Detailed solution

We can see before calling each phase, the input is stored at %rdi, so we should pay attention to what happened to %rdi.

phase 1

```
0000000000400ee0 <phase_1>:
  400ee0:   48 83 ec 08             sub    $0x8,%rsp
  400ee4:   be 00 24 40 00          mov    $0x402400,%esi
  400ee9:   e8 4a 04 00 00          callq  401338 <strings_not_equal>
  400eee:   85 c0                   test   %eax,%eax
  400ef0:   74 05                   je     400ef7 <phase_1+0x17>
  400ef2:   e8 43 05 00 00          callq  40143a <explode_bomb>
  400ef7:   48 83 c4 08             add    $0x8,%rsp
  400efb:   c3                      retq
```

pass when %eax = 0.

helper functions:

```
000000000040131b <string_length>:
  40131b:   80 3f 00                cmpb   $0x0,(%rdi)
  40131e:   74 12                   je     401332 <string_length+0x17>
  401320:   48 89 fa                mov    %rdi,%rdx
  401323:   48 83 c2 01             add    $0x1,%rdx
  401327:   89 d0                   mov    %edx,%eax
  401329:   29 f8                   sub    %edi,%eax
  40132b:   80 3a 00                cmpb   $0x0,(%rdx)
  40132e:   75 f3                   jne    401323 <string_length+0x8>
  401330:   f3 c3                   repz retq
  401332:   b8 00 00 00 00          mov    $0x0,%eax
  401337:   c3                      retq
```

```
int string_length(char *x){
    char *y;
    int result;
    if (*x == '\0') return 0;
loop:
    y = x;
    ++y;
    result = y - x;
    if (*y != '\0') goto loop;
    return result;
}
```

%rdi is the address of a string, %eax contains its length after executing this function.

```
0000000000401338 <strings_not_equal>:
# Suppose char *a in %rdi, char *b in %rsi
  401338:   41 54                   push   %r12
  40133a:   55                      push   %rbp
  40133b:   53                      push   %rbx
  40133c:   48 89 fb                mov    %rdi,%rbx
  40133f:   48 89 f5                mov    %rsi,%rbp
  401342:   e8 d4 ff ff ff          callq  40131b <string_length>
  401347:   41 89 c4                mov    %eax,%r12d
  40134a:   48 89 ef                mov    %rbp,%rdi
```

```
  40134d:   e8 c9 ff ff ff          callq  40131b <string_length>
  # %r12d = length(a), %rax = length(b)
  401352:   ba 01 00 00 00          mov    $0x1,%edx
  # %edx = true
  401357:   41 39 c4                cmp    %eax,%r12d
  40135a:   75 3f                   jne    40139b <strings_not_equal+0x63>
  # if (length(a) != length(b)) return true;
  40135c:   0f b6 03                movzbl (%rbx),%eax
  40135f:   84 c0                   test   %al,%al
  401361:   74 25                   je     401388 <strings_not_equal+0x50>
  401363:   3a 45 00                cmp    0x0(%rbp),%al
  401366:   74 0a                   je     401372 <strings_not_equal+0x3a>
  401368:   eb 25                   jmp    40138f <strings_not_equal+0x57>
  40136a:   3a 45 00                cmp    0x0(%rbp),%al
  40136d:   0f 1f 00                nopl   (%rax)
  401370:   75 24                   jne    401396 <strings_not_equal+0x5e>
  401372:   48 83 c3 01             add    $0x1,%rbx
  401376:   48 83 c5 01             add    $0x1,%rbp
  40137a:   0f b6 03                movzbl (%rbx),%eax
  40137d:   84 c0                   test   %al,%al
  40137f:   75 e9                   jne    40136a <strings_not_equal+0x32>
  401381:   ba 00 00 00 00          mov    $0x0,%edx
  401386:   eb 13                   jmp    40139b <strings_not_equal+0x63>
  401388:   ba 00 00 00 00          mov    $0x0,%edx
  40138d:   eb 0c                   jmp    40139b <strings_not_equal+0x63>
  40138f:   ba 01 00 00 00          mov    $0x1,%edx
  401394:   eb 05                   jmp    40139b <strings_not_equal+0x63>
  401396:   ba 01 00 00 00          mov    $0x1,%edx
  40139b:   89 d0                   mov    %edx,%eax
  40139d:   5b                      pop    %rbx
  40139e:   5d                      pop    %rbp
  40139f:   41 5c                   pop    %r12
  4013a1:   c3                      retq
```

%rdi, %rsi are two string addresses, %eax = 1 if the strings are not equal, otherwise 0.

So for phase_1, %rdi is the input line, %rsi = 0x402400, we need to figure out what's in this memory address.

