

Debugging and Design

Computer Systems

Friday, November 01, 2024

Outline

■ Debugging

- Tools
- Code with a Bug

■ Design

- Managing complexity
- Communication
- Naming
- Comments

GDB

■ No longer stepping through assembly!

- Use the step/next commands
- break on line numbers, functions
- Use list to display code at line-numbers and functions
- Use print with variables

■ Use pwndbg

- Nice display for viewing source/executing commands

```

nshc@nshcdell: ~/computer_system/hexdump (ssh)
0x7fffffff350 00 10 00 00 00 00 00 00 01 00 00 00 05 00 00 00 .....0x7fffffff360 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00 00 .....[Inferior 1 (process 2031547) exited normally]
pwndbg> b main
Breakpoint 1 at 0x5555555553ce
pwndbg> r
Starting program: /home/nshc/moon/computer_system/hexdump/hexdump

Breakpoint 1, 0x0000555555553ce in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]  

[RAX 0x5555555553ce (main) ← endbr64
[RBX 0x555555555470 (.libc_csu_init) ← endbr64
[RDX 0x555555555470 (.libc_csu_init) ← endbr64
[RDX 0x7fffffff488 → 0x7fffffff70f ← 'SHELL=/bin/bash'
[RDI 0x1
[RSI 0x7fffffff478 → 0x7fffffff6df ← '/home/nshc/moon/computer_system/hexdump/hexdump'
[R8 0x0
[R9 0x7ffff7fe0d60 (_dl_fini) ← endbr64
[R10 0x3
[R11 0x0
[R12 0x555555555120 (_start) ← endbr64
[R13 0x7fffffff470 ← 0x1
[R14 0x0
[R15 0x0
[RBP 0x0
[RSP 0x7ffff7de388 → 0x7ffff7de6083 (_libc_start_main+243) ← mov edi, eax
[RIP 0x5555555553ce (main) ← endbr64  

[ DISASM / x86-64 / set emulate on ]  

[ STACK ]  

▶ 0x5555555553ce <main> endbr64
0x5555555553d2 <main+4> push rbp
0x5555555553d5 <main+8> mov rbp, rsp
0x5555555553d8 <main+8> sub rsp, 0x120
0x5555555553d9 <main+15> mov rax, qword ptr fs:[0x28]
0x5555555553e0 <main+24> mov qword ptr [rbp - 8], rax
0x5555555553e1 <main+28> xor eax, eax
0x5555555553e3 <main+30> lea rsi, [rip + 0xc1c]
0x5555555553e5 <main+37> lea rdi, [rip + 0xc18]
0x5555555553f0 <main+44> call fopen@plt <open@plt>
0x5555555553ff <main+49> mov qword ptr [rbp - 0x118], rax  

[ BACKTRACE ]  

00:0000 rsp 0x7fffffff388 → 0x7ffff7de6083 (_libc_start_main+243) ← mov edi, eax
01:0008 0x7fffffff390 → 0x7ffff7ffc620 (.rtld_global_ro) ← 0x504eb00000000
02:0010 0x7fffffff398 → 0x7fffffff478 → 0x7fffffff6df ← '/home/nshc/moon/computer_system/hexdump/hexdump'  

03:0018 0x7fffffff3a0 ← 0x100000000
04:0020 0x7fffffff3a8 → 0x5555555553ce (main) ← endbr64
05:0028 0x7fffffff3b0 → 0x555555555470 (.libc_csu_init) ← endbr64
06:0030 0x7fffffff3b8 ← 0x927d733d677e2cd
07:0038 0x7fffffff3c0 → 0x555555555120 (_start) ← endbr64  

[ BACKTRACE ]  

▶ 0 0x5555555553ce main
1 0x7ffff7de6083 _libc_start_main+243  

pwndbg> [ BACKTRACE ]

```

<https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>

Valgrind

- Find memory errors, detect memory leaks
- Common errors:
 - Illegal read/write errors
 - Use of uninitialized values
 - Illegal frees
 - Overlapping source/destination addresses
- Typical solutions
 - Did you allocate enough memory?
 - Did you accidentally free stack variables/something twice?
 - Did you initialize all your variables?
 - Did you use something that you just freed?
- **--leak-check=full**
 - Memcheck gives details for each definitely/possibly lost memory block (where it was allocated)



The screenshot shows a terminal window titled "Terminal". The command run is `valgrind ./memleak`. The output is as follows:

```
[pwellis2@newcell ~]# valgrind ./memleak
==16738== Memcheck, a memory error detector
==16738== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==16738== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==16738== Command: ./memleak
==16738==
==16738== Invalid write of size 4
==16738==   at 0x400589: main (mem_leak.c:32)
==16738==   Address 0x4c26068 is 0 bytes after a block of size 40 alloc'd
==16738==     at 0xA0646F: malloc (vg_replace_malloc.c:236)
==16738==   by 0x400505: main (mem_leak.c:17)
==16738==
==16738== Invalid read of size 4
==16738==   at 0x400598: main (mem_leak.c:33)
==16738==   Address 0x4c26068 is 0 bytes after a block of size 40 alloc'd
==16738==     at 0xA0646F: malloc (vg_replace_malloc.c:236)
==16738==   by 0x400505: main (mem_leak.c:17)
==16738==
==16738== HEAP SUMMARY:
==16738==   in use at exit: 410 bytes in 8 blocks
==16738==   total heap usage: 11 allocs, 3 frees, 590 bytes allocated
==16738==
==16738== LEAK SUMMARY:
==16738==   definitely lost: 410 bytes in 8 blocks
==16738==   indirectly lost: 0 bytes in 0 blocks
==16738==   possibly lost: 0 bytes in 0 blocks
==16738==   still reachable: 0 bytes in 0 blocks
==16738==   suppressed: 0 bytes in 0 blocks
==16738== Rerun with --leak-check=full to see details of leaked memory
==16738==
==16738== For counts of detected and suppressed errors, rerun with: -v
==16738== ERROR SUMMARY: 36 errors from 2 contexts (suppressed: 4 from 4)
[pwellis2@newcell ~]#
```

A blue box highlights the "HEAP SUMMARY" and "LEAK SUMMARY" sections.

Code with a Bug

```
#include <stdio.h>

int fact(int n) {
    if(n == 1)
        return n;
    else
        return n * fact(n-1);
}

int main() {
    int n;
    for (n=0; n<20; n++)
    {
        printf("factorial of %d: \t %d\n" , n, fact(n));
    }
    return 0;
}
```

\$./fact
Segmentation fault (core dumped)

Code with a Bug

```

pwndbg> r
Starting program: /home/nshc/moon/computer_system/07_debugging/fact

Program received signal SIGSEGV, Segmentation fault.
0x00000000 in fact (fact+41)

LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]-
[EAX 0xffffc005a ← 0x91c4ffff
*EBX 0x804c000 (_GLOBAL_OFFSET_TABLE_) → 0x804bf14 (_DYNAMIC) ← 0x1
*ECX 0xfffffd500 ← 0x1
*EDX 0xfffffd524 ← 0x0
*EDI 0xf7fb1000 (_GLOBAL_OFFSET_TABLE_) ← 0xlead6c
*ESI 0xf7fb1000 (_GLOBAL_OFFSET_TABLE_) ← 0xlead6c
*EBP 0xff7fe018 → 0xff7fe038 → 0xff7fe058 → 0xff7fe078 → 0xff7fe098 ← ...
*ESP 0xff7fe000 → 0xffffc005a ← 0x91c4ffff
*EIP 0x80491bf (fact+41) → 0xfffffd2e8 ← 0x3
[ DISASM / i386 / set emulate on ]-
► 0x80491bf <fact+41> call fact <fact>
    arg[0]: 0xffffc005a ← 0x91c4ffff
    arg[1]: 0x0
    arg[2]: 0x0
    arg[3]: 0x80491a5 (fact+15) ← add eax, 0x2e5b
0x80491c4 <fact+46> add esp, 0x10
0x80491c7 <fact+53> imul eax, dword ptr [ebp + 8]
0x80491cb <fact+53> leave
0x80491cc <fact+54> ret
0x80491cd <main> endbr32
0x80491d1 <main+4> lea ecx, [esp + 4]
0x80491d5 <main+8> and esp, 0xffffffff
0x80491d8 <main+11> push dword ptr [ecx - 4]
0x80491db <main+14> push ebp
0x80491dc <main+15> mov ebp, esp
[ STACK ]-
00:0000 esp 0xff7fe000 → 0xffffc005a ← 0x91c4ffff
01:0004 0xff7fe004 ← 0x0
02:0008 0xff7fe008 ← 0x0
03:000c 0xff7fe00c → 0x80491a5 (fact+15) ← add eax, 0x2e5b
04:0010 0xff7fe010 ← 0x0
05:0014 0xff7fe014 ← 0x0
06:0018 ebp 0xff7fe018 → 0xff7fe038 → 0xff7fe058 → 0xff7fe078 → 0xff7fe098 ← ...
07:001c 0xff7fe01c → 0x80491c4 (fact+46) ← add esp, 0x10
[ BACKTRACE ]-
► 0 0x80491bf fact+41
1 0x80491c4 fact+46
2 0x80491c4 fact+46
3 0x80491c4 fact+46
4 0x80491c4 fact+46
5 0x80491c4 fact+46
6 0x80491c4 fact+46
7 0x80491c4 fact+46
[ ]
pwndbg>

