

Lab 3 [Bomb Lab] Report

학번 : 20220312

이름 : 박준혁

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

전체 정답 및 폭탄 해체 화면

```
lab3 > bomb149 > ≡ solution.txt
1 You can Russia from land here in Alaska.
2 1 2 4 8 16 32
3 3 -609
4 13 31 DrEvil
5 5 115
6 1 2 4 5 3 6
7 36
8
```

```
(gdb) r solution.txt
Starting program: /home/std/joon363/lab3/bomb149/bomb solution.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2. Keep going!
Halfway there!
So you got that one. Try this one.
Good work! On to the next...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
[Inferior 1 (process 18147) exited normally]
```

Phase 1

```
Dump of assembler code for function phase_1:  
0x00000000000400ef0 <+0>:    sub    $0x8,%rsp  
0x00000000000400ef4 <+4>:    mov    $0x4024e0,%esi  
0x00000000000400ef9 <+9>:    callq  0x40132e <strings_not_equal>  
0x00000000000400efe <+14>:   test   %eax,%eax  
0x00000000000400f00 <+16>:   je     0x400f07 <phase_1+23>  
0x00000000000400f02 <+18>:   callq  0x401594 <explode_bomb>  
0x00000000000400f07 <+23>:   add    $0x8,%rsp  
0x00000000000400f0b <+27>:   retq  
End of assembler dump.
```

+9~+18: strings_not_equal이 참이면 폭탄이 터지게 되어 있다.

즉 올바른 문자열을 입력해야 해체할 수 있다.

9: 예 break를 걸고 info register를 해보았다.

```
(gdb) disas  
Dump of assembler code for function phase_1:  
0x00000000000400ef0 <+0>:    sub    $0x8,%rsp  
0x00000000000400ef4 <+4>:    mov    $0x4024e0,%esi  
=> 0x00000000000400ef9 <+9>:    callq  0x40132e <strings_not_equal>  
0x00000000000400efe <+14>:   test   %eax,%eax  
0x00000000000400f00 <+16>:   je     0x400f07 <phase_1+23>  
0x00000000000400f02 <+18>:   callq  0x401594 <explode_bomb>  
0x00000000000400f07 <+23>:   add    $0x8,%rsp  
0x00000000000400f0b <+27>:   retq  
End of assembler dump.  
(gdb) x/s $rsi  
0x4024e0:      "You can Russia from land here in Alaska."  
(gdb) █
```

strings_not_equal의 인자로 들어간 esi를 보면 You can Russia from land here in Alaska. 이며, 입력과 동일해야 한다. 따라서 주어진 문자열이 정답이다.

Phase 2

```
Dump of assembler code for function phase_2:  
=> 0x0000000000400f0c <+0>:    push    %rbp  
  0x0000000000400f0d <+1>:    push    %rbx  
  0x0000000000400f0e <+2>:    sub    $0x28,%rsp  
  0x0000000000400f12 <+6>:    mov    %rsp,%rsi  
  0x0000000000400f15 <+9>:    callq  0x4015ca <read_six_numbers>  
  0x0000000000400f1a <+14>:   cmpl   $0x1,(%rsp)  
  0x0000000000400f1e <+18>:   je     0x400f40 <phase_2+52>  
  0x0000000000400f20 <+20>:   callq  0x401594 <explode_bomb>  
  0x0000000000400f25 <+25>:   jmp    0x400f40 <phase_2+52>  
  0x0000000000400f27 <+27>:   mov    -0x4(%rbx),%eax  
  0x0000000000400f2a <+30>:   add    %eax,%eax  
  0x0000000000400f2c <+32>:   cmp    %eax,(%rbx)  
  0x0000000000400f2e <+34>:   je     0x400f35 <phase_2+41>  
  0x0000000000400f30 <+36>:   callq  0x401594 <explode_bomb>  
  0x0000000000400f35 <+41>:   add    $0x4,%rbx  
  0x0000000000400f39 <+45>:   cmp    %rbp,%rbx  
  0x0000000000400f3c <+48>:   jne    0x400f27 <phase_2+27>  
  0x0000000000400f3e <+50>:   jmp    0x400f4c <phase_2+64>  
  0x0000000000400f40 <+52>:   lea    0x4(%rsp),%rbx  
  0x0000000000400f45 <+57>:   lea    0x18(%rsp),%rbp  
  0x0000000000400f4a <+62>:   jmp    0x400f27 <phase_2+27>  
  0x0000000000400f4c <+64>:   add    $0x28,%rsp  
  0x0000000000400f50 <+68>:   pop    %rbx  
  0x0000000000400f51 <+69>:   pop    %rbp  
  0x0000000000400f52 <+70>:   retq  
End of assembler dump.
```

+9의 read_six_numbers:

```
Dump of assembler code for function read_six_numbers:  
  0x00000000004015ca <+0>:    sub    $0x18,%rsp  
  0x00000000004015ce <+4>:    mov    %rsi,%rdx  
  0x00000000004015d1 <+7>:    lea    0x4(%rsi),%rcx  
  0x00000000004015d5 <+11>:   lea    0x14(%rsi),%rax  
  0x00000000004015d9 <+15>:   mov    %rax,0x8(%rsp)  
  0x00000000004015de <+20>:   lea    0x10(%rsi),%rax  
  0x00000000004015e2 <+24>:   mov    %rax,(%rsp)  
  0x00000000004015e6 <+28>:   lea    0xc(%rsi),%r9  
  0x00000000004015ea <+32>:   lea    0x8(%rsi),%r8  
  0x00000000004015ee <+36>:   mov    $0x402801,%esi  
  0x00000000004015f3 <+41>:   mov    $0x0,%eax  
  0x00000000004015f8 <+46>:   callq  0x400c30 <__isoc99_sscanf@plt>  
=> 0x00000000004015fd <+51>:   cmp    $0x5,%eax  
  0x0000000000401600 <+54>:   jg    0x401607 <read_six_numbers+61>  
  0x0000000000401602 <+56>:   callq  0x401594 <explode_bomb>  
  0x0000000000401607 <+61>:   add    $0x18,%rsp  
  0x000000000040160b <+65>:   retq  
End of assembler dump.  
(gdb) x/s $eax  
0x6:    <Address 0x6 out of bounds>  
(gdb) info register  
rax          0x6          6
```

+51에서 5개 초과의 수가 들어와야 넘어가는 걸 보면, 6개의 수를 입력받는 함수임을 알 수 있다.

따라서 phase 2는 6개의 수를 받는다.