Use `gdb bomb`, then `(gdb) print (char*)0x402400`, it returns `$1 = 0x402400 "Border relations with Canada have never been better."`, and that's the answer of phase 1.

phase 2

```
0000000000400efc <phase_2>:
  400efc:   55                      push   %rbp
  400efd:   53                      push   %rbx
  400efe:   48 83 ec 28             sub    $0x28,%rsp
  400f02:   48 89 e6                mov    %rsp,%rsi
  400f05:   e8 52 05 00 00          callq  40145c <read_six_numbers>
  400f0a:   83 3c 24 01             cmpl   $0x1,(%rsp)
  # (int)(%rsp) = 1 (otherwise explode)
  400f0e:   74 20                   je     400f30 <phase_2+0x34>
  400f10:   e8 25 05 00 00          callq  40143a <explode_bomb>

  400f15:   eb 19                   jmp    400f30 <phase_2+0x34>
  # This line will never be executed.

  400f17:   8b 43 fc                mov    -0x4(%rbx),%eax
  400f1a:   01 c0                   add    %eax,%eax
  400f1c:   39 03                   cmp    %eax,(%rbx)
  # (%rbx) = 2 * (int)(%rbx - 4) (otherwise explode)
  400f1e:   74 05                   je     400f25 <phase_2+0x29>
  400f20:   e8 15 05 00 00          callq  40143a <explode_bomb>

  400f25:   48 83 c3 04             add    $0x4,%rbx
  400f29:   48 39 eb                cmp    %rbp,%rbx
  # when %rbx + 4 gets to %rsp + 24, go out, otherwise repeat.
```

```
  400f2c:   75 e9                  jne    400f17 <phase_2+0x1b>
  400f2e:   eb 0c                  jmp    400f3c <phase_2+0x40>

  400f30:   48 8d 5c 24 04         lea    0x4(%rsp),%rbx
  400f35:   48 8d 6c 24 18         lea    0x18(%rsp),%rbp
  # %rbx = %rsp + 4, %rbp = %rsp + 24
  400f3a:   eb db                  jmp    400f17 <phase_2+0x1b>

  # Go out!
  400f3c:   48 83 c4 28            add    $0x28,%rsp
  400f40:   5b                     pop    %rbx
  400f41:   5d                     pop    %rbp
  400f42:   c3                     retq
```

Phase 2 requires (int)(%rsp) = 1, (int)(%rsp + 4) = 2(int)(%rsp), (int)(%rsp + 8) = 2(int)(%rsp + 4), and so on.

So the 4 byte int in the stack should be 1 2 4 8 16 32, from top of stack to bottom.

Then take a look at `read_six_numbers` .

```
000000000040145c <read_six_numbers>:
  40145c:   48 83 ec 18            sub    $0x18,%rsp
  401460:   48 89 f2               mov    %rsi,%rdx
  401463:   48 8d 4e 04            lea    0x4(%rsi),%rcx
  401467:   48 8d 46 14            lea    0x14(%rsi),%rax
  40146b:   48 89 44 24 08         mov    %rax,0x8(%rsp)
  401470:   48 8d 46 10            lea    0x10(%rsi),%rax
  401474:   48 89 04 24            mov    %rax,(%rsp)
  401478:   4c 8d 4e 0c            lea    0xc(%rsi),%r9
  40147c:   4c 8d 46 08            lea    0x8(%rsi),%r8
  401480:   be c3 25 40 00         mov    $0x4025c3,%esi
  # gdb tells "%d %d %d %d %d %d" is in 0x4025c3
  401485:   b8 00 00 00 00         mov    $0x0,%eax
  40148a:   e8 61 f7 ff ff         callq  400bf0 <__isoc99_sscanf@plt>
  40148f:   83 f8 05               cmp    $0x5,%eax
  401492:   7f 05                  jg     401499 <read_six_numbers+0x3d>
  # Go out when scanf read more than 5 inputs
  401494:   e8 a1 ff ff ff         callq  40143a <explode_bomb>
  401499:   48 83 c4 18            add    $0x18,%rsp
  40149d:   c3                     retq
```

It pushes six input numbers into the stack.