```

Code with a Bug

```
pwndbg> info thread
   Id  Target Id          Frame
* 1    process 3280933 "fact" 0x080491bf in fact ()
pwndbg> cat /proc/3280933/maps
08048000-08049000 r--p 00000000 fd:00 4463972
08049000-0804a000 r-xp 00001000 fd:00 4463972
0804a000-0804b000 r--p 00002000 fd:00 4463972
0804b000-0804c000 r--p 00002000 fd:00 4463972
0804c000-0804d000 rw-p 00003000 fd:00 4463972
f7dc6000-f7ddf000 r--p 00000000 fd:00 6041625
f7ddf000-f7f3a000 r-xp 00019000 fd:00 6041625
f7f3a000-f7fae000 r--p 00174000 fd:00 6041625
f7fae000-f7faf000 ---p 001e8000 fd:00 6041625
f7faf000-f7fb1000 r--p 001e8000 fd:00 6041625
f7fb1000-f7fb2000 rw-p 001ea000 fd:00 6041625
f7fb2000-f7fb5000 rw-p 00000000 00:00 0
f7fca000-f7fcc000 rw-p 00000000 00:00 0
f7fcc000-f7fcf000 r--p 00000000 00:00 0
f7fcf000-f7fd1000 r-xp 00000000 00:00 0
f7fd1000-f7fd2000 r--p 00000000 fd:00 6041336
f7fd2000-f7ff0000 r-xp 00001000 fd:00 6041336
f7ff0000-f7ffb000 r--p 0001f000 fd:00 6041336
f7ffc000-f7ffd000 r--p 0002a000 fd:00 6041336
f7ffd000-f7ffe000 rw-p 0002b000 fd:00 6041336
ff7fe000-ffffe000 rw-p 00000000 00:00 0
[stack]
This command is deprecated in Pwndbg. Please use the GDB's built-in syntax for running shell commands instead: !cat <args>
pwndbg>
```

Code with a Bug

```
pwndbg> vmmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
      Start          End Perm      Size Offset File
0x8048000 0x8049000 r--p    1000      0 /home/nshc/moon/computer_system/07_debugging/fact
0x8049000 0x804a000 r-xp    1000  1000 /home/nshc/moon/computer_system/07_debugging/fact
0x804a000 0x804b000 r--p    1000  2000 /home/nshc/moon/computer_system/07_debugging/fact
0x804b000 0x804c000 r--p    1000  2000 /home/nshc/moon/computer_system/07_debugging/fact
0x804c000 0x804d000 rw-p    1000  3000 /home/nshc/moon/computer_system/07_debugging/fact
0xf7dc6000 0xf7ddf000 r--p   19000      0 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7ddf000 0xf7f3a000 r-xp  15b000  19000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7f3a000 0xf7fae000 r--p   74000 174000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7fae000 0xf7faf000 ---p   1000 1e8000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7faf000 0xf7fb1000 r--p   2000 1e8000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7fb1000 0xf7fb2000 rw-p   1000 1ea000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7fb2000 0xf7fb5000 rw-p   3000      0 [anon_f7fb2]
0xf7fc000 0xf7fcc000 rw-p   2000      0 [anon_f7fc0]
0xf7fcc000 0xf7fcf000 r--p   3000      0 [vvar]
0xf7fcf000 0xf7fd1000 r-xp   2000      0 [vdso]
0xf7fd1000 0xf7fd2000 r--p   1000      0 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7fd2000 0xf7ff0000 r-xp  1e000  1000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7ff0000 0xf7ffb000 r--p   b000 1f000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7ffc000 0xf7ffd000 r--p   1000 2a000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7ffd000 0xf7ffe000 rw-p   1000 2b000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xff7fe000 0xffffe000 rw-p  800000      0 [stack]
pwndbg>
```

Code with a Bug

```
nshc@nshcdell:~/computer_system/07_debugging$ valgrind ./fact
==3281737== Memcheck, a memory error detector
==3281737== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3281737== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3281737== Command: ./fact
==3281737==
==3281737== Stack overflow in thread #1: can't grow stack to 0xfe522000
==3281737==
==3281737== Process terminating with default action of signal 11 (SIGSEGV)
==3281737== Access not within mapped region at address 0xFE522FFC
==3281737== Stack overflow in thread #1: can't grow stack to 0xfe522000
==3281737==     at 0x80491A0: fact (in /home/nshc/moon/computer_system/07_debugging/fact)
==3281737== If you believe this happened as a result of a stack
==3281737== overflow in your program's main thread (unlikely but
==3281737== possible), you can try to increase the size of the
==3281737== main thread stack using the --main-stacksize= flag.
==3281737== The main thread stack size used in this run was 8388608.
==3281737== Stack overflow in thread #1: can't grow stack to 0xfe522000
--3281737-- VALGRIND INTERNAL ERROR: Valgrind received a signal 11 (SIGSEGV) - exiting
--3281737-- si_code=1; Faulting address: 0xFE522FFC; sp: 0x82c36f20

valgrind: the 'impossible' happened:
  Killed by fatal signal

host stacktrace:
==3281737==     at 0x580B5272: ??? (in /usr/lib/x86_64-linux-gnu/valgrind	memcheck-x86-linux)

sched status:
  running_tid=1

Thread 1: status = VgTs_Runnable (lwpid 3281737)
Segmentation fault (core dumped)
nshc@nshcdell:~/computer_system/07_debugging$
```

