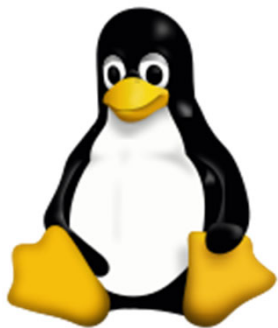
GNU/Linux

GNU 프로젝트 (GNU Projects)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



목 차



- **GCC**

- **GDB**

- **GNU make**



GCC



- **GCC**

- gcc 옵션

- 분할 컴파일



- **GDB**

- **GNU make**



GCC

- **GCC**(GNU Compiler Collection) : gcc.gnu.org

- **GNU C Compiler**

- **1987년 5월, Richard Stallman**
 - GNU 프로젝트의 컴파일러로 작성
 - » 1987년 3월, **GCC 0.9** (first beta release)
 - » 1987년 5월, **GCC 1.0**
 - » 1987년 12월, **C++ 컴파일러 확장** - GCC 1.15.3 (g++)



- **GNU Compiler Collection**

- **대부분의 유닉스 계열 운영체제의 표준 컴파일러로 채택**
 - GNU 운영체제의 공식 컴파일러 그리고 GNU/Linux 및 BSD 계열 운영체제
 - 공식 지원 언어
 - » **C(gcc), C++(g++)**, Objective-C, Fortran(**gfortran**), Ada(**GNAT**), Go(**gccgo**)
 - » **Java(gcj) : GCC 7.1 버전부터 지원 중단**
 - 2018년 1월, **GCC 7.3**
- **GNU GPL**(GNU General Public License)
 - 2007년 10월, GCC 4.2.2 부터 **GPLv3**

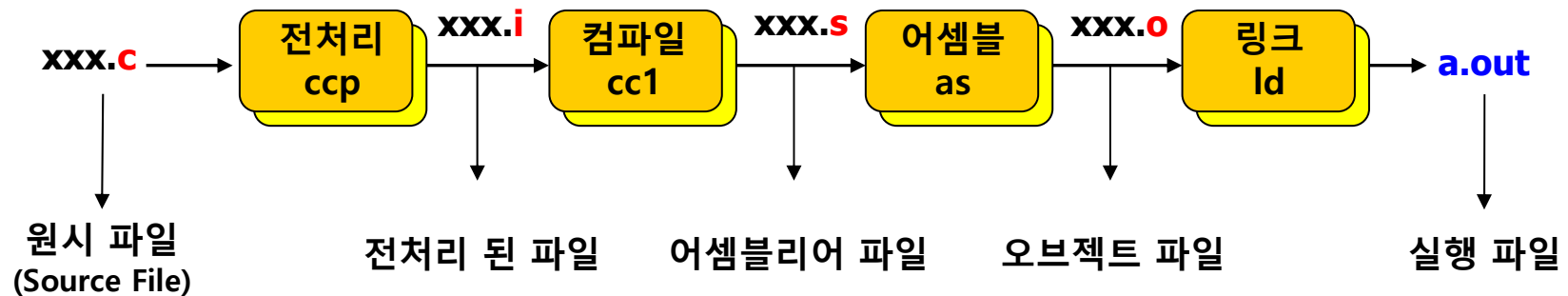


gcc : 컴파일 과정 (1/3)

● gcc : 컴파일 과정

○ 프로그램을 차례로 실행 시키는 툴

- 일반적으로 gcc 를 컴파일러 라고 한다.
- 소스 파일을 이용해 실행 파일을 만들 때까지 필요한 프로그램을 차례로 실행시키는 툴이다.



```
[clickseo@localhost ~]$ gcc hello.c
```

```
[clickseo@localhost ~]$ ./a.out
```

```
Hello World!!!
```

gcc : 컴파일 과정 (2/3)

- gcc : 파일 확장자

- gcc 는 파일 확장자에 따라 처리 방법이 다르다.

- 대표적인 확장자 : **.c**

- gcc 로 전처리, 컴파일, 어셈블, 링크 과정을 거쳐야 실행 파일이 완성

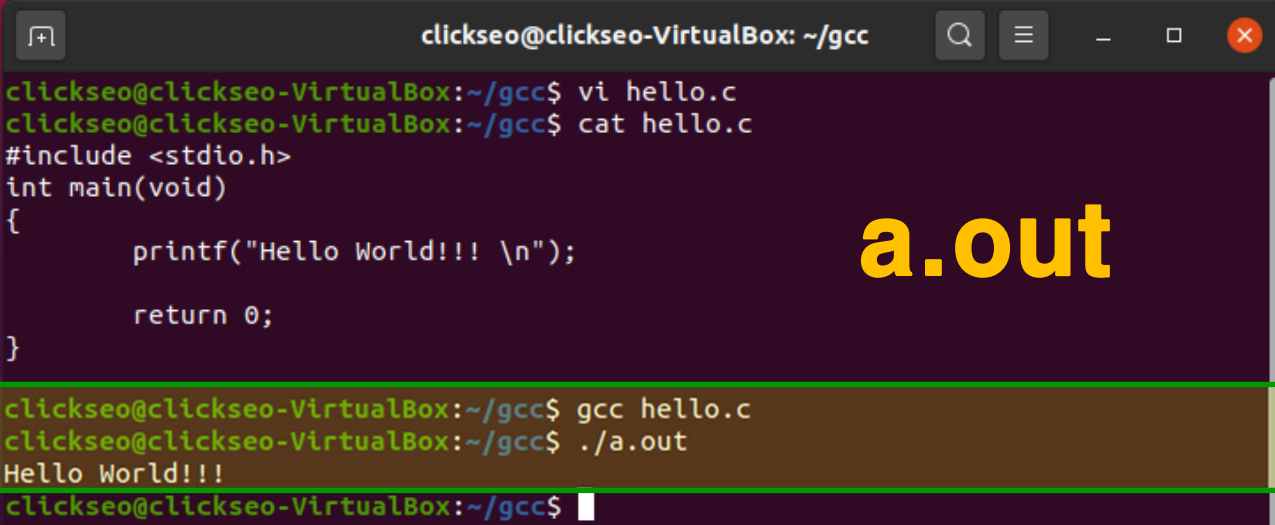
확장자	종류	처리 방법
.c	C, 원시 파일(Source File)	gcc 로 전처리, 컴파일, 어셈블, 링크
.cc .cpp	C++, 원시 파일(Source File)	g++ 로 전처리, 컴파일, 어셈블, 링크
.i	전처리 된 c 소스 코드	gcc 로 컴파일과 어셈블, 링크
.ii	전처리 된 c++ 소스 코드	g++ 로 컴파일과 어셈블, 링크
.s	어셈블리어로 된 파일	어셈블과 링크
.S	어셈블리어로 된 파일	전처리와 어셈블, 링크
.o	목적 파일(Object File)	링크
.a .so	컴파일 된 라이브러리 파일	링크

gcc : 컴파일 과정 (3/3)

예제 6-1 : gcc 컴파일과 프로그램 실행

```
#include <stdio.h>
int main(void)
{
    printf ("Hello Linux \n");

    return 0;
}
```



```
clickseo@clickseo-VirtualBox: ~/gcc
clickseo@clickseo-VirtualBox:~/gcc$ vi hello.c
clickseo@clickseo-VirtualBox:~/gcc$ cat hello.c
#include <stdio.h>
int main(void)
{
    printf("Hello World!!! \n");

    return 0;
}
clickseo@clickseo-VirtualBox:~/gcc$ gcc hello.c
clickseo@clickseo-VirtualBox:~/gcc$ ./a.out
Hello World!!!
clickseo@clickseo-VirtualBox:~/gcc$
```

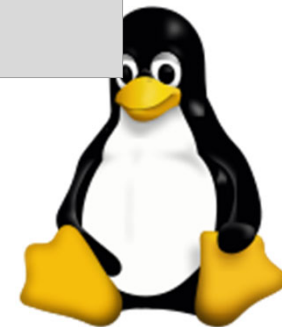
gcc 컴파일 실행

기본 실행 파일 생성 및 실행



GCC

gcc 옵션



gcc 옵션 (1/7)

● gcc : 옵션

옵 션	의 미
-o	지정된 파일명으로 실행 파일 생성 : 이진 파일(Binary File)
-c	목적 파일 생성(단, 링킹 과정은 수행하지 않는다)
-E	원시 소스 파일을 전 처리 과정까지 만 실행하고 컴파일을 중단한다.
-I	표준 디렉터리가 아닌 위치에 있는 헤더 파일의 디렉터리 지정(디렉터리 목록 추가)
-L	라이브러리 파일을 검색하는 디렉터리 목록을 추가
-l	라이브러리 파일을 컴파일 시 링크
-g	이진 파일에 표준 디버깅 정보를 포함하여 컴파일 gdb 를 이용하여 디버깅 하기 위해서는 -g 옵션을 이용하여 컴파일해야 한다.
-ggdb	이진 파일에 gdb 만이 이해할 수 있는 많은 디버깅 정보를 포함
-O	컴파일 코드 최적화
-O level	컴파일 최적화 level 단계 지정
-D FOO=BAR	명령 라인에서 BAR 값을 가지는 FOO 라는 선행 처리기 매크로를 정의한다.
-static	정적 라이브러리에 링크한다.

gcc 옵션 (2/7)

- gcc : 옵션

옵 션	의 미
-ansi	ANSI/ISO C 표준을 지원 : 표준과 충돌하는 GNU 확장안을 취소 ANSI 호환코드를 보장 안 함
-traditional	과거 스타일의 함수 정의 형식과 같이 전통적인 K&R C 언어 형식을 지원
-MM	make 호환의 의존성 목록을 출력
-V	컴파일의 각 단계에서 사용되는 명령 출력



gcc 옵션 (3/7)

- **gcc : -o 옵션**

- 실행 파일 생성 시 이름 지정 : 이진 파일(Binary File)
 - 지정하지 않을 시 **a.out** 의 기본 실행 파일 생성

일반 형식	<code>gcc -o [outputFileName] [sourceFileName]</code> <code>gcc [sourceFileName] -o [outputFileName]</code>
	출력파일과 소스파일의 순서는 바뀌어도 상관없다.

```
[clickseo@localhost ~]$ gcc -o hello hello.c
```

