

Intro. to Computer SW Systems Lab Report

[Bomb Lab]

20220100 박기현

kihyun@postech.ac.kr

명예 서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

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

main

```
(gdb) disas main
Dump of assembler code for function main:
0x000000000000400bd <+0>: push %rbx
0x000000000000400be <+1>: cmp $0x1,%edi
0x000000000000400c1 <+4>: jne 0x400dd3 <main+22>
0x000000000000400d3 <+6>: mov 0x2039be(%rip),%rax # 0x604788 <stdin@GLIBC_2.2.5>
0x000000000000400dc9 <+13>: mov %rax,0x2039cf(%ip)
0x000000000000400d1 <+20>: jmp 0x400e2c <main+111>
0x000000000000400d3 <+22>: mov %rsi,%rbx
0x000000000000400d9 <+25>: cmp $0x2,%edi
0x000000000000400dd9 <+28>: jne 0x400e10 <main+83>
0x000000000000400db <+30>: mov 0xb(%rsi),%rdi
0x000000000000400df <+34>: mov $0x402854,%esi
0x000000000000400d4 <+39>: callq 0x400c50 <open@plt>
0x000000000000400d9 <+44>: mov %rax,0x2039b0(%ip) # 0x6047a0 <infile>
0x000000000000400df0 <+51>: test %rax,%rax
0x000000000000400df3 <+54>: jne 0x400e2c <main+111>
0x000000000000400df5 <+56>: mov 0xb(%rbx),%rdx
0x000000000000400df9 <+60>: mov (%rbx),%rsi
0x000000000000400dfc <+63>: mov $0x402350,%edi
0x000000000000400e01 <+68>: callq 0x400b60 <printf@plt>
0x000000000000400e06 <+73>: mov $0x8,%edi
0x000000000000400eb9 <+78>: callq 0x400c80 <exit@plt>
0x000000000000400e10 <+83>: mov (%rsi),%rsi
0x000000000000400e13 <+86>: mov $0x40236d,%edi
0x000000000000400e18 <+91>: mov $0x0,%eax
0x000000000000400e1d <+96>: callq 0x400b60 <printf@plt>
0x000000000000400e22 <+101>: mov $0x8,%edi
0x000000000000400e27 <+106>: callq 0x400c80 <exit@plt>
0x000000000000400e2c <+111>: callq 0x401357 <initialize_bomb>
0x000000000000400e31 <+116>: mov $0x4023d8,%edi
0x000000000000400e36 <+121>: callq 0x400b40 <puts@plt>
0x000000000000400e3b <+126>: mov $0x402418,%edi
0x000000000000400e40 <+131>: callq 0x400b40 <puts@plt>
0x000000000000400e45 <+136>: callq 0x4015cc <read_line>
0x000000000000400e4a <+141>: mov %rax,%rdi
0x000000000000400e4d <+144>: callq 0x400ef0 <phase_1>
0x000000000000400e52 <+149>: callq 0x4016f2 <phase_defused>
0x000000000000400e57 <+154>: mov $0x402448,%edi
0x000000000000400e5c <+159>: callq 0x400b40 <puts@plt>
0x000000000000400e61 <+164>: callq 0x4015cc <read_line>
0x000000000000400e66 <+169>: mov %rax,%rdi
0x000000000000400e69 <+172>: callq 0x400f0c <phase_2>
0x000000000000400e66 <+177>: callq 0x4016f2 <phase_defused>
0x000000000000400e73 <+182>: mov $0x402387,%edi
0x000000000000400e78 <+187>: callq 0x400b40 <puts@plt>
0x000000000000400e7d <+192>: callq 0x4015cc <read_line>
0x000000000000400e82 <+197>: mov %rax,%rdi
0x000000000000400e85 <+200>: callq 0x400f53 <phase_3>
0x000000000000400e8a <+205>: callq 0x4016f2 <phase_defused>
0x000000000000400e8f <+210>: mov $0x4023a5,%edi
0x000000000000400e94 <+215>: callq 0x400b40 <puts@plt>
0x000000000000400e99 <+220>: callq 0x4015cc <read_line>
0x000000000000400e9e <+225>: mov %rax,%rdi
0x000000000000400ea1 <+228>: callq 0x40101c <phase_4>
0x000000000000400eab <+233>: callq 0x4016f2 <phase_defused>
0x000000000000400eb <+238>: mov $0x402478,%edi
0x000000000000400eb0 <+243>: callq 0x400b40 <puts@plt>
0x000000000000400eb5 <+248>: callq 0x4015cc <read_line>
0x000000000000400eba <+253>: mov %rax,%rdi
0x000000000000400ebd <+256>: callq 0x401073 <phase_5>
0x000000000000400ec2 <+261>: callq 0x4016f2 <phase_defused>
0x000000000000400e7 <+266>: mov $0x4023b4,%edi
0x000000000000400ecc <+271>: callq 0x400b40 <puts@plt>
0x000000000000400ed1 <+276>: callq 0x4015cc <read_line>
0x000000000000400ed6 <+281>: mov %rax,%rdi
0x000000000000400ed9 <+284>: callq 0x4010e0 <phase_6>
0x000000000000400de <+289>: callq 0x4016f2 <phase_defused>
0x000000000000400e3 <+294>: mov $0x0,%eax
0x000000000000400e8 <+299>: pop %rbx
0x000000000000400e9 <+300>: retq
End of assembler dump.
```

phase 1

(gdb) x/s 0x4024a0
0x4024a0: "We have to stand with our North Korean allies."

(gdb) disas phase_1

Dump of assembler code for function phase_1:

```
=> 0x0000000000400ef0 <+0>:    sub    $0x8,%rsp
 0x0000000000400ef4 <+4>:    mov    $0x4024a0,%esi
 0x0000000000400ef9 <+9>:    callq  0x4012ee <strings_not_equal>
 0x0000000000400efe <+14>:   test   %eax,%eax
 0x0000000000400f00 <+16>:   je     0x400f07 <phase_1+23>
 0x0000000000400f02 <+18>:   callq  0x401554 <explode_bomb>
 0x0000000000400f07 <+23>:   add    $0x8,%rsp
 0x0000000000400f0b <+27>:   retq
End of assembler dump.
```

```
%esi=0x4024a0 23 ←
<strings_not_equal> 푸시
%eax and %eax 같아
같아면 jump <strings_not_equal>+98
아니면 Bomb!
    같아 %eax=0x0
    아니면 %eax=0x1
    그 경우 zf=1이면 zf=0
```

