차량소프트웨어엔지니어링 (T3. 디버깅)

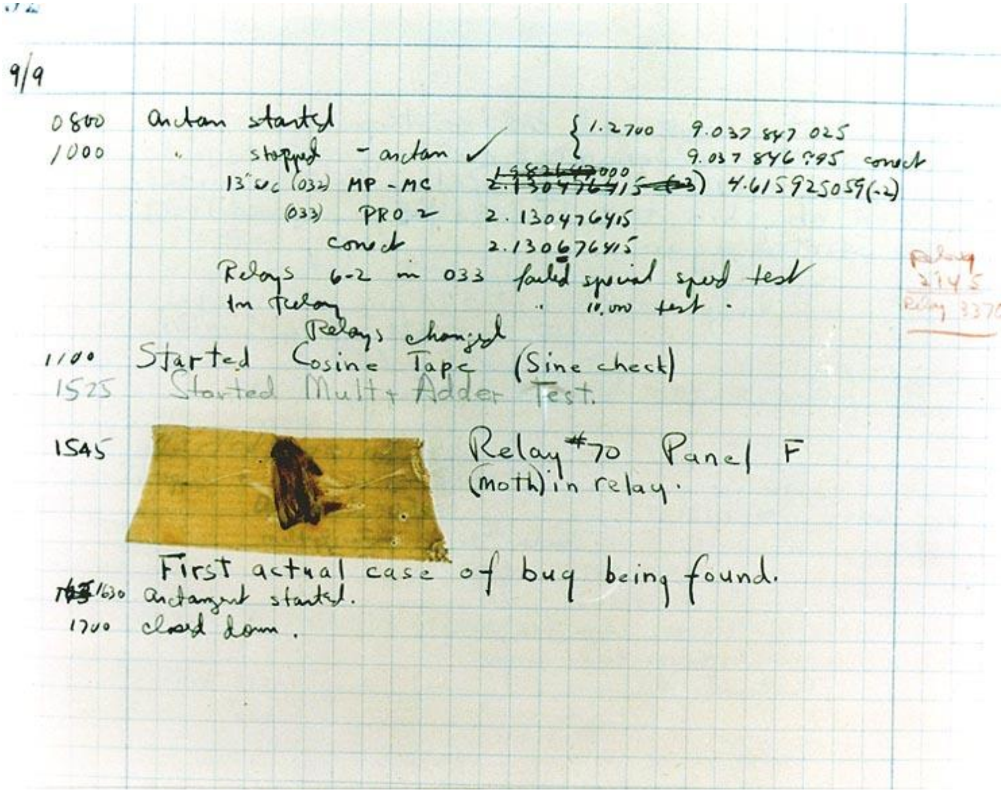
김종찬 교수

자동차공학과 / 자동차IT융합학과

KMU 국민대학교
KOOKMIN UNIVERSITY

최초의 디버깅

- Grace Hopper: 미 해군 제독이자 극초기 프로그래머
- 1947년 Harvard Mark II 컴퓨터 내부에서 나방 발견
- 제거 후 아래와 같이 기록. 최초의 debugging



9/9

0800 Antan started
1000 " stopped - antan ✓ {1.2700 9.037 847 025
9.037 846 995 correct
13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ (-3) 4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 11.000 test .
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 Antangent started.
1700 closed down.