Although some details are still obscure for me, now I am sure the helper functions just behave as their names indicate...

phase 3

```
0000000000400f43 <phase_3>:
  400f43:   48 83 ec 18            sub    $0x18,%rsp
  400f47:   48 8d 4c 24 0c         lea    0xc(%rsp),%rcx
  # gdb tells *$rcx is 0
  400f4c:   48 8d 54 24 08         lea    0x8(%rsp),%rdx
  400f51:   be cf 25 40 00         mov    $0x4025cf,%esi
  # gdb tells "%d %d" is in 0x4025cf
  400f56:   b8 00 00 00 00         mov    $0x0,%eax
  400f5b:   e8 90 fc ff ff         callq  400bf0 <__isoc99_sscanf@plt>
  400f60:   83 f8 01               cmp    $0x1,%eax
  400f63:   7f 05                  jg     400f6a <phase_3+0x27>
  # %eax > 1, i.e. input 2 numbers
  400f65:   e8 d0 04 00 00         callq  40143a <explode_bomb>
  400f6a:   83 7c 24 08 07         cmpl   $0x7,0x8(%rsp)
  400f6f:   77 3c                  ja     400fad <phase_3+0x6a>
  # the first number (denote it as a) <= 7
  400f71:   8b 44 24 08            mov    0x8(%rsp),%eax
  400f75:   ff 24 c5 70 24 40 00   jmpq   *0x402470(,%rax,8)
  # get address in the memory 0x402470 + a * 8
  # for a = 0 ... 7, the results are (and the value of %eax, correspondingly)
  # 0x400f7c(0xcf), 0x400fb9(0x137), 0x400f83(0x2c3), 0x400f8a(0x100),
```

```
  # 0x400f91(0x185), 0x400f98(0xce), 0x400f9f(0x2aa), 0x400fa6(0x147)
  400f7c:   b8 cf 00 00 00          mov    $0xcf,%eax
  400f81:   eb 3b                   jmp    400fbe <phase_3+0x7b>
  400f83:   b8 c3 02 00 00          mov    $0x2c3,%eax
  400f88:   eb 34                   jmp    400fbe <phase_3+0x7b>
  400f8a:   b8 00 01 00 00          mov    $0x100,%eax
  400f8f:   eb 2d                   jmp    400fbe <phase_3+0x7b>
  400f91:   b8 85 01 00 00          mov    $0x185,%eax
  400f96:   eb 26                   jmp    400fbe <phase_3+0x7b>
  400f98:   b8 ce 00 00 00          mov    $0xce,%eax
  400f9d:   eb 1f                   jmp    400fbe <phase_3+0x7b>
  400f9f:   b8 aa 02 00 00          mov    $0x2aa,%eax
  400fa4:   eb 18                   jmp    400fbe <phase_3+0x7b>
  400fa6:   b8 47 01 00 00          mov    $0x147,%eax
  400fab:   eb 11                   jmp    400fbe <phase_3+0x7b>
  400fad:   e8 88 04 00 00          callq  40143a <explode_bomb>
  400fb2:   b8 00 00 00 00          mov    $0x0,%eax
  400fb7:   eb 05                   jmp    400fbe <phase_3+0x7b>
  400fb9:   b8 37 01 00 00          mov    $0x137,%eax
  400fbe:   3b 44 24 0c             cmp    0xc(%rsp),%eax
  400fc2:   74 05                   je     400fc9 <phase_3+0x86>
  # %eax = b
  400fc4:   e8 71 04 00 00          callq  40143a <explode_bomb>
  400fc9:   48 83 c4 18             add    $0x18,%rsp
  400fcd:   c3                      retq
```

It seems any one of the 8 pairs will survive? Really?

phase 4

```
000000000040100c <phase_4>:
  40100c:   48 83 ec 18             sub    $0x18,%rsp
  401010:   48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  401015:   48 8d 54 24 08          lea    0x8(%rsp),%rdx
  40101a:   be cf 25 40 00          mov    $0x4025cf,%esi
  40101f:   b8 00 00 00 00          mov    $0x0,%eax
  401024:   e8 c7 fb ff ff          callq  400bf0 <__isoc99_sscanf@plt>
  # above lines are the same as phase 3
  401029:   83 f8 02                cmp    $0x2,%eax
  40102c:   75 07                   jne    401035 <phase_4+0x29>
  # input two numbers, %rax = 2
  40102e:   83 7c 24 08 0e          cmpl   $0xe,0x8(%rsp)
  401033:   76 05                   jbe    40103a <phase_4+0x2e>
  # input a <= 0xe
  401035:   e8 00 04 00 00          callq  40143a <explode_bomb>
  40103a:   ba 0e 00 00 00          mov    $0xe,%edx
  40103f:   be 00 00 00 00          mov    $0x0,%esi
  401044:   8b 7c 24 08             mov    0x8(%rsp),%edi
  401048:   e8 81 ff ff ff          callq  400fce <func4>
  40104d:   85 c0                   test   %eax,%eax
  # %eax = 0
  40104f:   75 07                   jne    401058 <phase_4+0x4c>
  401051:   83 7c 24 0c 00          cmpl   $0x0,0xc(%rsp)
  # b = 0
  401056:   74 05                   je     40105d <phase_4+0x51>
  401058:   e8 dd 03 00 00          callq  40143a <explode_bomb>
  40105d:   48 83 c4 18             add    $0x18,%rsp
  401061:   c3                      retq
```