Practice : rec_main.c

■ rec_main

```
#include <stdio.h>

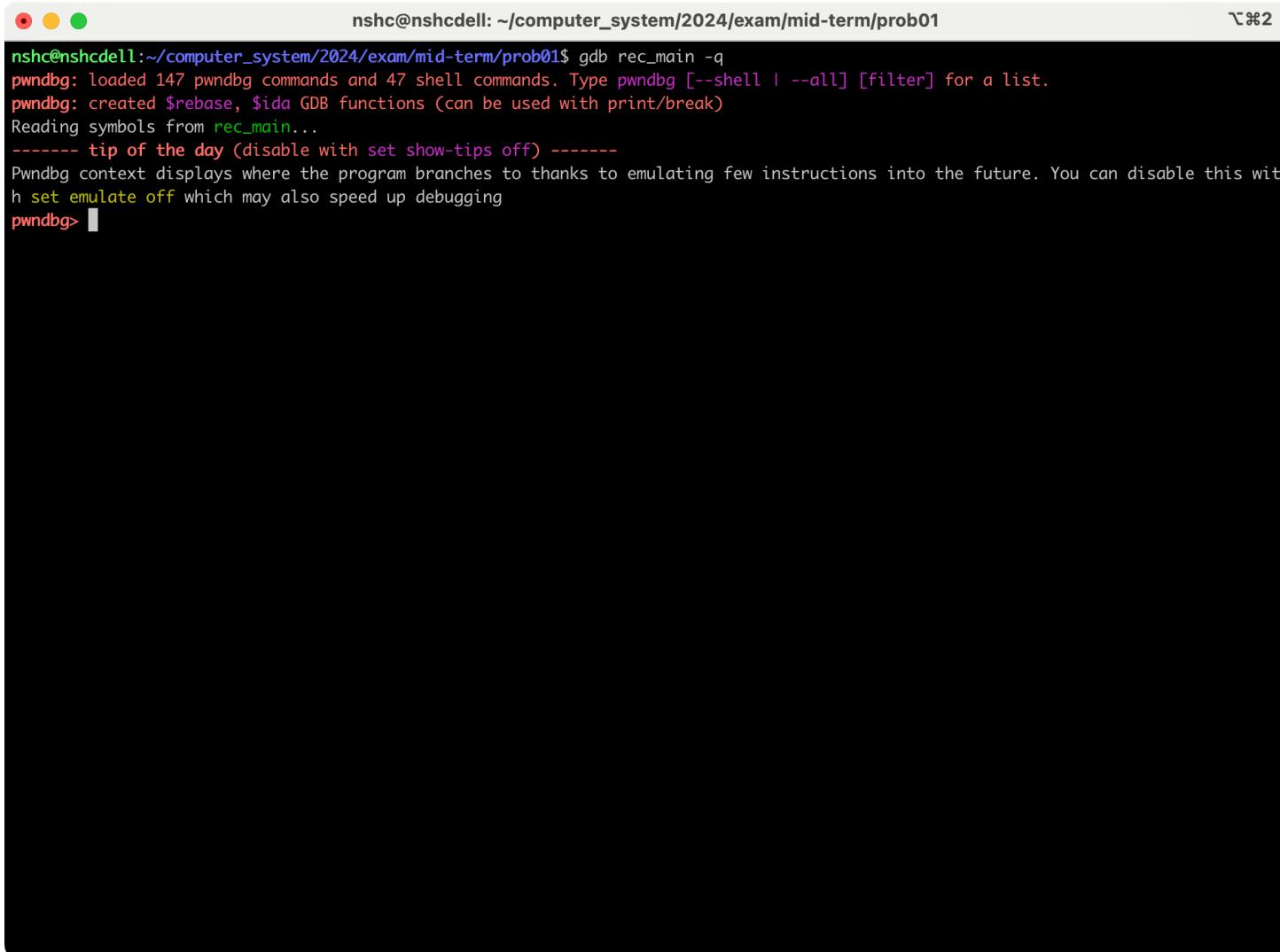
int main(int argc)
{
    if (argc > 0x800)
        printf("The result is %d\n", argc);
    else
        return main(argc<<1);
}
```

```
>>> while argc <= 0x800 :
...     argc = argc<<1
...
>>> argc
4096
```

```
nshc@nshcdell:~/computer_system/2024/exam/mid-term/prob01$ ./rec_main
The result is 4096
```

rec_main.c

■ rec_main (debugging)



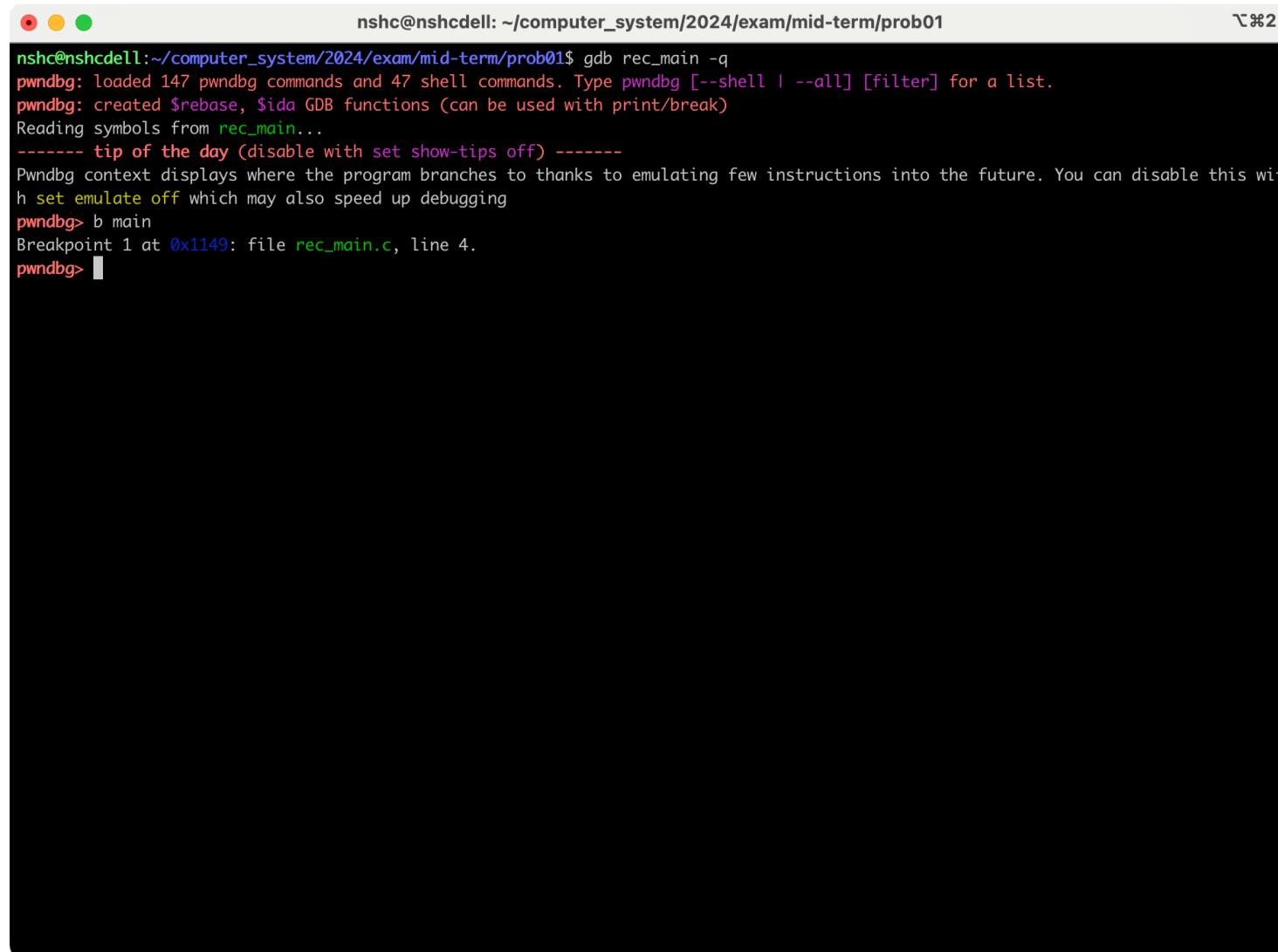
The screenshot shows a terminal window titled "nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01". The command entered is \$gdb rec_main -q. The output shows the GDB prompt (pwndbg>) and some initial setup messages.

```
nshc@nshcdell:~/computer_system/2024/exam/mid-term/prob01$ gdb rec_main -q
pwndbg: loaded 147 pwndbg commands and 47 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
Reading symbols from rec_main...
----- tip of the day (disable with set show-tips off) -----
Pwndbg context displays where the program branches to thanks to emulating few instructions into the future. You can disable this with set emulate off which may also speed up debugging
pwndbg> 
```

\$gdb rec_main -q

rec_main.c

■ rec_main (debugging)



The screenshot shows a terminal window with the following content:

```
nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01$ gdb rec_main -q
pwndbg: loaded 147 pwndbg commands and 47 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
Reading symbols from rec_main...
----- tip of the day (disable with set show-tips off) -----
Pwndbg context displays where the program branches to thanks to emulating few instructions into the future. You can disable this with set emulate off which may also speed up debugging
pwndbg> b main
Breakpoint 1 at 0x1149: file rec_main.c, line 4.
pwndbg>
```

```
$gdb rec_main -q
> break main
```

rec_main.c

■ rec_main (debugging)