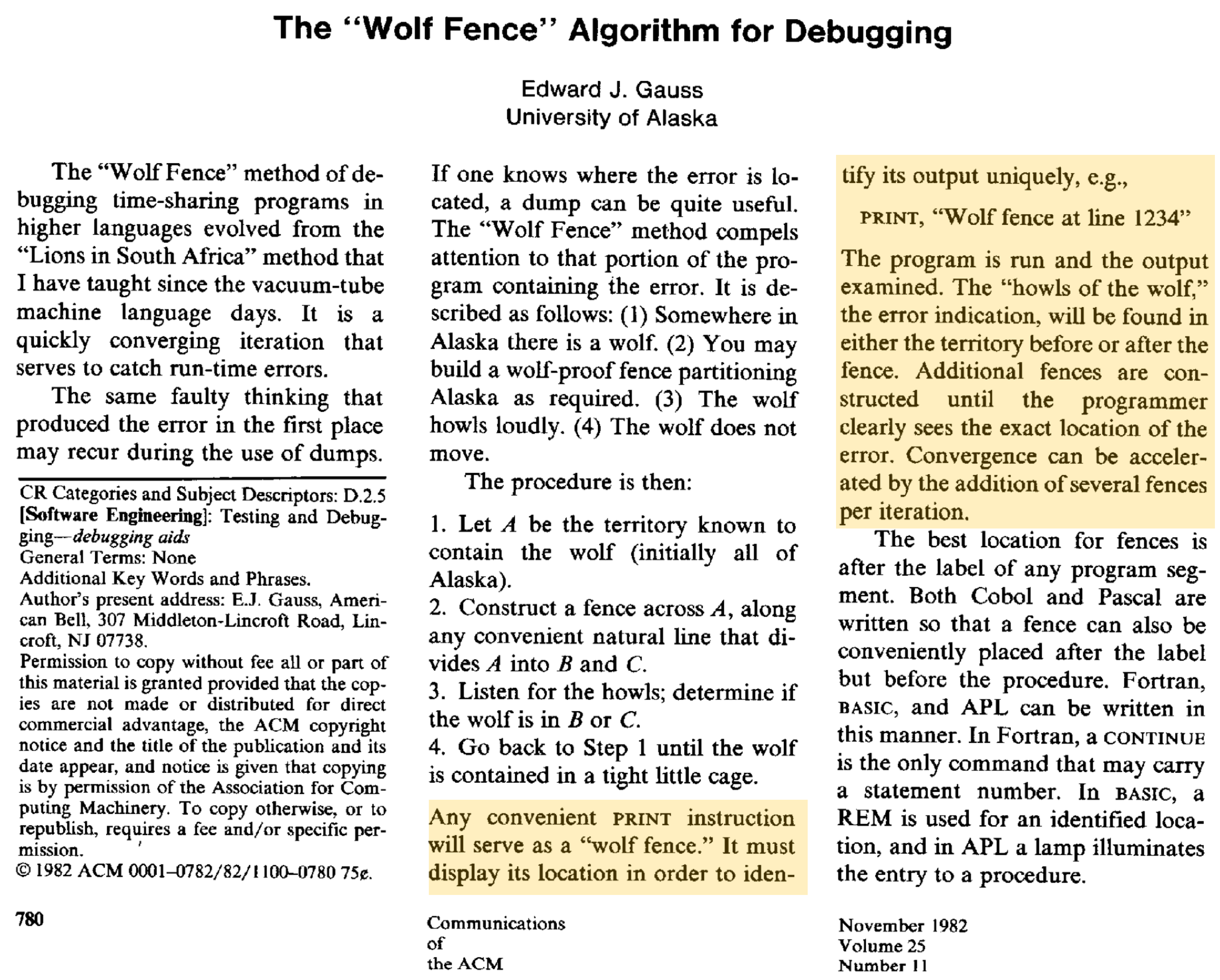
Relay 2145
Relay 3370

https://en.wikipedia.org/wiki/Debugging

Grace Brewster Murray Hopper
(1906~1992)

Mark I 컴퓨터 (1943)

Wolf Fence Algorithm



The "Wolf Fence" Algorithm for Debugging

Edward J. Gauss
University of Alaska

The "Wolf Fence" method of debugging time-sharing programs in higher languages evolved from the "Lions in South Africa" method that I have taught since the vacuum-tube machine language days. It is a quickly converging iteration that serves to catch run-time errors.

The same faulty thinking that produced the error in the first place may recur during the use of dumps. If one knows where the error is located, a dump can be quite useful. The "Wolf Fence" method compels attention to that portion of the program containing the error. It is described as follows: (1) Somewhere in Alaska there is a wolf. (2) You may build a wolf-proof fence partitioning Alaska as required. (3) The wolf howls loudly. (4) The wolf does not move.

The procedure is then:

1. Let *A* be the territory known to contain the wolf (initially all of Alaska).
2. Construct a fence across *A*, along any convenient natural line that divides *A* into *B* and *C*.
3. Listen for the howls; determine if the wolf is in *B* or *C*.
4. Go back to Step 1 until the wolf is contained in a tight little cage.

Any convenient PRINT instruction will serve as a "wolf fence." It must display its location in order to identify its output uniquely, e.g.,

```
PRINT, "Wolf fence at line 1234"
```

The program is run and the output examined. The "howls of the wolf," the error indication, will be found in either the territory before or after the fence. Additional fences are constructed until the programmer clearly sees the exact location of the error. Convergence can be accelerated by the addition of several fences per iteration.

The best location for fences is after the label of any program segment. Both Cobol and Pascal are written so that a fence can also be conveniently placed after the label but before the procedure. Fortran, BASIC, and APL can be written in this manner. In Fortran, a CONTINUE is the only command that may carry a statement number. In BASIC, a REM is used for an identified location, and in APL a lamp illuminates the entry to a procedure.