0x4024a0 = answer

(gdb) disas strings_not_equal

Dump of assembler code for function strings_not_equal:

```
0x00000000004012ee <+0>: push   %r12
 0x00000000004012f0 <+2>: push   %rbp
 0x00000000004012f1 <+3>: push   %rbx
 0x00000000004012f2 <+4>: mov    %rdi,%rbx
 0x00000000004012f5 <+7>: mov    %rsi,%rbp
 0x00000000004012f8 <+10>: callq  0x4012d1 <string_length>
 0x00000000004012fd <+15>: mov    %eax,%r12d
 0x0000000000401300 <+18>: mov    %rbp,%rdi
 0x0000000000401303 <+21>: callq  0x4012d1 <string_length>
 0x0000000000401308 <+26>: mov    $0x1,%edx
 0x000000000040130d <+31>: cmp    %eax,%r12d
 0x0000000000401310 <+34>: jne    0x401350 <strings_not_equal+98>
 0x0000000000401312 <+36>: movzb1 (%rbx),%eax
 0x0000000000401315 <+39>: test   %al,%al
 0x0000000000401317 <+41>: je     0x40133d <strings_not_equal+79>
 0x0000000000401319 <+43>: cmp    %o(%rbp),%al
 0x000000000040131c <+46>: je     0x401327 <strings_not_equal+57>
 0x000000000040131e <+48>: xchg   %ax,%ax
 0x0000000000401320 <+50>: jmp    0x401344 <strings_not_equal+86>
 0x0000000000401322 <+52>: cmp    %o(%rbp),%al
 0x0000000000401325 <+55>: jne    0x40134b <strings_not_equal+93>
 0x0000000000401327 <+57>: add    $0x1,%rbx
 0x000000000040132b <+61>: add    $0x1,%rbp
 0x000000000040132f <+65>: movzb1 (%rbx),%eax
 0x0000000000401332 <+68>: test   %al,%al
 0x0000000000401334 <+70>: jne    0x401322 <strings_not_equal+52>
 0x0000000000401336 <+72>: mov    $0x0,%edx
 0x000000000040133b <+77>: jmp    0x401350 <strings_not_equal+98>
 0x000000000040133d <+79>: mov    $0x0,%edx
 0x0000000000401342 <+84>: jmp    0x401350 <strings_not_equal+98>
 0x0000000000401344 <+86>: mov    $0x1,%edx
 0x0000000000401349 <+91>: jmp    0x401350 <strings_not_equal+98>
 0x000000000040134b <+93>: mov    $0x1,%edx
 0x0000000000401350 <+98>: mov    %edx,%eax ←
 0x0000000000401352 <+100>: pop    %rbx
 0x0000000000401353 <+101>: pop    %rbp
 0x0000000000401354 <+102>: pop    %r12
 0x0000000000401356 <+104>: retq
```

```
%rbx에 unk 문자열 string(%rdi) 저장
%rbp에 answer(%rsi) 23 ←
<string_length> 문자열 + %rdi를 전자로 짜여
%r12d에 unk 문자열 (%eax) 저장
%rdi에 %rbx로 저장
    ↳ answer
<string_length> 문자열 + %rdi를 전자로 짜여
%eax에 unk 문자열 저장되는 위치에 현재 저장된 값 zero extension되어 23 ←
    ↳ answer

    ↳ 문자열 문자열 %rbx 값 비교
    ↳ 문자열 문자열 jump

%eax와 %rbx 값이 동일한지 비교하는 위치에 현재 저장된 값 zero extension되어 23 ←
    ↳ answer
```

즉 문자열 문자열 answer
 ↳ 문자열 문자열 %rbx 값 비교
 ↳ 문자열 문자열 answer

End of assembler dump.

Phase 1

<phase_1> 함수에서는 %esi 레지스터에 0x4024a0 을 저장하고, <strings_not_equal> 함수를 호출하여 내가 입력한 string 과 0x4024a0 에 저장된 string 을 비교한다. 이때, <strings_not_equal> 함수는 string 이 서로 같으면 0x0 을, 다르면 0x1 을 반환한다. 반환값은 %eax 레지스터에 저장되며, 함수 종료 후 다시 <phase_1> 함수로 돌아와 test 인스트럭션에 의해 %eax and %eax 연산이 수행된다. test 인스트럭션의 결과 ZF(Zero Flag)가 1 로 설정되면 je 인스트럭션에 의해 <phase_1+23>으로 jump 하여 <explode_bomb> 함수를 호출하지 않고 <phase_1> 함수를 종료할 수 있다.

즉, phase_1 을 해결하기 위해서는 내가 입력한 string 과 %esi 레지스터에 저장된 0x4024a0 에 저장된 string 이 같아야 한다. x/s 0x4024a0 을 통해 string 을 확인하면 "We have to stand with our North Korean allies."인 것을 알 수 있다.

따라서 정답은 "We have to stand with our North Korean allies."

phase 2

(gdb) disas phase_2

Dump of assembler code for function phase_2:

```

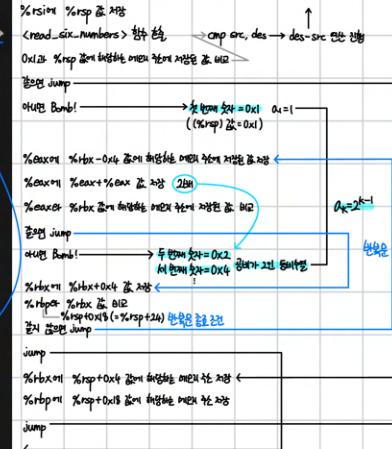
0x0000000000400f0c <+0>: push %rbp
0x0000000000400fed <+1>: push %rbx
0x0000000000400f0e <+2>: sub $0x28,%rsp
0x0000000000400f12 <+6>: mov %rsp,%rsi
0x0000000000400f15 <+9>: callq 0x40158a <read_six_numbers>
0x0000000000400f1a <+14>: cmpl $0x1,(%rsp)
0x0000000000400f1e <+18>: je 0x400f40 <phase_2+52>
0x0000000000400f20 <+20>: callq 0x401554 <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 0x401554 <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.

ZF(Zero Flag): 짐스터 값이 0이면 ZF=1
 CF(Carry Flag): 부호를 처리하는 때에 발생하는 CF=1
 OF(Overflow Flag): 부호를 처리하는 때에 발생하는 OF=1
 SF(Sign Flag): 짐스터 값에 따라 SF=1

des-src<0	0	ZF
des-src>0	0	CF
des-src=0	1	OF



Memory
%rsp 0x1 (=1)
%rsp+4 0x2 (=2)
%rsp+8 0x4 (=4)
%rsp+12 0x8 (=8)
%rsp+16 0x10 (=16)
%rsp+20 0x20 (=32)

(gdb) disas read_six_numbers

Dump of assembler code for function read_six_numbers:

```

0x000000000040158a <+0>: sub $0x18,%rsp
0x000000000040158e <+4>: mov %rsi,%rdx
0x0000000000401591 <+7>: lea 0x4(%rsi),%rcx first argument
0x0000000000401595 <+11>: lea 0x14(%rsi),%rax second argument
0x0000000000401599 <+15>: mov %rax,0x8(%rsp) third argument
0x000000000040159e <+20>: lea 0x10(%rsi),%rax fifth argument
0x00000000004015a2 <+24>: mov %rax,(%rsp)
0x00000000004015a6 <+28>: lea 0xc(%rsi),%rcx fourth argument
0x00000000004015aa <+32>: lea 0x8(%rsi),%r8 third argument
0x00000000004015ae <+36>: mov $0x4027c1,%esi
0x00000000004015b3 <+41>: mov $0x0,%eax
0x00000000004015b8 <+46>: callq 0x400c30 <_isoc99_sscanf@plt>
0x00000000004015bd <+51>: cmp $0x5,%eax
0x00000000004015c0 <+54>: jg jump_if_faster((%rsp)+0x20+1)
0x00000000004015c2 <+56>: callq 0x401554 <explode_bomb>
0x00000000004015c7 <+61>: add $0x18,%rsp
0x00000000004015cb <+65>: retq

```

End of assembler dump.

↳ 짐스터 %rsp 값 0, %rsi sub \$0x18, %rsp 값을 System에
 %rcx(%rsi)(%rsp+0x18) 값 저장
 %rcx(%rsi+0x4)(%rsp+0x18+0x4) 값 저장되는 순간 짐스터 %rsi
 %rcx(%rsi+0x8)(%rsp+0x18+0x8) 값 저장되는 순간 짐스터 %rcx
 %rcx(%rsi+0x12)(%rsp+0x18+0x12) 값 저장되는 순간 짐스터 %rax
 %rcx(%rsi+0x16)(%rsp+0x18+0x16) 값 저장되는 순간 짐스터 %rcx
 %rcx(%rsi+0x20)(%rsp+0x18+0x20) 값 저장되는 순간 짐스터 %rcx
 %rcx(%rsi+0x24)(%rsp+0x18+0x24) 값 저장되는 순간 짐스터 %rcx
 %rcx(%rsi+0x28)(%rsp+0x18+0x28) 값 저장되는 순간 짐스터 %rcx
 %rcx(%rsi+0x32)(%rsp+0x18+0x32) 값 저장되는 순간 짐스터 %rcx
 %esi(%rsi+0x4027c1) 저장 : "%d %d %d %d %d %d"

%eax -0x5 > 0 실행
 실행 Bomb! 6개 블록 실행

Phase 2

<phase_2> 함수에서는 <read_six_numbers> 함수를 호출한다. 함수 이름에서도 알 수 있지만, cmp \$0x5, %eax 와 jg 인스트럭션을 통해서도 6 개의 숫자가 입력되어야 함을 알 수 있다. 입력한 6 개의 숫자는 %rsp 레지스터 값에 해당하는 메모리 주소부터 4 바이트씩 순서대로 저장된다.

cmpl \$0x1, (%rsp)에서 ZF가 1로 설정되어야 je 인스트럭션에 의해 <explode_bomb> 함수를 호출하지 않고 jump 할 수 있다. 따라서 %rsp 레지스터 값에 해당하는 메모리 주소에 저장된 값은 0x1임을 알 수 있다. 즉, 첫 번째 숫자는 "1"이다.

그 이후에 %rbx 레지스터에 %rsp 레지스터에서 4 바이트만큼 떨어진 주소를 저장한다. 또한, %eax 레지스터에는 앞서 구한 값을 저장하고, add %eax, %eax를 통해 2 배의 값을 저장한다. %rbx 레지스터 값에 해당하는 메모리 주소에 저장된 값과 %eax 레지스터에 저장된 값을 비교하여 ZF가 1로 설정되면 je 인스트럭션에 의해 <explode_bomb> 함수를 호출하지 않고 jump 할 수 있다. 따라서 %rsp + 0x4에 해당하는 메모리 주소에 저장된 값은 첫 번째 숫자의 2 배인 0x2임을 알 수 있다. 즉, 두 번째 숫자는 "2"이다.

%rbx 레지스터에 저장된 값이 %rbp 레지스터 값(= %rsp + 0x18)과 같아질 때까지, 즉 6 개의 숫자를 모두 확인할 때까지 위 과정을 반복한다. 따라서 6 개의 숫자는 공비가 2인 등비수열임을 알 수 있다.

따라서 정답은 "1 2 4 8 16 32"

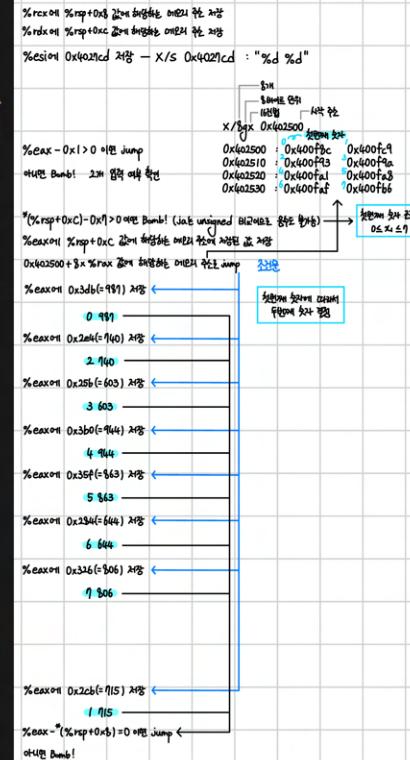
phase 3

```

(gdb) disas phase_3
Dump of assembler code for function phase_3:
0x000000000040053 <+0>:    sub    $0x18,%rsp
0x000000000040057 <+4>:    lea    0x8(%rsp),%rcx
0x00000000004005c <+9>:    lea    0xc(%rsp),%rdx second argument
0x000000000040061 <+14>:   mov    $0x4027cd,%esi first argument
0x000000000040066 <+19>:   mov    $0x0,%eax
0x00000000004006b <+24>:   callq 0x400c30 <_isoc99_sscanf@plt>
0x000000000040070 <+29>:   cmp    $0x1,%eax
0x000000000040075 <+32>:   jg    0x400f7a <phase_3+39>
0x000000000040075 <+34>:   callq 0x401554 <explode_bomb>
0x00000000004007a <+39>:   cmpl  $0x7,0xc(%rsp)
0x00000000004007f <+44>:   ja    0x400fb9 <phase_3+106>
0x000000000040081 <+46>:   mov    $0xc(%rsp),%eax
0x000000000040085 <+50>:   jmpq  *0x402500(,%rax,8)
0x00000000004008c <+57>:   mov    $0x3db,%eax
0x000000000040091 <+62>:   1 jmp  0x400fce <phase_3+123>
0x000000000040093 <+64>:   2 mov    $0x2e4,%eax
0x000000000040098 <+69>:   3 jmp  0x400fce <phase_3+123>
0x00000000004009a <+71>:   4 mov    $0x25b,%eax
0x00000000004009f <+76>:   5 jmp  0x400fce <phase_3+123>
0x0000000000400a1 <+78>:   6 mov    $0x3b0,%eax
0x0000000000400a6 <+83>:   7 jmp  0x400fce <phase_3+123>
0x0000000000400f8 <+85>:   8 mov    $0x35f,%eax
0x0000000000400fa <+90>:   9 jmp  0x400fce <phase_3+123>
0x0000000000400f9 <+92>:   10 mov   $0x284,%eax
0x0000000000400fb4 <+97>:  11 jmp  0x400fce <phase_3+123>
0x0000000000400fb6 <+99>:  12 mov   $0x326,%eax
0x0000000000400fb <+104>: 13 jmp  0x400fce <phase_3+123>
0x0000000000400fbd <+106>: 14 callq 0x401554 <explode_bomb>
0x0000000000400fc2 <+111>: 15 mov    $0x0,%eax
0x0000000000400fc7 <+116>: 16 jmp  0x400fce <phase_3+123>
0x0000000000400fc9 <+118>: 17 mov   $0x2cb,%eax
0x0000000000400fce <+123>: 18 cmp    $0x8(%rsp),%eax
0x0000000000400fd2 <+127>: 19 je    0x400fd9 <phase_3+134>
0x0000000000400fd4 <+129>: 20 callq 0x401554 <explode_bomb>
0x0000000000400fd9 <+134>: 21 add    $0x18,%rsp
0x0000000000400ffd <+138>: 22 retq

End of assembler dump.

```



Phase 3

<phase_3> 함수에서는 먼저 %rdx 레지스터에 첫 번째 argument 값을, %rcx 레지스터에 두 번째 argument 값을 저장한다. 이는 x/s 0x4027cd 를 통해서도 두 개의 정수를 읽는다는 것을 알 수 있으며, cmp \$0x1, %eax 와 jg 인스트럭션을 통해서도 2 개의 숫자가 입력되지 않으면 <explode_bomb> 함수를 호출한다는 것을 알 수 있다.

cmpl \$0x7, 0xc(%rsp)와 ja 인스트럭션을 통해서 첫 번째 숫자의 조건이 $0 \leq x1 \leq 7$ 임을 알 수 있는데, 이때 ja 인스트럭션은 unsigned 비교이므로 음수가 불가능하다.

%eax 레지스터에 0xc(%rsp), 즉 %rsp + 0xc 에 해당하는 메모리 주소에 저장된 값을 저장한다. 그리고 jmpq *0x402500(%rax,8)을 통해 0x402500 으로부터 8 X %rax 만큼 떨어진 주소로 jump 한다.

x/8gx 0x402500 을 통해 첫 번째 숫자가 0 이면 0x400f8c 로, 1 이면 0x400fc9 로, 2 이면 0x400f93 으로, 3 이면 0x400f9a 로, 4 이면 0x400fa1 로, 5 이면 0x400fa8 로, 6 이면 0x400faf 로, 7 이면 0x400fb6 으로 jump 한다는 것을 알 수 있다.

첫 번째 숫자에 따라서 알맞은 주소로 jump 한 후, %eax 레지스터에 특정 값을 저장한다. 그 후, 0x8(%rsp), 즉 내가 입력한 두 번째 값과 %eax 레지스터에 저장된 값을 비교하여 ZF 가 1 로 설정되면 <explode_bomb> 함수를 호출하지 않고 <phase_3> 함수를 종료할 수 있다.

따라서 정답은 "0 987", "1 715", "2 740", "3 603", "4 944", "5 863", "6 644", "7 806"

phase 4

(gdb) disas phase_4

Dump of assembler code for function phase_4:

```

0x000000000040101c <+0>:    sub    $0x18,%rsp
0x0000000000401020 <+4>:    lea    0x8(%rsp),%rcx
0x0000000000401025 <+9>:    lea    0xc(%rsp),%rdx
0x000000000040102a <+14>:   mov    $0x4027cd,%esi
0x000000000040102f <+19>:   mov    $0x0,%eax
0x0000000000401034 <+24>:   callq 0x400c30 <_isoc99_sscanf@plt>
0x0000000000401039 <+29>:   cmp    $0x2,%eax
0x000000000040103c <+32>:   jne    0x401045 <phase_4+41>
0x000000000040103e <+34>:   cmpl   $0xe,0xc(%rsp)
0x0000000000401043 <+39>:   jbe    0x40104a <phase_4+46>
0x0000000000401045 <+41>:   callq 0x401554 <explode_bomb>
0x000000000040104a <+46>:   mov    $0xe,%edx
0x000000000040104f <+51>:   mov    $0x0,%esi
0x0000000000401054 <+56>:   mov    0xc(%rsp),%edi
0x0000000000401058 <+60>:   callq 0x400fde <func4>
0x000000000040105d <+65>:   cmp    $0x2,%eax
0x0000000000401060 <+68>:   jne    0x401069 <phase_4+77>
0x0000000000401062 <+70>:   cmpl   $0x2,0x8(%rsp)
0x0000000000401067 <+75>:   je     0x40106e <phase_4+82>
0x0000000000401069 <+77>:   callq 0x401554 <explode_bomb>
0x000000000040106e <+82>:   add    $0x18,%rsp
0x0000000000401072 <+86>:   retq

```

End of assembler dump.

%rcx에 %rsp+0x8 를 넣어 주면 second argument
%rdx에 %rsp+0xc 를 넣어 주면 first argument
%esi에 0x4027cd 저장 - X/S 0x4027cd : "%d %d"

%eax-0x2=0x4 대체 Bomb! 2진수 출력 예상

"(%rsp+0xc)-0x2=0x4 jump
대체Bomb!" → 대체 %eax-0x2
%eax에 0xe 저장
%esi에 0x0 저장
%rdx에 %rcp+0xc 를 넣어 주면 second argument 값 저장
<func4> 출력

%eax-0x2=0x4 대체 Bomb! → 대체 %eax-0x2
대체Bomb!" → 대체 %eax-0x2
대체 %eax-0x2=0x4 jump

수행된 블록	다음 블록
0	0
1	0
2	4
3	0
4	2
5	2
6	6
7	0
8	/
9	/
10	5
11	/
12	3
13	3
14	7

%eax이 %eax-0x2
%eax- %esi (= %edx- %esi) & 255 → %eax- %esi → 0x1
%eax- %esi (= %edx- %esi) & 255 → %eax- %esi → 0x1
%eax > 0x1f (high) (= %ecx > 148) & 255
%eaxon %eax+ %esi (= %edx- %esi+ %esi) & 255
%eaxon %eax > 1 (middle) (= %edx- %esi+ %esi) & 255
%eaxon %eax+ %esi (= (%edx+ %esi+ %esi)/2) & 255 대체로 대체 %eax & 255
→ 대체 %eax
%eax- %esi & 255 대체 %eax- %esi & 255
<func4> 출력 2진수
%eaxon %eax+ %eax & 255

%eaxon 0x0 & 255 ←
%eax- %esi & 255 대체 %eax- %esi & 255
<func4> 출력
%eaxon %eax+ %eax & 255 대체 %eax & 255
<func4> 출력
%eaxon %eax+ %eax & 255 대체 %eax & 255 ←

(gdb) disas func4

Dump of assembler code for function func4:

```

0x0000000000400fde <+0>:    sub    $0x8,%rsp
0x0000000000400fe2 <+4>:    mov    %edx,%eax
0x0000000000400fe4 <+6>:    sub    %esi,%eax
0x0000000000400fe6 <+8>:    mov    %eax,%ecx
0x0000000000400fe8 <+10>:   shr    $0x1f,%ecx
0x0000000000400feb <+13>:   add    %ecx,%eax
0x0000000000400fed <+15>:   sar    %eax
0x0000000000400fef <+17>:   lea    (%rax,%rsi,1),%ecx
0x0000000000400ff2 <+20>:   cmp    %edi,%ecx
0x0000000000400ff4 <+22>:   jle    0x401002 <func4+36>
0x0000000000400ff6 <+24>:   lea    0x1(%rcx,%rsi),%edx
0x0000000000400ff9 <+27>:   callq 0x400fde <func4>
0x0000000000400ffe <+32>:   add    %eax,%eax
0x0000000000401000 <+34>:   jmp    0x401017 <func4+57>
0x0000000000401002 <+36>:   mov    $0x0,%eax
0x0000000000401007 <+41>:   cmp    %edi,%ecx
0x0000000000401009 <+43>:   jge    0x401017 <func4+57>
0x000000000040100b <+45>:   lea    0x1(%rcx,%rsi),%esi
0x000000000040100e <+48>:   callq 0x400fde <func4>
0x0000000000401013 <+53>:   lea    0x1(%rax,%rax,1),%eax
0x0000000000401017 <+57>:   add    $0x8,%rsp
0x000000000040101b <+61>:   retq

```

End of assembler dump.

```

    ↓ Return ECX
    ↓ Return ECX == EDX
    ↓ Return ECX == EDX
int funct(int edi, int esi, int edx)
{
    int exx = edx;
    exx -= esi;

    int exx = exx;
    exx += (edx >= esi) ? 0 : 1 // == (exx >= 0x1f);

    exx += exx;
    exx /= 0x2; // == (exx >= 0x1);
    exx = exx+esi;

    if(exx-edi > 0)
    {
        exx = exx-0x1;
        exx = funct(edi,esi,edx);
        exx += exx;
    }

    return exx; // ECX == EDX == ECX == ECX
}

ECX = 0x0;

if(exx-edi<0)
{
    esi = exx+0x1;
    exx = funct(edi,esi,edx);
    exx = exx+exx;
}