The screenshot shows the pwndbg debugger interface with the following sections:

- [DISASM / x86-64 / set emulate on]**: Assembly code listing for the main function.
- In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c**: Source code for the main function.
- [STACK]**: Stack dump showing memory locations from 00:0000 to 07:0038.
- [BACKTRACE]**: Backtrace showing the call stack.

```

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
[ DISASM / x86-64 / set emulate on ]
▶ 0x55555555149 <main>      endbr64
    push rbp
    mov rbp, rsp
    sub rsp, 0x10
    mov dword ptr [rbp - 4], edi
    cmp dword ptr [rbp - 4], 0x800
    jle main+53                <main+53>
    +
    mov eax, dword ptr [rbp - 4]
    add eax, eax
    mov edi, eax
    call main                  <main>
[ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
1 #include <stdio.h>
2
3 int main(int argc)
▶ 4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }
[ STACK ]
00:0000| rsp 0x7fffffff348 --> 0x7ffff7de6083 (_libc_start_main+243) ← mov edi, eax
01:0008| 0x7fffffff350 --> 0x7ffff7ffc620 (_rtld_global_ro) ← 0x504ff00000000
02:0010| 0x7fffffff358 --> 0x7fffffff438 --> 0x7fffffff691 ← '/home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main'
03:0018| 0x7fffffff360 ← 0x100000000
04:0020| 0x7fffffff368 --> 0x55555555149 (main) ← endbr64
05:0028| 0x7fffffff370 --> 0x55555555190 (_libc_csu_init) ← endbr64
06:0030| 0x7fffffff378 ← 0xe39123f564679e12
07:0038| 0x7fffffff380 --> 0x55555555060 (_start) ← endbr64
[ BACKTRACE ]
▶ 0 0x55555555149 main
  1 0x7ffff7de6083 _libc_start_main+243

```

```
$gdb rec_main -q
> break main
> run
```

rec_main.c

■ rec_main (debugging)

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01

```

0x55555555514d <main+4>    push   rbp
0x55555555514e <main+5>    mov    rbp, rsp
0x555555555151 <main+8>    sub    rbp, 0x10
0x555555555155 <main+12>   mov    dword ptr [rbp - 4], edi
0x555555555158 <main+15>   cmp    dword ptr [rbp - 4], 0x800
0x55555555515f <main+22>   jle    main+53           <main+53>
    ↓
0x55555555517e <main+53>   mov    eax, dword ptr [rbp - 4]
0x555555555181 <main+56>   add    eax, eax
0x555555555183 <main+58>   mov    edi, eax
0x555555555185 <main+60>   call   main              <main>
[ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
1 #include <stdio.h>
2
3 int main(int argc)
▶ 4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }

[ STACK ]
00:0000 | rsp 0x7fffffff348 --> 0x7ffff7de6083 (_libc_start_main+243) ← mov edi, eax
01:0008 | 0x7fffffff350 --> 0x7ffff7fc620 (_rtld_global_ro) ← 0x504ff00000000
02:0010 | 0x7fffffff358 --> 0x7fffffff438 --> 0x7fffffff691 ← '/home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main'
03:0018 | 0x7fffffff360 ← 0x1000000000
04:0020 | 0x7fffffff368 --> 0x555555555149 (main) ← endbr64
05:0028 | 0x7fffffff370 --> 0x555555555190 (_libc_csu_init) ← endbr64
06:0030 | 0x7fffffff378 ← 0xe39123f564679e12
07:0038 | 0x7fffffff380 --> 0x555555555060 (_start) ← endbr64
[ BACKTRACE ]
▶ 0 0x555555555149 main
1 0x7ffff7de6083 _libc_start_main+243

pwndbg> b 8
Breakpoint 2 at 0x55555555517e: file rec_main.c, line 8.
pwndbg> █

```

```
$gdb rec_main -q
> break main
> run
> break 8
```

rec_main.c

■ rec_main (debugging)

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01

```
*RIP 0x555555555517e (main+53) ← mov eax, dword ptr [rbp - 4]
[ DISASM / x86-64 / set emulate on ]
▶ 0x555555555517e <main+53>      mov    eax, dword ptr [rbp - 4]
0x5555555555181 <main+56>        add    eax, eax
0x5555555555183 <main+58>        mov    edi, eax
0x5555555555185 <main+60>        call   main           <main>

0x555555555518a <main+65>        leave
0x555555555518b <main+66>        ret

0x555555555518c                  nop    dword ptr [rax]
0x5555555555190 <__libc_csu_init> endbr64
0x5555555555194 <__libc_csu_init+4> push   r15
0x5555555555196 <__libc_csu_init+6> lea    r15, [rip + 0x2c1b]     <__init_array_start>
0x555555555519d <__libc_csu_init+13> push   r14
[ SOURCE (CODE) ]
```

In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c

```
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }
```

[STACK]

```
00:0000| rsp 0x7fffffff330 → 0x7fffffff340 ← 0x1
01:0008| 0x7fffffff338 ← 0x100000000
02:0010| rbp 0x7fffffff340 ← 0x0
03:0018| 0x7fffffff348 → 0x7ffff7de6083 (__libc_start_main+243) ← mov edi, eax
04:0020| 0x7fffffff350 → 0x7ffff7ffc620 (.rtld_global_ro) ← 0x504ff00000000
05:0028| 0x7fffffff358 → 0x7fffffff438 → 0x7fffffff691 ← '/home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main'
06:0030| 0x7fffffff360 ← 0x100000000
07:0038| 0x7fffffff368 → 0x55555555149 (main) ← endbr64
[ BACKTRACE ]
```

```
▶ 0 0x555555555517e main+53
1 0x7ffff7de6083 __libc_start_main+243
```

pwndbg>

```
$gdb rec_main -q
> break main
> run
> break 8
> continue
```

rec_main.c

■ rec_main (debugging)

The screenshot shows a terminal window with the following content:

```

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
[ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }

[ STACK ]
00:0000| rsp 0x7fffffff330 -> 0x7fffffff430 ← 0x1
01:0008|   0x7fffffff338 ← 0x100000000
02:0010| rbp 0x7fffffff340 ← 0x0
03:0018|   0x7fffffff348 -> 0x7ffff7de6083 (__libc_start_main+243) ← mov edi, eax
04:0020|   0x7fffffff350 -> 0x7ffff7ffc620 (_rtld_global_ro) ← 0x504ff00000000
05:0028|   0x7fffffff358 -> 0x7fffffff438 -> 0x7fffffff691 ← '/home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main'
06:0030|   0x7fffffff360 ← 0x100000000
07:0038|   0x7fffffff368 -> 0x55555555149 (main) ← endbr64

[ BACKTRACE ]
▶ 0 0x5555555517e main+53
    1 0x7ffff7de6083 __libc_start_main+243

pwndbg> p/d argc
$1 = 1
pwndbg>

```

```

$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc

```

rec_main.c

■ rec_main (debugging)

```
nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
```

```

0x555555555518a <main+65>      leave
0x555555555518b <main+66>      ret

0x555555555518c      nop    dword ptr [rax]
0x5555555555190 <__libc_csu_init> endbr64
0x5555555555194 <__libc_csu_init+4> push   r15
0x5555555555196 <__libc_csu_init+6> lea    r15, [rip + 0x2c1b]      <__init_array_start>
0x555555555519d <__libc_csu_init+13> push   r14
[ SOURCE (CODE) ]
```

In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c

```

3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }
```