CR Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging—*debugging aids*
General Terms: None
Additional Key Words and Phrases.
Author's present address: E.J. Gauss, American Bell, 307 Middleton-Lincroft Road, Lincroft, NJ 07738.

Printf 디버깅

test.c

```
#include <stdio.h>

int main(void)
{
    int a, b, c;
    a = -1;
    b = 1;
    c = -1;
    a = b + c;
    b = a + c;
    c = a + b + 1;
    a = b / c;
    b = a * c;
    printf("a b c = %d %d %d\n", a, b, c);

    return 0;
}
```

- 어느 라인에서 죽나?

```
$ gcc test.c
$ ./a.out
Floating point exception
```

Printf 디버깅

```c
#include <stdio.h>

int main(void)
{
    int a, b, c;
    printf("*");
    a = -1;
    printf("*");
    b = 1;
    printf("*");
    c = -1;
    printf("*");
    a = b + c;
    printf("*");
    b = a + c;
    printf("*");
    c = a + b + 1;
    printf("*");
    a = b / c;
    printf("*");
    b = a * c;
    printf("a b c = %d %d %d\n", a, b, c);

    return 0;
}
```

어떤 차이?

```c
#include <stdio.h>

int main(void)
{
    int a, b, c;
    printf("*\n");
    a = -1;
    printf("*\n");
    b = 1;
    printf("*\n");
    c = -1;
    printf("*\n");
    a = b + c;
    printf("*\n");
    b = a + c;
    printf("*\n");
    c = a + b + 1;
    printf("*\n");
    a = b / c;
    printf("*\n");
    b = a * c;
    printf("a b c = %d %d %d\n", a, b, c);

    return 0;
}
```

디버깅 매크로

dbg.c

```c
#include <stdio.h>

#ifdef DEBUG
#define DBG(fmt, ...) printf("(%s:%d) " fmt "\n", __FILE__, __LINE__, __VA_ARGS__)
#else
#define DBG(fmt, ...)
#endif

int main(void)
{
    int a, b, c;

    a = -1;
    DBG("a = %d", a);
    b = 1;
    DBG("b = %d", b);
    c = -1;
    DBG("c = %d", c);
    a = b + c;
    DBG("a = %d", a);
    b = a + c;
    DBG("b = %d", b);
    c = a + b + 1;
    DBG("c = %d", c);
    a = b / c;
    DBG("a = %d", a);
    b = a * c;
    printf("a b c = %d %d %d\n", a, b, c);

    return 0;
}
```

```
$ gcc dbg.c
$ ./a.out
Floating point exception


$ gcc dbg.c -DDEBUG
$ ./a.out
(dbg.c:16) a = -1
(dbg.c:18) b = 1
(dbg.c:20) c = -1
(dbg.c:22) a = 0
(dbg.c:24) b = -1
(dbg.c:26) c = 0
Floating point exception
```