지정된 실행 파일
hello

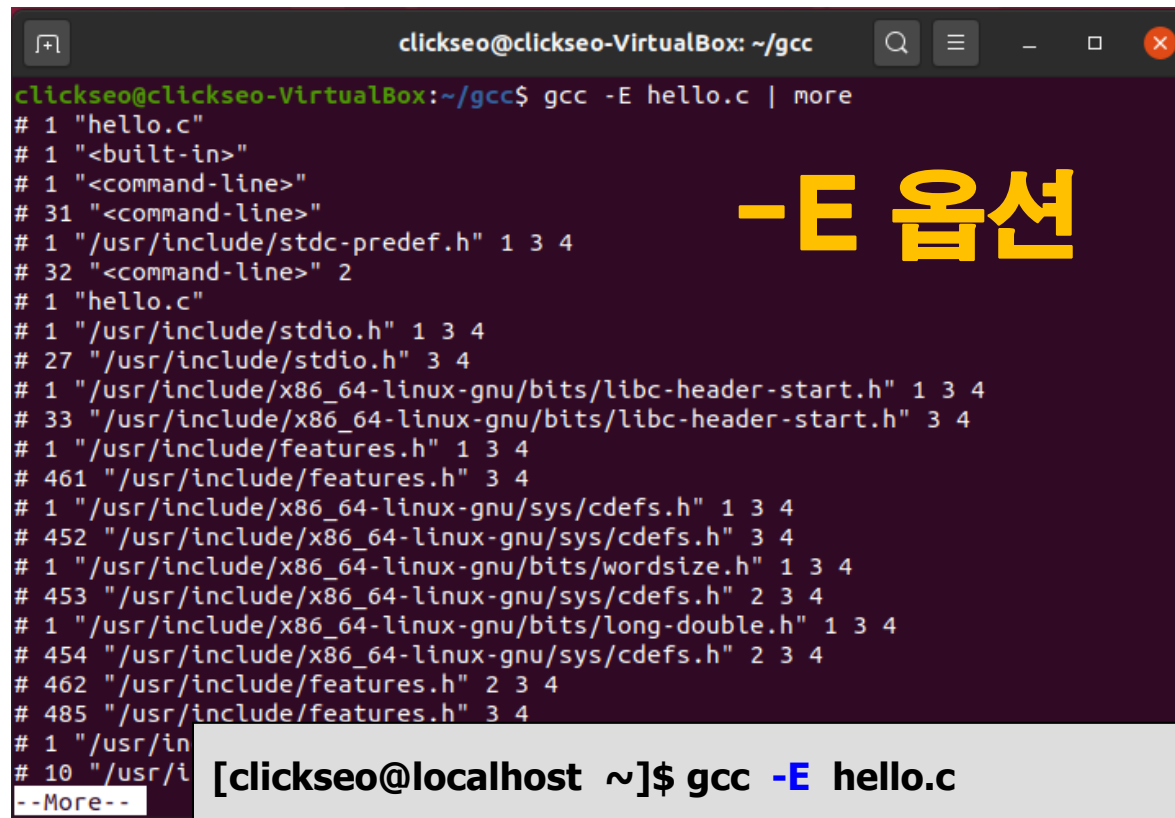
```
clickseo@clickseo-VirtualBox: ~/gcc
clickseo@clickseo-VirtualBox:~/gcc$ gcc hello.c
clickseo@clickseo-VirtualBox:~/gcc$ ./a.out
Hello World!!!
clickseo@clickseo-VirtualBox:~/gcc$ gcc -o hello hello.c
clickseo@clickseo-VirtualBox:~/gcc$ ./hello
Hello World!!!
clickseo@clickseo-VirtualBox:~/gcc$
```

-o 옵션

gcc 옵션 (4/7)

- **gcc : -E 옵션**

- 원시 파일을 전 처리 과정만을 실행하고, 컴파일을 중단한다.



```
clickseo@clickseo-VirtualBox: ~/gcc
clickseo@clickseo-VirtualBox:~/gcc$ gcc -E hello.c | more
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 1 3 4
# 33 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 461 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4
# 452 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 453 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/long-double.h" 1 3 4
# 454 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
# 462 "/usr/include/features.h" 2 3 4
# 485 "/usr/include/features.h" 3 4
# 1 "/usr/in
# 10 "/usr/i
--More--
```

```
[clickseo@localhost ~]$ gcc -E hello.c
[clickseo@localhost ~]$ gcc -E hello.c > preHello
```

gcc 옵션 (5/7)

- **gcc : -c 옵션**

- **목적 파일(Object File) 생성 : 확장자 .o 파일**

- 원시 파일(Source File)을 목적 파일로만 컴파일하고, 링크 과정 생략

```
[clickseo@localhost ~]$ gcc -c hello.c
```

```
[clickseo@localhost ~]$ gcc -o hello hello.o # hello 실행 파일 생성
```

```
[clickseo@localhost ~]$ gcc hello.o # a.out 실행 파일 생성
```

목적 파일 생성

hello.o

```
clickseo@clickseo-VirtualBox: ~/gcc
clickseo@clickseo-VirtualBox:~/gcc$ ls
hello.c
clickseo@clickseo-VirtualBox:~/gcc$ gcc -c hello.c
clickseo@clickseo-VirtualBox:~/gcc$ ls
hello.c hello.o
clickseo@clickseo-VirtualBox:~/gcc$ gcc -o hello hello.o
clickseo@clickseo-VirtualBox:~/gcc$ ls
hello hello.c hello.o
clickseo@clickseo-VirtualBox:~/gcc$ ./hello
Hello World!!!
clickseo@clickseo-VirtualBox:~/gcc$
```

-c 옵션

gcc 옵션 (6/7)

- **gcc : -I 옵션**

- 표준 디렉터리가 아닌 위치에 있는 헤더 파일의 디렉터리를 지정한다.

- 소스 파일과 헤더 파일이 다른 경로에 존재할 때 사용한다.

```
[clickseo@localhost ~]$ gcc main.c -I <헤더파일이 있는 디렉터리 경로 및 이름>
```



gcc 옵션 (7/7)

예제 6-2 : gcc 컴파일과 -I 옵션 `#include "./myheader/myheader.h`

```
// ./myheader/myheader.h  
#define MAXSIZE 1024
```

```
// ./main.c  
#include <stdio.h>  
#include "myheader.h"  
int main(void)  
{  
    printf ("%d \n", MAXSIZE);  
  
    return 0;  
}
```

소스 파일과
헤더 파일이
다른 경로에 존재

헤더 파일 경로 지정
`./myheader/`

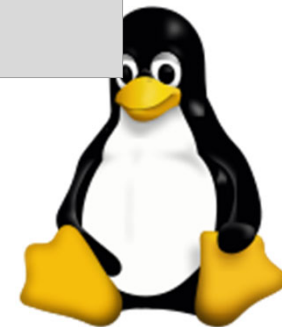
```
clickseo@clickseo-VirtualBox: ~/gcc  
clickseo@clickseo-VirtualBox:~/gcc$ gcc main.c  
main.c:2:10: fatal error: myheader.h: 그런 파일이나 디렉터리가 없습니다  
 2 | #include "myheader.h"  
    | ^~~~~~  
compilation terminated.  
clickseo@clickseo-VirtualBox:~/gcc$ gcc main.c -I ./myheader/  
clickseo@clickseo-VirtualBox:~/gcc$ ./a.out  
1024  
clickseo@clickseo-VirtualBox:~/gcc$
```

-I 옵션



GCC

분할 컴파일



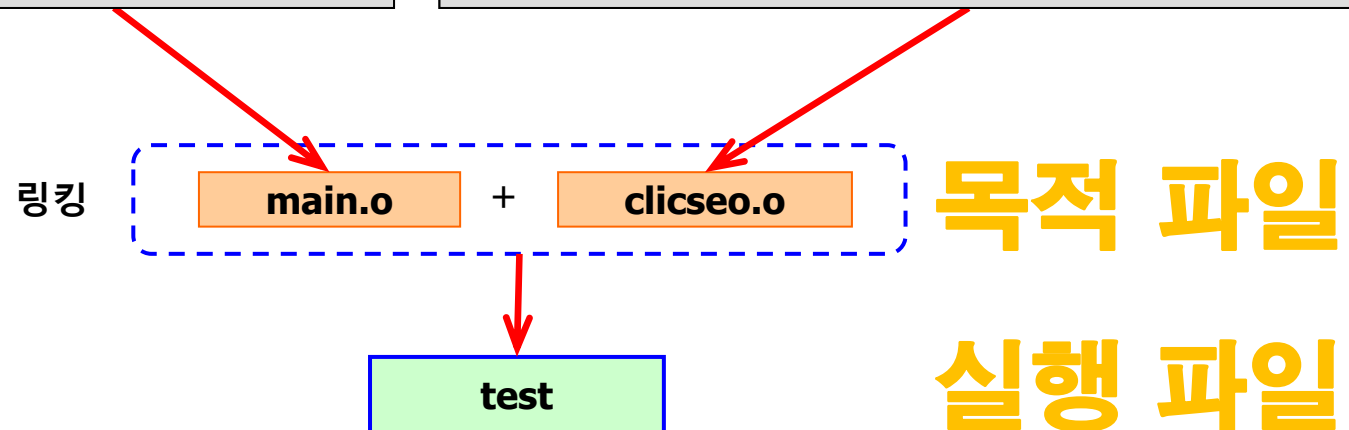
분할 컴파일 (1/2)

- 분할 컴파일

- 여러 파일로 분리 작성된 하나의 프로그램을 컴파일

```
// main.c
extern void clickseo(void);
int main(void)
{
    clickseo();
    return 0;
}
```

```
// clickseo.c
#include <stdio.h>
void clickseo(void)
{
    printf ("Hi~ Clickseo ^..^ \n");
    return;
}
```



분할 컴파일 (2/2)

- 분할 컴파일 : -c 옵션과 -o 옵션

- 분할 컴파일 : 목적 파일과 실행 파일 생성 후 실행 결과

```
[clickseo@localhost ~]$ gcc -c main.c  
[clickseo@localhost ~]$ gcc -c clicskeo.c  
[clickseo@localhost ~]$ gcc main.o clickseo.o -o test
```

컴파일과

목적 파일 생성

실행 파일 생성

```
clickseo@clickseo-VirtualBox: ~/gcc  
clickseo@clickseo-VirtualBox:~/gcc$ ls  
clickseo.c main.c  
clickseo@clickseo-VirtualBox:~/gcc$ gcc -c main.c  
clickseo@clickseo-VirtualBox:~/gcc$ gcc -c clickseo.c  
clickseo@clickseo-VirtualBox:~/gcc$ ls  
clickseo.c clickseo.o main.c main.o  
clickseo@clickseo-VirtualBox:~/gcc$ gcc main.o clickseo.o -o test  
clickseo@clickseo-VirtualBox:~/gcc$ ls  
clickseo.c clickseo.o main.c main.o test  
clickseo@clickseo-VirtualBox:~/gcc$ ./test  
Hi~ Clickseo ^..^  
clickseo@clickseo-VirtualBox:~/gcc$
```

```
[clickseo@localhost ~]$ gcc main.c clickseo.c -o test
```

