Let's use "try" as our random input:

```
That's number 2.  Keep going!
try

Breakpoint 4, 0x0000000000400f15 in phase_3 ()
(gdb) disas
Dump of assembler code for function phase_3:
=> 0x0000000000400f15 <+0>:   sub    $0x18,%rsp
   0x0000000000400f19 <+4>:   mov    %fs:0x28,%rax
   0x0000000000400f22 <+13>:  mov    %rax,0x8(%rsp)
   0x0000000000400f27 <+18>:  xor    %eax,%eax
   0x0000000000400f29 <+20>:  lea    0x4(%rsp),%rcx
   0x0000000000400f2e <+25>:  mov    %rsp,%rdx
   0x0000000000400f31 <+28>:  mov    $0x4025af,%esi  // see what 0x4025af stored
   0x0000000000400f36 <+33>:  call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f3b <+38>:  cmp    $0x1,%eax // it compares to the number of input given
   0x0000000000400f3e <+41>:  jg     0x400f45 <phase_3+48>
   0x0000000000400f40 <+43>:  call   0x40142a <explode_bomb>
   0x0000000000400f45 <+48>:  cmpl   $0x7,(%rsp) //checkes if the first input is less than 7
   0x0000000000400f49 <+52>:  ja     0x400f86 <phase_3+113>
   0x0000000000400f4b <+54>:  mov    (%rsp),%eax
   0x0000000000400f4e <+57>:  jmp    *0x402420(,%rax,8) // jumpes to a particular case
depending to the first input

   0x0000000000400f55 <+64>:  mov    $0xc6,%eax
   0x0000000000400f5a <+69>:  jmp    0x400f97 <phase_3+130>

   0x0000000000400f5c <+71>:  mov    $0x31e,%eax
   0x0000000000400f61 <+76>:  jmp    0x400f97 <phase_3+130>

   0x0000000000400f63 <+78>:  mov    $0x299,%eax
   0x0000000000400f68 <+83>:  jmp    0x400f97 <phase_3+130>

   0x0000000000400f6a <+85>:  mov    $0x3a,%eax
   0x0000000000400f6f <+90>:  jmp    0x400f97 <phase_3+130>

   0x0000000000400f71 <+92>:  mov    $0x270,%eax
   0x0000000000400f76 <+97>:  jmp    0x400f97 <phase_3+130>

   0x0000000000400f78 <+99>:  mov    $0x10b,%eax
   0x0000000000400f7d <+104>: jmp    0x400f97 <phase_3+130>

   0x0000000000400f7f <+106>: mov    $0x80,%eax
   0x0000000000400f84 <+111>: jmp    0x400f97 <phase_3+130>

   0x0000000000400f86 <+113>: call   0x40142a <explode_bomb>

   0x0000000000400f8b <+118>: mov    $0x0,%eax
   0x0000000000400f90 <+123>: jmp    0x400f97 <phase_3+130>

   0x0000000000400f92 <+125>: mov    $0x3f,%eax
```

```
--Type <RET> for more, q to quit, c to continue without paging--
   0x0000000000400f97 <+130>: cmp    0x4(%rsp),%eax //if the second user input is equal to the value in %eax jump
   0x0000000000400f9b <+134>: je     0x400fa2 <phase_3+141>
   0x0000000000400f9d <+136>: call   0x40142a <explode_bomb>
   0x0000000000400fa2 <+141>: mov    0x8(%rsp),%rax
   0x0000000000400fa7 <+146>: xor    %fs:0x28,%rax
   0x0000000000400fb0 <+155>: je     0x400fb7 <phase_3+162>
   0x0000000000400fb2 <+157>: call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400fb7 <+162>: add    $0x18,%rsp
   0x0000000000400fbb <+166>: ret
End of assembler dump.
```

From above code, we can see this is a switch statement. I seperated each case with a space in between. Notice each case ends with
a call to 0x400f97 which compares our second digit with the correct second digit

At first we see that the value of %eax is 0 which means that our input length is not valid so lets move the pointer to <+28> to figure out what 0x4025af is:

```
(gdb) x/s 0x4025af
0x4025af:       "%d %d"
```

Here we see that this the input format

let's give input: 9 3

Now Lets figure out what %rsp holds when pointer is at this instruction:

```
0x0000000000400f45 <+48>:  cmpl   $0x7,(%rsp)

(gdb) x/d $rsp
0x7fffffffdf00: 9
//We see %rsp holds our first digit!
```

```
Dump of assembler code for function phase_3:

   0x0000000000400f36 <+33>:  call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f3b <+38>:  cmp    $0x1,%eax
   0x0000000000400f3e <+41>:  jg     0x400f45 <phase_3+48>
=> 0x0000000000400f40 <+43>:  call   0x40142a <explode_bomb>
   0x0000000000400f45 <+48>:  cmpl   $0x7,(%rsp)
   0x0000000000400f49 <+52>:  ja     0x400f86 <phase_3+113> //This will jump to bomb!!
   0x0000000000400f4b <+54>:  mov    (%rsp),%eax
```