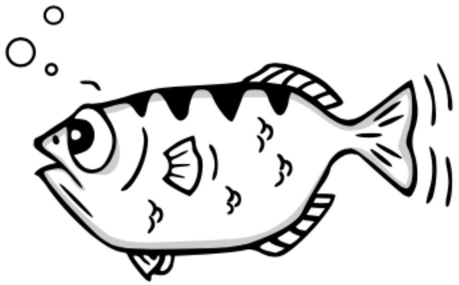
Interactive Debugging

- GDB: The GNU Project Debugger
  - 설치

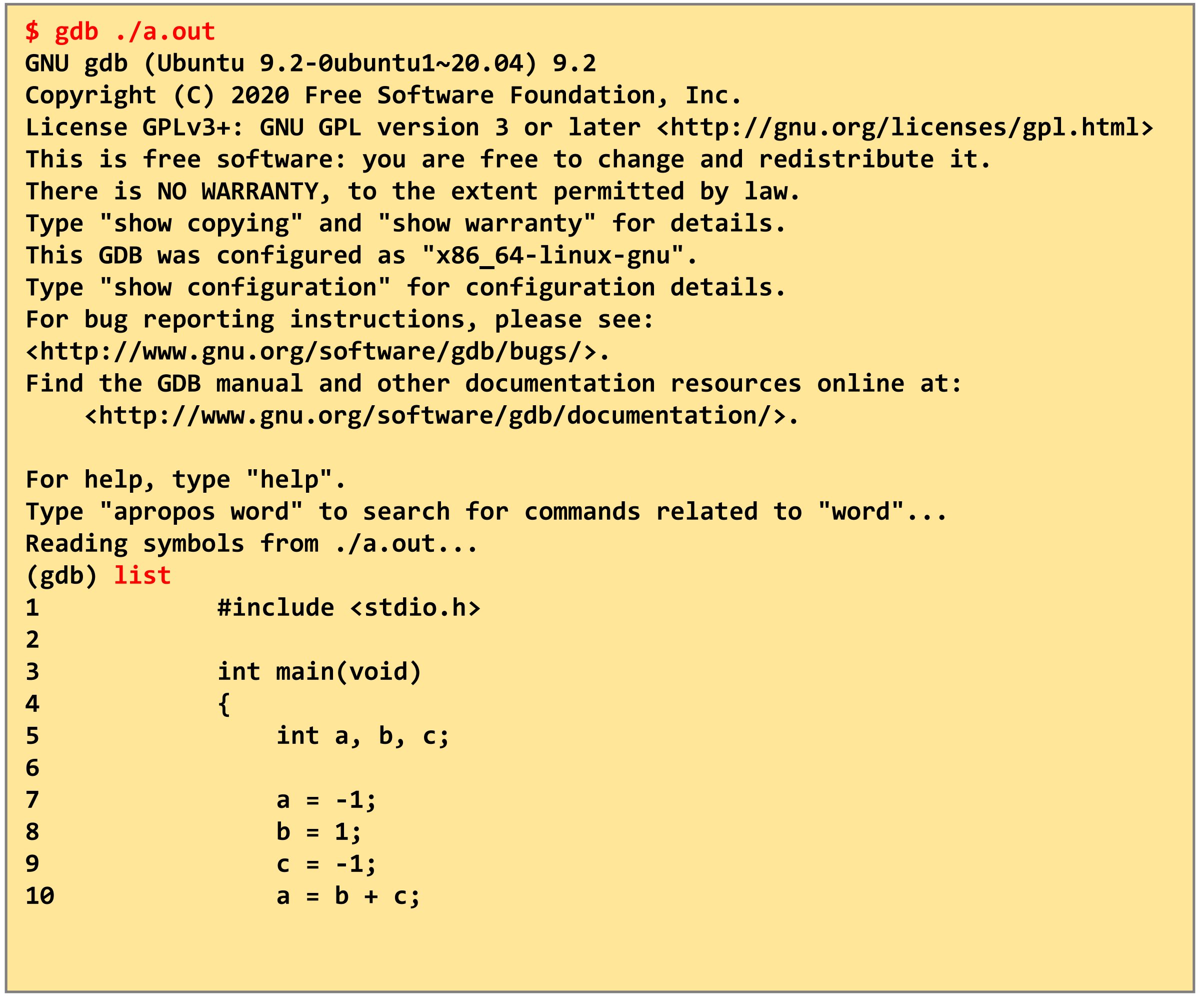
```
$ sudo apt install gdb
```

  - 컴파일시 디버깅 정보 추가

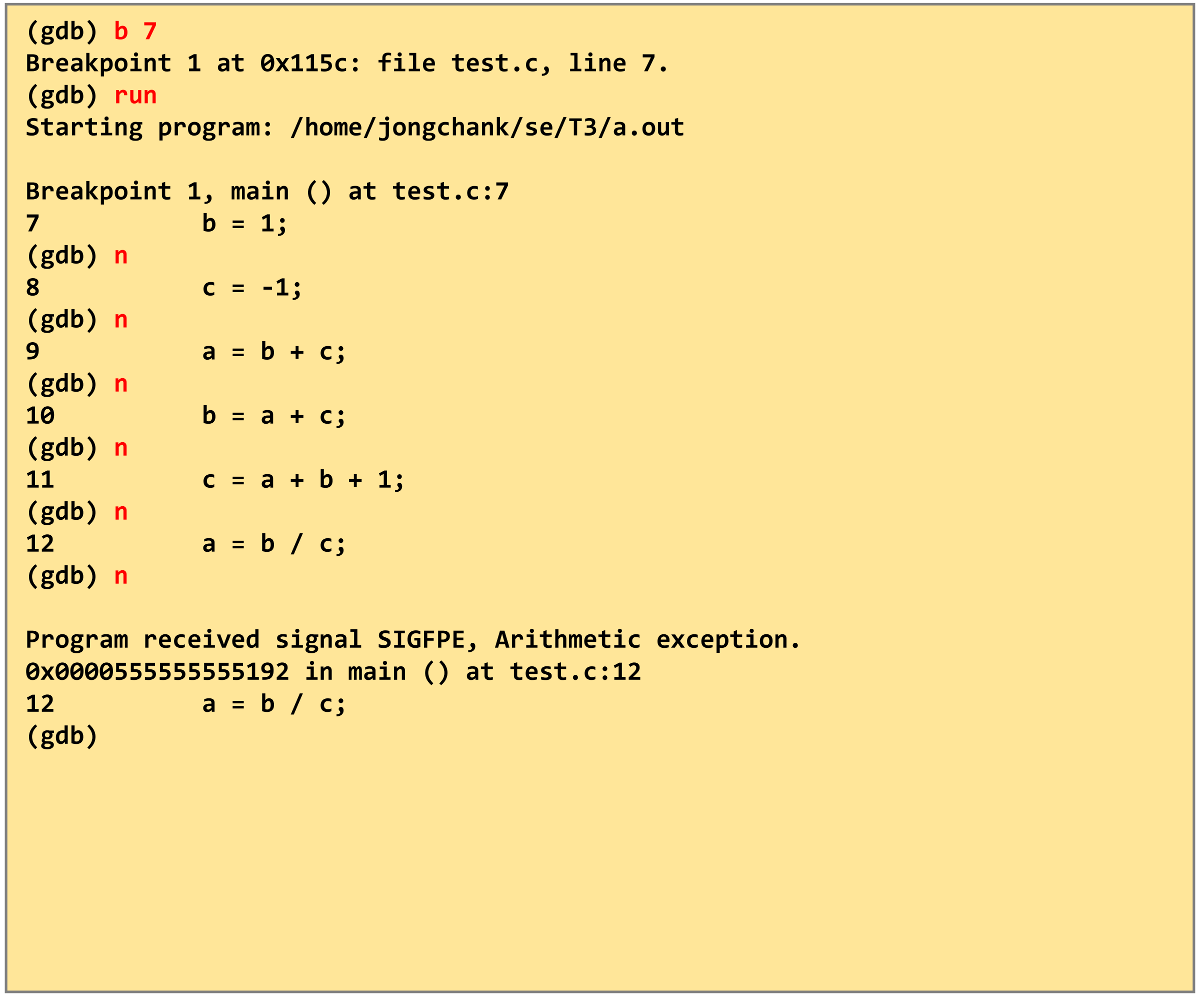
```
$ gcc -g test.c
```



Interactive Debugging



```
$ gdb ./a.out
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
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 ./a.out...
(gdb) list
1             #include <stdio.h>
2
3             int main(void)
4             {
5                 int a, b, c;
6
7                 a = -1;
8                 b = 1;
9                 c = -1;
10                a = b + c;
```