GDB



- GCC

- GDB

- gdb 명령어

- 변수와 레지스터

- GNU make



GDB (1/4)

- **GDB**(GNU Project debugger)

- 1988년, 리처드 스톨만(Richard Stallman)

- GNU 소프트웨어 시스템을 위한 기본 디버거

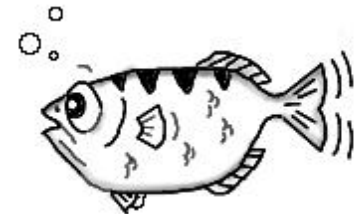
- 컴퓨터 프로그램의 실행을 추적하고 수정할 수 있는 많은 기능을 제공

- <https://www.gnu.org/software/gdb/>

- GDB 릴리즈

- 2018년 1월, GDB 8.1

- 라이선스 : GNU GPL



GDB (2/4)

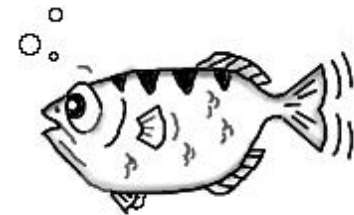
● GDB : 주요 특징

○ 다양한 프로그래밍 언어 지원

- C, C++, Objective-C, Fortran, Ada 그리고 Go, Rust 등의 프로그래밍 언어 지원

○ 주요 특징

- 프로그램의 내부 변수들의 값을 모니터링하거나 변경 가능
- 프로그램의 일반적인 실행 과정과 독립적으로 함수 호출이 가능
- 명령어를 통해서 메모리, 스택, 시그널, 레지스터 등의 정보를 보는 것이 가능
- 제한적으로 멀티 프로세스 멀티 스레드 디버깅 가능
- **원격 디버깅**
 - 임베디드 시스템을 디버깅할 때 사용되는 원격 모드 지원
 - GDB가 한 머신 상에 동작하고, 디버그 할 프로그램은 다른 머신 상에 서 동작



GDB (3/4)

- **gdb : 실행과 종료**

- gcc 컴파일 시 **-g** 옵션을 이용하여 컴파일

- -g 옵션을 붙이면 디버깅 정보가 실행 파일에 삽입된다.
- -g 옵션을 붙이지 않으면 어셈블리 코드만 보면서 디버깅해야 한다.

- gdb 실행

```
[clickseo@localhost ~]$ gdb [ programName ] # 일반적인 방법...
```

```
# 치명적 예외 상황
```

```
[clickseo@localhost ~]$ gdb [ programName ] [ coreFileName ]
```

```
# 수행 중인 프로세스
```

```
[clickseo@localhost ~]$ gdb [ programName ] [ processID ]
```

- gdb 종료

```
(gdb) d
```

```
(gdb) Ctrl + d
```

GDB (4/4)

- **gdb** : 프로그램 실행

```
$ gcc -g -o helloDebug hello.c
```

```
$ gdb helloDebug
```

컴파일 시 실행 파일에
디버깅 정보를 포함

```
clickseo@clickseo-VirtualBox:~/gcc$ gcc -g -o helloDebug hello.c
clickseo@clickseo-VirtualBox:~/gcc$ ls
hello.c  helloDebug
clickseo@clickseo-VirtualBox:~/gcc$ ./helloDebug
Hello Wrold!!!
```

-g 옵션

gdb 실행

```
clickseo@clickseo-VirtualBox:~/gcc$ gdb helloDebug
GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1
Copyright (C) 2020 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from helloDebug...
```

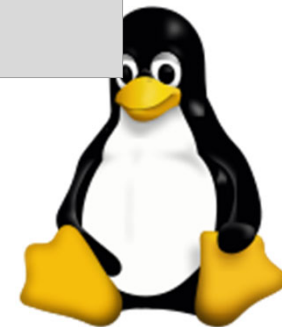
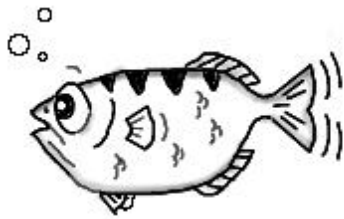
(gdb) 명령 프롬프트

```
(gdb) █
```



GDB

gdb 명령어



gdb : 명령어 (1/6)

● **gdb : List**

○ **l** : (List) **gdb** 를 이용한 프로그램 소스 출력

- 다시 소문자 **l** 을 누르거나 Enter 를 입력하면 나머지 소스 부분이 출력된다.

명령	내용
l	main 함수를 기점으로 프로그램 소스 출력
l -	출력된 행의 이전 행 출력
l n	n 번째 행을 기점으로 프로그램 소스 출력
l functionName	지정된 함수의 소스 출력
l file.c:n	file.c 파일의 n 번째 행을 기준으로 출력
l file.c:functionName	file.c 파일의 functionName 함수 부분 출력

(gdb) **l 10** # 10행의 소스를 보려고 할 때...

(gdb) **l main** # main 함수 부분을 보려고 할 때...

gdb : 명령어 (2/6)

● **GDB : Break Point**

○ **b : (Break Point) 중단점 설정**

- 프로그램의 수행을 어느 곳에서 멈출 것인지를 결정

명령	내용
b n	n 번째 행에서 중단점 설정
b functionName	functionName 함수의 시작부분에 중단점 설정
b n if var = 0	n 번째 행에 중단점을 설정하는데, 변수 var 값이 0 일 때 작동
b file.c:n	file.c 파일의 n 번째 행에 중단점 설정
b file.c:functionName	file.c 파일의 functionName 함수에 중단점 설정
b +n	현재 행에서 n 개 행 이후 지점에 중단점 설정
b -n	현재 행에서 n 개 행 이전 지점에 중단점 설정
b *0x8049000	메모리 주소 0x8049000 에 중단점 설정 (어셈블러 디버깅 시 사용)

```
(gdb) b 10
```

```
# 10행에서 중단점을 설정할 때...
```

```
(gdb) b main
```

```
# main 함수에서 중단점을 설정할 때...
```

gdb : 명령어 (3/6)

- **GDB : Break Point**

- **d : (Break Point) 중단점 설정 해제**

명령	내용
d	모든 중단점 설정 삭제
d n	n 번째 행의 중단점 설정 삭제
d functionName	functionName 함수의 시작부분에 중단점 설정 삭제
d file.c:10	file.c 파일의 10행의 중단점 설정 삭제
d file.c:functionName	file.c 파일의 functionName 함수의 중단점 설정 삭제

- **전체 출력 : 설정된 중단점 정보**

```
(gdb) info b
```

```
(gdb) info breakpoints
```

gdb : 명령어 (4/6)

- **GDB : 프로그램 실행 및 종료**

- **r : (run) 프로그램 실행**

명 령	내 용
r	프로그램 실행
r <i>argc1</i> <i>argc2</i>	<i>argc1</i> 과 <i>argc2</i> 를 인자로 프로그램 실행
k	프로그램 수행 종료

- **r** 명령을 수행하면 이전에 설정한 중단점 부분에서 멈춘다.

gdb : 명령어 (5/6)

- **GDB : 프로그램 진행**

- 프로그램 진행

명령	내용
s	(step) 현재 행 수행 후 정지(함수 호출 시, 함수 내부로 디버깅 모드 진입) (gdb) s 5 : s 를 5 번 입력한 것과 동일
n	(next) 현재 행 수행 후 정지(함수 호출 시, 함수 수행 후 다음 행으로 이동) (gdb) n 5 : n 을 5 번 입력한 것과 동일
c	(continue) 중단점을 만날 때까지 계속 진행
u	(until) 현재 루프를 종료
finish	현재 함수를 수행하고 종료
return	현재 함수를 수행하지 않고 종료
return 100	현재 함수를 수행하지 않고 종료(반환 값은 100)
si	현재의 명령을 수행(함수 호출 시, 함수 내부로 디버깅 모드 진입)
ni	현재의 명령을 수행(함수 호출 시, 함수 수행 후 다음 행으로 이동)

gdb : 명령어 (6/6)

- **GDB : Watch Point**

- 어떤 변수 값이 바뀔 때마다 브레이크를 걸고 싶을 때 사용

- 변수 값이 어떻게 바뀌는지 그리고 어떤 코드가 바뀌는지에 대해서 확인할 때 편리

```
(gdb) watch variableName
```

```
File Edit View Window Help
(gdb) watch i
Hardware watchpoint 2: i
(gdb) c
Continuing.
Hardware watchpoint 2: i

Old value = 134513196
New value = 0
0x08048433 in main () at bugprg.c:30
30         for(i = 0; i < 100; i++)
(gdb) c
Continuing.
array[i] = one
Hardware watchpoint 2: i

Old value = 0
New value = 1
0x08048480 in main () at bugprg.c:30
30         for(i = 0; i < 100; i++)
(gdb) █
```

gdb : 명령어 실습 (1/5)

● 예제 프로그램

[실습예제] bugprg.c

```
#include <stdio.h>

struct time
{
    int hour;
    int min;
    int sec;
} gtime = {1,2,3}, gtimes[4];

char *array[4] = {"one", "two", "three"};

int hello()
{
    int i = 10;
    while(i--);
    return 4;
}
```

