

HW5. Bomblab

부산대학교 정보컴퓨터공학부

2017-24542

이준영

제출일: 2018.11.16

1. 숙제 환경 및 점수, Github

1) 아이디

CSE DELL SERVER ID: 201724542

Bomb ID: 201724542

Bomb Email: rubinstory@naver.com

Bomb Number: bomb44

2) 결과 화면 스크린샷 (아래 화면 참조)

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb12	Thu Nov 8 06:04	6	0	70	valid
2	bomb20	Thu Nov 8 10:44	6	0	70	valid
3	bomb32	Sat Nov 10 07:24	6	0	70	valid
4	bomb11	Tue Nov 13 21:45	6	0	70	valid
5	bomb27	Wed Nov 14 02:58	6	0	70	valid
6	bomb37	Thu Nov 15 07:57	6	0	70	valid
7	bomb55	Thu Nov 15 03:20	6	1	70	valid
8	bomb57	Thu Nov 15 20:30	6	1	70	valid
9	bomb6	Thu Nov 15 22:46	6	1	70	valid
10	bomb35	Mon Nov 12 08:07	6	4	68	valid
11	bomb53	Thu Nov 15 04:34	6	4	68	valid
12	bomb44	Fri Nov 16 04:31	6	4	68	valid

3) Github 보고서 업로드 스크린샷

 rubinstory first	Latest commit 019ccdb a minute ago
 bomb44	first a minute ago
 README.md	Update README.md 14 days ago
 SS_HW5.docx	first a minute ago

2. Phase_1 해결 방법

(1) 문제 정의

Phase_1에서는 사용자가 입력한 문자열과 정답 문자열이 같은지를 비교한다. Bomb44의 경우, "Verbosity leads to unclear, inarticulate things."를 입력해야 Phase_2로 넘어갈 수 있다.

(2) 문제 해결 방법을 최대한 자세히 설명

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
This is not a answer.
```

```
Breakpoint 2, 0x00005555555552b4 in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
=> 0x00005555555552b4 <+0>:    sub    $0x8,%rsp
 0x00005555555552b8 <+4>:    lea     0x1831(%rip),%rsi      # 0x555555556af0
 0x00005555555552bf <+11>:   callq   0x5555555557c1 <strings_not_equal>
 0x00005555555552c4 <+16>:   test    %eax,%eax
 0x00005555555552c6 <+18>:   jne     0x5555555552cd <phase_1+25>
 0x00005555555552c8 <+20>:   add    $0x8,%rsp
 0x00005555555552cc <+24>:   retq
 0x00005555555552cd <+25>:   callq   0x555555555ac5 <explode_bomb>
 0x00005555555552d2 <+30>:   jmp     0x5555555552c8 <phase_1+20>
End of assembler dump.
(gdb) x/s 0x555555556af0
0x555555556af0: "Verbosity leads to unclear, inarticulate things."
(gdb)
```

위의 그림은 phase_1 함수의 내용을 gdb를 통해 나타낸 것이다. 현재 11번째 줄에서 입력한 문자열과 정답이 같은지를 비교하는 <strings_not_equal> 함수를 불러오는 것으로 보아, 정답은 0x555555556af0에 저장되어 있음을 유추할 수 있다.

x/s 0x555555556af0 명령어를 통해 확인해보면 정답인 "Verbosity leads to unclear, inarticulate things."가 저장되어 있음을 알 수 있다.

3. Phase_2 문제 해결 방법

(1) 문제 정의

Phase_2의 경우, 사용자가 입력한 6개의 숫자가 정답과 같은지를 비교한다. 6개의 숫자는 일정한 규칙을 가진다. Bomb44의 경우 정답이 0 1 1 2 3 5로, 피보나치 수열이라는 규칙을 가진다.

(2) 문제 해결 방법을 최대한 자세히 설명