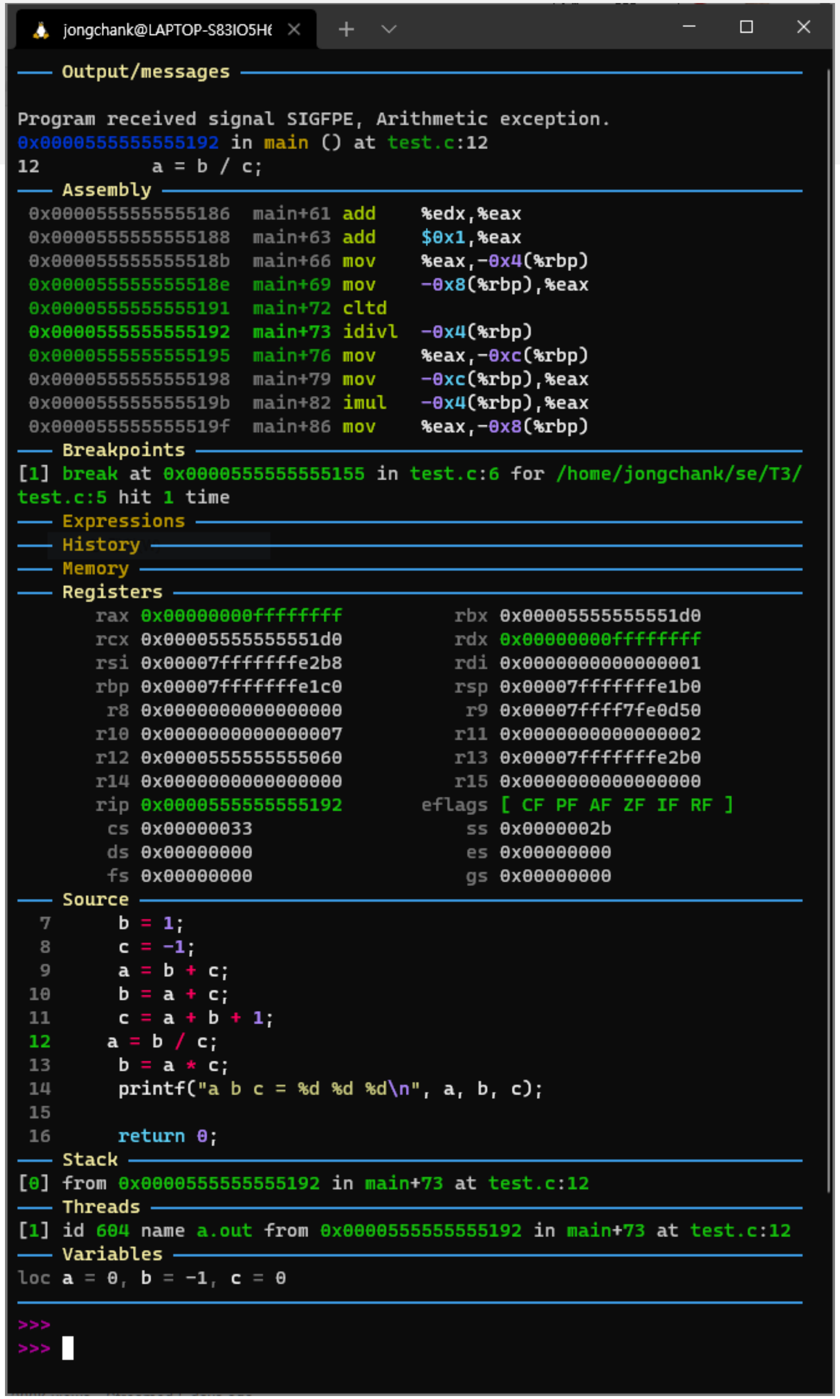


```
(gdb) b 7
Breakpoint 1 at 0x115c: file test.c, line 7.
(gdb) run
Starting program: /home/jongchank/se/T3/a.out

Breakpoint 1, main () at test.c:7
7             b = 1;
(gdb) n
8             c = -1;
(gdb) n
9             a = b + c;
(gdb) n
10            b = a + c;
(gdb) n
11            c = a + b + 1;
(gdb) n
12            a = b / c;
(gdb) n

Program received signal SIGFPE, Arithmetic exception.
0x0000555555555192 in main () at test.c:12
12            a = b / c;
(gdb)
```

GDB 프론트엔드: GDB dashboard

- https://github.com/cyrus-and/gdb-dashboard

```
$ sudo apt install python3-pip
$ wget -P ~ https://git.io/.gdbinit
$ pip install pygments
```



```
─── Output/messages ─────────────────────────────────────────

Program received signal SIGFPE, Arithmetic exception.
0x0000555555555192 in main () at test.c:12
12          a = b / c;
─── Assembly ────────────────────────────────────────────────
 0x0000555555555186  main+61 add    %edx,%eax
 0x0000555555555188  main+63 add    $0x1,%eax
 0x000055555555518b  main+66 mov    %eax,-0x4(%rbp)
 0x000055555555518e  main+69 mov    -0x8(%rbp),%eax
 0x0000555555555191  main+72 cltd
 0x0000555555555192  main+73 idivl  -0x4(%rbp)
 0x0000555555555195  main+76 mov    %eax,-0xc(%rbp)
 0x0000555555555198  main+79 mov    -0xc(%rbp),%eax
 0x000055555555519b  main+82 imul   -0x4(%rbp),%eax
 0x000055555555519f  main+86 mov    %eax,-0x8(%rbp)
─── Breakpoints ─────────────────────────────────────────────
[1] break at 0x0000555555555155 in test.c:6 for /home/jongchank/se/T3/
test.c:5 hit 1 time
─── Expressions ─────────────────────────────────────────────
─── History ─────────────────────────────────────────────────
─── Memory ──────────────────────────────────────────────────
─── Registers ───────────────────────────────────────────────
      rax 0x00000000ffffffff          rbx 0x00005555555551d0
      rcx 0x00005555555551d0          rdx 0x00000000ffffffff
      rsi 0x00007fffffffe2b8          rdi 0x0000000000000001
      rbp 0x00007fffffffe1c0          rsp 0x00007fffffffe1b0
       r8 0x0000000000000000           r9 0x00007ffff7fe0d50
      r10 0x0000000000000007          r11 0x0000000000000002
      r12 0x0000555555555060          r13 0x00007fffffffe2b0
      r14 0x0000000000000000          r15 0x0000000000000000
      rip 0x0000555555555192       eflags [ CF PF AF ZF IF RF ]
       cs 0x00000033                   ss 0x0000002b
       ds 0x00000000                   es 0x00000000
       fs 0x00000000                   gs 0x00000000
─── Source ──────────────────────────────────────────────────
 7        b = 1;
 8        c = -1;
 9        a = b + c;
10        b = a + c;
11        c = a + b + 1;
12       a = b / c;
13        b = a * c;
14        printf("a b c = %d %d %d\n", a, b, c);
15
16        return 0;
─── Stack ───────────────────────────────────────────────────
[0] from 0x0000555555555192 in main+73 at test.c:12
─── Threads ─────────────────────────────────────────────────
[1] id 604 name a.out from 0x0000555555555192 in main+73 at test.c:12
─── Variables ───────────────────────────────────────────────
loc a = 0, b = -1, c = 0
─────────────────────────────────────────────────────────────

>>>
>>> 
```