Take a look at `func4` :

```
0000000000400fce <func4>:
  400fce:   48 83 ec 08             sub    $0x8,%rsp
  400fd2:   89 d0                   mov    %edx,%eax
  400fd4:   29 f0                   sub    %esi,%eax # %eax = 0xe - 0
  400fd6:   89 c1                   mov    %eax,%ecx
  400fd8:   c1 e9 1f                shr    $0x1f,%ecx # %ecx = %eax >> 31(logical) = 0
  400fdb:   01 c8                   add    %ecx,%eax
  400fdd:   d1 f8                   sar    %eax # %eax = (%eax + %ecx) >> 1 (arith) = 7
```

```
  400fdf:   8d 0c 30                lea    (%rax,%rsi,1),%ecx # %ecx = %rax + 0 = 7
  400fe2:   39 f9                   cmp    %edi,%ecx
  400fe4:   7e 0c                   jle    400ff2 <func4+0x24>

  # 7 < a
  400fe6:   8d 51 ff                lea    -0x1(%rcx),%edx # %edx = 7 - 1 = 6
  400fe9:   e8 e0 ff ff ff          callq  400fce <func4>
  400fee:   01 c0                   add    %eax,%eax
  400ff0:   eb 15                   jmp    401007 <func4+0x39>

  # 7 >= a here
  400ff2:   b8 00 00 00 00          mov    $0x0,%eax
  400ff7:   39 f9                   cmp    %edi,%ecx
  400ff9:   7d 0c                   jge    401007 <func4+0x39>
  400ffb:   8d 71 01                lea    0x1(%rcx),%esi
  400ffe:   e8 cb ff ff ff          callq  400fce <func4>
  401003:   8d 44 00 01             lea    0x1(%rax,%rax,1),%eax

  401007:   48 83 c4 08             add    $0x8,%rsp
  40100b:   c3                      retq
```

Oh, it's a little bit complex, let's translate it into C:

```c
int func4(int a, int b, int c)
// a in %esi, b in %edi, c in %edx
// func4 are called by phase_4 with (0, input a, 0xe), and should return 0.
{
    int result, tmp;
    result = c - a;
    tmp = result < 0;
    result = (result + tmp) >> 1;
    tmp = result + a;
    if(tmp <= b)
    {
        result = 0;
        if(tmp >= b) // == actually
            return result;
        else
        {
            result = func4(1+tmp, b, c);
            return 1 + 2 * result;
        }
    }
    else
    {
        c = tmp - 1;
        result = func4(a, b, c);
        return 2 * result;
    }
}
```

The result is not obvious, but we can easily try all the cases!

```c
int main()
{
    for(int i = 0; i <= 0xe; ++i)
        printf("%d ",func4(0,i,0xe));
    return 0;
}
```

It prints `0 0 4 0 2 2 6 0 1 1 5 1 3 3 7`, so `a` can be 0/1/3/7, still so many possibilities? `b` is 0, that's easy.

phase 5

```
0000000000401062 <phase_5>:
  401062:   53                      push   %rbx
  401063:   48 83 ec 20             sub    $0x20,%rsp
```

```
  401067:   48 89 fb                mov    %rdi,%rbx
  40106a:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  401071:   00 00
  401073:   48 89 44 24 18          mov    %rax,0x18(%rsp)
  401078:   31 c0                   xor    %eax,%eax
  40107a:   e8 9c 02 00 00          callq  40131b <string_length>
  40107f:   83 f8 06                cmp    $0x6,%eax # input 6 chars
  401082:   74 4e                   je     4010d2 <phase_5+0x70>
  401084:   e8 b1 03 00 00          callq  40143a <explode_bomb>

  401089:   eb 47                   jmp    4010d2 <phase_5+0x70> # seems useless

  40108b:   0f b6 0c 03             movzbl (%rbx,%rax,1),%ecx
  40108f:   88 0c 24                mov    %cl,(%rsp)
  401092:   48 8b 14 24             mov    (%rsp),%rdx
  401096:   83 e2 0f                and    $0xf,%edx
  401099:   0f b6 92 b0 24 40 00    movzbl 0x4024b0(%rdx),%edx
  4010a0:   88 54 04 10             mov    %dl,0x10(%rsp,%rax,1)
  4010a4:   48 83 c0 01             add    $0x1,%rax
  4010a8:   48 83 f8 06             cmp    $0x6,%rax
  4010ac:   75 dd                   jne    40108b <phase_5+0x29>
  # Above is the main loop processing the input.

  4010ae:   c6 44 24 16 00          movb   $0x0,0x16(%rsp)
  4010b3:   be 5e 24 40 00          mov    $0x40245e,%esi # "flyers"
  4010b8:   48 8d 7c 24 10          lea    0x10(%rsp),%rdi
  4010bd:   e8 76 02 00 00          callq  401338 <strings_not_equal>
  4010c2:   85 c0                   test   %eax,%eax # (char *)0x10(%rsp) = "flyers"
  4010c4:   74 13                   je     4010d9 <phase_5+0x77>
  4010c6:   e8 6f 03 00 00          callq  40143a <explode_bomb>
  4010cb:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
  4010d0:   eb 07                   jmp    4010d9 <phase_5+0x77>

  4010d2:   b8 00 00 00 00          mov    $0x0,%eax
  4010d7:   eb b2                   jmp    40108b <phase_5+0x29>
  4010d9:   48 8b 44 24 18          mov    0x18(%rsp),%rax
  4010de:   64 48 33 04 25 28 00    xor    %fs:0x28,%rax
  4010e5:   00 00
  4010e7:   74 05                   je     4010ee <phase_5+0x8c> # 0x28 = 0x18(%rsp)
  4010e9:   e8 42 fa ff ff          callq  400b30 <__stack_chk_fail@plt>
  4010ee:   48 83 c4 20             add    $0x20,%rsp
  4010f2:   5b                      pop    %rbx
  4010f3:   c3                      retq
```