Phase 2를 분석해보면 다음과 같다

```
<+0>: push %rbp
<+1>: push %rbx
<+2>: sub $0x28,%rsp
<+6>: mov %rsp,%rsi
<+9>: callq 0x4015ca <read_six_numbers>
<+14>: cmpl $0x1,(%rsp)
<+18>: je 0x400f40 <phase_2+52>
<+20>: callq 0x401594 <explode_bomb>
<+25>: jmp 0x400f40 <phase_2+52>
<+27>: mov -0x4(%rbx),%eax
<+30>: add %eax,%eax
<+32>: cmp %eax,(%rbx)
<+34>: je 0x400f35 <phase_2+41>
<+36>: callq 0x401594 <explode_bomb>
<+41>: add $0x4,%rbx
<+45>: cmp %rbp,%rbx
<+48>: jne 0x400f27 <phase_2+27>
<+50>: jmp 0x400f4c <phase_2+64>
<+52>: lea 0x4(%rsp),%rbx
<+57>: lea 0x18(%rsp),%rbp
<+62>: jmp 0x400f27 <phase_2+27>
<+64>: add $0x28,%rsp
<+68>: pop %rbx
<+69>: pop %rbp
<+70>: retq
```

rsi : rsp 저장
수 입력
첫번째 = 1이면 52
아니면 bomb .: 1번째 = 1

1번간자 (1) → eax
두배
두번째 수 = 1번수 × 2면 만더감 .: 2번째 = 2

rbx에 다음 수 주소 입력
만족 마지막 수 주소 아니면 Loop

rbx에 2번째 수 주소 입력

- 14: 첫번째 수가 1이 아니면 터진다 -> 1번 수 = 1
- 25: 우선 52로 간다.
- 52: 두번째 수를 rbx에 넣는다.
- 62: 27로 돌아간다.
- 27: 첫번째 수를 eax에 넣는다.
- 30: eax를 두배 한다.
- 32: 방금 두배가 된 eax와 두번째 수가 동일하면 41로 넘어간다. 따라서 2번 수=2
- 41: 세번째 수를 rbx에 넣는다.
- 45: 마지막 수가 아니면 27로 돌아가 루프를 돈다.

따라서 각 숫자들은 두배씩 되어야 하며 첫번째는 1, 두번째는 2, 세번째는 8, ... 이다.

즉 1,2,4,8,16,32가 정답이다.

```
lab3 > bomb149 > solution.txt
1 You can Russia from land here in Alaska.
2 1 2 4 8 16 32
```

(gdb) r solution.txt

```
Starting program: /home/std/joon363/lab3/bomb149/bomb solution.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2. Keep going!
```

Phase 3

```
Dump of assembler code for function phase_3:  
0x0000000000400f53 <+0>:    sub    $0x18,%rsp  
0x0000000000400f57 <+4>:    lea    0x8(%rsp),%rcx  
0x0000000000400f5c <+9>:    lea    0xc(%rsp),%rdx  
0x0000000000400f61 <+14>:   mov    $0x40280d,%esi  
0x0000000000400f66 <+19>:   mov    $0x0,%eax  
0x0000000000400f6b <+24>:   callq 0x400c30 <__isoc99_sscanf@plt>
```

```
End of assembler dump.  
(gdb) x/s 0x40280d  
0x40280d:      "%d %d"  
(gdb) █
```

scanf의 인수를 보면 "%d %d"로, phase_3에는 정수 두개를 입력한다.

```
=> 0x0000000000400f7a <+39>:  cmpl  $0x7,0xc(%rsp)          ↗ 첫 수 = 3 입렵시  
0x0000000000400f7f <+44>:  ja    0x400fe7 <phase_3+148>  
0x0000000000400f81 <+46>:  mov   0xc(%rsp),%eax  
0x0000000000400f85 <+50>:  jmpq  *0x402540(%rax,8)          ↘ ex = 0  
0x0000000000400f8c <+57>:  mov   $0x0,%eax  
0x0000000000400f91 <+62>:  jmp   0x400f98 <phase_3+69>  
0x0000000000400f93 <+64>:  mov   $0x2ca,%eax  
0x0000000000400f98 <+69>:  sub   $0x29c,%eax  
0x0000000000400f9d <+74>:  jmp   0x400fa4 <phase_3+81>  
0x0000000000400f9f <+76>:  mov   $0x0,%eax  
0x0000000000400fa4 <+81>:  add   $0x22f,%eax  
0x0000000000400fa9 <+86>:  jmp   0x400fb0 <phase_3+93>  
0x0000000000400fab <+88>:  mov   $0x0,%eax  
0x0000000000400fb0 <+93>:  sub   $0x261,%eax  
0x0000000000400fb5 <+98>:  jmp   0x400fb0 <phase_3+105>  
0x0000000000400fb7 <+100>: mov   $0x0,%eax  
0x0000000000400fb0 <+105>: add   $0x261,%eax  
0x0000000000400fc1 <+110>: jmp   0x400fc8 <phase_3+117>  
0x0000000000400fc3 <+112>: mov   $0x0,%eax  
0x0000000000400fc8 <+117>: sub   $0x261,%eax  
0x0000000000400fc0 <+122>: jmp   0x400fd4 <phase_3+129>  
0x0000000000400fcf <+124>: mov   $0x0,%eax  
0x0000000000400fd4 <+129>: add   $0x261,%eax  
0x0000000000400fd9 <+134>: jmp   0x400fe0 <phase_3+141>  
0x0000000000400fdb <+136>: mov   $0x0,%eax  
0x0000000000400fe0 <+141>: sub   $0x261,%eax  
0x0000000000400fe5 <+146>: jmp   0x400ff1 <phase_3+158>  
---Type <return> to continue, or q <return> to quit---  
0x0000000000400fe7 <+148>: callq 0x401594 <explode_bomb>  
0x0000000000400fec <+153>: mov   $0x0,%eax  
0x0000000000400ff1 <+158>: cmpl $0x5,0xc(%rsp)  
0x0000000000400ff6 <+163>: jg    0x400ffe <phase_3+171>  
0x0000000000400ff8 <+165>: cmp   0x8(%rsp),%eax  
0x0000000000400ffc <+169>: je    0x401003 <phase_3+176>  
0x0000000000400ffe <+171>: callq 0x401594 <explode_bomb>  
0x0000000000401003 <+176>: add   $0x18,%rsp  
0x0000000000401007 <+180>: retq
```

↳ 0x261 = 두 번째 수여야
↳ bomb 안터짐
∴ (3, -609)

44: 첫번째 수가 7보다 크면 bomb이 터지므로 첫번째 수 = 7 이하.

예를 들어 첫번째가 3이면, +50의 jump table에 해당하는 곳으로 이동해, +88로 이동하였다.

그때부터는 eax 값에 특정 값을 더하기 빼기를 반복하고, 3의 경우에는 -609가 된다.

이후 맨 마지막에서 두번째로 입력한 값과 -609를 비교해 동일하면 터지지 않는다.

각 숫자마다 다른 정답이 있다. (여러개의 정답 쌍 존재)

따라서 정답 중 하나인 3, -609가 정답이 된다.

```
1 You can Russia from land here in Alaska.
2 1 2 4 8 16 32
3 3 -609

문제    출력    디버그 콘솔    터미널    포트 1

0x0000000000401007 <+180>:    retq
End of assembler dump.
(gdb) c
Continuing.
Halfway there!
```

phase 3을 풀어 halfway there이라는 출력이 나왔다.