Post-Mortem Debugging (死後 디버깅)

core.c

```c
#include <stdio.h>
#include <string.h>
void outer_function(void);
void inner_function(void);
void bad_function(void);
int main(void)
{
    outer_function();
    return 0;
}
void outer_function(void)
{
    inner_function();
}
void inner_function(void)
{
    bad_function();
}
void bad_function(void)
{
    char a[100];
    char *p = 0;

    memcpy(p, a, sizeof(a));
}
```

```
$ gcc -g core.c
$ ./a.out
Segmentation fault
$ ulimit -c unlimited
$ ./a.out
Segmentation fault (core dumped)
$ ls -l core
-rw------- 1 jongchank jongchank 245760 Sep 29 23:05 core
$ gdb ./a.out core
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
…
#0  0x00005620db4571b6 in bad_function () at core.c:24
24          memcpy(p, a, sizeof(a));
>>> bt
#0  0x00005620db4571b6 in bad_function () at core.c:24
#1  0x00005620db45717a in inner_function () at core.c:17
#2  0x00005620db45716a in outer_function () at core.c:13
#3  0x00005620db457156 in main () at core.c:8
>>>
```

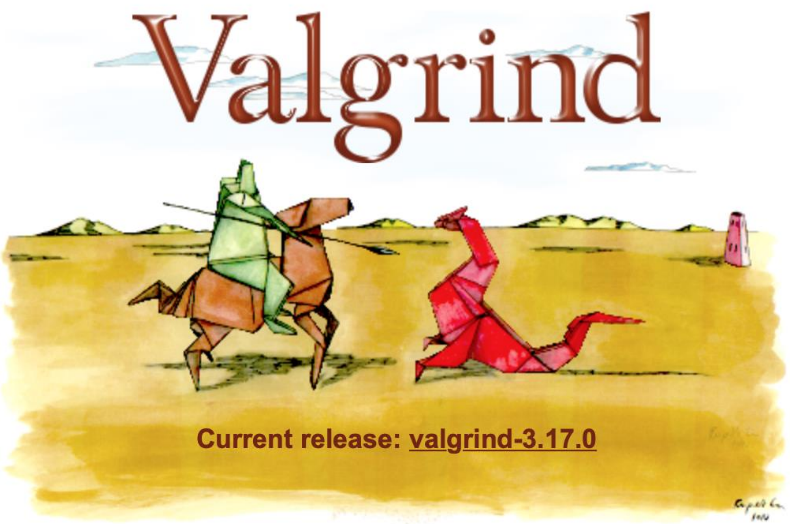
Call Stack

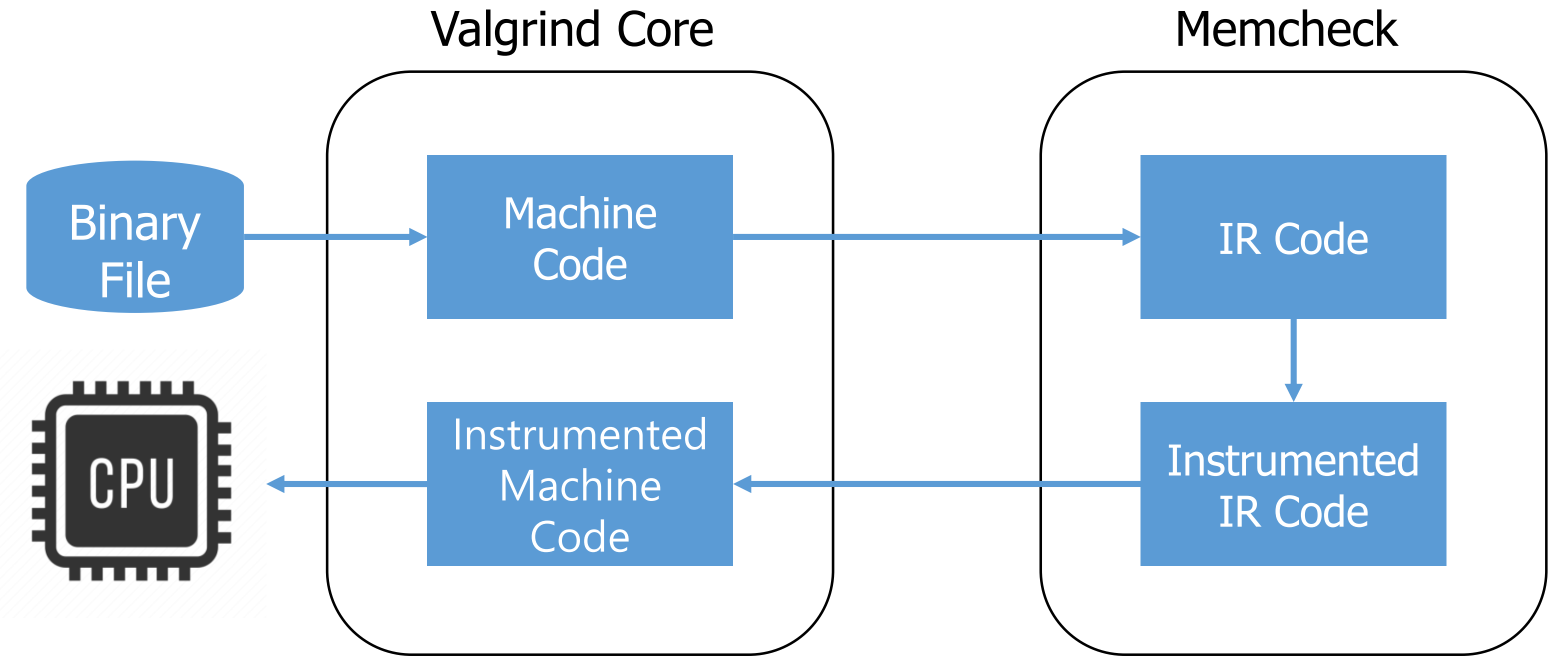
Binary Instrumentation Tools

- Valgrind
  - 설치

```
$ sudo apt install valgrind
```

  - 런타임 바이너리 변환을 지원하는 가상 머신
  - JIT (Just-In-Time) 컴파일





메모리 누수 (Memory Leakage) 검사

memory.c

```c
#include <stdio.h>
#include <stdlib.h>

void bad_function(void);