```
Breakpoint 2, 0x00005555555552d4 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
=> 0x00005555555552d4 <+0>:    push   %rbp
  0x00005555555552d5 <+1>:    push   %rbx
  0x00005555555552d6 <+2>:    sub    $0x28,%rsp
  0x00005555555552da <+6>:    mov    %fs:0x28,%rax
  0x00005555555552e3 <+15>:   mov    %rax,0x18(%rsp)
  0x00005555555552e8 <+20>:   xor    %eax,%eax
  0x00005555555552ea <+22>:   mov    %rsp,%rsi
  0x00005555555552ed <+25>:   callq 0x555555555b01 <read_six_numbers>
  0x00005555555552f2 <+30>:   cmpl   $0x0,(%rsp)
  0x00005555555552f6 <+34>:   jne    0x5555555552ff <phase_2+43>
  0x00005555555552f8 <+36>:   cmpl   $0x1,0x4(%rsp)
  0x00005555555552fd <+41>:   je     0x555555555304 <phase_2+48>
  0x00005555555552ff <+43>:   callq 0x555555555ac5 <explode_bomb>
  0x0000555555555304 <+48>:   mov    %rsp,%rbx
  0x0000555555555307 <+51>:   lea    0x10(%rbx),%rbp
  0x000055555555530b <+55>:   jmp    0x555555555316 <phase_2+66>
  0x000055555555530d <+57>:   add    $0x4,%rbx
  0x0000555555555311 <+61>:   cmp    %rbp,%rbx
  0x0000555555555314 <+64>:   je     0x555555555327 <phase_2+83>
  0x0000555555555316 <+66>:   mov    0x4(%rbx),%eax
  0x0000555555555319 <+69>:   add    (%rbx),%eax
  0x000055555555531b <+71>:   cmp    %eax,0x8(%rbx)
  0x000055555555531e <+74>:   je     0x55555555530d <phase_2+57>
  0x0000555555555320 <+76>:   callq 0x555555555ac5 <explode_bomb>
  0x0000555555555325 <+81>:   jmp    0x55555555530d <phase_2+57>
  0x0000555555555327 <+83>:   mov    0x18(%rsp),%rax
  0x000055555555532c <+88>:   xor    %fs:0x28,%rax
  0x0000555555555335 <+97>:   jne    0x55555555533e <phase_2+106>
  0x0000555555555337 <+99>:   add    $0x28,%rsp
  0x000055555555533b <+103>:  pop    %rbx
  0x000055555555533c <+104>:  pop    %rbp
  0x000055555555533d <+105>:  retq
  0x000055555555533e <+106>:  callq 0x55555554ef0 <__stack_chk_fail@plt>
End of assembler dump.
(gdb)
```

위의 그림은 phase_2 함수의 내용을 gdb를 통해 나타낸 것이다. 25번째 줄을 보면 <read_six_numbers>라는 함수를 불러오는 것으로 보아, 6개의 숫자를 입력해야 함을 알 수 있다. 그리고 36과 41번째 줄을 통해 두번째 숫자는 1이어야 함을 알 수 있다. 그리고 57과 61번째 줄을 통해 3,4,5,6번째 숫자를 계속 비교함을 알 수 있다. 그런데 여기서 71번째 줄을 보면 현재값을 계속 eax에 더하고 있음을 알 수 있다. 따라서 정답은 피보나치 수열을 따른다는 것을 알 수 있다.

4. Phase_3 문제 해결 방법

(1) 문제 정의

Phase_3의 경우, 사용자가 입력한 2개의 숫자가 정답과 같은지를 비교한다.

Bomb44의 경우, 0 140을 입력해야 다음 단계로 넘어갈 수 있다.

(2) 문제 해결 방법을 최대한 자세히 설명