Phase 4

```
Dump of assembler code for function phase_4:
0x000000000040103b <+0>:    sub    $0x18,%rsp
0x000000000040103f <+4>:    lea    0x8(%rsp),%rcx :첫번째 입력
0x0000000000401044 <+9>:    lea    0xc(%rsp),%rdx :둘번째 입력
0x0000000000401049 <+14>:   mov    $0x40280d,%esi
0x000000000040104e <+19>:   mov    $0x0,%eax
0x0000000000401053 <+24>:   callq 0x400c30 <__isoc99_sscanf@plt
0x0000000000401058 <+29>:   cmp    $0x2,%eax
0x000000000040105b <+32>:   jne    0x401064 <phase_4+41>
0x000000000040105d <+34>:   cmpl   $0xe,0xc(%rsp)      1번숫자가 15면 bomb
0x0000000000401062 <+39>:   jbe    0x401069 <phase_4+46>
0x0000000000401064 <+41>:   callq 0x401594 <explode_bomb>
0x0000000000401069 <+46>:   mov    $0xe,%edx
0x000000000040106e <+51>:   mov    $0x0,%esi
0x0000000000401073 <+56>:   mov    0xc(%rsp),%edi
0x0000000000401077 <+60>:   callq 0x401008 <func4>
0x000000000040107c <+65>:   cmp    $0x1f,%eax
0x000000000040107f <+68>:   jne    0x401088 <phase_4+77>
0x0000000000401081 <+70>:   cmpl   $0x1f,0x8(%rsp)
0x0000000000401086 <+75>:   je     0x40108d <phase_4+82>
0x0000000000401088 <+77>:   callq 0x401594 <explode_bomb>
0x000000000040108d <+82>:   add    $0x18,%rsp
0x0000000000401091 <+86>:   retq

→ 입력 2개 아니면 bomb
edx = e
esi = 0
edi = 1번숫자
리턴 = Dxf 아니면 bomb
두번째 = 0xf면 ok.
```

29: 숫자 두개를 받지 않으면 터진다; 정수 두개를 입력으로 받는다.

그러면 첫번째 두번째 입력이 0x8(%rsp), 0xc(%rsp)에 저장된다.

34: 첫번째 입력이 15보다 크면 폭탄이 터진다.

46, 51, 56: 레지스터 세팅

60: func4를 호출한다.

65: func4의 리턴이 31이 아니라면 터진다. 즉 리턴이 31인 입력을 찾아야 한다.

70: 두번째 입력은 31이 아니면 터진다. 두번째 입력은 31로 고정이다.

func4를 분석하면 다음과 같다.

```
End of assembler dump.          edx=15 esi=0 edi=A1
Dump of assembler code for function func4:
0x000000000000401008 <+0>: push %rbx
0x000000000000401009 <+1>: mov %edx,%eax      eax=15
0x00000000000040100b <+3>: sub %esi,%eax      eax=15-0=15
0x00000000000040100d <+5>: mov %eax,%ebx      ebx=15
0x00000000000040100f <+7>: shr $0x1,%ebx      ebx>>1 : ebx=15
0x000000000000401012 <+10>: add %ebx,%eax      eax=15+15=30
0x000000000000401014 <+12>: sar %eax           eax>>1 : eax=7
0x000000000000401016 <+14>: lea (%rax,%rsi,1),%ebx  ebx=rax+rsi=15
0x000000000000401019 <+17>: cmp %edi,%ebx      A1 ≤ ebx=7 ?
0x00000000000040101b <+19>: jle 0x401029 <func4+33>  THE     False
0x00000000000040101d <+21>: lea -0x1(%rbx),%edx
0x000000000000401020 <+24>: callq 0x401008 <func4>
0x000000000000401025 <+29>: add %ebx,%eax
0x000000000000401027 <+31>: jmp 0x401039 <func4+49>
0x000000000000401029 <+33>: mov %ebx,%eax
0x00000000000040102b <+35>: cmp %edi,%ebx
0x00000000000040102d <+37>: jge 0x401039 <func4+49>
0x00000000000040102f <+39>: lea 0x1(%rbx),%esi
0x000000000000401032 <+42>: callq 0x401008 <func4>
0x000000000000401037 <+47>: add %ebx,%eax
0x000000000000401039 <+49>: pop %rbx
0x00000000000040103a <+50>: retq
End of assembler dump.

}
{ 장면
  %edi의 값은 바뀌지 않음
  }
```

잘 따라가 보면,

17: 첫 번째 입력값이 7 이하면 바로 7을 반환해버린다. 그런데 위에서 리턴이 31이어야 하므로, 첫 번째 입력값 > 7임을 알 수 있다.

21: %edx에 %rbx-1의 주소를 넣는다. 값을 1 빼는게 아니기 때문에 복잡하다.

그 뒤에 재귀호출을 또 하고, 리턴값을 두 배 하는 등의 일이 일어난다.

그러나 edi를 관찰해 보면 함수 내부에서 바뀌지 않는다.

이 말은, 더 이상 로직을 관찰하지 않아도 31이 나올 때까지 edi를 넣어보면 알 수 있다는 것이다.

범위는 8~15의 정수이며 다음과 같다.

8->35, 9->27, 10->37, 11->18, 12->43, 13->31 (ok)

따라서 정답은 13, 31이 된다.

```
Breakpoint 3, 0x00000000000040108d in phase_4 ()
(gdb) info register
rax          0x1f    31
rbx          0xfffffffffe0d8  140737488347352
rcx          0x20    32
rdx          0xe     14
rsi          0xc     12
rdi          0xd     13
rbp          0x0     0x0
```

정답 상황의 레지스터 값은 위와 같다. rdi에 13이 그대로 있고, rax= 31을 반환하였다.

Phase 5

```
Dump of assembler code for function phase_5:
=> 0x0000000000401092 <+0>:    sub    $0x18,%rsp
0x0000000000401096 <+4>:    lea    0x8(%rsp),%rcx → 입력2 = A2      rcx=A2
0x000000000040109b <+9>:    lea    0xc(%rsp),%rdx → 입력1 = A1      rdx=A1
0x00000000004010a0 <+14>:   mov    $0x40280d,%esi      esi=0x40280d
0x00000000004010a5 <+19>:   mov    $0x0,%eax      eax=0
0x00000000004010aa <+24>:  callq  0x400c30 <__isoc99_sscanf@plt>
0x00000000004010af <+29>:  cmp    $0x1,%eax      eax=A1
0x00000000004010b2 <+32>:  jg    0x4010b9 <phase_5+39> ) 입력 1개 이상 받아야 함.
0x00000000004010b4 <+34>:  callq  0x401594 <explode_bomb>
0x00000000004010b9 <+39>:  mov    0xc(%rsp),%eax
0x00000000004010bd <+43>:  and    $0xf,%eax      eax=A1 & 0xf → 뒤 4자리
0x00000000004010c0 <+46>:  mov    %eax,0xc(%rsp)      A1 = A1 & 0xf
0x00000000004010c4 <+50>:  cmp    $0xf,%eax      A1 & 0xf = 0xf → bomb
0x00000000004010c7 <+53>:  je    0x4010f5 <phase_5+99>      :: 뒤 4자리 != 0xf
0x00000000004010c9 <+55>:  mov    $0x0,%ecx      ecx=0
0x00000000004010ce <+60>:  mov    $0x0,%edx      edx=0
0x00000000004010d3 <+65>:  add    $0x1,%edx      edx=1
0x00000000004010d6 <+68>:  cltq
0x00000000004010d8 <+70>:  mov    0x402580(%rax,4),%eax      eax = eax sign extension
0x00000000004010df <+77>:  add    %eax,%ecx      ecx+=eax
0x00000000004010e1 <+79>:  cmp    $0xf,%eax      eax=0xf 까지 loop
0x00000000004010e4 <+82>:  jne    0x4010d3 <phase_5+65>
0x00000000004010e6 <+84>:  mov    %eax,0xc(%rsp)
0x00000000004010ea <+88>:  cmp    $0xf,%edx
0x00000000004010ed <+91>:  jne    0x4010f5 <phase_5+99>
0x00000000004010ef <+93>:  cmp    $0x8(%rsp),%ecx
0x00000000004010f3 <+97>:  je    0x4010fa <phase_5+104>
0x00000000004010f5 <+99>:  callq  0x401594 <explode_bomb>
0x00000000004010fa <+104>: add    $0x18,%rsp
0x00000000004010fe <+108>: retq
```