By the first few attempts, I found that input characters are transformed according to fixed rules, so we can just find the code table instead of delving into the boring assembly codes!

The table is:

```
| before | abcdef    ghijkl    mnopqr    stuvwx    yz |
| after  | aduier    snfotv    bylmad    uiersn    fo |
```

So to get `flyers`, we can input `ionefg`.

Phase 6

This phase is so long, so we'd better split it into parts.

```
00000000004010f4 <phase_6>:
  ###### start, read six numbers
  4010f4:   41 56                   push   %r14
  4010f6:   41 55                   push   %r13
  4010f8:   41 54                   push   %r12
  4010fa:   55                      push   %rbp
  4010fb:   53                      push   %rbx
  4010fc:   48 83 ec 50             sub    $0x50,%rsp
  401100:   49 89 e5                mov    %rsp,%r13
  401103:   48 89 e6                mov    %rsp,%rsi
  401106:   e8 51 03 00 00          callq  40145c <read_six_numbers>
```

```
40110b:   49 89 e6                mov    %rsp,%r14
40110e:   41 bc 00 00 00 00       mov    $0x0,%r12d


###### Part 1: the six numbers <= 6, and not equal
401114:   4c 89 ed                mov    %r13,%rbp
401117:   41 8b 45 00             mov    0x0(%r13),%eax # 1st input
40111b:   83 e8 01                sub    $0x1,%eax
40111e:   83 f8 05                cmp    $0x5,%eax
401121:   76 05                   jbe    401128 <phase_6+0x34> # 1st input <= 6
401123:   e8 12 03 00 00          callq  40143a <explode_bomb>

401128:   41 83 c4 01             add    $0x1,%r12d # %r12d = 1
40112c:   41 83 fc 06             cmp    $0x6,%r12d
401130:   74 21                   je     401153 <phase_6+0x5f> # OUT HERE
401132:   44 89 e3                mov    %r12d,%ebx

401135:   48 63 c3                movslq %ebx,%rax # %rax = 1
401138:   8b 04 84                mov    (%rsp,%rax,4),%eax # %eax = 2nd input
40113b:   39 45 00                cmp    %eax,0x0(%rbp) # 2nd input != 1st input
40113e:   75 05                   jne    401145 <phase_6+0x51>
401140:   e8 f5 02 00 00          callq  40143a <explode_bomb>
401145:   83 c3 01                add    $0x1,%ebx
401148:   83 fb 05                cmp    $0x5,%ebx
40114b:   7e e8                   jle    401135 <phase_6+0x41>
# the 2~6 inputs != 1st input
40114d:   49 83 c5 04             add    $0x4,%r13
401151:   eb c1                   jmp    401114 <phase_6+0x20>
# next iteration, 1st input -> 2nd input


###### Part 2: x = 7 - x, for each input number
401153:   48 8d 74 24 18          lea    0x18(%rsp),%rsi
401158:   4c 89 f0                mov    %r14,%rax # %rsp
40115b:   b9 07 00 00 00          mov    $0x7,%ecx

401160:   89 ca                   mov    %ecx,%edx
401162:   2b 10                   sub    (%rax),%edx
401164:   89 10                   mov    %edx,(%rax)
401166:   48 83 c0 04             add    $0x4,%rax
40116a:   48 39 f0                cmp    %rsi,%rax
40116d:   75 f1                   jne    401160 <phase_6+0x6c>


###### Part 3: Push the nodes into the stack.
###### If the 1st input is x, then the (7-x)-th node will be pushed first.
## Data at 0x6032d0 are:
##                    |data     |id        |next
## 0x6032d0 <node1>:  0x0000014c  0x00000001  0x006032e0  0x00000000
## 0x6032e0 <node2>:  0x000000a8  0x00000002  0x006032f0  0x00000000
## 0x6032f0 <node3>:  0x0000039c  0x00000003  0x00603300  0x00000000
## 0x603300 <node4>:  0x000002b3  0x00000004  0x00603310  0x00000000
## 0x603310 <node5>:  0x000001dd  0x00000005  0x00603320  0x00000000
## 0x603320 <node6>:  0x000001bb  0x00000006  0x00000000  0x00000000
## It's a linked list!
40116f:   be 00 00 00 00          mov    $0x0,%esi