[실습예제] bugprg.c

```
int gethour(short sw)
{
    if(sw) return 2;
    else return hello();
}

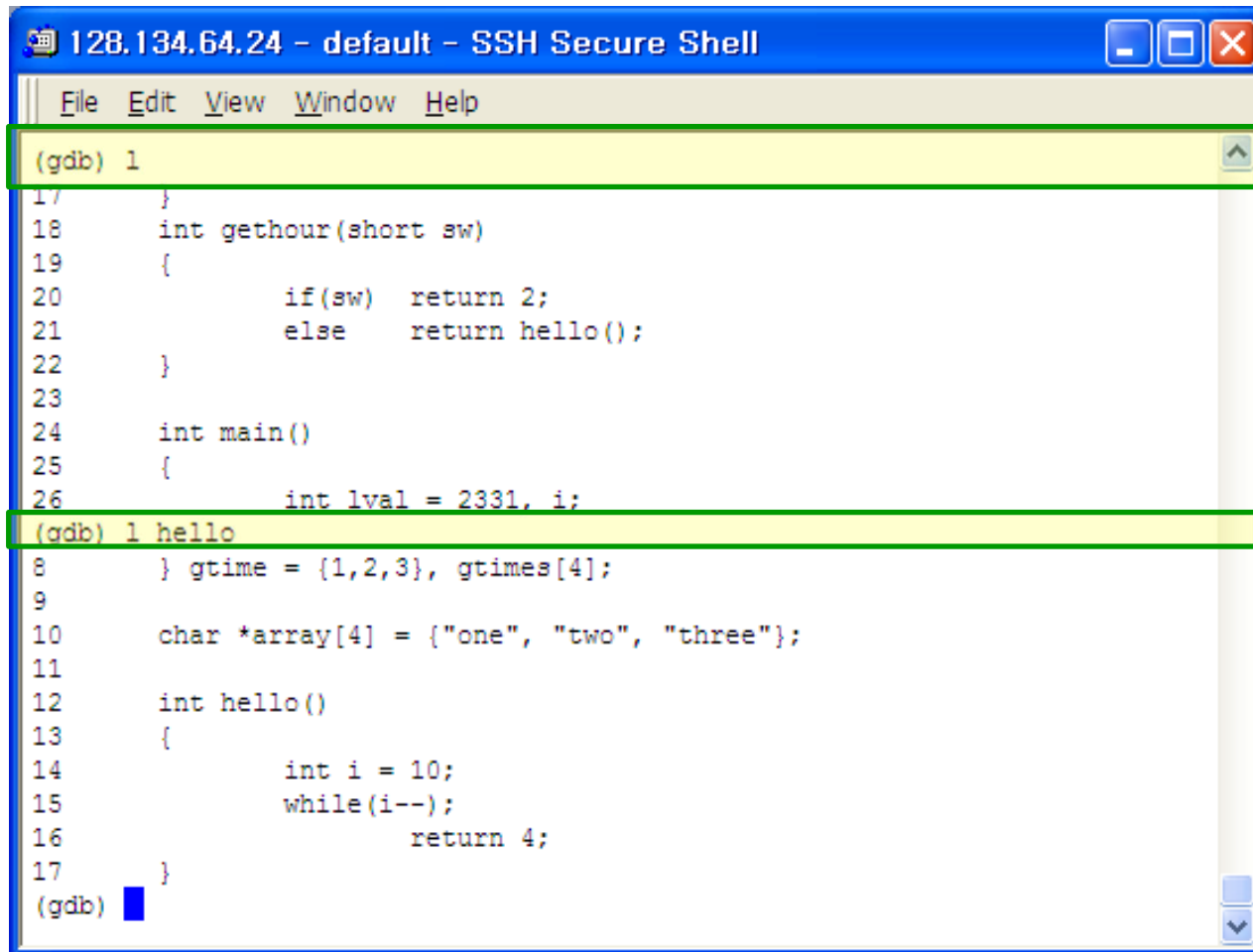
int main()
{
    int lval = 2331, i;
    char *lstr = "I like you.";
    struct time *pt = &gtime;

    for(i = 0; i < 100; i++)
    {
        printf("array[i] = %s\n", array[i]);
        gtime.hour += gethour(i % 2);
    }

    return 0;
}
```

gdb : 명령어 실습 (2/5)

- **소스 보기 (cont'd)**



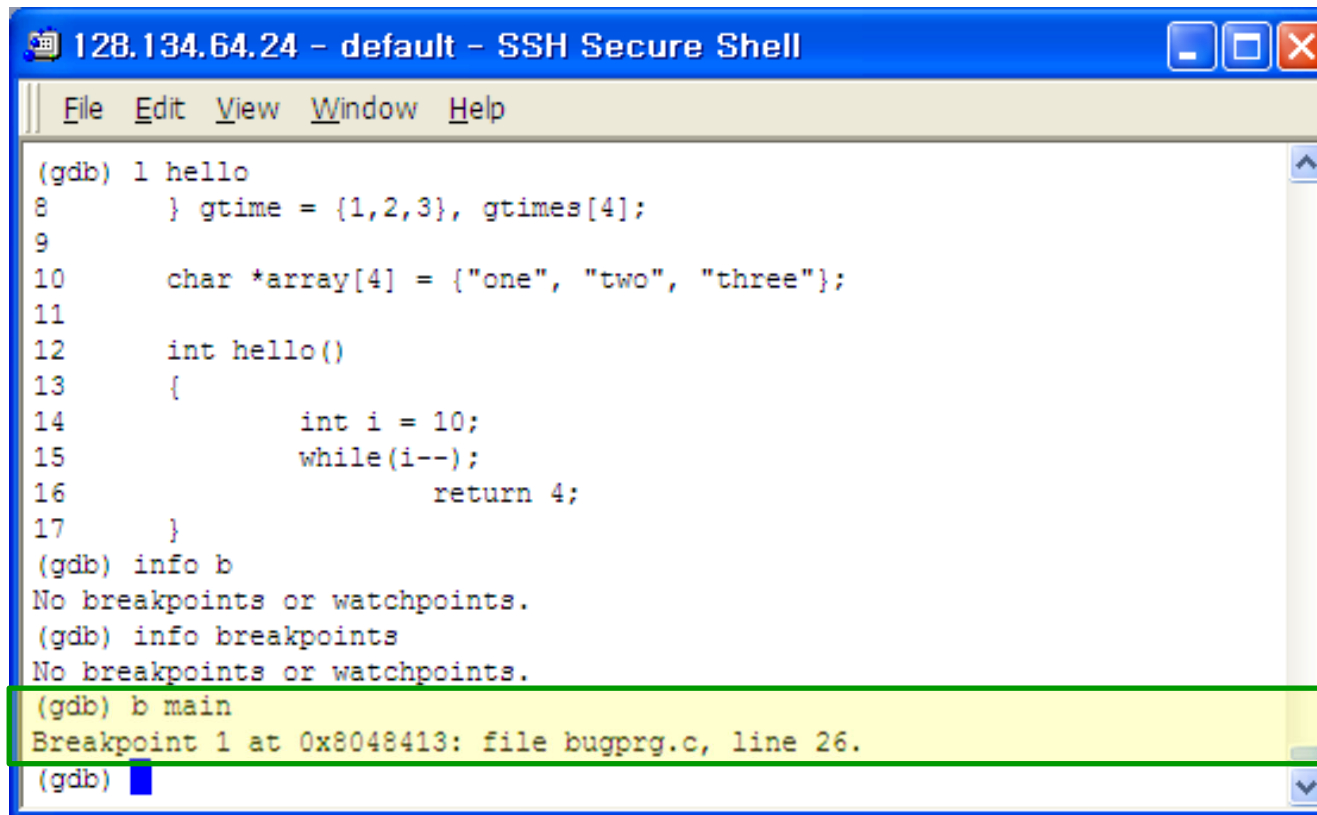
The screenshot shows a terminal window titled "128.134.64.24 - default - SSH Secure Shell". The window contains a GDB session where the user has entered the command `(gdb) l`. The terminal displays the source code for the `hello` program, with line numbers 17 through 26 visible. The code includes a `gethour` function and a `main` function. The `main` function contains a `hello` call. The terminal also shows the source code for the `hello` function, with line numbers 8 through 17 visible. The prompt `(gdb) █` is shown at the bottom of the terminal.

```
128.134.64.24 - default - SSH Secure Shell
File Edit View Window Help
(gdb) l
17     }
18     int gethour(short sw)
19     {
20         if(sw) return 2;
21         else  return hello();
22     }
23
24     int main()
25     {
26         int lval = 2331, i;
(gdb) l hello
8     } gtime = {1,2,3}, gtimes[4];
9
10    char *array[4] = {"one", "two", "three"};
11
12    int hello()
13    {
14        int i = 10;
15        while(i--);
16        return 4;
17    }
(gdb) █
```


gdb : 명령어 실습 (3/5)

- Break Point (cont'd)

- main 함수에 브레이크 포인트 설정

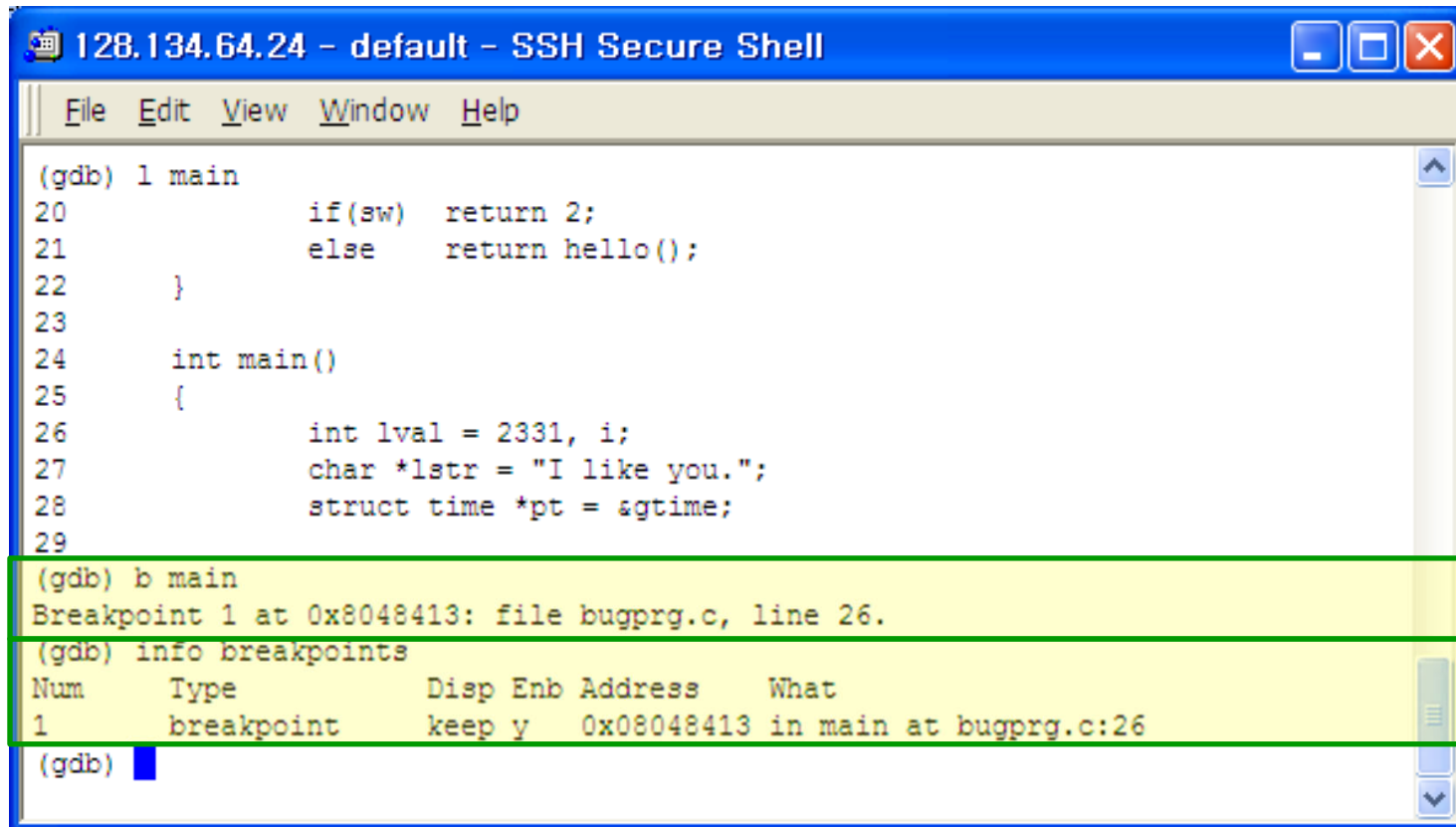


```
128.134.64.24 - default - SSH Secure Shell
File Edit View Window Help
(gdb) l hello
8      } gtime = {1,2,3}, gtimes[4];
9
10     char *array[4] = {"one", "two", "three"};
11
12     int hello()
13     {
14         int i = 10;
15         while(i--);
16         return 4;
17     }
(gdb) info b
No breakpoints or watchpoints.
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) b main
Breakpoint 1 at 0x8048413: file bugprg.c, line 26.
(gdb)
```

gdb : 명령어 실습 (4/5)