```
Dump of assembler code for function phase_3:
=> 0x000055555555343 <+0>:    sub   $0x18,%rsp
0x000055555555347 <+4>:    mov    %fs:0x28,%rax
0x000055555555350 <+13>:   mov    %rax,0x8(%rsp)
0x000055555555355 <+18>:   xor    %eax,%eax
0x000055555555357 <+20>:   lea    0x4(%rsp),%rcx
0x00005555555535c <+25>:   mov    %rsp,%rdx
0x00005555555535f <+28>:   lea    0x1aa7(%rip),%rsi      # 0x555555556e0d
0x000055555555366 <+35>:   callq 0x555555554f90 <__isoc99_sscanf@plt>
0x00005555555536b <+40>:   cmp    $0x1,%eax
0x00005555555536e <+43>:   jle    0x5555555538d <phase_3+74>
0x000055555555370 <+45>:   cmpl   $0x7,(%rsp)
0x000055555555374 <+49>:   ja    0x55555555411 <phase_3+206>
0x00005555555537a <+55>:   mov    (%rsp),%eax
0x00005555555537d <+58>:   lea    0x17dc(%rip),%rdx      # 0x555555556b60
0x000055555555384 <+65>:   movsq 0(%rdx,%rax,4),%rax
0x000055555555388 <+69>:   add    %rdx,%rax
0x00005555555538b <+72>:   jmpq   *%rax
0x00005555555538d <+74>:   callq  0x55555555ac5 <explode_bomb>
0x000055555555392 <+79>:   jmp    0x55555555370 <phase_3+45>
0x000055555555394 <+81>:   mov    $0x183,%eax
0x000055555555399 <+86>:   jmp    0x555555553a0 <phase_3+93>
0x00005555555539b <+88>:   mov    $0x0,%eax
0x0000555555553a0 <+93>:   sub    $0x7f,%eax
0x0000555555553a3 <+96>:   add    $0x279,%eax
0x0000555555553a8 <+101>:  sub    $0x2f1,%eax
0x0000555555553ad <+106>:  add    $0x2f1,%eax
0x0000555555553b2 <+111>:  sub    $0x2f1,%eax
0x0000555555553b7 <+116>:  add    $0x2f1,%eax
0x0000555555553bc <+121>:  sub    $0x2f1,%eax
0x0000555555553c1 <+126>:  cmpl   $0x5,(%rsp)
0x0000555555553c5 <+130>:  jg    0x555555553cd <phase_3+138>
0x0000555555553c7 <+132>:  cmp    %eax,0x4(%rsp)
0x0000555555553cb <+136>:  je    0x555555553d2 <phase_3+143>
0x0000555555553cd <+138>:  callq  0x55555555ac5 <explode_bomb>
0x0000555555553d2 <+143>:  mov    %fs:(%rsp),%rax
0x0000555555553d7 <+148>:  xor    $0x0,%rax
0x0000555555553e0 <+157>:  jne    0x5555555541d <phase_3+218>
0x0000555555553e2 <+159>:  add    $0x18,%rsp
0x0000555555553e6 <+163>:  retq
0x0000555555553e7 <+164>:  mov    $0x0,%eax
0x0000555555553ec <+169>:  jmp    0x555555553a3 <phase_3+96>
0x0000555555553ee <+171>:  mov    $0x0,%eax
0x0000555555553f3 <+176>:  jmp    0x555555553a8 <phase_3+101>
0x0000555555553f5 <+178>:  mov    $0x0,%eax
0x0000555555553fa <+183>:  jmp    0x555555553ad <phase_3+106>
0x0000555555553fc <+185>:  mov    $0x0,%eax
0x000055555555401 <+190>:  jmp    0x555555553b2 <phase_3+111>
0x000055555555403 <+192>:  mov    $0x0,%eax
0x000055555555408 <+197>:  jmp    0x555555553b7 <phase_3+116>
0x00005555555540a <+199>:  mov    $0x0,%eax
0x00005555555540f <+204>:  jmp    0x555555553bc <phase_3+121>
0x000055555555411 <+206>:  callq  0x55555555ac5 <explode_bomb>
0x000055555555416 <+211>:  mov    $0x0,%eax
0x00005555555541b <+216>:  jmp    0x555555553c1 <phase_3+126>
0x00005555555541d <+218>:  callq  0x55555554ef0 <__stack_chk_fail@plt>
End of assembler dump.
(gdb)
```

위의 그림은 phase_3 함수 내용을 gdb를 통해 나타낸 것이다.

40, 43번째 줄을 통해 입력해야하는 값이 2개임을 알 수 있다. 그리고 45, 49번째 줄을 통해 입력한 첫번째 값은 7보다 작아야 함을 알 수 있다.

그리고 143번줄 이후를 보면 rax 레지스터의 값과 2번째 입력값을 비교함을 알 수 있다.
이때, 비교하는 값은 140이므로 정답은 0 140이 된다.

5. Phase_4 문제 해결 방법

(1) 문제 정의

Phase_4는 입력한 숫자 2개가 정답과 같은지를 비교한다.

이때 입력한 첫번째 숫자는 func4에 인자로 들어가고, 그 결과값이 정답과 일치해야한다.

(2) 문제 해결 방법을 최대한 자세히 설명