401174:   eb 21                   jmp    401197 <phase_6+0xa3>

401176:   48 8b 52 08             mov    0x8(%rdx),%rdx # take next
40117a:   83 c0 01                add    $0x1,%eax
40117d:   39 c8                   cmp    %ecx,%eax
40117f:   75 f5                   jne    401176 <phase_6+0x82>
401181:   eb 05                   jmp    401188 <phase_6+0x94>
# for(i = 1; i < %ecx; ++i) %rdx = %rdx -> next;
# i.e., if %ecx = i, %rdx = nodei

401183:   ba d0 32 60 00          mov    $0x6032d0,%edx
401188:   48 89 54 74 20          mov    %rdx,0x20(%rsp,%rsi,2)
40118d:   48 83 c6 04             add    $0x4,%rsi
401191:   48 83 fe 18             cmp    $0x18,%rsi
401195:   74 14                   je     4011ab <phase_6+0xb7> # OUT HERE

401197:   8b 0c 34                mov    (%rsp,%rsi,1),%ecx
40119a:   83 f9 01                cmp    $0x1,%ecx
```

```
  40119d:   7e e4                   jle    401183 <phase_6+0x8f>
  40119f:   b8 01 00 00 00          mov    $0x1,%eax
  4011a4:   ba d0 32 60 00          mov    $0x6032d0,%edx
  4011a9:   eb cb                   jmp    401176 <phase_6+0x82>


  ###### Part 4: Relink the list according to the order in the stack
  4011ab:   48 8b 5c 24 20          mov    0x20(%rsp),%rbx # &head
  4011b0:   48 8d 44 24 28          lea    0x28(%rsp),%rax
  4011b5:   48 8d 74 24 50          lea    0x50(%rsp),%rsi
  4011ba:   48 89 d9                mov    %rbx,%rcx

  4011bd:   48 8b 10                mov    (%rax),%rdx
  4011c0:   48 89 51 08             mov    %rdx,0x8(%rcx)
  4011c4:   48 83 c0 08             add    $0x8,%rax
  4011c8:   48 39 f0                cmp    %rsi,%rax
  4011cb:   74 05                   je     4011d2 <phase_6+0xde>
  4011cd:   48 89 d1                mov    %rdx,%rcx
  4011d0:   eb eb                   jmp    4011bd <phase_6+0xc9>

  4011d2:   48 c7 42 08 00 00 00    movq   $0x0,0x8(%rdx) # tail -> next = null
  4011d9:   00


  ###### Part 5: Check the linked list is in descending order
  4011da:   bd 05 00 00 00          mov    $0x5,%ebp

  4011df:   48 8b 43 08             mov    0x8(%rbx),%rax
  4011e3:   8b 00                   mov    (%rax),%eax
  4011e5:   39 03                   cmp    %eax,(%rbx) # (%rbx)>=(%rax)=0x8(%rbx)
  4011e7:   7d 05                   jge    4011ee <phase_6+0xfa>
  4011e9:   e8 4c 02 00 00          callq  40143a <explode_bomb>
  4011ee:   48 8b 5b 08             mov    0x8(%rbx),%rbx
  4011f2:   83 ed 01                sub    $0x1,%ebp
  4011f5:   75 e8                   jne    4011df <phase_6+0xeb>


  ###### end
  4011f7:   48 83 c4 50             add    $0x50,%rsp
  4011fb:   5b                      pop    %rbx
  4011fc:   5d                      pop    %rbp
  4011fd:   41 5c                   pop    %r12
  4011ff:   41 5d                   pop    %r13
  401201:   41 5e                   pop    %r14
  401203:   c3                      retq
```

By the analysis above, the order should be `3 4 5 6 1 2`, so the input should be `4 3 2 1 6 5`.