↑
입력은 숫자 두 개이다.
분석해보면 다음과 같다.
입력 1을 받고 뒤에 4자리만 끊은 다음 rax에 넣는다. (최대 0xf = 15)
0x402580+4*rax에 가서, 해당 값을 rax로 넣는다. 위 array는 아래와 같이 16개의 수로 되어 있다.
그리고 rax=15가 될 때까지 루프를 도는데, bomb 해제 조건은
처음에 1번 증가했고, 루프를 돌 때마다 1씩 증가하는 rdx가 15가 될 것,
루프를 돌 때마다 rax값을 더해서 얻는 ecx와 입력2가 동일할 것이다.
따라서 14번 루프를 돌게 만드는 입력 1을 찾고, (그때까지의 합)=입력 2로 하면 된다.

```
(gdb) x/20wd 0x402580
0x402580 <array.3161>: 10      2      14      7
0x402590 <array.3161+16>: 8      12      15      11
0x4025a0 <array.3161+32>: 0      4      1       13
0x4025b0 <array.3161+48>: 3      9      6       5
0x4025c0: 2032168787      1948284271      1802398056      1970239776
(gdb) █
```

위 array를 도식화 해 보았다.

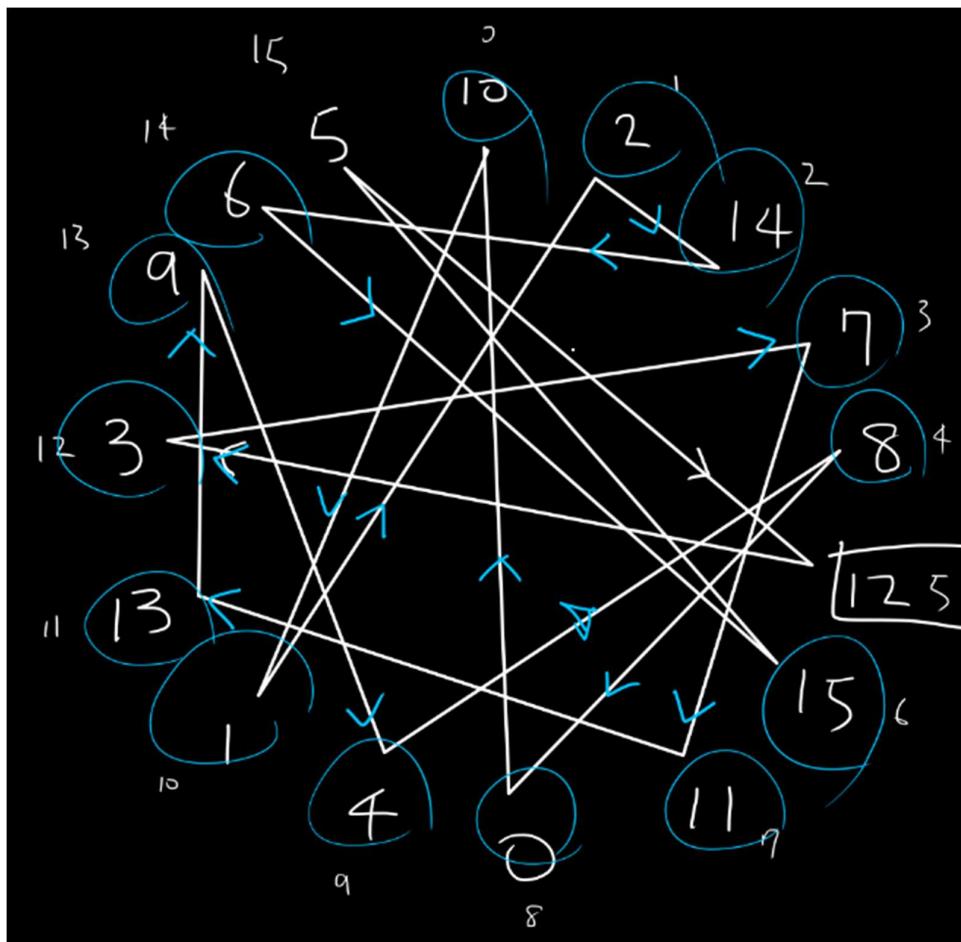
5번 자리에 12가 들어 있으므로, 그 다음에 12번 자리로 이동한다.

12번 자리에 3이 들어 있으므로, 그 다음에 3번 자리로 이동한다.

...

14번 자리에 6이 들어 있으므로, 그 다음에 6번 자리로 이동한다.

6번 자리에 15가 들어 있으므로 루프가 멈추었으며 이때까지 14번 루프가 돌았다.



따라서, 14번 이동하여 rax=15에 도달하려면 rax=5(값은 12)에서 출발해야 한다.

이때의 ecx 값은 info register로 확인한 결과 115이다.

(사실 위 루프에서 0~15의 합 = 120, 더해지지 않은 숫자 5를 빼면 115이다)

rax	0xf	15
rbx	0x7fffffffffe0d8	140
rcx	0x73	115
rdx	0xf	15
rsi	0x0	0

Final Phase 6

어셈블리 분석:

6개의 입력(1~6의 서로 다른 6개의 정수)를 받아서

1. 각 입력 숫자를 7에서 뺀 수로 치환
2. 1의 입력 배열에 대해, 주어진 배열을 입력 배열을 인덱스로 해서 재배열
3. 재배열된 배열이 내림차순이어야 함

정답 추론:

주어진 배열을 크기순으로 순서를 매기고 그 순서를 뒤집고 7에서 빼면 된다.

bomb 149의 경우 주어진 배열이 127 133 179 128 641 729이며,

크기 순서는 1 3 4 2 5 6, 뒤집으면 6 5 3 2 4 1, 7에서 빼면 [1 2 4 5 3 6] <-정답

문제가 점프를 경계로 여러 파트로 나누어져 있으며,

part 1 입력 제한, part 2 배열 수정, part 3 노드 값 저장, part 4 정렬 확인이 된다.