```
Dump of assembler code for function phase_4:  
> 0x0000555555555461 <+0>:    sub    $0x18,%rsp  
 0x0000555555555465 <+4>:    mov    %fs:0x28,%rax  
 0x000055555555546e <+13>:   mov    %rax,0x8(%rsp)  
 0x0000555555555473 <+18>:   xor    %eax,%eax  
 0x0000555555555475 <+20>:   lea    0x4(%rsp),%rcx  
 0x000055555555547a <+25>:   mov    %rsp,%rdx  
 0x000055555555547d <+28>:   lea    0x1989(%rip),%rsi      # 0x555555556e0d  
 0x0000555555555484 <+35>:   callq 0x55555554f90 <__isoc99_sscanf@plt>  
 0x0000555555555489 <+40>:   cmp    $0x2,%eax  
 0x000055555555548c <+43>:   jne    0x55555555494 <phase_4+51>  
 0x000055555555548e <+45>:   cmpl   $0xe,(%rsp)  
 0x0000555555555492 <+49>:   jbe    0x55555555499 <phase_4+56>  
 0x0000555555555494 <+51>:   callq 0x55555555ac5 <explode_bomb>  
 0x0000555555555499 <+56>:   mov    $0xe,%edx  
 0x000055555555549e <+61>:   mov    $0x0,%esi  
 0x00005555555554a3 <+66>:   mov    (%rsp),%edi  
 0x00005555555554a6 <+69>:   callq 0x55555555422 <func4>  
 0x00005555555554ab <+74>:   cmp    $0x6,%eax  
 0x00005555555554ae <+77>:   jne    0x555555554b7 <phase_4+86>  
 0x00005555555554b0 <+79>:   cmpl   $0x6,0x4(%rsp)  
 0x00005555555554b5 <+84>:   je     0x555555554bc <phase_4+91>  
 0x00005555555554b7 <+86>:   callq 0x55555555ac5 <explode_bomb>  
 0x00005555555554bc <+91>:   mov    0x8(%rsp),%rax  
 0x00005555555554c1 <+96>:   xor    %fs:0x28,%rax  
 0x00005555555554ca <+105>:  jne    0x555555554d1 <phase_4+112>  
 0x00005555555554cc <+107>:  add    $0x18,%rsp  
 0x00005555555554d0 <+111>:  retq  
 0x00005555555554d1 <+112>:  callq 0x55555554ef0 <__stack_chk_fail@plt>  
End of assembler dump.  
(gdb)
```

위의 그림은 phase_4 함수 내용을 gdb를 통해 나타낸 것이다.

45번째 줄을 보면 입력한 첫번째 값이 14보다 작은지를 확인한다. 따라서 첫번째 숫자는 14 보다 작아야한다. 그리고 69, 74번째 줄을 보면 func4의 결과값이 6이어야함을 알 수 있다. 그리고 79번째 줄을 보면, 입력한 숫자 중 2번째 값이 6과 같은지를 비교한다. 따라서 두번째 숫자는 6임을 알 수 있다. 이제 첫번째 숫자를 알아내기 위해 func4의 내용을 보자.

```
Dump of assembler code for function func4:  
=> 0x0000555555555422 <+0>:    sub    $0x8,%rsp  
0x0000555555555426 <+4>:    mov    %edx,%eax  
0x0000555555555428 <+6>:    sub    %esi,%eax  
0x000055555555542a <+8>:    mov    %eax,%ecx  
0x000055555555542c <+10>:   shr    $0x1f,%ecx  
0x000055555555542f <+13>:   add    %eax,%ecx  
0x0000555555555431 <+15>:   sar    %ecx  
0x0000555555555433 <+17>:   add    %esi,%ecx  
0x0000555555555435 <+19>:   cmp    %edi,%ecx  
0x0000555555555437 <+21>:   jg    0x555555555447 <func4+37>  
0x0000555555555439 <+23>:   mov    $0x0,%eax  
0x000055555555543e <+28>:   cmp    %edi,%ecx  
0x0000555555555440 <+30>:   jl    0x555555555453 <func4+49>  
0x0000555555555442 <+32>:   add    $0x8,%rsp  
0x0000555555555446 <+36>:   retq  
0x0000555555555447 <+37>:   lea    -0x1(%rcx),%edx  
0x000055555555544a <+40>:   callq 0x555555555422 <func4>  
0x000055555555544f <+45>:   add    %eax,%eax  
0x0000555555555451 <+47>:   jmp    0x555555555442 <func4+32>  
0x0000555555555453 <+49>:   lea    0x1(%rcx),%esi  
0x0000555555555456 <+52>:   callq 0x555555555422 <func4>  
0x000055555555545b <+57>:   lea    0x1(%rax,%rax,1),%eax  
0x000055555555545f <+61>:   jmp    0x555555555442 <func4+32>  
End of assembler dump.  
(gdb)
```

위 사진은 gdb를 통해 func4의 내용을 나타낸 것이다. 이 코드들을 c언어로 번역하면 아래와 같다.

