```
//phase two

0000000000400ea9 <phase_2>:
  400ea9:       55                      push   %rbp
  400eaa:       53                      push   %rbx
  400eab:       48 83 ec 28             sub    $0x28,%rsp
  400eaf:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  400eb6:       00 00
  400eb8:       48 89 44 24 18          mov    %rax,0x18(%rsp)
  400ebd:       31 c0                   xor    %eax,%eax
  400ebf:       48 89 e6                mov    %rsp,%rsi
  400ec2:       e8 85 05 00 00          call   40144c <read_six_numbers> //we come to know the input
is six number
  400ec7:       83 3c 24 00             cmpl   $0x0,(%rsp)
  400ecb:       75 07                   jne    400ed4 <phase_2+0x2b>
  400ecd:       83 7c 24 04 01          cmpl   $0x1,0x4(%rsp)
  400ed2:       74 05                   je     400ed9 <phase_2+0x30>
  400ed4:       e8 51 05 00 00          call   40142a <explode_bomb>
  400ed9:       48 89 e3                mov    %rsp,%rbx
  400edc:       48 8d 6c 24 10          lea    0x10(%rsp),%rbp
  400ee1:       8b 43 04                mov    0x4(%rbx),%eax
  400ee4:       03 03                   add    (%rbx),%eax
  400ee6:       39 43 08                cmp    %eax,0x8(%rbx)
  400ee9:       74 05                   je     400ef0 <phase_2+0x47>
  400eeb:       e8 3a 05 00 00          call   40142a <explode_bomb>
  400ef0:       48 83 c3 04             add    $0x4,%rbx
  400ef4:       48 39 eb                cmp    %rbp,%rbx
  400ef7:       75 e8                   jne    400ee1 <phase_2+0x38>
  400ef9:       48 8b 44 24 18          mov    0x18(%rsp),%rax
  400efe:       64 48 33 04 25 28 00    xor    %fs:0x28,%rax
  400f05:       00 00
  400f07:       74 05                   je     400f0e <phase_2+0x65>
  400f09:       e8 f2 fb ff ff          call   400b00 <__stack_chk_fail@plt>
  400f0e:       48 83 c4 28             add    $0x28,%rsp
  400f12:       5b                      pop    %rbx
  400f13:       5d                      pop    %rbp
  400f14:       c3                      ret

Lets put in "0 1 2 3 4 5" as our test input and disas <read_six_numbers>:
Dump of assembler code for function read_six_numbers:

   0x000000000040144c <+0>:   sub    $0x8,%rsp
   0x0000000000401450 <+4>:   mov    %rsi,%rdx
   0x0000000000401453 <+7>:   lea    0x4(%rsi),%rcx
   0x0000000000401457 <+11>:  lea    0x14(%rsi),%rax
   0x000000000040145b <+15>:  push   %rax
   0x000000000040145c <+16>:  lea    0x10(%rsi),%rax
   0x0000000000401460 <+20>:  push   %rax
   0x0000000000401461 <+21>:  lea    0xc(%rsi),%r9
   0x0000000000401465 <+25>:  lea    0x8(%rsi),%r8
   0x0000000000401469 <+29>:  mov    $0x4025a3,%esi
   0x000000000040146e <+34>:  mov    $0x0,%eax
   0x0000000000401473 <+39>:  call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000401478 <+44>:  add    $0x10,%rsp
   0x000000000040147c <+48>:  cmp    $0x5,%eax // here we compare the length of our user input
to 5
=> 0x000000000040147f <+51>:  jg     0x401486 <read_six_numbers+58> //if %eax is greater
than 5, then we pass explode bomb.
   0x0000000000401481 <+53>:  call   0x40142a <explode_bomb>
   0x0000000000401486 <+58>:  add    $0x8,%rsp
   0x000000000040148a <+62>:  ret

End of assembler dump.
```

Lets check what 0x4025a3 is:

```
(gdb) x/s 0x4025a3
0x4025a3:       "%d %d %d %d %d %d"
(gdb)
```

//Must be the format of our answer, which is 6 digits with spaces in between. Looking at:

```
  0x000000000040147c <+48>:  cmp    $0x5,%eax
=>0x000000000040147f <+51>:  jg     0x401486 <read_six_numbers+58>
  0x0000000000401481 <+53>:  call   0x40142a <explode_bomb>
```

We can see that its probably comparing our input format to the format in %esi. If we have more than 5 digits, aka 6, we can pass
the explode bomb. Lets see if this works for our "0 1 2 3 4 5":

```
      0x00000000004015e0 <+54>:    jg     0x4015e7 <read_six_numbers+61>
      0x00000000004015e2 <+56>:    callq  0x401574 <explode_bomb>
 => 0x00000000004015e7 <+61>:    add    $0x18,%rsp
      0x00000000004015eb <+65>:    retq
    End of assembler dump.
```

We pass the bomb! So the format is definitely %d %d %d %d %d %d.