9 will call the bomb!
It looks like we are comparing our first digit. We want it to be greater than one, but less than 7. So lets try "2 3".

```
Dump of assembler code for function phase_3:
   0x0000000000400f15 <+0>:   sub    $0x18,%rsp
   0x0000000000400f19 <+4>:   mov    %fs:0x28,%rax
   0x0000000000400f22 <+13>:  mov    %rax,0x8(%rsp)
   0x0000000000400f27 <+18>:  xor    %eax,%eax
   0x0000000000400f29 <+20>:  lea    0x4(%rsp),%rcx
   0x0000000000400f2e <+25>:  mov    %rsp,%rdx
   0x0000000000400f31 <+28>:  mov    $0x4025af,%esi
   0x0000000000400f36 <+33>:  call   0x400bb0 <__isoc99_sscanf@plt>
=> 0x0000000000400f3b <+38>:  cmp    $0x1,%eax
   0x0000000000400f3e <+41>:  jg     0x400f45 <phase_3+48>
   0x0000000000400f40 <+43>:  call   0x40142a <explode_bomb>
   0x0000000000400f45 <+48>:  cmpl   $0x7,(%rsp)
   0x0000000000400f49 <+52>:  ja     0x400f86 <phase_3+113>
   0x0000000000400f4b <+54>:  mov    (%rsp),%eax
   0x0000000000400f4e <+57>:  jmp    *0x402420(,%rax,8)
   0x0000000000400f55 <+64>:  mov    $0xc6,%eax
   0x0000000000400f5a <+69>:  jmp    0x400f97 <phase_3+130>
   0x0000000000400f5c <+71>:  mov    $0x31e,%eax
   0x0000000000400f61 <+76>:  jmp    0x400f97 <phase_3+130>
   0x0000000000400f63 <+78>:  mov    $0x299,%eax
   0x0000000000400f68 <+83>:  jmp    0x400f97 <phase_3+130>
   0x0000000000400f6a <+85>:  mov    $0x3a,%eax
   0x0000000000400f6f <+90>:  jmp    0x400f97 <phase_3+130>
   0x0000000000400f71 <+92>:  mov    $0x270,%eax
   0x0000000000400f76 <+97>:  jmp    0x400f97 <phase_3+130>
   0x0000000000400f78 <+99>:  mov    $0x10b,%eax
   0x0000000000400f7d <+104>: jmp    0x400f97 <phase_3+130>
   0x0000000000400f7f <+106>: mov    $0x80,%eax
   0x0000000000400f84 <+111>: jmp    0x400f97 <phase_3+130>
   0x0000000000400f86 <+113>: call   0x40142a <explode_bomb>
   0x0000000000400f8b <+118>: mov    $0x0,%eax
   0x0000000000400f90 <+123>: jmp    0x400f97 <phase_3+130>
   0x0000000000400f92 <+125>: mov    $0x3f,%eax
--Type <RET> for more, q to quit, c to continue without paging--
=> 0x0000000000400f97 <+130>: cmp    0x4(%rsp),%eax
   0x0000000000400f9b <+134>: je     0x400fa2 <phase_3+141>
   0x0000000000400f9d <+136>: call   0x40142a <explode_bomb>
   0x0000000000400fa2 <+141>: mov    0x8(%rsp),%rax
   0x0000000000400fa7 <+146>: xor    %fs:0x28,%rax
   0x0000000000400fb0 <+155>: je     0x400fb7 <phase_3+162>
   0x0000000000400fb2 <+157>: call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400fb7 <+162>: add    $0x18,%rsp
   0x0000000000400fbb <+166>: ret

End of assembler dump.
(gdb) i r
rax            0x31e               798
rbx            0x4021e0            4202976
rcx            0x0                 0
```

//As we can see, our digit of 2 gets us pass that first requirement of the digit being between 0 and 7. Now we see that we are comparing
%rax to 0x4(%rsp). Lets see whats in %rsp+4:

```
(gdb) x/d $rsp+4
```

```
0x7fffffffe4a8: 3
```

The second user input is being compared to the value 798 which calls explode bomb. Since we want our second digit to be 798, lets try it with an input of "2 798" --> this works!!

We notice that for first input ranging from 0-7 we have coresponding number which is our second input