[STACK]

```

00:0000 | rsp 0x7fffffff310 -> 0x7ffff7fb32e8 (__exit_funcs_lock) ← 0x0
01:0008 | 0x7fffffff318 ← 0x255555190
02:0010 | rbp 0x7fffffff320 -> 0x7fffffff340 ← 0x0
03:0018 | 0x7fffffff328 -> 0x5555555518a (main+65) ← leave
04:0020 | 0x7fffffff330 -> 0x7fffffff430 ← 0x1
05:0028 | 0x7fffffff338 ← 0x100000000
06:0030 | 0x7fffffff340 ← 0x0
07:0038 | 0x7fffffff348 -> 0x7fff7de6083 (__libc_start_main+243) ← mov edi, eax
[ BACKTRACE ]
```

```

▶ 0 0x555555555517e main+53
  1 0x555555555518a main+65
  2 0x7fff7de6083 __libc_start_main+243
```

pwndbg> info b

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x000055555555149	in main at rec_main.c:4
					breakpoint already hit 2 times
2	breakpoint	keep	y	0x00005555555517e	in main at rec_main.c:8
					breakpoint already hit 2 times

pwndbg>

```
$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
```

rec_main.c

■ rec_main (debugging)

```

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
[ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }

[ STACK ]
00:0000 | rsp 0x7fffffff310 -> 0x7ffff7fb32e8 (__exit_funcs_lock) ← 0x0
01:0008 | 0x7fffffff318 ← 0x255555190
02:0010 | rbp 0x7fffffff320 -> 0x7fffffff340 ← 0x0
03:0018 | 0x7fffffff328 -> 0x5555555518a (main+65) ← leave
04:0020 | 0x7fffffff330 -> 0x7fffffff430 ← 0x1
05:0028 | 0x7fffffff338 ← 0x100000000
06:0030 | 0x7fffffff340 ← 0x0
07:0038 | 0x7fffffff348 -> 0x7ffff7de6083 (__libc_start_main+243) ← mov edi, eax

[ BACKTRACE ]
▶ 0 0x5555555517e main+53
  1 0x5555555518a main+65
  2 0x7ffff7de6083 __libc_start_main+243

pwndbg> info b
Num   Type      Disp Enb Address          What
1    breakpoint  keep y  0x000055555555149 in main at rec_main.c:4
      breakpoint already hit 2 times
2    breakpoint  keep y  0x00005555555517e in main at rec_main.c:8
      breakpoint already hit 2 times
pwndbg> print/d argc
$2 = 2
pwndbg> disa 1
pwndbg>

```

```

$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1

```

rec_main.c

■ rec_main (debugging)

```

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
[ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }

[ STACK ]
00:0000 | rsp 0x7fffffff310 --> 0x7ffff7fb32e8 (__exit_funcs_lock) ← 0x0
01:0008 | 0x7fffffff318 ← 0x255555190
02:0010 | rbp 0x7fffffff320 --> 0x7fffffff340 ← 0x0
03:0018 | 0x7fffffff328 --> 0x55555555518a (main+65) ← leave
04:0020 | 0x7fffffff330 --> 0x7fffffff430 ← 0x1
05:0028 | 0x7fffffff338 ← 0x100000000
06:0030 | 0x7fffffff340 ← 0x0
07:0038 | 0x7fffffff348 --> 0x7ffff7de6083 (__libc_start_main+243) ← mov edi, eax

[ BACKTRACE ]
▶ 0 0x55555555517e main+53
  1 0x55555555518a main+65
  2 0x7ffff7de6083 __libc_start_main+243

[ BACKTRACE ]
pwndbg> info b
Num Type Disp Enb Address What
1 breakpoint keep y 0x000055555555149 in main at rec_main.c:4
breakpoint already hit 2 times
2 breakpoint keep y 0x00005555555517e in main at rec_main.c:8
breakpoint already hit 2 times
pwndbg> print/d argc
$2 = 2
pwndbg> disa 1
pwndbg> info b
Num Type Disp Enb Address What
1 breakpoint keep n 0x000055555555149 in main at rec_main.c:4
breakpoint already hit 2 times
2 breakpoint keep y 0x00005555555517e in main at rec_main.c:8
breakpoint already hit 2 times
pwndbg>

```

```

$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1
> info break

```

rec_main.c

■ rec_main (debugging)

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01

```
[ DISASM / x86-64 / set emulate on ]
▶ 0x5555555517e <main+53>      mov    eax, dword ptr [rbp - 4]
0x55555555181 <main+56>        add    eax, eax
0x55555555183 <main+58>        mov    edi, eax
0x55555555185 <main+60>        call   main             <main>
                                leave
0x5555555518a <main+65>        ret
0x5555555518b <main+66>
                                nop    dword ptr [rax]
0x55555555190 <__libc_csu_init> endbr64
0x55555555194 <__libc_csu_init+4> push   r15
0x55555555196 <__libc_csu_init+6> lea    r15, [rip + 0x2c1b]    <_init_array_start>
0x5555555519d <__libc_csu_init+13> push   r14
[ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }
[ STACK ]
00:0000  rsp 0x7fffffff2f0 ← 0xc2
01:0008  0x7fffffff2f8 ← 0x4ffffe327
02:0010  rbp 0x7fffffff300 → 0x7fffffff320 → 0x7fffffff340 ← 0x0
03:0018  0x7fffffff308 → 0x5555555518a (main+65) ← leave
04:0020  0x7fffffff310 → 0x7fff7fb32e8 (_exit_funcs_lock) ← 0x0
05:0028  0x7fffffff318 ← 0x255555190
06:0030  0x7fffffff320 → 0x7fffffff340 ← 0x0
07:0038  0x7fffffff328 → 0x5555555518a (main+65) ← leave
[ BACKTRACE ]
▶ 0 0x5555555517e main+53
1 0x5555555518a main+65
2 0x5555555518a main+65
3 0x7ffff7de6083 __libc_start_main+243
[ pwndbg ]
```

```
$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1
> info break
> continue
```

rec_main.c

■ rec_main (debugging)

```
nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
0x555555555181 <main+56>      add    eax, eax
0x555555555183 <main+58>      mov    edi, eax
0x555555555185 <main+60>      call   main             <main>
                                leave
0x55555555518a <main+65>      ret
0x55555555518b <main+66>
                                [ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }
                                [ STACK ]
00:0000 | rsp 0x7fffffff2f0 ← 0xc2
01:0008 | 0x7fffffff2f8 ← 0x4ffffe327
02:0010 | rbp 0x7fffffff300 → 0x7fffffff320 → 0x7fffffff340 ← 0x0
03:0018 | 0x7fffffff308 → 0x5555555518a (main+65) ← leave
04:0020 | 0x7fffffff310 → 0x7ffff7fb32e8 (__exit_funcs_lock) ← 0x0
05:0028 | 0x7fffffff318 ← 0x255555190
06:0030 | 0x7fffffff320 → 0x7fffffff340 ← 0x0
07:0038 | 0x7fffffff328 → 0x5555555518a (main+65) ← leave
                                [ BACKTRACE ]
▶ 0 0x55555555517e main+53
1 0x55555555518a main+65
2 0x55555555518a main+65
3 0x7ffff7de6083 __libc_start_main+243
                                [ ]
pwndbg> p/d argc
$3 = 4
pwndbg>
```

```
$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1
> info break
> continue
> p/d argc
```

rec_main.c

■ rec_main (debugging)

```
nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
0x555555555181 <main+56>      add    eax, eax
0x555555555183 <main+58>      mov    edi, eax
0x555555555185 <main+60>      call   main             <main>
                                leave
0x55555555518a <main+65>      ret
0x55555555518b <main+66>
                                [ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }
                                [ STACK ]
00:0000 | rsp 0x7fffffff2f0 ← 0xc2
01:0008 | 0x7fffffff2f8 ← 0x4ffffe327
02:0010 | rbp 0x7fffffff300 → 0x7fffffff320 → 0x7fffffff340 ← 0x0
03:0018 | 0x7fffffff308 → 0x55555555518a (main+65) ← leave
04:0020 | 0x7fffffff310 → 0x7ffff7fb32e8 (__exit_funcs_lock) ← 0x0
05:0028 | 0x7fffffff318 ← 0x255555190
06:0030 | 0x7fffffff320 → 0x7fffffff340 ← 0x0
07:0038 | 0x7fffffff328 → 0x55555555518a (main+65) ← leave
                                [ BACKTRACE ]
▶ 0 0x55555555517e main+53
1 0x55555555518a main+65
2 0x55555555518a main+65
3 0x7ffff7de6083 __libc_start_main+243
                                [ ]
pwndbg> p/d argc
$3 = 4
pwndbg>
```

```
$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1
> info break
> continue
> p/d argc
```