Lets look at what the compare statement is comparing:

```
gdb) disas
Dump of assembler code for function phase_2:
  0x0000000000400ea9 <+0>:   push   %rbp
  0x0000000000400eaa <+1>:   push   %rbx
  0x0000000000400eab <+2>:   sub    $0x28,%rsp
  0x0000000000400eaf <+6>:   mov    %fs:0x28,%rax
  0x0000000000400eb8 <+15>:  mov    %rax,0x18(%rsp)
  0x0000000000400ebd <+20>:  xor    %eax,%eax
  0x0000000000400ebf <+22>:  mov    %rsp,%rsi
  0x0000000000400ec2 <+25>:  call   0x40144c <read_six_numbers>
=> 0x0000000000400ec7 <+30>:  cmpl   $0x0,(%rsp) // comparing 0 to the value in (%rsp) which
is the first user input pointing to the array type "0 1 2 3 4 5"
  0x0000000000400ecb <+34>:  jne    0x400ed4 <phase_2+43> //jump if not equal
  0x0000000000400ecd <+36>:  cmpl   $0x1,0x4(%rsp) // comparing 1 to the value in (4+%rsp)
which is the second user input pointing to the array type "0 1 2 3 4 5"
  0x0000000000400ed2 <+41>:  je     0x400ed9 <phase_2+48> //jump if equal
```

```
   0x0000000000400ed4 <+43>:  call   0x40142a <explode_bomb>
   0x0000000000400ed9 <+48>:  mov    %rsp,%rbx // assigning all array pointing address to %rbx
   0x0000000000400edc <+51>:  lea    0x10(%rsp),%rbp
   0x0000000000400ee1 <+56>:  mov    0x4(%rbx),%eax //moving the user input to the %eax
   0x0000000000400ee4 <+59>:  add    (%rbx),%eax //it adds the value to the current value in
%eax eg:"0+1->%eax"
   0x0000000000400ee6 <+61>:  cmp    %eax,0x8(%rbx) //it compares if the added value in %eax
is equal to the user input or not
   0x0000000000400ee9 <+64>:  je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:  call   0x40142a <explode_bomb>
   0x0000000000400ef0 <+71>:  add    $0x4,%rbx //it adds 4 to the address of %ebx to check the
condition of loop
   0x0000000000400ef4 <+75>:  cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:  jne    0x400ee1 <phase_2+56> //if the two address if not equal,
loop takes place
   0x0000000000400ef9 <+80>:  mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:  xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:  je     0x400f0e <phase_2+101>
   0x0000000000400f09 <+96>:  call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>: add    $0x28,%rsp
   0x0000000000400f12 <+105>: pop    %rbx
   0x0000000000400f13 <+106>: pop    %rbp
   0x0000000000400f14 <+107>: ret
End of assembler dump.

   0x0000000000400f2c <+32>:    cmp    %eax,(%rbx)
   0x0000000000400f2e <+34>:    je     0x400f35 <phase_2+41>
   0x0000000000400f30 <+36>:    callq  0x401574 <explode_bomb>
=> 0x0000000000400f35 <+41>:    add    $0x4,%rbx
   0x0000000000400f39 <+45>:    cmp    %rbp,%rbx
```

//We can see that %rbx holds each digit that we inputted from "0 1 2 3 4 5". We are looping through each digit
in our input and comparing it to the correct digits, which are in %rax.

```
   0x0000000000400ee1 <+56>:  mov    0x4(%rbx),%eax
   0x0000000000400ee4 <+59>:  add    (%rbx),%eax
   0x0000000000400ee6 <+61>:  cmp    %eax,0x8(%rbx)
   0x0000000000400ee9 <+64>:  je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:  call   0x40142a <explode_bomb>
   0x0000000000400ef0 <+71>:  add    $0x4,%rbx
   0x0000000000400ef4 <+75>:  cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:  jne    0x400ee1 <phase_2+56>
```

Looking at <+59>, we see we are adding first(0) value in %rbx to value in %eax which was moved from 4(%rbx) to %eax, which was 1. lets change our input to "0 1 1 3 4 5" to see if we can pass this iteration for third place value 1. ==> when we do this, adding 1 does pass this iteration of the loop. At this point, we can see at <+59> when looped again, we get a pattern of fibonacci number.

When we put "0 1 1 2 3 5" this works!!

//Solution:
0 1 1 2 3 5