return exx; // ECX == EDX == ECX == ECX
}

```

Return ECX	Return ECX
0	0
1	0
2	4
3	0
4	2
5	2
6	6
7	0
8	1
9	1
10	5
11	1
12	3
13	3
14	7

Phase 4

<phase_4> 함수에서는 먼저 %rdx 레지스터에 첫 번째 argument 값을, %rcx 레지스터에 두 번째 argument 값을 저장한다. 이는 x/s 0x4027cd 를 통해서도 두 개의 정수를 읽는다는 것을 알 수 있으며, cmp \$0x2, %eax 와 jne 인스트럭션을 통해서도 2 개의 숫자가 입력되지 않으면 <explode_bomb> 함수를 호출한다는 것을 알 수 있다.

cmpl \$0xe, 0xc(%rsp)와 jbe 인스트럭션을 통해서 첫 번째 숫자의 조건이 $0 \leq x1 \leq 14$ 임을 알 수 있는데, 이때 jbe 인스트럭션은 unsigned 비교이므로 음수가 불가능하다.

%edi 레지스터에는 %rsp + 0xc 에 해당하는 메모리 주소에 저장된 값, 즉 첫 번째 숫자를 저장하고, %edx 레지스터에는 0xe 를, %esi 레지스터에는 0x0 을 저장한다.

그 이후 <func4> 함수를 호출하고, 함수의 결과가 %eax 레지스터에 저장되며, cmpl \$0x2, 0x8(%rsp)의 결과로 ZF 가 1 로 설정되어야 je 인스트럭션에 의해 <explode_bomb> 함수를 호출하지 않고 <phase_4>를 종료할 수 있다. 즉, 두 번째 숫자는 0x2 이어야 하고, <func4> 함수가 2 를 반환하도록 첫 번째 숫자를 입력해 주어야 한다.

<func4> 함수를 C 코드로 역 번역하면, 위와 같은 함수를 얻을 수 있다. 이를 통해 (첫 번째 숫자, 두 번째 숫자)의 관계로, (0, 0), (1, 0), (2, 4), (3, 0), (4, 2), (5, 2), (6, 6), (7, 0), (8, 1), (9, 1), (10, 5), (11, 1), (12, 3), (13, 3), (14, 7)을 얻을 수 있다.

따라서 정답은 "4 2", "5 2"

phase 5

(gdb) disas phase_5

Dump of assembler code for function phase_5:

```

0x0000000000401073 <+0>:    sub    $0x18,%rsp
0x0000000000401077 <+4>:    lea    0x8(%rsp),%rcx
0x000000000040107c <+9>:    lea    0xc(%rsp),%rdx
0x0000000000401081 <+14>:   mov    $0x4027cd,%esi
0x0000000000401086 <+19>:   mov    $0x0,%eax
0x0000000000401088 <+24>:   callq 0x400c30 <_isoc99_sscanf@plt>
0x0000000000401090 <+29>:   cmp    $0x1,%eax
0x0000000000401093 <+32>:   jg    0x40109a <phase_5+39>
0x0000000000401095 <+34>:   callq 0x401554 <explode_bomb>
0x000000000040109a <+39>:   mov    0xc(%rsp),%eax
0x000000000040109e <+43>:   and    $0xf,%eax
0x00000000004010a1 <+46>:   mov    %eax,0xc(%rsp)
0x00000000004010a5 <+50>:   cmp    $0xf,%eax
0x00000000004010a8 <+53>:   je    0x4010d6 <phase_5+99>
0x00000000004010aa <+55>:   mov    $0x0,%ecx
0x00000000004010af <+60>:   mov    $0x0,%edx
0x00000000004010b4 <+65>:   add    $0x1,%edx
0x00000000004010b7 <+68>:   cltq
0x00000000004010b9 <+70>:   mov    0x402540(%rax,4),%eax
0x00000000004010c0 <+77>:   add    %eax,%ecx
0x00000000004010c2 <+79>:   cmp    $0xf,%eax
0x00000000004010c5 <+82>:   jne    0x4010b4 <phase_5+65>
0x00000000004010c7 <+84>:   mov    %eax,0xc(%rsp)
0x00000000004010cb <+88>:   cmp    $0xf,%edx
0x00000000004010ce <+91>:   jne    0x4010d6 <phase_5+99>
0x00000000004010d0 <+93>:   cmp    $0x8(%rsp),%ecx
0x00000000004010d4 <+97>:   je    0x4010db <phase_5+104>
0x00000000004010d6 <+99>:   callq 0x401554 <explode_bomb>
0x00000000004010db <+104>:  add    $0x18,%rsp
0x00000000004010df <+108>:  retq

```

End of assembler dump.

%rcx=0x0 %rsp+0x8 끝에 배열을 만들 때는 %rdx=0x0 %rsp+0x4 끝에 배열을 만들 때는 %esi=0x4027cd 저장 - X-S 0x4027cd : "%d %d"

%eax->0 = 0 이면 jump

아직은 Bomb! 2회 끝에 때를 확인

%eax=0x0 %rsp+0xc 끝에 배열을 만들 때에는 배열을 만들 때는 %rdx=0x0 %rsp+0x4 끝에 배열을 만들 때는 %esi=0x4027cd 저장

%eax=0x0 and 0xf 을 저장

%rsp+0xc 뒤에 대해서 맨마지막에 %eax 을 저장

%eax=0xf = 0 이면 Bomb! —> %eax=0x0

%ecx=0x0 저장

%edx=0x0 저장

for(edx=0x1; edx<0x1; edx++)

%eax=0x0 %ecx=0x0 %edx=0x1 저장

%eax=0x0 %ecx=0x0 %edx=0x2 저장

%eax=0x0 %ecx=0x0 %edx=0x3 저장

%eax=0x0 %ecx=0x0 %edx=0x4 저장

%eax=0x0 %ecx=0x0 %edx=0x5 저장

%eax=0xf = 0 이면 Bomb! —> %eax=0x0

%ecx=0x0 %edx=0x0 저장

%eax=0x0 %ecx=0x1 %edx=0x0 저장

%eax=0x0 %ecx=0x1 %edx=0x1 저장

%eax=0x0 %ecx=0x1 %edx=0x2 저장

%eax=0x0 %ecx=0x1 %edx=0x3 저장

%eax=0x0 %ecx=0x1 %edx=0x4 저장

%eax=0x0 %ecx=0x1 %edx=0x5 저장

array [%eax]	
[0]	10
[1]	2
[2]	14
[3]	1
[4]	8
[5]	12
[6]	5
[7]	11
[8]	0
[9]	4
[10]	1
[11]	3
[12]	3
[13]	9
[14]	6
[15]	5

X-locus 0x00000040102540 ←

0x00000040102540 <array 3161> : 10 2 14 1

0x00000040102550 <array 3161+16> : 8 12 15 11

0x00000040102560 <array 3161+32> : 0 4 1 13

0x00000040102570 <array 3161+48> : 3 9 6 5

array_j [0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15]

%eax = array_j[%eax] + %eax*4 (5+ 8+14+ 1+ 2+ 1+ 0+ 1+ 3+ 3+ 9+ 6+ 5)

5 ← 6 ← 14 ← 2 ← 1 ← 0 ← 0 ← 8 ← 4 ← 9 ← 3 ← 11 ← 7 ← 3 ← 12 ← 5
a[5] a[4] ...

$\sum_{i=1}^5 \text{array}[i] = 5 = 1/5$
평균 5 = 15

Phase 5

<phase_5> 함수에서는 먼저 %rdx 레지스터에 첫 번째 argument 값을, %rcx 레지스터에 두 번째 argument 값을 저장한다. 이는 x/s 0x4027cd 를 통해서도 두 개의 정수를 읽는다는 것을 알 수 있으며, cmp \$0x1, %eax 와 jg 인스트럭션을 통해서도 2 개의 숫자가 입력되지 않으면 <explode_bomb> 함수를 호출한다는 것을 알 수 있다.

%eax 레지스터에 %rsp + 0xc 에 해당하는 메모리 주소에 저장된 값, 즉 첫 번째 숫자를 저장하고, 0xf 와 and 연산을 수행한다. 이를 %rsp + 0xc 에 해당하는 메모리 주소에 저장하고, cmp \$0xf, %eax 의 결과로 ZF 가 1 로 설정되어야 je 인스트럭션에 의해 <explode_bomb> 함수를 호출하지 않는다는 것을 알 수 있다. 즉, 첫 번째 숫자의 조건은 $x1 \neq 0xf$ 이다.

그 이후에는 %ecx 레지스터와 %edx 레지스터에 0x0 을 저장하고, %edx 레지스터 값에 1 을 더해준 후, %eax 레지스터에는 $0x402540 + 4 \times \%rax$ 에 해당하는 메모리 주소에 저장된 값을 저장한다. 그 값을 %ecx 레지스터 값에 더해주고, cmp \$0xf, %eax 의 결과로 ZF 가 0 이면 jne 인스트럭션에 의해 다시 위로 jump 하여 %edx 레지스터 값에 1 을 더해주는 과정부터 다시 반복한다. 이는 for 문으로도 해석할 수 있다.

최종적으로 위 과정이 끝난 후 cmp \$0xf, %edx 의 결과로 ZF 가 0 이 아니면 jne 인스트럭션에 의해 <explode_bomb> 함수를 호출하므로 15 번 반복이 되어야 한다는 것을 알 수 있고, cmp \$0x8(%rsp), %ecx 의 결과로 ZF 가 1 이 아니면 je 인스트럭션에 의해 <explode_bomb> 함수를 호출하므로 두 번째 숫자와 %ecx 레지스터 값이 같아야 한다는 것을 알 수 있다.

x/16uw 0x402540 을 통해 4 바이트 단위로 정수 값이 저장된 것을 보아 배열임을 알 수 있고, 순서대로 10, 2, 14, 7, 8, 12, 15, 11, 0, 4, 1, 13, 3, 9, 6, 5 가 저장되어 있다.

위의 조건들을 종합하면 총 15 번 반복되어야 하며, 반복이 종료되는 조건은 값이 15 일 때이다. 즉, 15 번째 반복에서 15 가 나와야 한다. 만약 이전 값이 5 였다면, 다음 값은 array[5] 이므로 15 부터 역 추적하여 15 번째의 값이 첫 번째 숫자임을 알 수 있다. 따라서 $15 \leftarrow 6 \leftarrow 14 \leftarrow 2 \leftarrow 1 \leftarrow 10 \leftarrow 0 \leftarrow 8 \leftarrow 4 \leftarrow 9 \leftarrow 13 \leftarrow 11 \leftarrow 7 \leftarrow 3 \leftarrow 12 \leftarrow 5$ 이므로 첫 번째 숫자는 5이고, 두 번째 숫자는 array[5] 부터 대응되는 숫자들을 모두 더한 값으로, $12 + 3 + 7 + 11 + 13 + 9 + 4 + 8 + 0 + 10 + 1 + 2 + 14 + 6 + 15 = 115$ 이다.

따라서 정답은 "5 115"

phase 6

(gdb) disas phase_6