rec_main.c

■ rec_main (debugging)

```

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
0x555555555185 <main+60>      call   main           <main>
0x55555555518a <main+65>      leave
0x55555555518b <main+66>      ret
0x55555555518c                 nop    dword ptr [rax]
0x555555555190 <__libc_csu_init> endbr64
0x555555555194 <__libc_csu_init+4> push   r15
0x555555555196 <__libc_csu_init+6> lea    r15, [rip + 0x2c1b]    <__init_array_start>
0x55555555519d <__libc_csu_init+13> push   r14
[ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }
[ STACK ]
00:0000| rsp 0x7fffffff270 ← 0x38000000380
01:0008| 0x7fffffff278 ← 0x4000000380
02:0010| rbp 0x7fffffff280 → 0x7fffffff2e0 → 0x7fffffff2e20 → 0x7fffffff2e0 → 0x7fffffff300 ← ...
03:0018| 0x7fffffff288 → 0x5555555518a (main+65) ← leave
04:0020| 0x7fffffff290 ← 0x0
05:0028| 0x7fffffff298 ← 0x2000000000
06:0030| 0x7fffffff2a0 → 0x7fffffff2e0 → 0x7fffffff2e20 → 0x7fffffff300 → 0x7fffffff320 ← ...
07:0038| 0x7fffffff2a8 → 0x5555555518a (main+65) ← leave
[ BACKTRACE ]
▶ 0 0x55555555517e main+53
1 0x55555555518a main+65
2 0x55555555518a main+65
3 0x55555555518a main+65
4 0x55555555518a main+65
5 0x55555555518a main+65
6 0x55555555518a main+65
7 0x7ffff7de6083 __libc_start_main+243

```

pwndbg>

```

$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1
> info break
> continue
> p/d argc
> continue 4

```

rec_main.c

■ rec_main (debugging)

The screenshot shows a terminal window with the following content:

```

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
0x55555555518a <main+65>      leave
0x55555555518b <main+66>      ret

0x55555555518c      nop    dword ptr [rax]
0x555555555190 <__libc_csu_init> endbr64
0x555555555194 <__libc_csu_init+4> push   r15
0x555555555196 <__libc_csu_init+6> lea    r15, [rip + 0x2c1b]    <__init_array_start>
0x55555555519d <__libc_csu_init+13> push   r14

[ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }

[ STACK ]
00:0000  rsp 0x7fffffff270 ← 0x38000000380
01:0008  0x7fffffff278 ← 0x4000000380
02:0010  rbp 0x7fffffff280 → 0x7fffffff2e2a0 → 0x7fffffff2e2c0 → 0x7fffffff2e2e0 → 0x7fffffff300 ← ...
03:0018  0x7fffffff288 → 0x55555555518a (main+65) ← leave
04:0020  0x7fffffff290 ← 0x0
05:0028  0x7fffffff298 ← 0x2000000000
06:0030  0x7fffffff2e2a0 → 0x7fffffff2e2c0 → 0x7fffffff2e2e0 → 0x7fffffff300 → 0x7fffffff320 ← ...
07:0038  0x7fffffff2e2a8 → 0x55555555518a (main+65) ← leave

[ BACKTRACE ]
▶ 0 0x55555555517e main+53
  1 0x55555555518a main+65
  2 0x55555555518a main+65
  3 0x55555555518a main+65
  4 0x55555555518a main+65
  5 0x55555555518a main+65
  6 0x55555555518a main+65
  7 0x7ffff7de6083 __libc_start_main+243

pwndbg> p/d argc
$4 = 64
pwndbg>

```

```

$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1
> info break
> continue
> p/d argc
> continue 4
> p/d argc

```

rec_main.c

■ rec_main (debugging)

The screenshot shows a GDB session for the file `rec_main.c`. The assembly code at the top shows the main function's entry point at `0x55555555185`, followed by a `call main` instruction. The source code below shows the implementation of the `main` function. The stack dump in the middle shows the current stack state with registers `rsp` and `rbp`. The backtrace at the bottom shows the call chain starting from the current instruction at address `0x5555555517e`.

```

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
0x55555555185 <main+60>      call   main           <main>
0x5555555518a <main+65>      leave
0x5555555518b <main+66>      ret
0x5555555518c      nop    dword ptr [rax]
0x55555555190 <__libc_csu_init> endbr64
0x55555555194 <__libc_csu_init+4> push   r15
0x55555555196 <__libc_csu_init+6> lea    r15, [rip + 0x2c1b]    <__init_array_start>
0x5555555519d <__libc_csu_init+13> push   r14
[ SOURCE (CODE) ]
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }
[ STACK ]
00:0000| rsp 0x7fffffff1f0 ← 0x0
01:0008|          0x7fffffff1f8 ← 0x400000000000
02:0010| rbp 0x7fffffff200 → 0x7fffffff220 → 0x7fffffff240 → 0x7fffffff260 → 0x7fffffff280 ← ...
03:0018|          0x7fffffff208 → 0x5555555518a (main+65) ← leave
04:0020|          0x7fffffff210 ← 0x38000000380
05:0028|          0x7fffffff218 ← 0x20000000380
06:0030|          0x7fffffff220 → 0x7fffffff240 → 0x7fffffff260 → 0x7fffffff280 → 0x7fffffff2a0 ← ...
07:0038|          0x7fffffff228 → 0x5555555518a (main+65) ← leave
[ BACKTRACE ]
▶ 0 0x5555555517e main+53
1 0x5555555518a main+65
2 0x5555555518a main+65
3 0x5555555518a main+65
4 0x5555555518a main+65
5 0x5555555518a main+65
6 0x5555555518a main+65
7 0x5555555518a main+65
[ pwndbg> ]

```

```

$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1
> info break
> continue
> p/d argc
> continue 4
> p/d argc
> continue 4
> continue 4

```

rec_main.c

■ rec_main (debugging)

```
nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
0x55555555518a <main+65>      leave
0x55555555518b <main+66>      ret

0x55555555518c      nop    dword ptr [rax]
0x555555555190 <__libc_csu_init> endbr64
0x555555555194 <__libc_csu_init+4> push   r15
0x555555555196 <__libc_csu_init+6> lea    r15, [rip + 0x2c1b]      <_init_array_start>
0x55555555519d <__libc_csu_init+13> push   r14
[ SOURCE (CODE) ]-----
```

In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob01/rec_main.c

```
3 int main(int argc)
4 {
5     if (argc > 0x800)
6         printf("The result is %d\n", argc);
7     else
8         return main(argc<<1);
9 }
```

[STACK]-----

```
00:0000 | rsp 0x7fffffff1f0 ← 0x0
01:0008 | 0x7fffffff1f8 ← 0x400000000000
02:0010 | rbp 0x7fffffff200 → 0x7fffffff220 → 0x7fffffff240 → 0x7fffffff260 → 0x7fffffff280 ← ...
03:0018 | 0x7fffffff208 → 0x5555555518a (main+65) ← leave
04:0020 | 0x7fffffff210 ← 0x38000000380
05:0028 | 0x7fffffff218 ← 0x20000000380
06:0030 | 0x7fffffff220 → 0x7fffffff240 → 0x7fffffff260 → 0x7fffffff280 → 0x7fffffff2a0 ← ...
07:0038 | 0x7fffffff228 → 0x5555555518a (main+65) ← leave
[ BACKTRACE ]-----
```

```
▶ 0 0x55555555517e main+53
  1 0x55555555518a main+65
  2 0x55555555518a main+65
  3 0x55555555518a main+65
  4 0x55555555518a main+65
  5 0x55555555518a main+65
  6 0x55555555518a main+65
  7 0x55555555518a main+65
```

pwndbg> p/d argc
\$5 = 1024
pwndbg> █

```
$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1
> info break
> continue
> p/d argc
> continue 4
> p/d argc
> continue 4
> p/d argc
> continue 4
> p/d argc
```