```

1 #include <stdio.h>
2
3 int func(int arg1, int arg2, int arg3) {
4     int r = arg3;
5     r -= arg2;
6     unsigned int t = r;
7     t >>= 31;
8     t += r;
9     t /= 2;
10    t += arg2;
11    if (t > arg1) {
12        arg3 = t-1;
13        int r = func(arg1, arg2, arg3);
14        return r * 2;
15    }
16    else if (t < arg1) {
17        r = 0;
18        arg2 = t + 1;
19        int r = func(arg1, arg2, arg3);
20        return 2 * r + 1;
21    }
22    return 0;
23 }
24
25 int main() {
26     int d = func(6, 0, 14);
27     printf("result == %d\n", d);
28     return 0;
29 }

```

위의 번역된 C언어 코드를 돌려보면, 결과값이 6이 나오고 14보다 작으면 6을 입력해야함을 알 수 있다. 따라서 phase_4의 정답은 6 6이다.

6. Phase_5 문제 해결 방법

(1) 문제 정의

Phase_5는 사용자가 입력한 2개의 숫자와 정답이 일치하는지를 비교한다.
Bomb44의 경우, 정답은 5 115이다.

(2) 문제 해결 방법을 최대한 자세히 설명

```

Dump of assembler code for function phase_5:
=> 0x0000555555554d6 <+0>:    sub    $0x18,%rsp
 0x0000555555554da <+4>:    mov    %fs:0x28,%rax
 0x0000555555554e3 <+13>:   mov    %rax,0x8(%rsp)
 0x0000555555554e8 <+18>:   xor    %eax,%eax
 0x0000555555554ea <+20>:   lea    0x4(%rsp),%rcx
 0x0000555555554ef <+25>:   mov    %rsp,%rdx
 0x0000555555554f2 <+28>:   lea    0x1914(%rip),%rsi      # 0x555555556e0d
 0x0000555555554f9 <+35>:   callq 0x55555554f90 <__isoc99_sscanf@plt>
 0x0000555555554fe <+40>:   cmp    $0x1,%eax
 0x000055555555501 <+43>:   jle    0x55555555555d <phase_5+135>
 0x000055555555503 <+45>:   mov    (%rsp),%eax
 0x000055555555506 <+48>:   and    $0xf,%eax
 0x000055555555509 <+51>:   mov    %eax,(%rsp)
 0x00005555555550c <+54>:   cmp    $0xf,%eax
 0x00005555555550f <+57>:   je     0x555555555543 <phase_5+109>
 0x000055555555511 <+59>:   mov    $0x0,%ecx
 0x000055555555516 <+64>:   mov    $0x0,%edx
 0x00005555555551b <+69>:   lea    0x165e(%rip),%rsi      # 0x555555556b80 <array.3417>
 0x000055555555522 <+76>:   add    $0x1,%edx
 0x000055555555525 <+79>:   cltq
 0x000055555555527 <+81>:   mov    (%rsi,%rax,4),%eax
 0x00005555555552a <+84>:   add    %eax,%ecx
 0x00005555555552c <+86>:   cmp    $0xf,%eax
 0x00005555555552f <+89>:   jne    0x555555555522 <phase_5+76>
 0x000055555555531 <+91>:   movl   $0xf,(%rsp)
 0x000055555555538 <+98>:   cmp    $0xf,%edx
 0x00005555555553b <+101>:  jne    0x555555555543 <phase_5+109>
 0x00005555555553d <+103>:  cmp    %ecx,0x4(%rsp)
 0x000055555555541 <+107>:  je     0x555555555548 <phase_5+114>
 0x000055555555543 <+109>:  callq 0x5555555555ac5 <explode_bomb>
 0x000055555555548 <+114>:  mov    0x8(%rsp),%rax
 0x00005555555554d <+119>:  xor    %fs:0x28,%rax
 0x000055555555556 <+128>:  jne    0x555555555564 <phase_5+142>
 0x000055555555558 <+130>:  add    $0x18,%rsp
 0x00005555555555c <+134>:  retq
 0x00005555555555d <+135>:  callq 0x5555555555ac5 <explode_bomb>
 0x000055555555562 <+140>:  jmp    0x555555555503 <phase_5+45>
 0x000055555555564 <+142>:  callq 0x55555554ef0 <__stack_chk_fail@plt>