Secret Phase

Browsing the whole codes, we can find there is a `secret_phase`. The only entrance is in `phase_defused`.

```
00000000004015c4 <phase_defused>:
  4015c4:   48 83 ec 78             sub    $0x78,%rsp
  4015c8:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  4015cf:   00 00
  4015d1:   48 89 44 24 68          mov    %rax,0x68(%rsp)
  4015d6:   31 c0                   xor    %eax,%eax
  4015d8:   83 3d 81 21 20 00 06    cmpl   $0x6,0x202181(%rip)
  4015df:   75 5e                   jne    40163f <phase_defused+0x7b>
  4015e1:   4c 8d 44 24 10          lea    0x10(%rsp),%r8
  4015e6:   48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  4015eb:   48 8d 54 24 08          lea    0x8(%rsp),%rdx

  ###### HERE!
  4015f0:   be 19 26 40 00          mov    $0x402619,%esi # "%d %d %s"
  4015f5:   bf 70 38 60 00          mov    $0x603870,%edi
  4015fa:   e8 f1 f5 ff ff          callq  400bf0 <__isoc99_sscanf@plt>
  4015ff:   83 f8 03                cmp    $0x3,%eax
  401602:   75 31                   jne    401635 <phase_defused+0x71>
```

```
  401604:   be 22 26 40 00          mov    $0x402622,%esi # "DrEvil"
  401609:   48 8d 7c 24 10          lea    0x10(%rsp),%rdi
  40160e:   e8 25 fd ff ff          callq  401338 <strings_not_equal>
  401613:   85 c0                   test   %eax,%eax
  401615:   75 1e                   jne    401635 <phase_defused+0x71>
  401617:   bf f8 24 40 00          mov    $0x4024f8,%edi
  40161c:   e8 ef f4 ff ff          callq  400b10 <puts@plt>
  401621:   bf 20 25 40 00          mov    $0x402520,%edi
  401626:   e8 e5 f4 ff ff          callq  400b10 <puts@plt>
  40162b:   b8 00 00 00 00          mov    $0x0,%eax
  401630:   e8 0d fc ff ff          callq  401242 <secret_phase> # ENTRENCE


  401635:   bf 58 25 40 00          mov    $0x402558,%edi
  40163a:   e8 d1 f4 ff ff          callq  400b10 <puts@plt>
  40163f:   48 8b 44 24 68          mov    0x68(%rsp),%rax
  401644:   64 48 33 04 25 28 00    xor    %fs:0x28,%rax
  40164b:   00 00
  40164d:   74 05                   je     401654 <phase_defused+0x90>
  40164f:   e8 dc f4 ff ff          callq  400b30 <__stack_chk_fail@plt>
  401654:   48 83 c4 78             add    $0x78,%rsp
  401658:   c3                      retq
  401659:   90                      nop
  40165a:   90                      nop
  40165b:   90                      nop
  40165c:   90                      nop
  40165d:   90                      nop
  40165e:   90                      nop
  40165f:   90                      nop
```

Since the answers of phase 3 and 4 contains two numbers, I guessed putting "DrEvil" after one of them will unlock the secret phase, and I found it's phase 4, not phase 3.

Let's solve it!

```
0000000000401242 <secret_phase>:
  401242:   53                      push   %rbx
  401243:   e8 56 02 00 00          callq  40149e <read_line>
  401248:   ba 0a 00 00 00          mov    $0xa,%edx
  40124d:   be 00 00 00 00          mov    $0x0,%esi
  401252:   48 89 c7                mov    %rax,%rdi
  401255:   e8 76 f9 ff ff          callq  400bd0 <strtol@plt>
  40125a:   48 89 c3                mov    %rax,%rbx
  40125d:   8d 40 ff                lea    -0x1(%rax),%eax
  401260:   3d e8 03 00 00          cmp    $0x3e8,%eax
  401265:   76 05                   jbe    40126c <secret_phase+0x2a>
  401267:   e8 ce 01 00 00          callq  40143a <explode_bomb>
  40126c:   89 de                   mov    %ebx,%esi
  40126e:   bf f0 30 60 00          mov    $0x6030f0,%edi
  401273:   e8 8c ff ff ff          callq  401204 <fun7>
  401278:   83 f8 02                cmp    $0x2,%eax # fun7 returns 2
  40127b:   74 05                   je     401282 <secret_phase+0x40>
  40127d:   e8 b8 01 00 00          callq  40143a <explode_bomb>
  401282:   bf 38 24 40 00          mov    $0x402438,%edi
  401287:   e8 84 f8 ff ff          callq  400b10 <puts@plt>
  40128c:   e8 33 03 00 00          callq  4015c4 <phase_defused>
  401291:   5b                      pop    %rbx
  401292:   c3                      retq
  401293:   90                      nop
  401294:   90                      nop
  401295:   90                      nop
  401296:   90                      nop
  401297:   90                      nop
  401298:   90                      nop
  401299:   90                      nop
  40129a:   90                      nop
  40129b:   90                      nop
  40129c:   90                      nop
  40129d:   90                      nop
  40129e:   90                      nop
  40129f:   90                      nop
```