rec_main.c

■ rec_main (debugging)

```

nshc@nshcdell: ~/computer_system/2024/exam/mid-term/prob01
[ STACK ]
00:0000| rsp 0x7fffffff1f0 ← 0x0
01:0008|     0x7fffffff1f8 ← 0x40000000000
02:0010| rbp 0x7fffffff200 → 0x7fffffff220 → 0x7fffffff240 → 0x7fffffff260 → 0x7fffffff280 ← ...
03:0018|     0x7fffffff208 → 0x55555555518a (main+65) ← leave
04:0020|     0x7fffffff210 ← 0x38000000380
05:0028|     0x7fffffff218 ← 0x20000000380
06:0030|     0x7fffffff220 → 0x7fffffff240 → 0x7fffffff260 → 0x7fffffff280 → 0x7fffffff2a0 ← ...
07:0038|     0x7fffffff228 → 0x55555555518a (main+65) ← leave
[ BACKTRACE ]
▶ 0 0x55555555517e main+53
  1 0x55555555518a main+65
  2 0x55555555518a main+65
  3 0x55555555518a main+65
  4 0x55555555518a main+65
  5 0x55555555518a main+65
  6 0x55555555518a main+65
  7 0x55555555518a main+65

pwndbg> p/d argc
$5 = 1024
pwndbg> x/20xg $rsp
0x7fffffff1f0: 0x0000000000000000 0x0000040000000000
0x7fffffff200: 0x00007fffffff220 0x00005555555518a
0x7fffffff210: 0x0000038000000380 0x0000020000000380
0x7fffffff220: 0x00007fffffff240 0x00005555555518a
0x7fffffff230: 0x0000038000000380 0x0000010000000380
0x7fffffff240: 0x00007fffffff260 0x00005555555518a
0x7fffffff250: 0x0000038000000380 0x0000008000000380
0x7fffffff260: 0x00007fffffff280 0x00005555555518a
0x7fffffff270: 0x0000038000000380 0x0000004000000380
0x7fffffff280: 0x00007fffffff2a0 0x00005555555518a
pwndbg>

```

```

$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1
> info break
> continue
> p/d argc
> continue 4
> p/d argc
> continue 4
> p/d argc
> x/20xg $rsp

```

rec_main.c

■ rec_main (debugging)

The terminal window shows the following information:

- Assembly View:** Shows the assembly code for the main function, including instructions like `nop`, `dword ptr [rax]`, and `endbr64`.
- Source View:** Shows the C source code for the main function, with line 8 highlighted as the current instruction being executed.
- Stack View:** Shows a dump of the stack memory from address 00:0000 to 07:0038, with frame pointers (rsp, rbp) and their corresponding addresses.
- Backtrace View:** Shows the call stack with 7 frames, all pointing to the main function at address 0x5555555518a.
- pwndbg Prompt:** The user is interacting with the debugger via the pwndbg command-line interface.
- Output:** The result of the program execution is displayed as "The result is 4096".

```
$gdb rec_main -q
> break main
> run
> break 8
> continue
> p/d argc
> info break
> disa 1
> info break
> continue
> p/d argc
> continue 4
> p/d argc
> continue 4
> p/d argc
> x/20xg $rsp
> continue
```

Practice : prob6

■ C code

- (a) newData->start = 0x
 - (b) newData->raw[0] = 0
 - (c) newData->raw[2] = 0
 - (d) newData->raw[4] = 0
 - (e) newData->sense = 0x

```
In file: /home/nshc/moon/computer_system/2024/exam/mid-term/prob06/prob6.c
30     oldData->data = 1.5;
31
32     newData = (NewSensorData *) oldData;
33
34     printf("newData->code %x\n", newData->code);
35     printf("newData->start %x\n", newData->start);
36     printf("newData->raw[0] %x\n", newData->raw[0]);
37     printf("newData->raw[1] %x\n", newData->raw[1]);
38     printf("newData->raw[2] %x\n", newData->raw[2]);
39     printf("newData->raw[3] %x\n", newData->raw[3]);
40     printf("newData->raw[4] %x\n", newData->raw[4]);
[ STACK ]
00:0000 | esp 0xfffffd450 -> 0x56557008 ← 'newData->code %x\n'
01:0004 |           0xfffffd454 ← 0x104f
02:0008 |           0xfffffd458 ← 0x9e
03:000c |           0xfffffd45c -> 0xf7faf224 (_elf_set__libc_subfreeres_element_free_mem__)
_mem) ← endbr32
04:0010 |           0xfffffd460 ← 0x0
05:0014 |           0xfffffd464 ← 0x14
06:0018 |           0xfffffd468 -> 0xf7fb1000 (_GLOBAL_OFFSET_TABLE_) ← 0x1ead6c
07:001c |           0xfffffd46c -> 0x56556314 (main+44) ← add esp, 0x10
[ BACKTRACE ]
▶ 0 0x56556241 foo+84
  1 0x56556314 main+44
  2 0xf7de0ed5 __libc_start_main+245

pwndbg> p *newData
$1 = {
    code = 4175,
    start = -256,
    raw = "\270\032P\200", <incomplete sequence \341>,
    sense = 143,
    ext = 0,
    data = -3.1566918117587027e+78
}
pwndbg>
```

Homework 5

➤ Homework #05

- Overview

- **Released date:** 11/1 (Fri.)
- **Due date:** 11/8 (Fri.)
- **Where to submit:** to e-class (<http://eclass.seoultech.ac.kr>)
 - Late submission is not allowed.
- **Assigned score:** 1 points

1. Refer to the following source code.

```
#include <stdio.h>

unsigned long long fibonacci(unsigned long long n) {
    if(n == 0){
        return 0;
    } else if(n == 1) {
        return 1;
    } else {
        return (fibonacci(n-1) + fibonacci(n-2));
    }
}

int main() {
    unsigned long long i;
    unsigned long long n = 90;

    printf("Fibonacci of %lld: \n" , n);

    for(i = 0;i<n;i++) {
        printf("%lld ",fibonacci(i));
    }
    printf("\n");
    return 0;
}
```

Outline

- Debugging
 - Tools
- Design ●
 - Managing complexity
 - Communication
 - Naming
 - Comments

Design

■ A good design needs to achieve many things:

- Performance
- Availability
- Modifiability, portability
- Scalability
- Security
- Testability
- Usability
- Cost to build, cost to operate

Design

- A good design needs to achieve many things:

- Performance
- Availability
- Modifiability, portability
- Scalability
- Security
- Testability
- Usability
- Cost to build, cost to operate

But above all else: it must be readable

Design

Good Design does:

**Complexity Management &
Communication**

Complexity

- There are well known limits to how much complexity a human can manage easily.

VOL. 63, NO. 2

MARCH, 1956

THE PSYCHOLOGICAL REVIEW

THE MAGICAL NUMBER SEVEN, PLUS OR MINUS TWO:
SOME LIMITS ON OUR CAPACITY FOR
PROCESSING INFORMATION ¹

GEORGE A. MILLER

Harvard University

Complexity Management

- However, patterns can be very helpful...

COGNITIVE PSYCHOLOGY 4, 55–81 (1973)

Perception in Chess¹

WILLIAM G. CHASE AND HERBERT A. SIMON
Carnegie-Mellon University

This paper develops a technique for isolating and studying the perceptual structures that chess players perceive. Three chess players of varying strength — from master to novice — were confronted with two tasks: (1) A perception task, where the player reproduces a chess position in plain view, and (2) de Groot's (1965) short-term recall task, where the player reproduces a chess position after viewing it for 5 sec. The successive glances at the position in the perceptual task and long pauses in the memory task were used to segment the structures in the reconstruction protocol. The size and nature of these structures were then analyzed as a function of chess skill.

Complexity Management

Many techniques have been developed to help manage complexity:

- Separation of concerns
- Modularity
- Reusability
- Extensibility
- DRY
- Abstraction
- Information Hiding
- ...

Managing Complexity

■ Given the many ways to manage complexity

- Design code to be testable
- Try to reuse testable chunks

Complexity Example

- **Split a cache access into three+ testable components**
 - State all of the steps that a cache access requires
 - Which steps depend on the operation being a load or a store?