End of assembler dump.
(gdb) █

```

위 사진은 gdb를 통해 phase_5의 내용을 확인한 것이다.

첫번째 인자의 경우, 입력하는 값에 따라 eax의 값이 변한다. 어셈블리 코드를 보면 loop이 존재함을 알 수 있는데, 문제가 요구하는 것은 loop이 몇번을 돌고 나야 eax가 14가 되는지이다. 여기서 5를 입력하면, loop가 14번을 돌아 나오는 것을 알 수 있고, 86과 98의 비교문을 통과하는 것을 확인할 수 있다.

그리고 두번째 숫자는 ecx 레지스터의 값과 비교를 하는데, 이를 확인해보면 16진수로 0x73임을 알 수 있다. 이를 10진수로 변환하면 115가 되므로, 정답은 5 115임을 알 수 있다.

7. Phase_6 문제 해결 방법

(1) 문제 정의

Phase_6은 사용자가 입력한 6개의 숫자와 정답이 일치하는지를 비교한다.

이때, 6개의 숫자는 각 node가 가지고 있는 값에 따라 결정된다.

(2) 문제 해결 방법을 최대한 자세히 설명

```
(gdb) disas
Dump of assembler code for function phase_6:
=> 0x0000555555555569 <+0>:    push %r13
 0x000055555555556b <+2>:    push %r12
 0x000055555555556d <+4>:    push %rbp
 0x000055555555556e <+5>:    push %rbx
 0x000055555555556f <+6>:    sub $0x68,%rsp
 0x0000555555555573 <+10>:   mov %fs:0x28,%rax
 0x000055555555557c <+19>:   mov %rax,%x58(%rsp)
 0x0000555555555581 <+24>:   xor %eax,%eax
 0x0000555555555583 <+26>:   mov %rsp,%r12
 0x0000555555555586 <+29>:   mov %r12,%rsi
 0x0000555555555589 <+32>:   callq 0x5555555555b01 <read_six_numbers>
 0x000055555555558e <+37>:   mov $0x0,%r13d
 0x0000555555555594 <+43>:   jmp 0x5555555555bb <phase_6+82>
 0x0000555555555596 <+45>:   callq 0x5555555555ac5 <explode_bomb>
 0x000055555555559b <+50>:   jmp 0x5555555555ca <phase_6+97>
 0x000055555555559d <+52>:   add $0x1,%ebx
 0x00005555555555a0 <+55>:   cmp $0x5,%ebx
 0x00005555555555a3 <+58>:   jg 0x5555555555b7 <phase_6+78>
 0x00005555555555a5 <+60>:   movslq %ebx,%rax
 0x00005555555555a8 <+63>:   mov (%rsp,%rax,4),%eax
 0x00005555555555ab <+66>:   cmp %eax,0x0(%rbp)
 0x00005555555555ae <+69>:   jne 0x55555555559d <phase_6+52>
 0x00005555555555b0 <+71>:   callq 0x5555555555ac5 <explode_bomb>
 0x00005555555555b5 <+76>:   jmp 0x55555555559d <phase_6+52>
 0x00005555555555b7 <+78>:   add $0x4,%r12
 0x00005555555555b9 <+82>:   mov %r12,%rbp
 0x00005555555555be <+85>:   mov (%r12),%eax
 0x00005555555555c2 <+89>:   sub $0x1,%eax
 0x00005555555555c5 <+92>:   cmp $0x5,%eax
 0x00005555555555c8 <+95>:   ja 0x555555555596 <phase_6+45>
 0x00005555555555ca <+97>:   add $0x1,%r13d
 0x00005555555555ce <+101>:  cmp $0x6,%r13d
 0x00005555555555d2 <+105>:  je 0x5555555555609 <phase_6+160>