Data at 0x6030f0 are:

```
0x6030f0 <n1>:  0x0000000000000024  0x0000000000603110
0x603100 <n1+16>:   0x0000000000603130  0x0000000000000000
0x603110 <n21>: 0x0000000000000008  0x0000000000603190
0x603120 <n21+16>:  0x0000000000603150  0x0000000000000000
0x603130 <n22>: 0x0000000000000032  0x0000000000603170
0x603140 <n22+16>:  0x00000000006031b0  0x0000000000000000
0x603150 <n32>: 0x0000000000000016  0x0000000000603270
0x603160 <n32+16>:  0x0000000000603230  0x0000000000000000
0x603170 <n33>: 0x000000000000002d  0x00000000006031d0
0x603180 <n33+16>:  0x0000000000603290  0x0000000000000000
0x603190 <n31>: 0x0000000000000006  0x00000000006031f0
0x6031a0 <n31+16>:  0x0000000000603250  0x0000000000000000
0x6031b0 <n34>: 0x000000000000006b  0x0000000000603210
0x6031c0 <n34+16>:  0x00000000006032b0  0x0000000000000000
0x6031d0 <n45>: 0x0000000000000028  0x0000000000000000
0x6031e0 <n45+16>:  0x0000000000000000  0x0000000000000000
0x6031f0 <n41>: 0x0000000000000001  0x0000000000000000
0x603200 <n41+16>:  0x0000000000000000  0x0000000000000000
0x603210 <n47>: 0x0000000000000063  0x0000000000000000
0x603220 <n47+16>:  0x0000000000000000  0x0000000000000000
0x603230 <n44>: 0x0000000000000023  0x0000000000000000
0x603240 <n44+16>:  0x0000000000000000  0x0000000000000000
0x603250 <n42>: 0x0000000000000007  0x0000000000000000
0x603260 <n42+16>:  0x0000000000000000  0x0000000000000000
0x603270 <n43>: 0x0000000000000014  0x0000000000000000
0x603280 <n43+16>:  0x0000000000000000  0x0000000000000000
0x603290 <n46>: 0x000000000000002f  0x0000000000000000
0x6032a0 <n46+16>:  0x0000000000000000  0x0000000000000000
0x6032b0 <n48>: 0x00000000000003e9  0x0000000000000000
0x6032c0 <n48+16>:  0x0000000000000000  0x0000000000000000
```

This is a binary tree:

```
                    0x24
                    /\
          0x8                       0x32
          /\                        /\
    0x6         0x16          0x2d          0x6b
    /\          /\            /\            /\
0x1   0x7   0x14   0x23   0x28   0x2f   0x63   0x3e9
```

Then we can know what happens in `fun7` more easily.

```
0000000000401204 <fun7>:
  401204:   48 83 ec 08             sub    $0x8,%rsp
  401208:   48 85 ff                test   %rdi,%rdi
  40120b:   74 2b                   je     401238 <fun7+0x34>
  # if(a == NULL) return 0xffffffff;
  40120d:   8b 17                   mov    (%rdi),%edx
  40120f:   39 f2                   cmp    %esi,%edx
  401211:   7e 0d                   jle    401220 <fun7+0x1c>

  401213:   48 8b 7f 08             mov    0x8(%rdi),%rdi
  401217:   e8 e8 ff ff ff          callq  401204 <fun7>
  40121c:   01 c0                   add    %eax,%eax
  40121e:   eb 1d                   jmp    40123d <fun7+0x39>

  401220:   b8 00 00 00 00          mov    $0x0,%eax
  401225:   39 f2                   cmp    %esi,%edx
  401227:   74 14                   je     40123d <fun7+0x39>
  401229:   48 8b 7f 10             mov    0x10(%rdi),%rdi
  40122d:   e8 d2 ff ff ff          callq  401204 <fun7>
  401232:   8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  401236:   eb 05                   jmp    40123d <fun7+0x39>

  401238:   b8 ff ff ff ff          mov    $0xffffffff,%eax
```

```
  40123d:   48 83 c4 08             add    $0x8,%rsp
  401241:   c3                      retq
```

The C code is:

```
int fun7(treenode *p, int x)
{
    int tmp;
    if(p == NULL) return 0xffffffff;
    tmp = p -> data;
    if(tmp <= x)
    {
        if(tmp == x) return 0;
        return 2 * fun7(p -> right, x) + 1;
    }
    else return 2 * fun7(p -> left, x);
}
```

To get 2 returned, `fun7` should be called with `root` , `root->left` , `root->left->right` (or plus `root->left->right->left`).

The answer is 22(or 20).