- **Break Point (cont'd)**

- 브레이크 포인트 리스트 확인



```
128.134.64.24 - default - SSH Secure Shell
File Edit View Window Help
(gdb) 1 main
20         if(sw) return 2;
21         else  return hello();
22     }
23
24     int main()
25     {
26         int lval = 2331, i;
27         char *lstr = "I like you.";
28         struct time *pt = localtime();
29
(gdb) b main
Breakpoint 1 at 0x8048413: file bugprg.c, line 26.
(gdb) info breakpoints
Num     Type           Disp Enb Address          What
1       breakpoint     keep y   0x08048413 in main at bugprg.c:26
(gdb) █
```

gdb : 명령어 실습 (5/5)

- 프로그램 실행 (cont'd)

- r (run)을 이용해 프로그램 수행

```
128.134.64.24 - default - SSH Secure Shell
File Edit View Window Help
(gdb) b main
Breakpoint 1 at 0x8048413: file bugprg.c, line 26.
(gdb) info breakpoints
Num      Type          Disp Enb Address      What
1        breakpoint    keep y   0x08048413  in main at bugprg.c:26
(gdb) r
Starting program: /home/clickseo/temp/bugprg

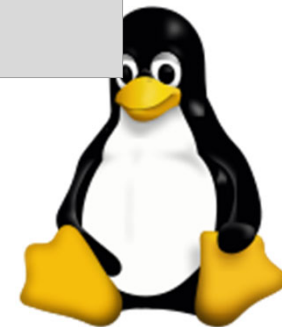
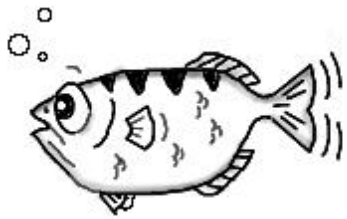
Breakpoint 1, main () at bugprg.c:26
26      int lval = 2331, i;
Missing separate debuginfos, use: debuginfo-install glibc-2.10.1-2.i686
(gdb) █
```

브레이크 포인트를 걸어준 main함수에서 멈춘다.



GDB

변수와 레지스터



gdb : 변수와 레지스터 (1/10)

● **변수와 레지스터 값 검사**

- 디버거를 사용하는 목적은 특정한 시점에서 메모리 값과 레지스터 값을 알기 위함이다.

- **전체 변수의 출력**

- 어떤 상태에서 어떤 지역변수들이 있으며 각각의 값들은 무엇인지 보인다.

```
(gdb) info locals
```

- **개별 변수의 출력**

```
(gdb) p [변수명]
```

```
# 해당 변수값 출력
```

```
(gdb) p [함수명]
```

```
# 해당 함수의 주소 값 출력
```

- **포인터 변수의 출력**

```
(gdb) p [포인터 변수명]
```

```
# 해당 포인터 변수 값 출력
```

gdb : 변수와 레지스터 (2/10)

- **변수와 레지스터 값 검사 (cont'd)**

- 레지스터 값 출력

```
(gdb) p $[레지스트명]
```

- 전체 레지스터 값 출력

```
(gdb) info registers
```

The screenshot shows a terminal window with a menu bar (File, Edit, View, Window, Help) and a title bar (128.134.64.24 - default - SSH). The terminal displays the following content:

```
(gdb) p $eax
$7 = -1073744172

(gdb) p $ebx
$8 = 3338228

(gdb) $ecx
Undefined command: "$ecx". Try "help $command".

(gdb) #edx

(gdb) $eip
Undefined command: "$eip". Try "help $command".

(gdb) █
```

Overlaid on the terminal is a window showing the output of the `info registers` command:

```
(gdb) info registers
eax          0xbffff6d4      -1073744172
ecx          0x531c56a4      1394366116
edx          0x1              1
ebx          0x32eff4 3338228
esp          0xbffff5f0      0xbffff5f0
ebp          0xbffff628      0xbffff628
esi          0x0              0
edi          0x0              0
eip          0x804842b      0x804842b <main+34>
eflags      0x286      [ PF SF IF ]
cs          0x73          115
ss          0x7b          123
ds          0x7b          123
es          0x7b          123
fs          0x0              0
gs          0x33          51

(gdb) █
```

gdb : 변수와 레지스터 (3/10)

● 변수와 레지스터 값 검사 (cont'd)

○ 출력 형식의 지정

```
(gdb) p/[출력 형식] [변수]
```

ex) p lval : lval 변수 값이 10진수 형태로 출력

p/x lval : lval 변수 값이 16진수 형태로 출력

○ 출력 형식의 종류

형 식	내 용
t	2진수로 출력
o	8진수로 출력
d	부호가 있는 10진수로 출력(int)
u	부호가 없는 10진수로 출력(unsigned int)
x	16진수로 출력
c	최초 1바이트 값을 문자형으로 출력
f	부동 소수점 값 형식으로 출력
a	가장 가까운 심볼의 오프셋을 출력

gdb : 변수와 레지스터 (4/10)

● **변수와 레지스터 값 검사 (cont'd)**

○ 화면에 변하는 변수 값을 자동으로 디스플레이 하기

```
(gdb) display [변수명]
```

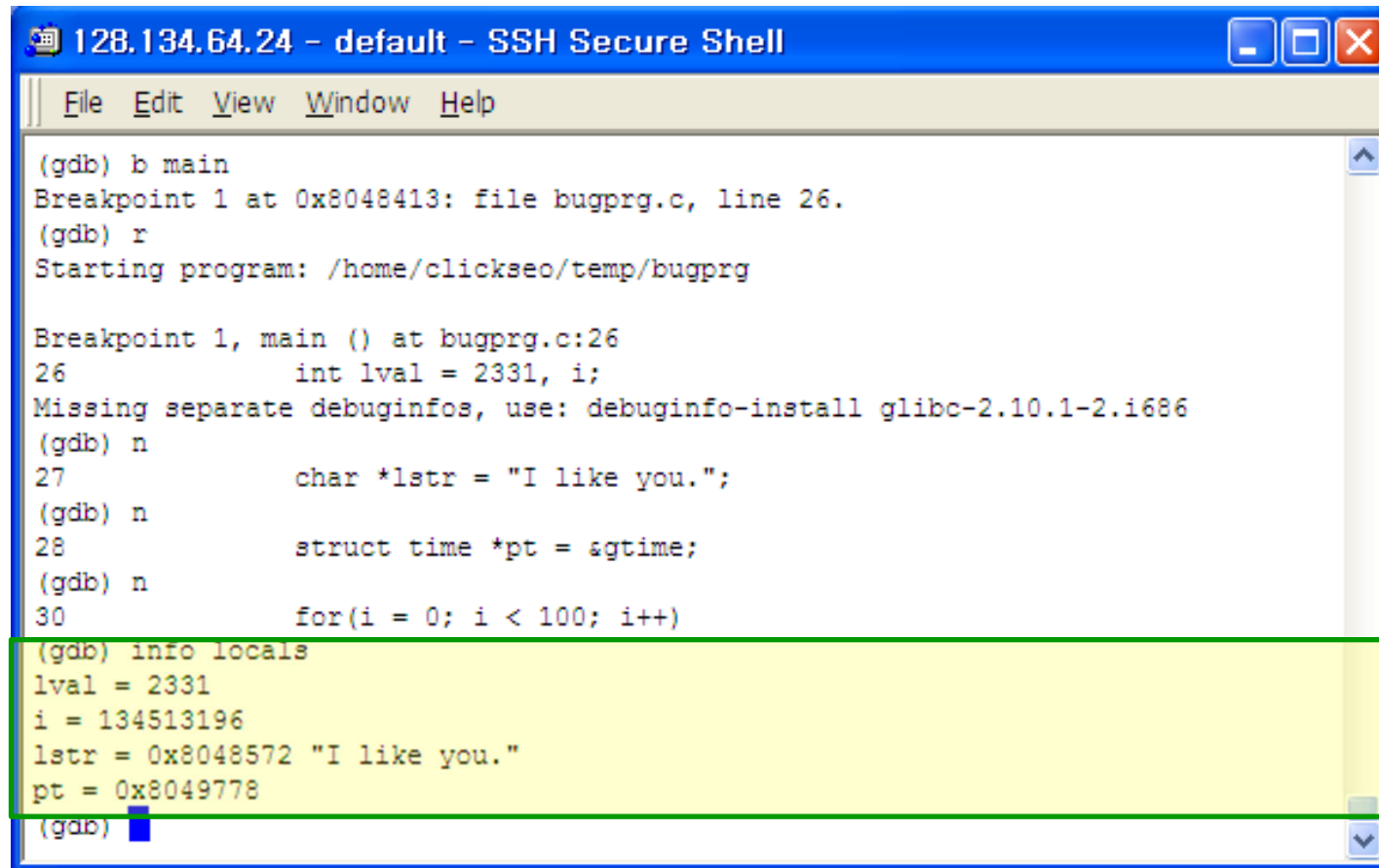
- display 명령은 지역변수를 디스플레이 하는데 함수가 바뀌어서 변수의 스코프 범위를 벗어나면 해당 변수를 디스플레이 하지 않는다.