 0x00005555555555d4 <+107>:  mov %r13d,%ebx
 0x00005555555555d7 <+110>:  jmp 0x5555555555a5 <phase_6+60>
 0x00005555555555d9 <+112>:  mov %r13d,%rdx
 0x00005555555555d9 <+116>:  add $0x1,%eax
 0x00005555555555e0 <+119>:  cmp %ecx,%eax
 0x00005555555555e2 <+121>:  jne 0x5555555555d9 <phase_6+112>
 0x00005555555555e4 <+123>:  mov %rdx,0x20(%rsp,%rsi,8)
 0x00005555555555e9 <+128>:  add $0x1,%rsi
 0x00005555555555ed <+132>:  cmp $0x6,%rsi
 0x00005555555555f1 <+136>:  je 0x5555555555610 <phase_6+167>
 0x00005555555555f3 <+138>:  mov (%rsp,%rsi,4),%ecx
 0x00005555555555f6 <+141>:  mov $0x1,%eax
 0x00005555555555fb <+146>:  lea 0x202c2e(%rip),%rdx      # 0x555555758230 <node1>
 0x0000555555555602 <+153>:  cmp $0x1,%ecx
 0x0000555555555605 <+156>:  jg 0x5555555555d9 <phase_6+112>
 0x0000555555555607 <+158>:  jmp 0x5555555555e4 <phase_6+123>
 0x0000555555555609 <+160>:  mov $0x0,%esi
 0x000055555555560e <+165>:  jmp 0x5555555555f3 <phase_6+138>
 0x0000555555555610 <+167>:  mov %r13d,%rbx
 0x0000555555555615 <+172>:  mov %r13d,%rax
 0x000055555555561a <+177>:  mov %rax,0x8(%rbx)
 0x000055555555561e <+181>:  mov %r13d,%rdx
 0x0000555555555623 <+186>:  mov %rdx,0x8(%rax)
 0x0000555555555627 <+190>:  mov %r13d,%rax
```

위의 코드는 gdb를 통해 phase_6의 내용을 확인한 것이다.

코드를 보면, 숫자를 6개를 입력해야 하고 모든 숫자는 6보다 작아야 하며, 중복되면 안된다 는 것을 알 수 있다.

여기 146번째 줄에 주석으로 node1의 주소가 있음을 확인할 수 있다.

아래의 사진은 node의 정보를 gdb로 확인한 것이다.

```
(gdb) x/24x 0x555555758230
0x555555758230 <node1>: 0x00000030e    0x000000001    0x55758240    0x00005555
0x555555758240 <node2>: 0x0000008b     0x000000002    0x55758250    0x00005555
0x555555758250 <node3>: 0x00000348     0x000000003    0x55758260    0x00005555
0x555555758260 <node4>: 0x00000393     0x000000004    0x55758270    0x00005555
0x555555758270 <node5>: 0x00000349     0x000000005    0x55758110    0x00005555
0x555555758280 <host_table>: 0x55556e67    0x00005555    0x55556e6f    0x00005555
(gdb)
```

현재 node가 총 5개가 있고, 각각 16진수를 값으로 가지고 있음을 알 수 있다.
이 값에 따라 내림차순으로 숫자를 쓰면 4 5 3 1 2가 된다.
숫자는 총 6개가 되어야 하므로, 적당한 위치에 6을 넣으면 정답이 된다.
따라서 phase_6의 정답은 4 5 3 1 6 2가 된다.

8. 결론

- 우선, 문제의 난이도가 상당했다. 그래서 몇몇 문제의 경우는 경우의 수를 모두 입력하여 정답을 찾아낸 후, 문제의 내용을 분석하는 방식으로 접근했다.