```
Dump of assembler code for function phase_6:
0x00000000004010e0 <+0>: push %r13
0x00000000004010e2 <+2>: push %r12
0x00000000004010e4 <+4>: push %rbp
0x00000000004010e5 <+5>: push %rbx
0x00000000004010e6 <+6>: sub $0x58,%rsp
0x00000000004010ea <+10>: lea 0x30(%rsp),%rsi
0x00000000004010ef <+15>: callq 0x40158a <read six numbers>
0x00000000004010f4 <+20>: lea 0x30(%rsp),%r13
0x00000000004010f9 <+25>: mov $0x13,%rbp
0x00000000004010ff <+31>: mov %r13,%rbp
0x0000000000401102 <+34>: mov 0x0(%r13),%eax
0x0000000000401106 <+38>: sub $0x1,%eax
0x0000000000401108 <+41>: cmp $0x5,%eax
0x000000000040110c <+44>: jbe 0x401113 <phase_6+51>
0x000000000040110e <+46>: callq 0x401154 <explode_bomb>
0x0000000000401113 <+51>: add $0x1,%r12d
0x0000000000401117 <+55>: cmp $0x6,%r12d
0x000000000040111b <+59>: jne 0x40112a <phase_6+68>
0x000000000040111d <+61>: mov $0x0,%esi
0x0000000000401122 <+66>: jmp 0x401166 <phase_6+134>
0x0000000000401124 <+68>: mov %r12d,%ebx
0x0000000000401127 <+71>: movslq %ebx,%rax
0x000000000040112a <+74>: mov 0x30(%rsp),%rax,4,%eax
0x000000000040112e <+78>: cmp %eax,0x0(%rbp)
0x0000000000401131 <+81>: jne 0x401138 <phase_6+88>
0x0000000000401133 <+83>: callq 0x401154 <explode_bomb>
0x0000000000401138 <+88>: add $0x1,%ebx
0x000000000040113b <+91>: cmp $0x5,%ebx
0x000000000040113e <+94>: jle 0x401127 <phase_6+71>
0x0000000000401140 <+96>: add $0x4,%r13
0x0000000000401144 <+100>: jmp 0x4010ff <phase_6+31>
0x0000000000401146 <+102>: mov 0x8(%rdx),%rdx
0x000000000040114a <+106>: add $0x1,%eax
0x000000000040114d <+109>: cmp %rcx,%eax
0x000000000040114f <+111>: jne 0x401146 <phase_6+102>
0x0000000000401151 <+113>: jmp 0x401158 <phase_6+120>
0x0000000000401153 <+115>: mov $0x042f0,%edx
0x0000000000401158 <+120>: mov %rdx,%rsi,%rsi,2
0x000000000040115c <+124>: add $0x4,%rsi
0x0000000000401160 <+128>: cmp $0x18,%rsi
0x0000000000401164 <+132>: je 0x40117b <phase_6+155>
0x0000000000401166 <+134>: mov 0x30(%rsp,%rsi,1),%ecx
0x000000000040116a <+138>: cmp $0x1,%ecx
0x000000000040116d <+141>: jle 0x401153 <phase_6+115>
0x000000000040116f <+143>: mov $0x1,%eax
0x0000000000401174 <+148>: mov $0x6042f0,%edx
0x0000000000401179 <+153>: jmp 0x401146 <phase_6+102>
0x000000000040117b <+155>: mov (%rsp),%rbx
0x000000000040117f <+159>: lea 0x8(%rsp),%rax
0x0000000000401184 <+164>: lea 0x30(%rsp),%rsi
0x0000000000401189 <+169>: mov %rbx,%rcx
0x000000000040118c <+172>: mov (%rax),%rdx
0x000000000040118f <+175>: mov %rdx,0x8(%rcx)
0x0000000000401193 <+179>: add $0x8,%rax
0x0000000000401197 <+183>: cmp %rsi,%rax
0x000000000040119a <+186>: je 0x4011a1 <phase_6+193>
0x000000000040119c <+188>: mov %rdx,%rcx
0x000000000040119f <+191>: jmp 0x40118c <phase_6+172>
0x00000000004011a1 <+193>: movq $0x0,0x8(%rdx)
0x00000000004011a9 <+201>: mov $0x5,%rbp
0x00000000004011ae <+206>: mov 0x8(%rbx),%rax
0x00000000004011b2 <+210>: mov %rax,%eax
0x00000000004011b4 <+212>: cmp %eax,(%rbx)
0x00000000004011b6 <+214>: jle 0x4011bd <phase_6+221>
0x00000000004011b8 <+216>: callq 0x401154 <explode_bomb>
0x00000000004011bd <+221>: mov 0x8(%rbx),%rbx
0x00000000004011c1 <+225>: sub $0x1,%rbp
0x00000000004011c4 <+228>: jne 0x4011a1 <phase_6+206>
0x00000000004011c6 <+230>: add $0x58,%rsp
0x00000000004011ca <+234>: pop %rbx
0x00000000004011cb <+235>: pop %rbp
0x00000000004011cc <+236>: pop %r12
0x00000000004011cd <+238>: pop %r13
0x00000000004011d0 <+240>: retq
End of assembler dump.
```

%rsi = %r0x30 를 데스팅이션으로 미리정한 지정

%rsi = %r0x30 를 데스팅을 미리정한 지정

%rsi = 0x0 지정

%rsi = 0x0 지정

미리정한 데스팅 / 0 유행지정 <+33>에 위치한 1이 되었을 때는 0을 표시한 지정

%eax = 0x5 미리정한

미리정한 데스팅 / 0 유행지정

Phase 6

<phase_6> 함수에서는 <read_six_numbers> 함수를 호출한다. 함수 이름에서도 알 수 있지만, cmp \$0x5, %eax 와 jg 인스트럭션을 통해서도 6 개의 숫자가 입력되어야 함을 알 수 있다. 입력한 6 개의 숫자는 %r13 레지스터 값에 해당하는 주소(= %rsp + 0x30)부터 4 바이트씩 순서대로 저장된다.

%r12d 레지스터에는 0x0 을 저장하고, %rbp 레지스터와 %eax 레지스터에는 첫 번째 숫자를 저장한 후, %eax 레지스터 값에 1 을 뺀다. cmp \$0x5, %eax 와 jbe 인스트럭션을 통해 첫 번째 숫자의 조건은 $1 \leq x_1 \leq 6$ 임을 알 수 있다. 이때, 1 보다 크거나 같은 이유는 jbe 인스트럭션은 unsigned 비교이므로 0 이 되면 <phase_6+38>에 의해 음수가 되기 때문이다.

위 과정을 마치면 %r12d 레지스터 값에 1 을 더해주고, cmp \$0x6, %r12d 의 결과로 ZF 가 0 이면 jne 인스트럭션에 의해 jump 한다. %ebx 레지스터에 %r12d 레지스터 값을 저장하고, %eax 레지스터에 %rsp + 4 X %rax + 0x30, 즉 두 번째 숫자를 저장한다. cmp %eax, 0x0(%rbp)의 결과로 ZF 가 0 이 아니면 jne 인스트럭션에 의해 <explode_bomb> 함수를 호출하므로 첫 번째 숫자와 두 번째 숫자 값이 달라야 함을 알 수 있다. cmp \$0x5, %ebx 와 jle 인스트럭션을 통해 총 5 번 반복하고, 이를 통해 첫 번째 숫자와 같은 숫자는 없어야 한다는 것을 알 수 있다.

위 과정을 마치면 %r13 레지스터에 %r13 + 0x4 값을 저장하고, 위로 jump 한다.

이 모든 과정을 %r12d 가 0x6 이 될 때까지, 즉 총 6 번 반복하며, 이를 통해 6 개 숫자 모두 1 보다 크거나 같고, 6 보다 작거나 같아야 하며, 서로 다른 숫자임을 알 수 있다.

위 반복문이 종료되면, %ecx 레지스터에 첫 번째 숫자를 저장한다. %edx 레지스터에 0x6042f0 을 저장하고, 첫 번째 숫자 값만큼 0x6042f0 에 0x8 만큼 반복해서 더해주어, %rsp 레지스터에 저장한다. x/24uw 0x6042f0 을 통해 <node1>부터 <node6>이 순서대로 저장되어 있는 것을 알 수 있으며, 위 과정을 모든 순서대로 반복하여 최종적으로 %rsp 레지스터 값에 해당하는 메모리 주소부터 8 바이트 단위로 숫자 값에 해당하는 node 의 주소를 저장한다는 것을 알 수 있다.

모든 node 의 주소를 저장하면, 반복문을 이용하여 각 node 의 주소에 저장된 값을 저장하고, cmp %eax, (%rbx), jle 인스트럭션을 통해 각 node 의 주소에 저장된 값이 오름차순이 되어야 함을 알 수 있다.

마찬가지로 x/24uw 0x6042f0 을 통해 node 에 저장된 값을 알아낼 수 있고, node 에 저장된 값의 대소관계는 *(node 6) < *(node 4) < *(node 2) < *(node 5) < *(node 3) < *(node 1)임을 알 수 있다.

최종적으로 정리하면 입력한 6 개의 숫자는 1 부터 6 까지 중복되지 않는 수이며, 각 숫자에 대응되는 node 의 주소에 저장된 값은 오름차순을 이뤄야 한다.

따라서 정답은 "6 4 2 5 3 1"

phase_defused