○ display 관련 명령어

명령어	내용
display [변수명]	변수 값을 매번 화면에 디스플레이한다.
display /[출력형식] [변수명]	변수 값을 출력 형식으로 디스플레이한다.
undisplay [디스플레이 번호]	디스플레이 설정을 없앤다.
disable display [디스플레이번호]	디스플레이를 일시 중단한다.
enable display [디스플레이 번호]	디스플레이를 다시 활성화한다.

gdb : 변수와 레지스터 (5/10)

- 변수와 레지스터 값 검사 (cont'd)
 - 전체 변수의 출력 (for문으로 이동 후 실행)



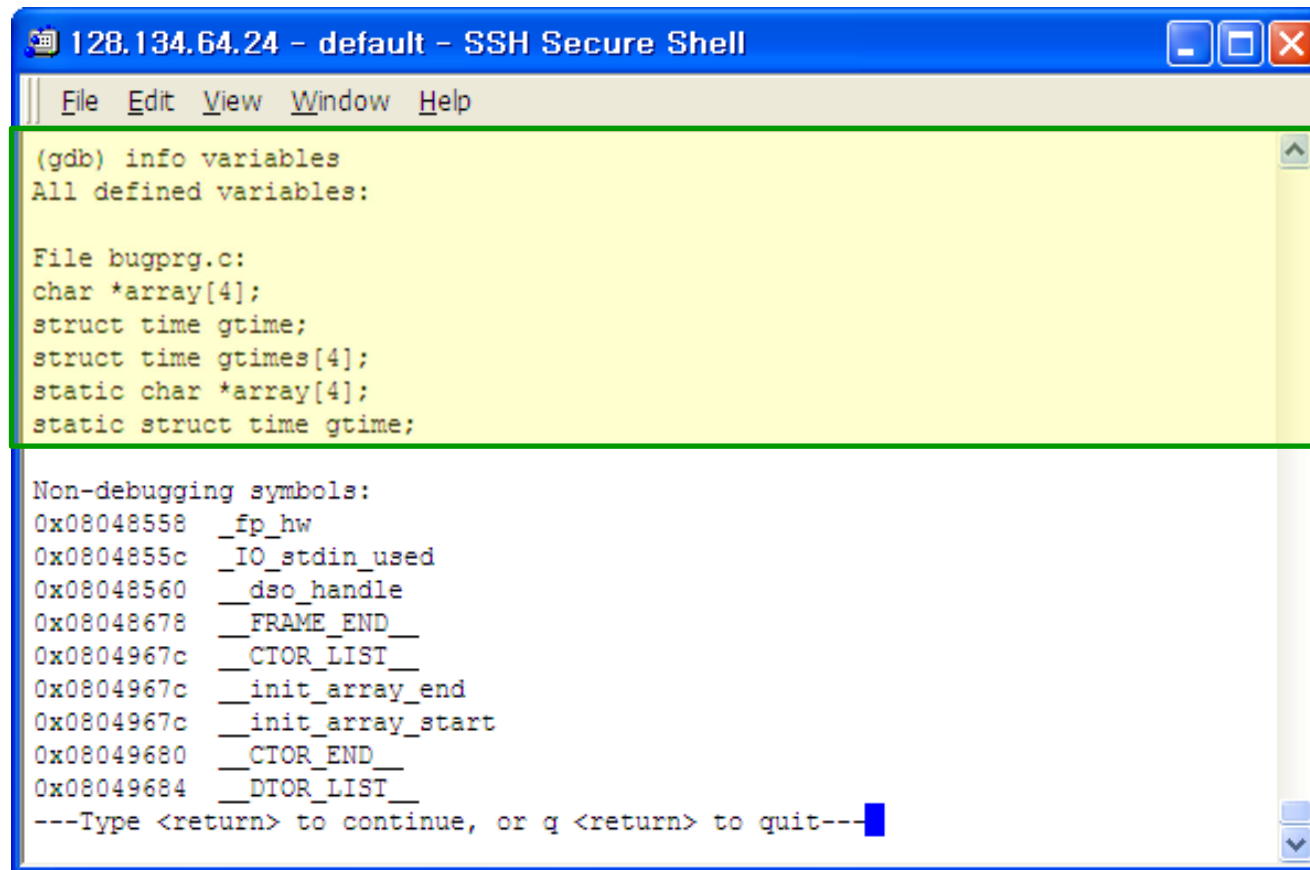
```
128.134.64.24 - default - SSH Secure Shell
File Edit View Window Help
(gdb) b main
Breakpoint 1 at 0x8048413: file bugprg.c, line 26.
(gdb) r
Starting program: /home/clickseo/temp/bugprg

Breakpoint 1, main () at bugprg.c:26
26         int lval = 2331, i;
Missing separate debuginfos, use: debuginfo-install glibc-2.10.1-2.i686
(gdb) n
27         char *lstr = "I like you.";
(gdb) n
28         struct time *pt = &gtime;
(gdb) n
30         for(i = 0; i < 100; i++)
(gdb) info locals
lval = 2331
i = 134513196
lstr = 0x8048572 "I like you."
pt = 0x8049778
(gdb)
```

gdb : 변수와 레지스터 (6/10)

- **변수와 레지스터 값 검사 (cont'd)**

- 전역 변수 리스트 출력



```
128.134.64.24 - default - SSH Secure Shell
File Edit View Window Help
(gdb) info variables
All defined variables:

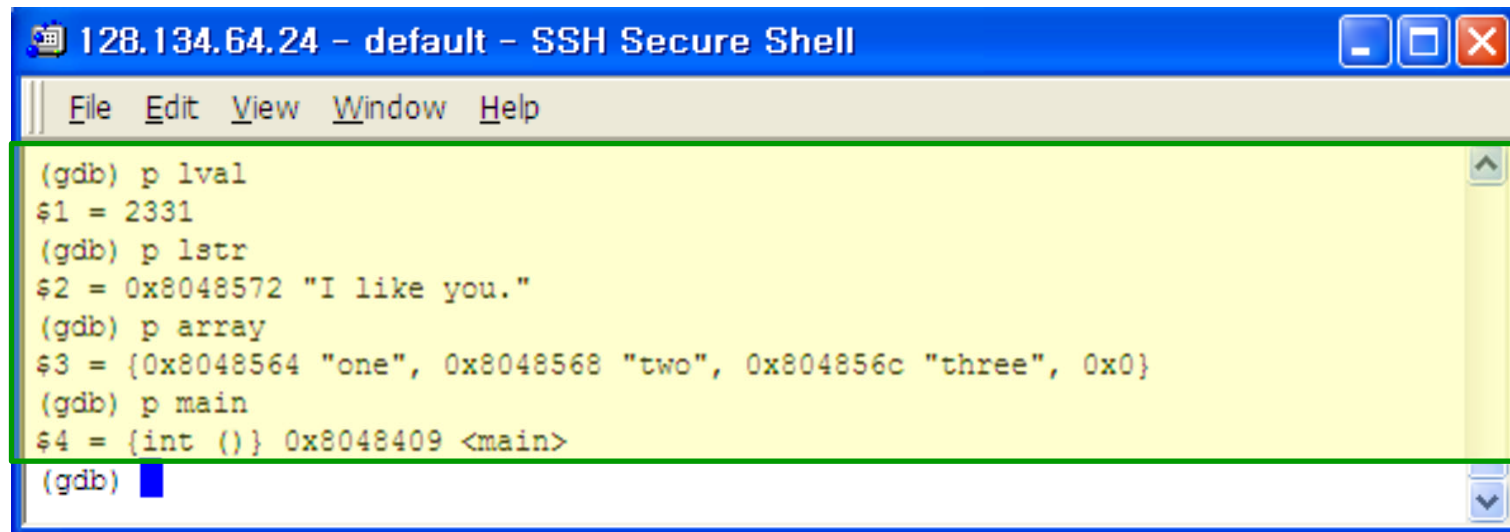
File bugprg.c:
char *array[4];
struct time gtime;
struct time gtimes[4];
static char *array[4];
static struct time gtime;

Non-debugging symbols:
0x08048558  _fp_hw
0x0804855c  _IO_stdin_used
0x08048560  __dso_handle
0x08048678  __FRAME_END__
0x0804967c  __CTOR_LIST__
0x0804967c  __init_array_end
0x0804967c  __init_array_start
0x08049680  __CTOR_END__
0x08049684  __DTOR_LIST__
---Type <return> to continue, or q <return> to quit---
```

gdb : 변수와 레지스터 (7/10)

- **변수와 레지스터 값 검사 (cont'd)**

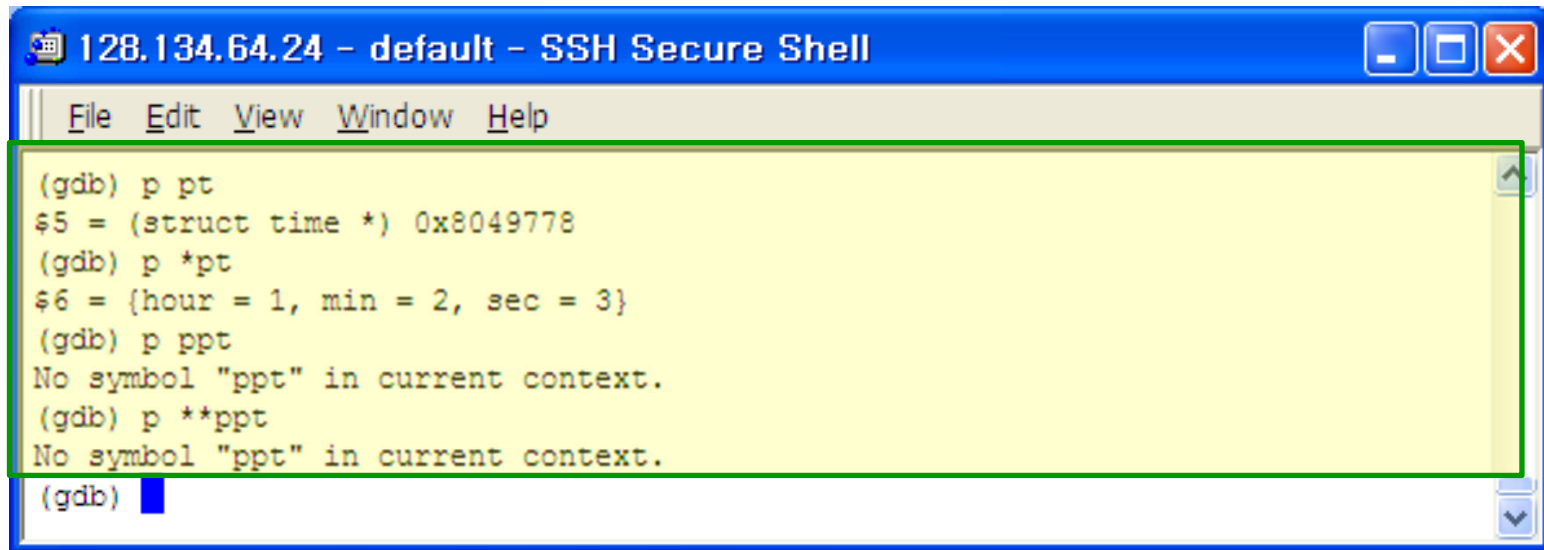
- 개별 변수의 출력



```
128.134.64.24 - default - SSH Secure Shell
File Edit View Window Help
(gdb) p lval
$1 = 2331
(gdb) p lstr
$2 = 0x8048572 "I like you."
(gdb) p array
$3 = {0x8048564 "one", 0x8048568 "two", 0x804856c "three", 0x0}
(gdb) p main
$4 = {int ()} 0x8048409 <main>
(gdb) █
```

gdb : 변수와 레지스터 (8/10)