Phase 6 - part 1: 입력 제한 걸기

```
Dump of assembler code for function phase_6:  
=> 0x00000000004010ff <+0>: push %r14  
0x0000000000401101 <+2>: push %r13  
0x0000000000401103 <+4>: push %r12  
0x0000000000401105 <+6>: push %rbp  
0x0000000000401106 <+7>: push %rbx  
0x0000000000401107 <+8>: sub $0x50,%rsp  
0x000000000040110b <+12>: lea 0x30(%rsp),%r13  
0x0000000000401110 <+17>: mov %r13,%rsi  
0x0000000000401113 <+20>: callq 0x4015ca <read_six_numbers> → rspoff A0,A1,A2,A3,A4,A5 저장  
0x0000000000401118 <+25>: mov %r13,%r14  
0x000000000040111b <+28>: mov $0x0,%r12d  
  
0x0000000000401121 <+34>: mov %r13,%rbp  
0x0000000000401124 <+37>: mov 0x0(%r13),%eax  
0x0000000000401128 <+41>: sub $0x1,%eax  
0x000000000040112b <+44>: cmp $0x5,%eax  
0x000000000040112e <+47>: jbe 0x401135 <phase_6+54>  
0x0000000000401130 <+49>: callq 0x401594 <explode_bomb>  
0x0000000000401135 <+54>: add $0x1,%r12d  
0x0000000000401139 <+58>: cmp $0x6,%r12d  
0x000000000040113d <+62>: je 0x401161 <phase_6+98>  
0x000000000040113f <+64>: mov %r12d,%ebx  
0x0000000000401142 <+67>: movslq %ebx,%rax  
0x0000000000401145 <+70>: mov 0x30(%rsp,%rax,4),%eax  
0x0000000000401149 <+74>: cmp %eax,0x0(%rbp)  
0x000000000040114c <+77>: jne 0x401153 <phase_6+84>  
0x000000000040114e <+79>: callq 0x401594 <explode_bomb>  
0x0000000000401153 <+84>: add $0x1,%ebx  
0x0000000000401156 <+87>: cmp $0x5,%ebx  
0x0000000000401159 <+90>: jle 0x401142 <phase_6+67>  
0x000000000040115b <+92>: add $0x4,%r13  
0x000000000040115f <+96>: jmp 0x401121 <phase_6+34>
```

Handwritten annotations on the assembly dump:

- Annotations for the first part of the loop:
 - Call to `read_six_numbers` at address 0x4015ca. Comment: `rspoff A0,A1,A2,A3,A4,A5 저장`
 - Assignment of `r13 = A0`
 - Assignment of `r12 = 0`
- Annotations for the main loop body:
 - Assignment of `r13 = Ai` (with i pointing to `r12`)
 - Assignment of `eax = Ai - 1`
 - Annotation: $5 \geq eax$
 - Annotation: $r12++$
 - Annotation: $6 = r12 ?$
 - Annotation: `False` leads to `explode_bomb` at 0x401594
 - Annotation: `True` leads to `if true, bomb` at 0x401159
 - Annotation: `ebx = ebx + 1`
 - Annotation: `rax = ebx`
 - Annotation: `eax = Arax`
 - Annotation: `eax != A0`
 - Annotation: $if true, bomb$
 - Annotation: $ebx++$
 - Annotation: $5 \geq ebx$
 - Annotation: `False`
 - Annotation: `Ti3 != null`
- Final annotation: `exit`

입력은 정수 6개(A0부터 A5로 부르자)이다. 이 입력값은 rsp와 r13 둘 다에 저장되어 있다.

```
(gdb) x/6wd $rsp+0x30
0x7fffffffdfa0: 1      2      3      4
0x7fffffffdfb0: 5      6
```

part 1에서 하는 일은 다음과 같다.

loop 1: rbp=A0, eax=A0-1, r12=1, ebx=1, rax=1, eax=A1, A1!=A0, ebx=2,
rax=2, eax=A2, A2!=A0, ebx=3, ... A5!=A0, ebx=6, r13=A1

loop 2: rbp=A1, eax=A1-1, r12=2, ...

loop 5: rbp=A4, eax=A4-1, r12=5, ...

loop 6: rbp=A5, eax=A5-1, r12=6 -> end of loop

즉 A0부터 A5는 전부 다르며 6 이하이고, An-1을 jbe에서 비교하므로(unsigned) 0이어도 안된다.

즉 입력은 1 2 3 4 5 6에서 하나씩 고른 형태여야 한다. (part 1 결론)

Phase 6 - part 2: 배열 값 바꾸기

```

0x0000000000401161 <+98>: lea    0x48(%rsp),%rsi
0x0000000000401166 <+103>: mov    %r14,%rax
--Type <return> to continue, or q <return> to quit---
0x0000000000401169 <+106>: mov    $0x7,%ecx
0x000000000040116e <+111>: mov    %ecx,%edx
0x0000000000401170 <+113>: sub    (%rax),%edx
0x0000000000401172 <+115>: mov    %edx,(%rax)
0x0000000000401174 <+117>: add    $0x4,%rax
0x0000000000401178 <+121>: cmp    %rsi,%rax
0x000000000040117b <+124>: jne    0x40116e <phase_6+111>
0x000000000040117d <+126>: mov    $0x0,%esi
0x0000000000401182 <+131>: jmp    0x4011a4 <phase_6+165>✓

rsi = rsp + 48 → A5 다음 수 = A6
rax = r14 = A0

ecx = 7
edx = ecx
edx = 7 - A0
rax = 7 - A0
rax != rax? ← F
rsi = rax? ↓ T: rax 순회 끝
rsi = 0

A0 ~ A5
7 - A0, ~ 7 - A5
변환

```

part 1의 +62에서 98로 점프해 왔다.

98: rsi에 A5 다음 수의 주소를 넣어 놓았다. 0x30 (48)부터 숫자가 있고, 0x48(72)를 더하면 숫자 6개 다음 위치인 것이다

103: rax에 A0를 넣었다.

106: ecx=7

111~121: rax의 각 위치 = An의 위치에 7-An을 넣는다. 루프는 A5 다음 수의 주소에 도달시 탈출한다. 즉 배열에서 EOF까지 순회하면서

각 배열의 원소를 7에서 뺀 값으로 교체한다. (part 2 결론)

Phase 6 - part 3: 노드 값 스택에 순서대로 저장하기

```

0x0000000000401184 <+133>: mov    0x8(%rdx),%rdx      rdx = rsi + 2
0x0000000000401188 <+137>: add    $0x1,%eax          eax ++
0x000000000040118b <+140>: cmp    %ecx,%eax          ecx != eax
0x000000000040118d <+142>: jne    0x401184 <phase_6+133>
0x000000000040118f <+144>: jmp    0x401196 <phase_6+151>
0x0000000000401191 <+146>: mov    $0x6042f0,%edx
0x0000000000401196 <+151>: mov    %rdx,(%rsp,%rsi,2) edx = 0x6042f0
0x000000000040119a <+155>: add    $0x4,%rsi           rsi += 4
0x000000000040119e <+159>: cmp    $0x18,%rsi          rsi = rsi
0x00000000004011a2 <+163>: je     0x4011b9 <phase_6+180>
0x00000000004011a4 <+165>: mov    0x30(%rsp,%rsi,1),%ecx
0x00000000004011a8 <+169>: cmp    $0x1,%ecx           ecx = rsp + rsi = 1-A
0x00000000004011ab <+172>: jle    0x401191 <phase_6+146>
0x00000000004011ad <+174>: mov    $0x1,%eax          eax = 1
0x00000000004011b2 <+179>: mov    $0x6042f0,%edx
0x00000000004011b7 <+184>: jmp    0x401184 <phase_6+133>