```
(gdb) disas phase_defused
Dump of assembler code for function phase_defused:
0x00000000004016f2 <+0>:    sub    $0x68,%rsp
0x00000000004016f6 <+4>:    mov    $0x1,%edi
0x00000000004016fb <+9>:    callq  0x401490 <send_msg>
0x0000000000401700 <+14>:   cmpl   $0x6,0x203095(%rip)          # 0x60479c <num_input_strings>
0x0000000000401707 <+21>:   jne    0x401776 <phase_defused+132>
0x0000000000401709 <+23>:   lea    0x10(%rsp),%r8
0x000000000040170e <+28>:   lea    0x8(%rsp),%rcx
0x0000000000401713 <+33>:   lea    0xc(%rsp),%rdx
0x0000000000401718 <+38>:   mov    $0x402817,%esi
0x000000000040171d <+43>:   mov    $0x6048b0,%edi
0x0000000000401722 <+48>:   mov    $0x0,%eax
0x0000000000401727 <+53>:   callq  0x400c30 <_isoc99_sscanf@plt>
0x000000000040172c <+58>:   cmp    $0x3,%eax
0x000000000040172f <+61>:   jne    0x401762 <phase_defused+112>
0x0000000000401731 <+63>:   mov    $0x402820,%esi
0x0000000000401736 <+68>:   lea    0x10(%rsp),%rdi
0x000000000040173b <+73>:   callq  0x4012ee <strings_not_equal>
0x0000000000401740 <+78>:   test   %eax,%eax
0x0000000000401742 <+80>:   jne    0x401762 <phase_defused+112>
0x0000000000401744 <+82>:   mov    $0x402678,%edi
0x0000000000401749 <+87>:   callq  0x400b40 <puts@plt>
0x000000000040174e <+92>:   mov    $0x4026a0,%edi
0x0000000000401753 <+97>:   callq  0x400b40 <puts@plt>
0x0000000000401758 <+102>:  mov    $0x0,%eax
0x000000000040175d <+107>:  callq  0x40120f <secret_phase>
0x0000000000401762 <+112>:  mov    $0x4026d8,%edi
0x0000000000401767 <+117>:  callq  0x400b40 <puts@plt>
0x000000000040176c <+122>:  mov    $0x402708,%edi
0x0000000000401771 <+127>:  callq  0x400b40 <puts@plt>
0x0000000000401776 <+132>:  add    $0x68,%rsp
0x000000000040177a <+136>:  retq
End of assembler dump.
```

3개의 실행문에 접근 jump
0x401762:

```
(gdb) x/s 0x402817
0x402817:    %ld %ld %%
이 실행문은 2개의 실행문, 즉 3개의 실행문
```

같이 같이 접근 jump
(gdb) x/s 0x6048b0
0x6048b0 <input_strings+240>; "4 2"
이 실행문은 2개의 4, 2 모드 텍스트