- 변수와 레지스터 값 검사 (cont'd)
 - 포인터 변수의 출력



```
128.134.64.24 - default - SSH Secure Shell
File Edit View Window Help
(gdb) p pt
$5 = (struct time *) 0x8049778
(gdb) p *pt
$6 = {hour = 1, min = 2, sec = 3}
(gdb) p ppt
No symbol "ppt" in current context.
(gdb) p **ppt
No symbol "ppt" in current context.
(gdb) █
```

gdb : 변수와 레지스터 (9/10)

- **변수와 레지스터 값 검사 (cont'd)**
 - 출력 형식의 지정

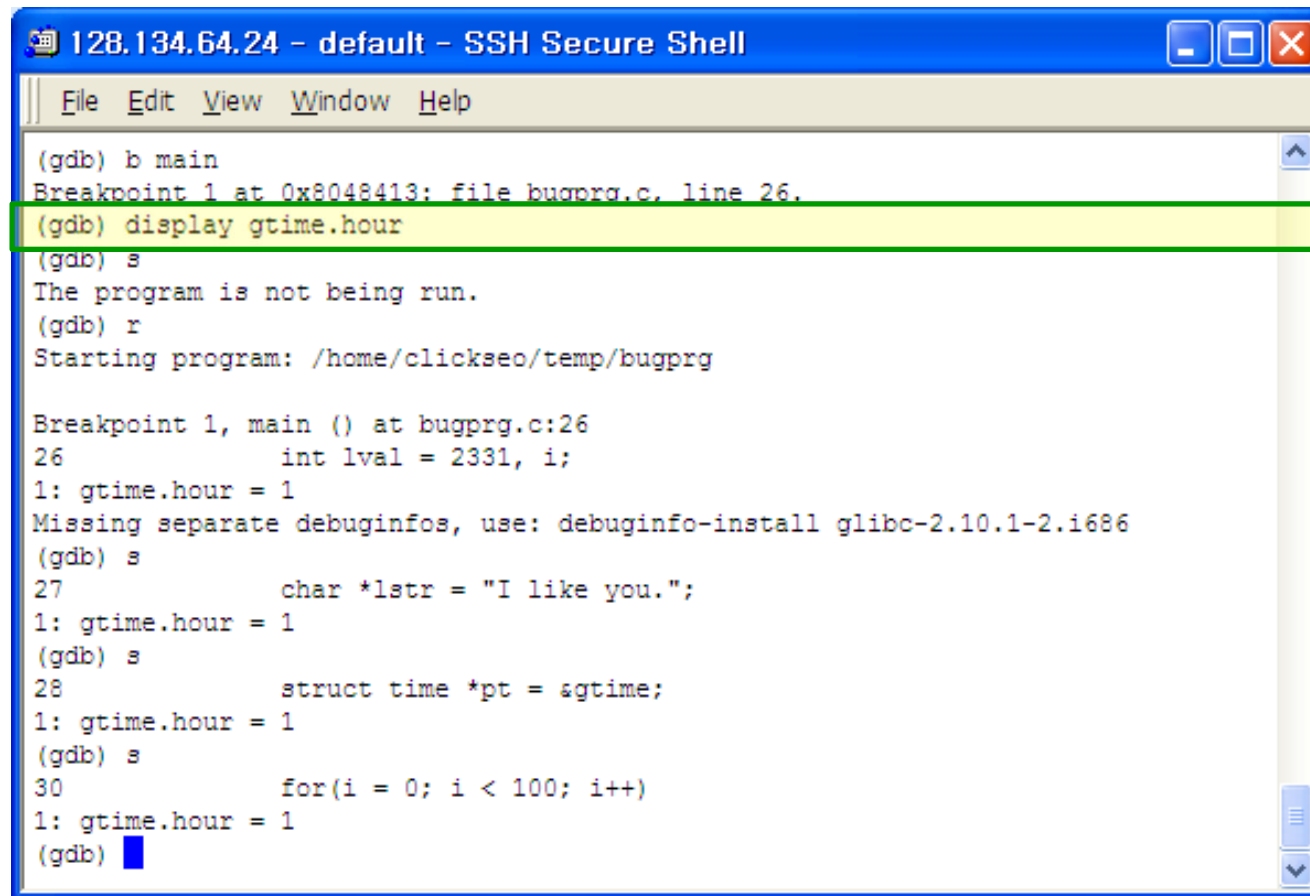
```
128.134.64.24 - default - SSH Secure Shell
File Edit View Window Help
(gdb) b main
Breakpoint 1 at 0x8048413: file bugprg.c, line 26.
(gdb) r
Starting program: /home/clickseo/temp/bugprg

Breakpoint 1, main () at bugprg.c:26
26         int lval = 2331, i;
Missing separate debuginfos, use: debuginfo-install glibc-2.10.1-2.i686
(gdb) n
27         char *lstr = "I like you.";
(gdb) n
28         struct time *pt = &gtime;
(gdb) n
30         for(i = 0; i < 100; i++)
(gdb) p lval
$1 = 2331
(gdb) p/x lval
$2 = 0x91b
(gdb) p/f lval
$3 = 3.26642672e-42
(gdb) █
```

gdb : 변수와 레지스터 (10/10)

- **변수와 레지스터 값 검사 (cont'd)**

- **display 명령**



```
128.134.64.24 - default - SSH Secure Shell
File Edit View Window Help
(gdb) b main
Breakpoint 1 at 0x8048413: file bugprg.c, line 26.
(gdb) display gtime.hour
(gdb) s
The program is not being run.
(gdb) r
Starting program: /home/clickseo/temp/bugprg

Breakpoint 1, main () at bugprg.c:26
26         int lval = 2331, i;
1: gtime.hour = 1
Missing separate debuginfos, use: debuginfo-install glibc-2.10.1-2.i686
(gdb) s
27         char *lstr = "I like you.";
1: gtime.hour = 1
(gdb) s
28         struct time *pt = &gtime;
1: gtime.hour = 1
(gdb) s
30         for(i = 0; i < 100; i++)
1: gtime.hour = 1
(gdb) █
```

GNU make



- GCC

- GDB

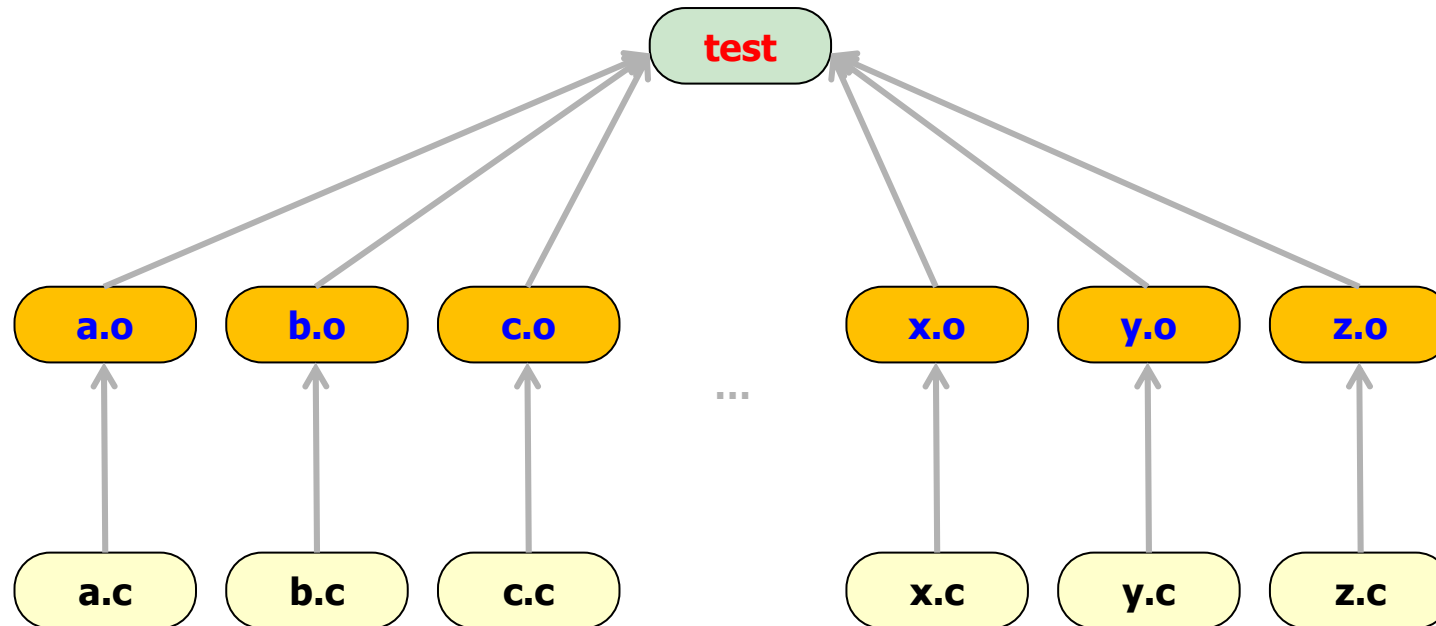
- **GNU make**

 - make 파일



make 파일 (1/5)

- make 파일

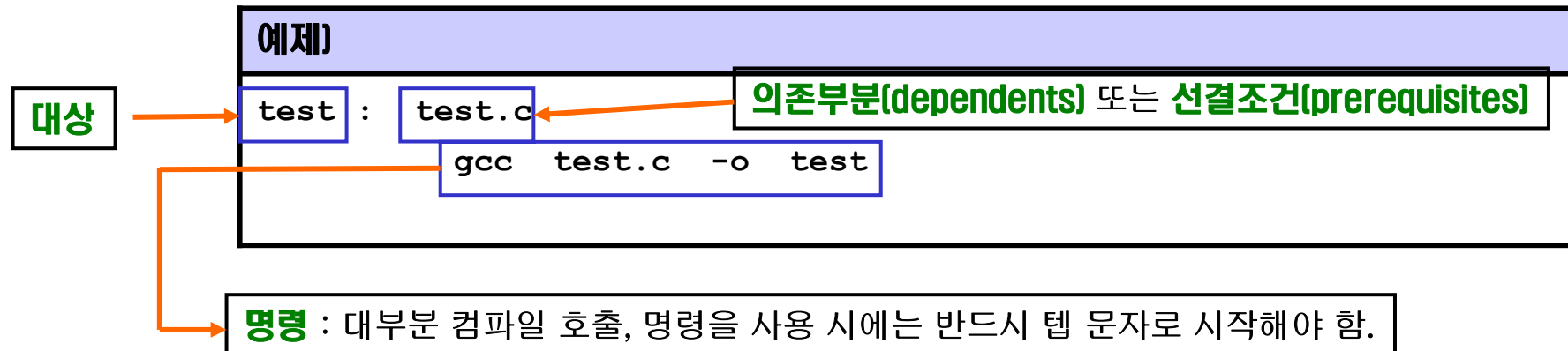


make 파일 (2/5)