```

예를 들어 4 3 6 5 2 1이 입력이라고 해보자. 여기 전까지 3 4 1 2 5 6이 저장될 것이다

(gdb) x/48wd 0x6042f0						
0x6042f0 <node1>:	127	1	6308608	0		
0x604300 <node2>:	133	2	6308624	0		
0x604310 <node3>:	179	3	6308640	0		
0x604320 <node4>:	128	4	6308656	0		
0x604330 <node5>:	641	5	6308672	0		
0x604340 <node6>:	729	6	0	0		
0x604350 <user_password>:			1262643049	126		
0x604360 <user_password+16>:			1113716810	812		

0x6042f0 의 “node” array는 다음과 같다. (0x604350 직전까지)

노드 1의 3번 원소인 6308608 = 0x604300인것으로 보아 다음 노드의 주소이다.

part 3에서 하는 일은 다음과 같다.

1. ecx=3 rsi=0 으로 시작

eax=1, rdx=127, rdx=133, eax=2, rdx=179, eax=3

rsp+0에 179

rsi=4

2. ecx=4

eax=1, rdx=127, rdx=133, eax=2, rdx=179, eax=3, rdx=128, eax=4,

rsp+8에 128

rsi=8

3. ecx=1:

edx=127,
rsp+16에 127
rsi=12

4. ecx=2

eax=1, rdx=127, rdx=133, eax=2
rsp+24에 133
rsi=16

5. ecx=5

eax=1, rdx=127, rdx=133, eax=2, rdx=179, eax=3, rdx=128, eax=4, rdx=641, eax=5
rsp+320에 641
rsi=20

6. ecx=6

eax=1, rdx=127, ..., rdx=729, eax=6
rsp+400에 729
rsi=24 break

따라서 part 3이 끝나면

(예를 들어) 입력 436521에 대해

rsp [+48 +52 +56 +60 +64 +68]에는 3,4,1,2,5,6이

rsp [+0 +8 +16 ... +40]에는 node 3,4,1,2,5,6의 값이 저장되게 된다. (part 3 결론)

Phase 6 - part 4: 정렬 여부 확인하기

```
0x00000000004011b9 <+186>:    mov    (%rsp),%rbx
0x00000000004011bd <+190>:    lea    0x8(%rsp),%rax
0x00000000004011c2 <+195>:    lea    0x30(%rsp),%rsi
0x00000000004011c7 <+200>:    mov    %rbx,%rcx
0x00000000004011ca <+203>:    mov    (%rax),%rdx
0x00000000004011cd <+206>:    mov    %rdx,0x8(%rcx)
0x00000000004011d1 <+210>:    add    $0x8,%rax
0x00000000004011d5 <+214>:    cmp    %rsi,%rax
0x00000000004011d8 <+217>:    je     0x4011df <phase_6+224>
0x00000000004011da <+219>:    mov    %rdx,%rcx
0x00000000004011dd <+222>:    jmp    0x4011ca <phase_6+203>
-Type <return> to continue, or q <return> to quit---
0x00000000004011df <+224>:    movq   $0x0,0x8(%rdx)
0x00000000004011e7 <+232>:    mov    $0x5,%ebp
0x00000000004011ec <+237>:    mov    0x8(%rbx),%rax
0x00000000004011f0 <+241>:    mov    (%rax),%eax
0x00000000004011f2 <+243>:    cmp    %eax,(%rbx)
0x00000000004011f4 <+245>:    jge    0x4011fb <phase_6+252>
0x00000000004011f6 <+247>:    callq  0x401594 <explode_bomb>
0x00000000004011fb <+252>:    mov    0x8(%rbx),%rbx
0x00000000004011ff <+256>:    sub    $0x1,%ebp
0x0000000000401202 <+259>:    jne    0x4011ec <phase_6+237>
0x0000000000401204 <+261>:    add    $0x50,%rsp
0x0000000000401208 <+265>:    pop    %rbx
0x0000000000401209 <+266>:    pop    %rbp
0x000000000040120a <+267>:    pop    %r12
0x000000000040120c <+269>:    pop    %r13
0x000000000040120e <+271>:    pop    %r14
0x0000000000401210 <+273>:    retq
d of assembler dump.
```

rsp [+0 +8 +16 ... +40] 을 node 1~6, 들어있는 값은 n3,n4,n1,n2,n5,n6이라고 하자.

186: rbx=(rsp)=n3

190: rax=*rsp+8 (node2)

195: rsi=*rsp+0x30 (input0)

200: rcx=rbx=n3

위와 같이 레지스터에 값을 넣고 루프를 돌게 된다.

이후 루프를 도는 부분은 다음과 같은 동작을 수행한다.

loop 1:

```
rdx=(rax)=n4, *(rcx+8)=node2=n4, rax+=8: rax=*rsp+16(node3), rcx=rdx=n4
```

loop 2:

```
rdx=(rax)=n1, *(rcx+8)=node3=n1, rax+=8: rax=*rsp+24(node4), rcx=rdx=n1
```

loop 3:

```
rdx=(rax)=n2, *(rcx+8)=node4=n2, rax+=8: rax=*rsp+32(node4), rcx=rdx=n2
```

...

loop 5:

```
dx=(rax)=n6, *(rcx+8)=node6=n6, rax+=8: rax=*rsp+48(node4), break;
```

224: (*rdx+8) = node 6 다음 주소 = 0

232: ebp=5

237: rax=n4

241: eax=n4

243: eax=n4 <= (rbx)=n3 **false -> bomb; n4가 n3보다 작다**

if true

rbx 한칸 이동: rbx=node2

ebp =4

237: rax=n1

241: eax=n1

243: eax=n1<=(rbx)=n4 **false -> bomb; n1이 n4보다 작다**

이하 루프 반복

즉 n3>=n4>=n1 ... 이어야, 정렬되어 있어야지 터지지 않는다. (part 4 결론)

therefore

주어진 node 배열 127 133 179 128 641 729에 대해 크기순서는 1 3 4 2 5 6

7-입력배열 = 6 5 3 2 4 1 이어야 하므로, 입력배열 = 1 2 4 5 3 6(정답)

(크기 순서를 앞뒤로 뒤집고 7에서 빼면 된다)

```
6 1 2 4 5 3 6
문제 출력 디버그 콘솔 터미널 포트 1
(gdb) c
Continuing.
Congratulations! You've defused the bomb!
```

Secret phase

secret phase 는 main 속 phase_defused 안에 숨어 있다.