```
(gdb) x/s 0x402820
0x402820:    "DrEvil"
이 실행문은 2개의 실행문, 즉 3개의 실행문
phase 4에서 "4 2 DrEvil"을 출력하고 실행문
secret_phase는 실행문 4로.
```

4, 2를 실행한 뒤 phase 4에서 phase 4에서 4, 2로 되어 있음을 알게 되다.

secret_phase

(gdb) disas secret_phase

Dump of assembler code for function secret_phase:

```

0x000000000040120f <+0>:    push    %rbx
0x0000000000401210 <+1>:    callq   0x4015cc <read_line>
0x0000000000401215 <+6>:    mov     $0xa,%edx
0x000000000040121a <+11>:   mov     $0x0,%esi
0x000000000040121f <+16>:   mov     %rax,%rdi
0x0000000000401222 <+19>:   callq   0x400f00 <strtol@plt>
0x0000000000401227 <+24>:   mov     %rax,%rbx
0x000000000040122a <+27>:   lea     -0x1(%rax),%eax
0x000000000040122d <+30>:   cmp     $0x3e8,%eax
0x0000000000401232 <+35>:   jbe    0x401239 <secret_phase+42>
0x0000000000401234 <+37>:   callq   0x401554 <explode_bomb>
0x0000000000401239 <+42>:   mov     %ebx,%esi
0x000000000040123b <+44>:   mov     $0x604110,%edi
0x0000000000401240 <+49>:   callq   0x4011d1 <fun7>
0x0000000000401245 <+54>:   cmp     $0x1,%eax
0x0000000000401248 <+57>:   je     0x40124f <secret_phase+64>
0x000000000040124a <+59>:   callq   0x401554 <explode_bomb>
0x000000000040124f <+64>:   mov     $0x4024d0,%edi
0x0000000000401254 <+69>:   callq   0x400b40 <puts@plt>
0x0000000000401259 <+74>:   callq   0x4016f2 <phase_defused>
0x000000000040125e <+79>:   pop    %rbx
0x000000000040125f <+80>:   retq

```

End of assembler dump.

%rdi=0xa 218
%rsi=0xb 208
%rdx=%rax=208

%rbx=%rax=208
%rdi=%rax=208 를 통해 면접한 후 208

%eax=0x208 50 ==> jump
0x401239 (%rdi) ==> %rax=0x208
%rdi=0x208
%rdx=0x208 (%rdi)=36
<fun7> 끝

%eax=%rdi=0 ==> jump
0x401554 (%rdi) ==> %rax=0
%rdi=0x208
%rdx=0x208 (%rdi)=

(gdb) disas fun7

Dump of assembler code for function fun7:

```

0x00000000004011d1 <+0>:    sub    $0x8,%rsp
0x00000000004011d5 <+4>:    test   %rdi,%rdi
0x00000000004011d8 <+7>:    je    0x401205 <fun7+52>
0x00000000004011da <+9>:    mov    (%rdi),%edx
0x00000000004011dc <+11>:   cmp    %esi,%edx
0x00000000004011de <+13>:   jle    0x4011ed <fun7+28>
0x00000000004011e0 <+15>:   mov    0x8(%rdi),%rdi
0x00000000004011e4 <+19>:   callq  0x4011d1 <fun7>
0x00000000004011e9 <+24>:   add    %eax,%eax
0x00000000004011eb <+26>:   jmp    0x40120a <fun7+57>
0x00000000004011ed <+28>:   mov    $0x0,%eax
0x00000000004011f2 <+33>:   cmp    %esi,%edx
0x00000000004011f4 <+35>:   je     0x40120a <fun7+57>
0x00000000004011f6 <+37>:   mov    0x10(%rdi),%rdi
0x00000000004011fa <+41>:   callq  0x4011d1 <fun7>
0x00000000004011ff <+46>:   lea    0x1(%rax,%rax,1),%eax
0x0000000000401203 <+50>:   jmp    0x40120a <fun7+57>
0x0000000000401205 <+52>:   mov    $0xffffffff,%eax
0x000000000040120a <+57>:   add    $0x8,%rsp
0x000000000040120e <+61>:   retq

```

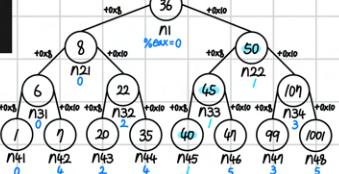
End of assembler dump.

%rdi and %rdi=0 ==> jump
0x401205 (%rdi) ==> %rax=0
%rdi=0 ==> jump
0x401205 (%rdi) ==> %rax=0
%rdi=0 ==> jump

%eax=%esi=0 ==> jump
%rdi=0 ==> jump
0x4011ed (%rdi) ==> %rax=0
<fun7> 끝
%rdi=0 ==> jump
0x4011d1 (%rdi) ==> %rax=0
X2

%eax=0 ==> jump

%eax=0 ==> jump
%rdi=0 ==> jump
0x40120a (%rdi) ==> %rax=0
<fun7> 끝
%rdi=0 ==> jump
0x4011d1 (%rdi) ==> %rax=0
X2+1



Secret Phase

우선, <phase_defused> 함수를 역 어셈블했을 때, <secret_phase> 함수가 있는 것을 확인할 수 있고, <secret_phase> 함수를 호출하기 위해서는 앞서 나온 총 6 개의 phase 를 모두 해결해야 함을 알 수 있다. 또한, 함수 호출 조건으로 x/s 0x402817 을 통해 정수 2 개와 string 1 개를 입력해야 하며, x/s 0x6048b0 을 통해 입력해야 할 정수 2 개는 "4 2"임을 알 수 있다.

따라서 정답으로 "4 2"를 입력할 수 있는 <phase_4>에서 4 2 와 함께 문자열 1 개를 추가로 입력하면 <secret_phase> 함수를 호출할 수 있다.

이때, 입력해야 할 문자열은 x/s 0x402820 을 통해 "DrEvil"임을 확인할 수 있다.

<secret_phase> 함수에서는 cmp \$0x3e8, %eax 와 jbe 인스트럭션을 통해 입력해야 하는 숫자의 조건으로 $0 \leq x \leq 1001$ 임을 알 수 있다.

%esi 레지스터에는 입력한 숫자 값을 저장하고, %edi 레지스터에는 0x604110 을 저장하고, <fun7> 함수를 호출한다. 이때, 0x604110 에 해당하는 메모리 주소에는 36 이 저장되어 있다.

cmp \$0x1, %eax 의 결과로 ZF 가 1 이면 je 인스트럭션에 의해 <explode_bomb> 함수를 호출하지 않고 <secret_phase> 함수를 종료할 수 있다. 즉, <fun7> 함수의 반환값이 1 이 되어야 한다.

%edx 레지스터에 %rdi 에 해당하는 메모리 주소에 저장된 값을 저장한다. %edx 레지스터 값과 %esi 레지스터의 값의 대소관계에 따라 함수의 반환값에 2 배를 하거나, 2 배 후 1 을 더하는 연산을 진행한다.

이는 트리 구조로 나타낼 수 있다. 최종적으로 함수의 반환값이 1 이 되기 위해서 $2 \times 0 + 1$ 또는 $2 \times (2 \times 0) + 1$, $2 \times (2 \times (2 \times 0)) + 1$ 이 되면 성립하고, 따라서 처음 주소인 0x604110 에서 0x10 만큼 더한 주소에 저장된 값, 혹은 그 값에 0x8, 0x10 만큼 더 더한 주소에 저장된 값이 되어야 한다.

따라서 정답은 "50", "45", "40"

```
Starting program: /home/std/kihyun/bomb20/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
We have to stand with our North Korean allies.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2. Keep going!
0 987
Halfway there!
4 2 DrEvil
So you got that one. Try this one.
5 115
Good work! On to the next...
6 4 2 5 3 1
Curses, you've found the secret phase!
But finding it and solving it are quite different...
50
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```