int main(void)
{
    bad_function();
    return 0;
}

void bad_function(void)
{
    char *p1, *p2, *p3;

    p1 = (char *)malloc(1024 * 204);
    p2 = (char *)malloc(1024 * 204);
    p3 = (char *)malloc(1024 * 204);

    free(p1);
    free(p2);
}
```

```
$ gcc -g memory.c
$ valgrind --tool=memcheck --leak-check=yes ./a.out
==880== Memcheck, a memory error detector
==880== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==880== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==880== Command: ./a.out
==880==
==880==
==880== HEAP SUMMARY:
==880==     in use at exit: 208,896 bytes in 1 blocks
==880==   total heap usage: 3 allocs, 2 frees, 626,688 bytes allocated
==880==
==880== 208,896 bytes in 1 blocks are definitely lost in loss record 1 of 1
==880==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_mem
check-amd64-linux.so)
==880==    by 0x1091AE: bad_function (memory.c:18)
==880==    by 0x109175: main (memory.c:8)
==880==
==880== LEAK SUMMARY:
==880==    definitely lost: 208,896 bytes in 1 blocks
==880==    indirectly lost: 0 bytes in 0 blocks
==880==      possibly lost: 0 bytes in 0 blocks
==880==    still reachable: 0 bytes in 0 blocks
==880==         suppressed: 0 bytes in 0 blocks
==880==
==880== For lists of detected and suppressed errors, rerun with: -s
==880== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Questions