phase_defused 코드는 다음과 같다.

```
(gdb) disas phase_defused
Dump of assembler code for function phase_defused:
0x0000000000401732 <+0>:    sub    $0x68,%rsp
0x0000000000401736 <+4>:    mov    $0x1,%edi
0x000000000040173b <+9>:    callq  0x4014d0 <send_msg>
0x0000000000401740 <+14>:   cmpl   $0x6,0x203055(%rip)        # 0x60479c <num_input_strings>
0x0000000000401747 <+21>:   jne    0x4017b6 <phase_defused+132>
0x0000000000401749 <+23>:   lea    0x10(%rsp),%r8
0x000000000040174e <+28>:   lea    0x8(%rsp),%rcx
0x0000000000401753 <+33>:   lea    0xc(%rsp),%rdx
0x0000000000401758 <+38>:   mov    $0x402857,%esi
0x000000000040175d <+43>:   mov    $0x6048b0,%edi
0x0000000000401762 <+48>:   mov    $0x0,%eax
0x0000000000401767 <+53>:   callq  0x400c30 <__isoc99_sscanf@plt>
0x000000000040176c <+58>:   cmp    $0x3,%eax
0x000000000040176f <+61>:   jne    0x4017a2 <phase_defused+112>
0x0000000000401771 <+63>:   mov    $0x402860,%esi
0x0000000000401776 <+68>:   lea    0x10(%rsp),%rdi
0x000000000040177b <+73>:   callq  0x40132e <strings_not_equal>
0x0000000000401780 <+78>:   test   %eax,%eax
0x0000000000401782 <+80>:   jne    0x4017a2 <phase_defused+112>
0x0000000000401784 <+82>:   mov    $0x4026b8,%edi
---Type <return> to continue, or q <return> to quit---
0x0000000000401789 <+87>:   callq  0x400b40 <puts@plt>
0x000000000040178e <+92>:   mov    $0x4026e0,%edi
0x0000000000401793 <+97>:   callq  0x400b40 <puts@plt>
0x0000000000401798 <+102>:  mov    $0x0,%eax
0x000000000040179d <+107>:  callq  0x40124f <secret_phase>
0x00000000004017a2 <+112>:  mov    $0x402718,%edi
0x00000000004017a7 <+117>:  callq  0x400b40 <puts@plt>
0x00000000004017ac <+122>:  mov    $0x402748,%edi
0x00000000004017b1 <+127>:  callq  0x400b40 <puts@plt>
0x00000000004017b6 <+132>:  add    $0x68,%rsp
0x00000000004017ba <+136>:  retq
End of assembler dump.
```

107 번의 secret_phase 를 접근하기 위해서는 여러 가지 조건이 필요하다.

+14: 0x60479c 가 6 이어야 secret 을 뛰어넘지 않는다
phase_6 까지 진행하고 main 이 종료되기 전 이 값을 보면 6 이 나온다.
즉 6 번 문제까지 풀고 나서 실행되는 코드이다.

```
0x60479c <num_input_strings>: 6
(gdb) 
```

+58: sscanf 의 리턴이 3 개여야 한다. 이 scanf 에 들어가는 esi, edi 는 다음과 같다.

```
(gdb) x/s 0x402857
0x402857: "%d %d %s"
(gdb) x/s 0x6048b0
0x6048b0 <input_strings+240>: "13 31"
(gdb) 
```

4 번 문제의 입력값인 13 31 이 edi 에 있고, esi 에는 %d %d %s 가 있다.

즉, 4 번 문제에서 스트링을 추가로 입력해야지 secret 에 접근할 수 있는 것이다.

그렇다면 그 스트링은 +73 에서 확인하는데,

+73 의 인풋인 0x402860 은 다음과 같다.

```
(gdb) x/s 0x402860
0x402860: "DrEvil"
```

따라서 phase_4 에서 13 31 DrEvil 을 입력하면 접근할 수 있다.

secret_phase 호출 직전의 코드들은 다음과 같은 문자열을 담고 있다.

```
End of assembler dump.
(gdb) x/s 0x4026b8
0x4026b8: "Curses, you've found the secret phase!"
(gdb) x/s 0x4026e0
0x4026e0: "But finding it and solving it are quite different..."
```

4 번 답에 DrEvil 을 넣고 돌리면 다음과 같다.

```
(gdb) r solution.txt
Starting program: /home/std/joon363/lab3/bomb149/bomb solution.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2. Keep going!
Halfway there!
So you got that one. Try this one.
Good work! On to the next...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```

secret_phase 분석

```
Dump of assembler code for function secret_phase:  
0x000000000040124f <+0>:    push    %rbx  
0x0000000000401250 <+1>:    callq   0x40160c <read_line>  
0x0000000000401255 <+6>:    mov     $0xa,%edx  
0x000000000040125a <+11>:   mov     $0x0,%esi  
0x000000000040125f <+16>:   mov     %rax,%rdi  
0x0000000000401262 <+19>:   callq   0x400c00 <strtol@plt>  
0x0000000000401267 <+24>:   mov     %rax,%rbx  
0x000000000040126a <+27>:   lea     -0x1(%rax),%eax  
0x000000000040126d <+30>:   cmp     $0x3e8,%eax  
0x0000000000401272 <+35>:   jbe     0x401279 <secret_phase+42>  
0x0000000000401274 <+37>:   callq   0x401594 <explode_bomb>  
0x0000000000401279 <+42>:   mov     %ebx,%esi  
0x000000000040127b <+44>:   mov     $0x604110,%edi  
0x0000000000401280 <+49>:   callq   0x401211 <fun7>  
0x0000000000401285 <+54>:   test    %eax,%eax  
0x0000000000401287 <+56>:   je      0x40128e <secret_phase+63>  
0x0000000000401289 <+58>:   callq   0x401594 <explode_bomb>  
0x000000000040128e <+63>:   mov     $0x402510,%edi  
0x0000000000401293 <+68>:   callq   0x400b40 <puts@plt>  
0x0000000000401298 <+73>:   callq   0x401732 <phase_defused>  
0x000000000040129d <+78>:   pop    %rbx  
0x000000000040129e <+79>:   retq  
  
End of assembler dump.
```

+6: edx=10

+11: esi=0

+16: rdi=rax (read_line 의 반환값)

+19: string to long 호출; rax 의 값을 long 으로 변환한다.

+24: rbx=rax (strtol 의 반환값)

+27: rax --

+30: rax<=0x3e8 (10 진수 1000)이어야 하고, 아니면 터진다.

즉 입력받은 값 – 1 <=1000 이어야 하며 **입력값 <= 1001** 이다.

+42: esi = ebx (strtol 의 반환값)

+44: edi = 0x604110

+49: fun7 호출, 인자는 위 두개

+54: fun7 의 반환값이 0 이라면 안전, 아니면 bomb

-> **fun7 은 0 을 반환해야 한다.**

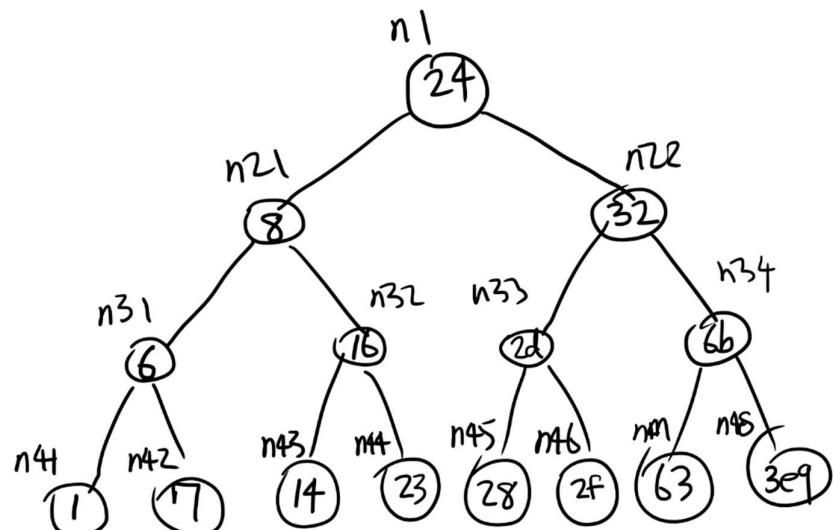
이러면 끝난다.

0x604110 은 다음과 같은 정보를 담고 있다.