Complexity Example

■ Split a cache access into three+ testable components

- State all of the steps that a cache access requires
 - Convert address into tag, set index, block offset
 - Look up the set using the set index
 - Check if the tag matches any line in the set
 - If so, hit
 - If not a match, miss, then
 - Find the LRU block
 - Evict the LRU block
 - Read in the new line from memory
 - Update LRU
 - Update dirty if the access was a store

- Which steps depend on the operation being a load or a store?

Designs need to be testable

■ Testable design

- Testing versus Contracts
- These are complementary techniques

■ Testing and Contracts are

- Acts of design more than verification
- Acts of documentation

Designs need to be testable

■ Testable design

- Testing versus Contracts
- These are complementary techniques

■ Testing and Contracts are

- Acts of design more than verification
- Acts of documentation: **executable documentation!**

Testing Example

- For your cache simulator, you can write your own traces

- Write a trace to test for a cache hit

- L 50, 1

- L 50, 1

- Write a trace to test dirty bytes in cache

- S 100, 1

Testable design is modular

- Modular code has: separation of concerns, encapsulation, abstraction
 - Leads to: reusability, extensibility, readability, testability
- Separation of concerns
 - Create helper functions so each function does “one thing”
 - Functions should neither do too much nor too little
 - Avoid duplicated code
- Encapsulation, abstraction, and respecting the interface
 - Each module is responsible for its own internals
 - No outside code “intrudes” on the inner workings of another module

Trust the Compiler!

- Use plenty of temporary variables
- Use plenty of functions
- Let compiler do the math

Communication

When writing code, the author is communicating with:

- The machine
- Other developers of the system
- Code reviewers
- Their future self

Communication

There are many techniques that have been developed around code communication:

- Tests
- Naming
- Comments
- Commit Messages
- Code Review
- Design Patterns
- ...

Naming

Avoid deliberately meaningless names:

The screenshot shows a GitHub search interface with the query 'foo'. The results are filtered to show 'Code' results, with a total of 8,937,025 available code results. The results are sorted by 'Best match'.

Repositories (493) | **Code** (8M+) | **Commits** (11M+) | **Issues** (33K) | **Packages** (34) | **Marketplace** (0) | **Topics** (507) | **Wikis** (74K) | **Users** (107)

Showing 8,937,025 available code results

Sort: Best match

alexef/gobject-introspection
tests/scanner/foo.h

```
1 #ifndef __FOO_OBJECT_H__
2 #define __FOO_OBJECT_H__
3 
4 #include <glib-object.h>
5 #include <gio/gio.h> /* GAsyncReadyCallback */
6 #include "utility.h"
7 
8 #define FOO_SUCCESS_INT 0x1138
9 
10 #define FOO_DEFINE_SHOULD_BE_EXPOSED "should be exposed"
```

● C Showing the top three matches Last indexed on Jun 25, 2018

alexef/gobject-introspection
tests/scanner/foo.c

```
1 #include "foo.h"
2 #include "girepository.h"
3 
4 /* A hidden type not exposed publicly, similar to GUPNP's XML wrapper
5    object */
6 typedef struct _FooHidden FooHidden;
7 
8 int foo_init_argv (int argc, char **argv);
```

● C Showing the top four matches Last indexed on Jun 25, 2018

Languages

PHP	26,699,388
JavaScript	8,942,989
C	X
Python	7,892,881
HTML	4,228,224
C++	4,093,394
Ruby	4,021,592
Java	2,891,173
Text	2,612,262
XML	2,599,848

Naming is understanding

“If you don’t know what a thing should be called, you cannot know what it is.

If you don’t know what it is, you cannot sit down and write the code.” - Sam Gardiner

Better naming practices

- 1. Start with meaning and intention**
- 2. Use words with precise meanings (avoid “data”, “info”, “perform”)**
- 3. Prefer fewer words in names**
- 4. Avoid abbreviations in names**
- 5. Use code review to improve names**
- 6. Read the code out loud to check that it sounds okay**
- 7. Actually rename things**

Naming guidelines – Use dictionary words

- Only use dictionary words and abbreviations that appear in a dictionary.
 - For example: FileCpy -> FileCopy
 - Avoid vague abbreviations such as acc, mod, auth, etc..

Avoid using single-letter names

- Single letters are unsearchable
 - Give no hints as to the variable's usage
- Exceptions are loop counters
 - Especially if you know why i, j, etc were originally used
 - C/unix systems have a few other common conventions, such as 'fd' for "file descriptor" and "str" for a string argument to a function.
Following existing style is fine & good.

Limit name character length

“Good naming limits individual name length, and reduces the need for specialized vocabulary” – Philip Relf

Limit name word count

- Keep names to a four word maximum
- Limit names to the number of words that people can read at a glance.
- Which of each pair do you prefer?
 - a1) arraysOfSetsOfLinesOfBlocks
 - a2) cache
 - b1) evictedData
 - b2) evictedDataBytes

Describe Meaning

- Use descriptive names.
- Avoid names with no meaning: a, foo, blah, tmp, etc
- There are reasonable exceptions:

```
void swap(int* a, int* b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}
```

Use a large vocabulary

- Be more specific when possible:
 - Person -> Employee
- What is size in this binaryTree?

```
struct binaryTree {  
    int size;  
    ...  
};
```

height
numChildren
subTreeNumNodes
keyLength

Use problem domain terms

- **Use the correct term in the problem domain's language.**
 - Hint: as a student, consider the terms in the assignment
- **In cachelab, consider the following:**

line

element

Use opposites precisely

- **Consistently use opposites in standard pairs**
 - first/end -> first/last

Comments

Don't Comments

- **Don't say what the code does**
 - because the code already says that
- **Don't explain awkward logic**
 - improve the code to make it clear
- **Don't add too many comments**
 - it's messy, and they get out of date

Awkward Code

- Imagine someone (TA, employer, etc) has to read your code
 - Would you rather rewrite or comment the following?

```
(* (void **) ( (* (void **) (bp)) + DSIZE)) = (* (void **) (bp + DSIZE));
```

- How about?

```
bp->prev->next = bp->next;
```

- Both lines update program state in the same way.

Do Comments

- Answer the question: why the code exists

- When should I use this code?
- When shouldn't I use it?
- What are the alternatives to this code?

Why does this exist?

- Explain why a magic number is what it is.

```
// Each address is 64-bit, which is 16 + 1 hex characters
const int MAX_ADDRESS_LENGTH = 17;
```

- When should this code be used? Is there an alternative?

```
unsigned power2(unsigned base, unsigned expo) {
    unsigned i;
    unsigned result = 1;
    for(i=0;i<expo;i++) {
        result+=result;
    }
    return result;
}
```

How to write good comments

1. Code by commenting!

Write short comment

1. Helps you think about design & overcome blank-page problem
2. Single line comments
3. Example: Write four one-line comments for quick sort

```
// Initialize locals  
// Pick a pivot value  
// Reorder array around the pivot  
// Recurse
```

How to write good comments

1. Write short comments of what the code will do.

1. Single line comments
2. Example: Write four one-line comments for quick sort

2. Write that code.

3. Revise comments / code

1. If the code or comments are awkward or complex
2. Join / Split comments as needed

4. Maintain code and revised comments

Commit Messages

- Committing code to a source repository is a vital part of development
 - Protects against system failures and typos:
 - cat foo.c versus cat > foo.c
 - The commit messages are your record of your work
 - Communicating to your future self
 - Describe in one line what you did
- Use branches

Summary

- **Programs have defects**
 - Be systematic about finding them
- **Programs are more complex than humans can manage**
 - Write code to be manageable
- **Programming is not solitary, even if you are communicating with a grader or a future self**
 - Be understandable in your communication

Acknowledgements

- Some debugging content derived from:
 - <http://www.whypartoffail.com/slides.php>
- Some code examples for design are based on:
 - “The Art of Readable Code”. Boswell and Foucher. 2011.
- Lecture originally written by
 - Michael Hilton and Brian Railing