- make 파일

- make 파일은 애플리케이션의 구성방법을 make에 알려주는 텍스트 파일

대상 (target) : 대상에 의존되는 파일1 [파일2 ...]
[tab간격] 명령 (command)



make 파일 (3/5)

- **make 파일 생성시 주의 사항**

- 각 요소를 구분하는데 있어 콤마(,) 같은 건 사용하지 않고 공백으로 한다.
- 명령을 시작하기 전에는 항상 <TAB>을 넣는다.
 - 절대 스페이스 키나 다른 키는 사용해선 안된다.
 - 그 밖의 다른 곳에서는 TAB을 사용하지 말라.
- make 파일 내에서 항목의 순서는 중요하지 않다.
 - make는 어떤 파일이 어느 곳에 의존적인지 알아내어 올바른 순서로 명령을 수행한다.

make 파일 (4/5)

- make 파일 예제

[예제 1] test1.c

```
#include <stdio.h>
#include "a.h"
void func1();
void func2();
main()
{
    printf("test1\n");
    func1 ();
    func2 ();
}
```

[예제 2] test2.c

```
#include <stdio.h>
#include "a.h"
#include "b.h"
extern void func1()
{
    printf("test2 \n");
}
```

[예제 3] test3.c

```
#include <stdio.h>
#include "b.h"
#include "c.h"
extern void func2()
{
    printf("test3 \n");
}
```

make 파일 (5/5)

● make 파일 예제 (cont'd)

① 헤더 파일 생성

```
[clickseo@comlab /]$ touch a.h
```

```
[clickseo@comlab /]$ touch b.h
```

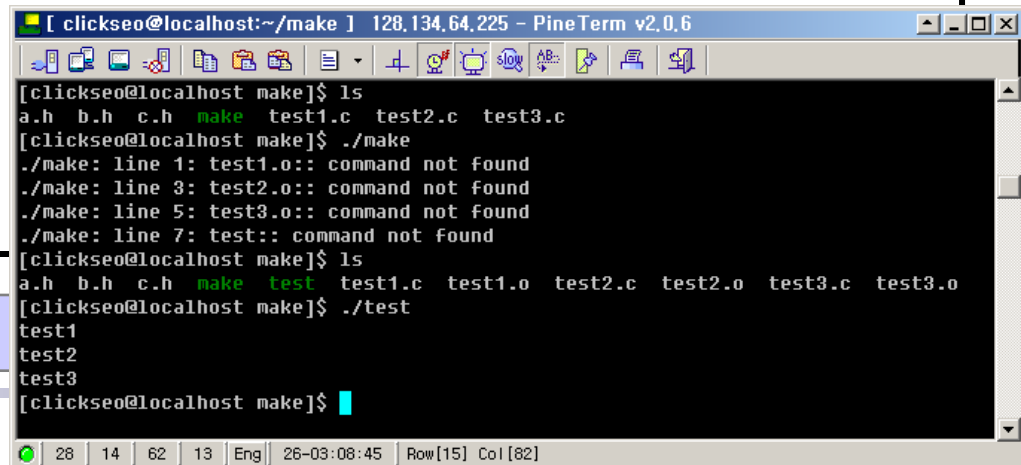
```
[clickseo@comlab /]$ touch c.h
```

② make 파일 생성

[makefile]

```
test: test1.o test2.o test3.o
    gcc -o test test1.o test2.o test3.o
test1.o: test1.c a.h
    gcc -c test1.c
test2.o: test2.c a.h b.h
    gcc -c test2.c
test3.o: test3.c b.h c.h
    gcc -c test3.c
```

③ make 실행



```
[clickseo@localhost ~/make] 128.134.64.225 - PineTerm v2.0.6
[clickseo@localhost make]$ ls
a.h b.h c.h make test1.c test2.c test3.c
[clickseo@localhost make]$ ./make
./make: line 1: test1.o:: command not found
./make: line 3: test2.o:: command not found
./make: line 5: test3.o:: command not found
./make: line 7: test:: command not found
[clickseo@localhost make]$ ls
a.h b.h c.h make test test1.c test1.o test2.c test2.o test3.c test3.o
[clickseo@localhost make]$ ./test
test1
test2
test3
[clickseo@localhost make]$
```

make 파일 : 문법 규칙 (1/8)

- 매크로 (Macro)

- make 파일을 작성하다 보면 같은 파일 이름을 여러 번 써야 하는 경우가 있다. 이런 경우에 매크로를 사용하면 편리하고 명령을 단순화 시킬 수 있다.

```
M_NAME = value
```

- 매크로 사용시 대소문자 모두 가능
 - 보통 대문자로 쓰는 것이 관례이다.
- make 파일 상단에 정의

make 파일 : 문법 규칙 (2/8)

- 매크로 사용 예제

```
[makefile]
OBJF = test1.o test2.o test3.o
test: $(OBJF)
    gcc -o test $(OBJF)
test1.o: test1.c a.h
    gcc -c test1.c
test2.o: test2.c a.h b.h
    gcc -c test2.c
test3.o: test3.c b.h c.h
    gcc -c test3.c
clean:
    rm $(OBJF)
```

Diagram illustrating macro usage in a Makefile:

- The macro `OBJF` is defined as `test1.o test2.o test3.o`. This definition is highlighted with a blue box and labeled "매크로 정의" (Macro Definition).
- The macro `$(OBJF)` is used in the `test` target's command and in the `clean` target's command. This usage is highlighted with a blue box and labeled with the expanded value: `$(OBJF) = test1.o test2.o test3.o`.
- Orange arrows indicate the flow from the macro definition to its usage in the `test` target and from the expanded value to the `clean` target.

make 파일 : 문법 규칙 (3/8)

- 내부 매크로

내부 매크로	의 미
\$@	현재 목표 파일의 이름
\$*	확장자를 제외한 현재 목표 파일의 이름
\$<	현재 필수 조건 파일 중 첫 번째 파일 이름
\$?	현재 대상보다 최근에 변경된 필수 조건 파일 이름
\$^	현재 모든 필수 조건 파일들

make 파일 : 문법 규칙 (4/8)

- 내부 매크로 사용 예제

[makefile]

```
OBJF = test1.o test2.o test3.o
test: $(OBJF)
    gcc -o $@ $^
test1.o: test1.c a.h
    gcc -c $<
test2.o: test2.c a.h b.h
    gcc -c $*.c
test3.o: test3.c b.h c.h
    gcc -c $*.c
clean:
    rm $(OBJF)
```

현재 대상 파일의 이름을 의미하므로 test를 나타냄

OBJF로 정의된 매크로 값

현재 대상 파일 test1.o가 의존하는 필수 조건 파일 중 첫 번째 파일 이름을 의미 test1.c

확장자 .c 를 제외한 현재 대상 파일의 이름을 의미 각각 test2 , test3 을 의미

make 파일 : 문법 규칙 (5/8)

- 매크로 치환

- 이미 정의된 매크로의 내용을 치환으로 변경

```
$(M_NAME:old=new)
```

예제)

```
OBJF = test1.o test2.o test3.o  
SRCS = $(OBJF:.o=.c)
```

- ➔ OBJF의 확장자 부분이 .o에서 .c로 바뀌게 되어 SRCS에 저장된다.
결국 SRCS 는 test1.c test2.c test3.c' 를 의미한다.

make 파일 : 문법 규칙 (6/8)

- 명시적 규칙

- make 가 해야 할 일을 명확히 지정

- 암시적 규칙

- make 내에 미리 정의된 규칙을 이용해 make파일을 단순화 시키는 규칙

[makefile]

```
OBJF = test1.o test2.o test3.o
test: $(OBJF)
    gcc -o $@ $(OBJF)
clean:
    rm $(OBJF)
```

make 실행 시 오브젝트 파일을 암시적 규칙에 따라 의존하는 파일을 찾고 컴파일러 호출까지 수행한다.
(단, 내장된 규칙에 의해 gcc 가 아니라 cc 를 호출)

make 파일 : 문법 규칙 (7/8)

● 접미사 규칙

예제)

```
.c.o:                                ...①  
    gcc -c $< $(CFLAGS)              ...②
```

① .c.o

.c 라는 확장자를 가진 파일을 사용해 .o 라는 확장자를 가진 파일을 만들 것임을 make에 알리는 역할을 한다.

② \$<

확장자가 .c인 파일명을 의미한다.

\$(CFLAGS)

C 컴파일러를 위한 플래그를 위해 미리 정의된 변수

make 파일 : 문법 규칙 (8/8)

● 패턴 규칙

- 암시적 규칙에 의존했을 경우 일어나는 오류방지
- 접미사 규칙과 비슷하나 더 뛰어난 기능을 가짐

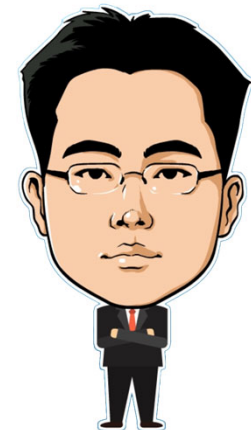
```
[makefile]
OBJF = test1_d.o test2_d.o test3_d.o
test: $(OBJF)
    gcc -o $@ $(OBJF)
%_d.o: %.c
    gcc -c -g $< -o $@
clean:
    rm $(OBJF)
```

확장자 .c 인 모든 파일에 대해 _d 를 붙인 오브젝트 파일을 생성하겠다는 의미

-g 옵션을 주어 컴파일 시 디버깅 정보를 삽입

참고문헌

- [1] 이종원, "IT CookBook, 우분투 리눅스(개정판) : 시스템 & 네트워크", 한빛아카데미, 2018.
- [2] 백창우, "유닉스 리눅스 프로그래밍 필수 유틸리티", 한빛미디어. 2010.
- [3] "GNU Software", GNU Operating System, FSF, 2020 of viewing the site, <https://www.gnu.org/software/>.



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며, 내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.