```
(gdb) x/64xg 0x604110
0x604110 <n1>: 0x0000000000000024      0x0000000000604130
0x604120 <n1+16>:          0x0000000000604150      0x0000000000000000
0x604130 <n21>: 0x0000000000000008      0x00000000006041b0
0x604140 <n21+16>:          0x0000000000604170      0x0000000000000000
0x604150 <n22>: 0x0000000000000032      0x0000000000604190
0x604160 <n22+16>:          0x00000000006041d0      0x0000000000000000
0x604170 <n32>: 0x0000000000000016      0x0000000000604290
0x604180 <n32+16>:          0x0000000000604250      0x0000000000000000
0x604190 <n33>: 0x000000000000002d      0x00000000006041f0
0x6041a0 <n33+16>:          0x00000000006042b0      0x0000000000000000
0x6041b0 <n31>: 0x0000000000000006      0x0000000000604210
0x6041c0 <n31+16>:          0x0000000000604270      0x0000000000000000
0x6041d0 <n34>: 0x000000000000006b      0x0000000000604230
0x6041e0 <n34+16>:          0x00000000006042d0      0x0000000000000000
0x6041f0 <n45>: 0x0000000000000028      0x0000000000000000
0x604200 <n45+16>:          0x0000000000000000      0x0000000000000000
0x604210 <n41>: 0x0000000000000001      0x0000000000000000
0x604220 <n41+16>:          0x0000000000000000      0x0000000000000000
0x604230 <n47>: 0x0000000000000063      0x0000000000000000
0x604240 <n47+16>:          0x0000000000000000      0x0000000000000000
0x604250 <n44>: 0x0000000000000023      0x0000000000000000
0x604260 <n44+16>:          0x0000000000000000      0x0000000000000000
0x604270 <n42>: 0x0000000000000007      0x0000000000000000
0x604280 <n42+16>:          0x0000000000000000      0x0000000000000000
0x604290 <n43>: 0x0000000000000014      0x0000000000000000
0x6042a0 <n43+16>:          0x0000000000000000      0x0000000000000000
0x6042b0 <n46>: 0x000000000000002f      0x0000000000000000
0x6042c0 <n46+16>:          0x0000000000000000      0x0000000000000000
0x6042d0 <n48>: 0x000000000000003e9      0x0000000000000000
0x6042e0 <n48+16>:          0x0000000000000000      0x0000000000000000
```

잘 보면 값, left, right 의 주소를 가지고 있는 노드들의 형태로, 이진 트리를 이루고 있다.

노드의 이름은 n(level)(position)으로, level 2 에는 n21 n22 가 level 3 에는 n31 n32 n33 이 있는 식



fun7 분석

```
Dump of assembler code for function fun7:  
0x0000000000401211 <+0>:    sub    $0x8,%rsp  
0x0000000000401215 <+4>:    test   %rdi,%rdi  
0x0000000000401218 <+7>:    je     0x401245 <fun7+52>  
0x000000000040121a <+9>:    mov    (%rdi),%edx  
0x000000000040121c <+11>:   cmp    %esi,%edx  
0x000000000040121e <+13>:   jle    0x40122d <fun7+28>  
0x0000000000401220 <+15>:   mov    0x8(%rdi),%rdi  
0x0000000000401224 <+19>:   callq  0x401211 <fun7>  
0x0000000000401229 <+24>:   add    %eax,%eax  
0x000000000040122b <+26>:   jmp    0x40124a <fun7+57>  
0x000000000040122d <+28>:   mov    $0x0,%eax  
0x0000000000401232 <+33>:   cmp    %esi,%edx  
0x0000000000401234 <+35>:   je     0x40124a <fun7+57>  
0x0000000000401236 <+37>:   mov    0x10(%rdi),%rdi  
0x000000000040123a <+41>:   callq  0x401211 <fun7>  
0x000000000040123f <+46>:   lea    0x1(%rax,%rax,1),%eax  
0x0000000000401243 <+50>:   jmp    0x40124a <fun7+57>  
0x0000000000401245 <+52>:   mov    $0xffffffff,%eax  
0x000000000040124a <+57>:   add    $0x8,%rsp  
0x000000000040124e <+61>:   retq  
End of assembler dump.
```

esi = strtol 의 반환값, edi = 0x604110; n1 의 값의 주소이다.

+4: rdi==0 이면 52로 이동해 -1을 반환한다.

+9: edx=(rdi): 노드 n1의 값인 24가 들어간다.

+11

edx<=esi 인 경우

+28: eax=0

+33:

esi==edx 이면

+57로 이동해 0을 반환한다.

esi!=edx 인 경우

+37: rdi에 16을 더해 오른쪽 노드로 이동한다.

+41: 맨 처음으로 돌아가 재귀 호출한다.

+46: 재귀한 값*2+1을 리턴한다.

edx>esi 인 경우

+15: rdi에 8을 더해 왼쪽 노드로 이동한다.

+19: 맨 처음으로 돌아가 재귀 호출한다.

+24: 재귀한 값의 두배를 리턴한다.

따라서 입력값 x에 따라 다음과 같이 동작한다.

```
if x>=24:  
    if x==24 return 0  
  
    else  
        move to n22  
        return fun7()*2+1  
  
if x<24:  
    move to n21  
    return fun7()*2
```

secret_phase 코드에서 fun7은 0을 반환해야 하므로

입력값 x는 0x24=36가 된다.

```
lab3 > bomb149 > ≡ solution.txt  
1 You can Russia from land here in Alaska.  
2 1 2 4 8 16 32  
3 3 -609  
4 13 31 DrEvil  
5 5 115  
6 1 2 4 5 3 6  
7 36  
8
```

```
(gdb) r solution.txt  
Starting program: /home/std/joon363/lab3/bomb149/bomb solution.txt  
Welcome to my fiendish little bomb. You have 6 phases with  
which to blow yourself up. Have a nice day!  
Phase 1 defused. How about the next one?  
That's number 2. Keep going!  
Halfway there!  
So you got that one. Try this one.  
Good work! On to the next...  
Curses, you've found the secret phase!  
But finding it and solving it are quite different...  
Wow! You've defused the secret stage!  
Congratulations! You've defused the bomb!  
Your instructor has been notified and will verify your solution.  
[Inferior 1 (process 18147) exited normally]
```