Bomblab

1. Phase_1

在 GDB 中运行 bomb 后，首先输入 test，可以发现“The bomb has blown up.”。在 bomb.c 中可以发现，此处调用了 phase_1 函数，在 GDB 中运行 disas phase_1:

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x4023b0,%esi
   0x0000000000400e96 <+9>:     callq  0x40131e <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    callq  0x40141d <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    retq
End of assembler dump.
```

可以发现其中调用了 strings_not_equal 函数，显然是判断两个字符串是否相同。在调用函数前，将一个地址存入了寄存器%esi，这个地址所存储的信息应当为 strings_not_equal 函数的参数之一。通过查看地址 0x4023b0 的存储内容:

```
(gdb) x/s 0x4023b0
0x4023b0:       "Houses will begat jobs, jobs will begat houses."
```

重新运行 bomb，输入“Houses will begat jobs, jobs will begat houses.", 显示:

```
which to blow yourself up. Have a nice day!
Houses will begat jobs, jobs will begat houses.
Phase 1 defused. How about the next one?
```

2. Phase_2

类似的，运行 disas phase_2:

```
(gdb) disas phase_2
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>:    callq  0x40143f <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jns    0x400ed2 <phase_2+41>
   0x0000000000400ecd <+36>:    callq  0x40141d <explode_bomb>
   0x0000000000400ed2 <+41>:    mov    %rsp,%rbp
   0x0000000000400ed5 <+44>:    mov    $0x1,%ebx
   0x0000000000400eda <+49>:    mov    %ebx,%eax
   0x0000000000400edc <+51>:    add    0x0(%rbp),%eax
   0x0000000000400edf <+54>:    cmp    %eax,0x4(%rbp)
   0x0000000000400ee2 <+57>:    je     0x400ee9 <phase_2+64>
   0x0000000000400ee4 <+59>:    callq  0x40141d <explode_bomb>
   0x0000000000400ee9 <+64>:    add    $0x1,%ebx
   0x0000000000400eec <+67>:    add    $0x4,%rbp
   0x0000000000400ef0 <+71>:    cmp    $0x6,%ebx
   0x0000000000400ef3 <+74>:    jne    0x400eda <phase_2+49>
---Type <return> to continue, or q <return> to quit---1 2 3 4 5 6
   0x0000000000400ef5 <+76>:    mov    0x18(%rsp),%rax
   0x0000000000400efa <+81>:    xor    %fs:0x28,%rax
   0x0000000000400f03 <+90>:    je     0x400f0a <phase_2+97>
   0x0000000000400f05 <+92>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0a <+97>:    add    $0x28,%rsp
   0x0000000000400f0e <+101>:   pop    %rbx
   0x0000000000400f0f <+102>:   pop    %rbp
   0x0000000000400f10 <+103>:   retq
```

可以发现调用了函数 read_six_number，而对函数 read_six_number，有:

```
(gdb) disas read_six_numbers
Dump of assembler code for function read_six_numbers:
   0x000000000040143f <+0>:     sub    $0x8,%rsp
   0x0000000000401443 <+4>:     mov    %rsi,%rdx
   0x0000000000401446 <+7>:     lea    0x4(%rsi),%rcx
   0x000000000040144a <+11>:    lea    0x14(%rsi),%rax
   0x000000000040144e <+15>:    push   %rax
   0x000000000040144f <+16>:    lea    0x10(%rsi),%rax
   0x0000000000401453 <+20>:    push   %rax
   0x0000000000401454 <+21>:    lea    0xc(%rsi),%r9
   0x0000000000401458 <+25>:    lea    0x8(%rsi),%r8
   0x000000000040145c <+29>:    mov    $0x4025a3,%esi
   0x0000000000401461 <+34>:    mov    $0x0,%eax
   0x0000000000401466 <+39>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x000000000040146b <+44>:    add    $0x10,%rsp
   0x000000000040146f <+48>:    cmp    $0x5,%eax
   0x0000000000401472 <+51>:    jg     0x401479 <read_six_numbers+58>
   0x0000000000401474 <+53>:    callq  0x40141d <explode_bomb>
   0x0000000000401479 <+58>:    add    $0x8,%rsp
   0x000000000040147d <+62>:    retq
End of assembler dump.
```

其中<+48>与<+51>行表示，当读入的数字超过 5 个时，会跳过<+53>行，也就是不会引爆炸弹，而是将读入的数字存入 phase_2 的 frame 栈中，因此此处应当输入至少 6 个数字才能避免引爆炸弹。

再考虑 phase_2, <+30>与<+34>判断输入的数字是否小于 0, 若小于 0 则炸弹爆炸，否则跳转到<+41>。从<+41>到<+44>，使%rsp 与%rbp 的值相同，并新建了变量 i = 1。从<+49>到<+74>为循环体，<+64>表示每次循环 i++，<+71>与<+74>表示当 i >= 6 时，跳出循环。而在循环体中，每次循环<+49>到<+57>先用第 i 个输入的数字加上 i，再判断它是否等于第 i + 1 个输入的数字，若不相等则炸弹爆炸，否则继续循环，直到 5 次循环结束。也即输入的数字必须满足 a[i + 1] = a[i] + i。

因此整体需要输入的数字必须大于等于六个，令 a[1] = n，其中前六个必须为: n, n + 1, n + 3, n + 6, n + 10, n +15，并且 n >= 0。

重新运行 bomb，输入“0 1 3 6 10 15”，显示:

```
Phase 1 defused. How about the next one?
0 1 3 6 10 15
That's number 2.  Keep going!
```

3. Phase_3

```
(gdb) disas phase_3
Dump of assembler code for function phase_3:
   0x0000000000400f11 <+0>:     sub    $0x18,%rsp
   0x0000000000400f15 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400f1e <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000400f23 <+18>:    xor    %eax,%eax
   0x0000000000400f25 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000400f2a <+25>:    mov    %rsp,%rdx
   0x0000000000400f2d <+28>:    mov    $0x4025af,%esi
   0x0000000000400f32 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f37 <+38>:    cmp    $0x1,%eax
   0x0000000000400f3a <+41>:    jg     0x400f41 <phase_3+48>
   0x0000000000400f3c <+43>:    callq  0x40141d <explode_bomb>
   0x0000000000400f41 <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400f45 <+52>:    ja     0x400f82 <phase_3+113>
   0x0000000000400f47 <+54>:    mov    (%rsp),%eax
   0x0000000000400f4a <+57>:    jmpq   *0x402420(,%rax,8)
   0x0000000000400f51 <+64>:    mov    $0x126,%eax
   0x0000000000400f56 <+69>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f58 <+71>:    mov    $0x32d,%eax
   0x0000000000400f5d <+76>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f5f <+78>:    mov    $0x273,%eax
   0x0000000000400f64 <+83>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f66 <+85>:    mov    $0xfa,%eax
---Type <return> to continue, or q <return> to quit---
   0x0000000000400f6b <+90>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f6d <+92>:    mov    $0x154,%eax
   0x0000000000400f72 <+97>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f74 <+99>:    mov    $0x6a,%eax
   0x0000000000400f79 <+104>:   jmp    0x400f93 <phase_3+130>
   0x0000000000400f7b <+106>:   mov    $0x51,%eax
   0x0000000000400f80 <+111>:   jmp    0x400f93 <phase_3+130>
   0x0000000000400f82 <+113>:   callq  0x40141d <explode_bomb>
   0x0000000000400f87 <+118>:   mov    $0x0,%eax
   0x0000000000400f8c <+123>:   jmp    0x400f93 <phase_3+130>
   0x0000000000400f8e <+125>:   mov    $0x1c7,%eax
   0x0000000000400f93 <+130>:   cmp    0x4(%rsp),%eax
   0x0000000000400f97 <+134>:   je     0x400f9e <phase_3+141>
   0x0000000000400f99 <+136>:   callq  0x40141d <explode_bomb>
   0x0000000000400f9e <+141>:   mov    0x8(%rsp),%rax
   0x0000000000400fa3 <+146>:   xor    %fs:0x28,%rax
   0x0000000000400fac <+155>:   je     0x400fb3 <phase_3+162>
   0x0000000000400fae <+157>:   callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400fb3 <+162>:   add    $0x18,%rsp
   0x0000000000400fb7 <+166>:   retq
End of assembler dump.
```

对<+28>中的地址使用 p (char*) 0x4025af，可以得到需要输入的数据类型:

```
(gdb) p (char*) 0x4025af
$4 = 0x4025af "%d %d"
```

可以发现<+48>到<+52>对输入的第一个数字做了限制，若大于 7，则跳转到<+113>，炸弹爆炸，因为进行了无符号比较，所以输入的第一个数字只能是 0,1,2,3,4,5,6,7。

<+57>到<+110>给出了一个跳转表，对应关系如下:

第一个数字	0	1	2	3	4	5	6	7
地址	0x402420	0x402428	0x402430	0x402438	0x402440	0x402448	0x402450	0x402458
跳转位置	0x400f8e	0x400f51	0x400f58	0x400f5f	0x400f66	0x400f6d	0x400f74	0x400f7b
行数	<+125>	<+64>	<+71>	<+78>	<+85>	<+92>	<+99>	<+106>
%eax	$0x1c7	$0x126	$0x32d	$0x273	$0xfa	$0x154	$0x6a	$0x51
对应十进制	455	294	813	627	250	340	106	81

```
(gdb) p/x *0x402420
$5 = 0x400f8e
(gdb) p/x *0x402428
$6 = 0x400f51
(gdb) p/x *0x402430
$7 = 0x400f58
(gdb) p/x *0x402438
$8 = 0x400f5f
(gdb) p/x *0x402440
$9 = 0x400f66
(gdb) p/x *0x402448
$10 = 0x400f6d
(gdb) p/x *0x402450
$11 = 0x400f74
(gdb) p/x *0x402458
$12 = 0x400f7b
```

因为<+57>采用了*0x402420(, %rax, 8)，而%rax 存储了第一个输入的数，也即{0,1,2,3,4,5,6,7}，因此分别跳转到{*0x402420, *0x402428, ……, *0x402458}，可以如左图查询到其指向的地址，最终得到上表。

在完成跳转后，进入<+130>，会判断输入的第二个数字是否等于%eax 中存储的数字，不相等，炸弹爆炸，否则会跳转到<+141>。如上表所示，即为第一个输入的数字与第二个数字的对应关系。

重新运行 bomb，输入“2 813”，显示：

```
That's number 2.  Keep going!
2 813
Halfway there!
```

4. Phase_4

```
(gdb) disas phase_4
Dump of assembler code for function phase_4:
   0x0000000000400ff3 <+0>:     sub    $0x18,%rsp
   0x0000000000400ff7 <+4>:     mov    %fs:0x28,%rax
   0x0000000000401000 <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000401005 <+18>:    xor    %eax,%eax
   0x0000000000401007 <+20>:    mov    %rsp,%rcx
   0x000000000040100a <+23>:    lea    0x4(%rsp),%rdx
   0x000000000040100f <+28>:    mov    $0x4025af,%esi
   0x0000000000401014 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000401019 <+38>:    cmp    $0x2,%eax
   0x000000000040101c <+41>:    jne    0x401029 <phase_4+54>
   0x000000000040101e <+43>:    mov    (%rsp),%eax
   0x0000000000401021 <+46>:    sub    $0x2,%eax
   0x0000000000401024 <+49>:    cmp    $0x2,%eax
   0x0000000000401027 <+52>:    jbe    0x40102e <phase_4+59>
   0x0000000000401029 <+54>:    callq  0x40141d <explode_bomb>
   0x000000000040102e <+59>:    mov    (%rsp),%esi
   0x0000000000401031 <+62>:    mov    $0x7,%edi
   0x0000000000401036 <+67>:    callq  0x400fb8 <func4>
   0x000000000040103b <+72>:    cmp    0x4(%rsp),%eax
   0x000000000040103f <+76>:    je     0x401046 <phase_4+83>
   0x0000000000401041 <+78>:    callq  0x40141d <explode_bomb>
   0x0000000000401046 <+83>:    mov    0x8(%rsp),%rax
---Type <return> to continue, or q <return> to quit---
   0x000000000040104b <+88>:    xor    %fs:0x28,%rax
   0x0000000000401054 <+97>:    je     0x40105b <phase_4+104>
   0x0000000000401056 <+99>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x000000000040105b <+104>:   add    $0x18,%rsp
   0x000000000040105f <+108>:   retq
End of assembler dump.
```

其中有函数 func4:

```
(gdb) disas func4
Dump of assembler code for function func4:
   0x0000000000400fb8 <+0>:     test   %edi,%edi
   0x0000000000400fba <+2>:     jle    0x400fe7 <func4+47>
   0x0000000000400fbc <+4>:     mov    %esi,%eax
   0x0000000000400fbe <+6>:     cmp    $0x1,%edi
   0x0000000000400fc1 <+9>:     je     0x400ff1 <func4+57>
   0x0000000000400fc3 <+11>:    push   %r12
   0x0000000000400fc5 <+13>:    push   %rbp
   0x0000000000400fc6 <+14>:    push   %rbx
   0x0000000000400fc7 <+15>:    mov    %esi,%ebp
   0x0000000000400fc9 <+17>:    mov    %edi,%ebx
   0x0000000000400fcb <+19>:    lea    -0x1(%rdi),%edi
   0x0000000000400fce <+22>:    callq  0x400fb8 <func4>
   0x0000000000400fd3 <+27>:    lea    0x0(%rbp,%rax,1),%r12d
   0x0000000000400fd8 <+32>:    lea    -0x2(%rbx),%edi
   0x0000000000400fdb <+35>:    mov    %ebp,%esi
   0x0000000000400fdd <+37>:    callq  0x400fb8 <func4>
   0x0000000000400fe2 <+42>:    add    %r12d,%eax
   0x0000000000400fe5 <+45>:    jmp    0x400fed <func4+53>
   0x0000000000400fe7 <+47>:    mov    $0x0,%eax
   0x0000000000400fec <+52>:    retq
   0x0000000000400fed <+53>:    pop    %rbx
   0x0000000000400fee <+54>:    pop    %rbp
   0x0000000000400fef <+55>:    pop    %r12
   0x0000000000400ff1 <+57>:    repz retq
End of assembler dump.
```

与 phase_3 类似，可以发现需要输入的是两个整型数字。

在原函数 phase_4 中，<+43>到<+52>先将输入的第二个数赋给%eax，再将%eax 减去 2，然后与 2 比较大小。若小于等于 2，也即输入的值小于等于 4，则跳转到<+59>，否则炸弹爆炸。因为进行的是无符号比较，因此还要满足%eax 大于等于 2，所以第二个数只能是 2,3,4。随后在<+59>和<+62>中将第二个输入的数赋给了%esi, 将 7 赋给了%edi，并调用 func4，此时的%eax 为 func4(%edi = 7, %esi = m)的返回值。通过比较输入的第一个数是否等于 fun4(7, m)，若不相等，则炸弹爆炸。

观察函数 func4，可以发现首先它会读入%esi 与%edi，也就是第二个输入的数 m 与 7。在<+0>到<+2>中，若%edi = 0，则跳转到<+47>，返回 0，也就是 func4(0, m) = 0。再把输入的%edi 在<+6>与 1 比较大小, 若其等于 1 则会跳转到<+57>, 返回%eax 等于%esi，也即 func4(1, m) = m。否则，用%ebp 和%ebx 分别保存%esi 和%edi 的值，也就是 m 和 7，再在<+19>对%edi 的值减一后递归调用 func4，也就是 func(6, m)，并在<+27>将返回值与%ebp 的值相加后赋给%r12d。然后对%ebx 的值减二后递归调用 func4，也就是 func4(5, m), 再将返回值与%r12d 保存的值相加, 将其作为函数的返回值, 也就是 func4(7, m) = func4(6, m) + func4(5, m) + m。综合考虑，也就是 F(m, n) = F(m - 1, n) + F(m - 2, n) + n，所以 func4(7, m) = func4(6, m) + func4(5, m) + m = func4(5, m) + func4(4, m) + func4(4, m) + func4(3, m) + 3m = 3 * func4(4, m) + 2 * func4(3, m) + 4m = 5 * func4(3, m) + 3 * func4(2, m) + 7m = 8 * func4(2, m) + 5 * func4(1, m) + 12m = 13 * func4(1, m) + 8 * func4(0, m) + 20m = 8 * func4(0, m) + 33m = 33m (5 > m > 1)。因此应输入的数只能是“132 4”或“99 3”或“66 2”。

因此，重新运行 bomb，输入“99 3”，显示:

```
Halfway there!
99 3
So you got that one.  Try this one.
```

5. Phase_5

```
(gdb) disas phase_5
Dump of assembler code for function phase_5:
   0x0000000000401060 <+0>:     sub    $0x18,%rsp
   0x0000000000401064 <+4>:     mov    %fs:0x28,%rax
   0x000000000040106d <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000401072 <+18>:    xor    %eax,%eax
   0x0000000000401074 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000401079 <+25>:    mov    %rsp,%rdx
   0x000000000040107c <+28>:    mov    $0x4025af,%esi
   0x0000000000401081 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000401086 <+38>:    cmp    $0x1,%eax
   0x0000000000401089 <+41>:    jg     0x401090 <phase_5+48>
   0x000000000040108b <+43>:    callq  0x40141d <explode_bomb>
   0x0000000000401090 <+48>:    mov    (%rsp),%eax
   0x0000000000401093 <+51>:    and    $0xf,%eax
   0x0000000000401096 <+54>:    mov    %eax,(%rsp)
   0x0000000000401099 <+57>:    cmp    $0xf,%eax
   0x000000000040109c <+60>:    je     0x4010cd <phase_5+109>
   0x000000000040109e <+62>:    mov    $0x0,%ecx
   0x00000000004010a3 <+67>:    mov    $0x0,%edx
   0x00000000004010a8 <+72>:    add    $0x1,%edx
   0x00000000004010ab <+75>:    cltq
   0x00000000004010ad <+77>:    mov    0x402460(,%rax,4),%eax
   0x00000000004010b4 <+84>:    add    %eax,%ecx
---Type <return> to continue, or q <return> to quit---
   0x00000000004010b6 <+86>:    cmp    $0xf,%eax
   0x00000000004010b9 <+89>:    jne    0x4010a8 <phase_5+72>
   0x00000000004010bb <+91>:    movl   $0xf,(%rsp)
   0x00000000004010c2 <+98>:    cmp    $0xf,%edx
   0x00000000004010c5 <+101>:   jne    0x4010cd <phase_5+109>
   0x00000000004010c7 <+103>:   cmp    0x4(%rsp),%ecx
   0x00000000004010cb <+107>:   je     0x4010d2 <phase_5+114>
   0x00000000004010cd <+109>:   callq  0x40141d <explode_bomb>
   0x00000000004010d2 <+114>:   mov    0x8(%rsp),%rax
   0x00000000004010d7 <+119>:   xor    %fs:0x28,%rax
   0x00000000004010e0 <+128>:   je     0x4010e7 <phase_5+135>
   0x00000000004010e2 <+130>:   callq  0x400b00 <__stack_chk_fail@plt>
   0x00000000004010e7 <+135>:   add    $0x18,%rsp
   0x00000000004010eb <+139>:   retq
End of assembler dump.
```

与 phase_3 和 phase_4 相似，<+28>表示本题同样需要输入两个整型数字。<+38>到<+43>对输入数字的数量进行了限制，若只输入了小于等于 1 个数字，则炸弹爆炸。

在<+48>首先将输入的第一个数字赋给%eax，再将%eax 与 0xf 进行位与操作，也就是只保留输入的数字的最小的十六进制位 m，再把其重新赋给(%rsp)保存的地址。接下来比较%eax，也就是输入的第一个数字的最小十六进制位 m 是否等于 0xf，若等于，炸弹爆炸。否则，在<+62>到<+72>中，为%ecx 赋 0x0，为%edx 中赋 0x0，而后加 1。在<+75>中，做了有符号数的扩展。

在<+77>中，可以猜测内存 0x402460 开始存储了一个数组，经查看：

```
(gdb) p *0x402460@16
$6 = {10, 2, 14, 7, 8, 12, 15, 11, 0, 4, 1, 13, 3, 9, 6, 5}
```

所以这一步是将%eax 的值加上数组中第%eax 个数后赋给%eax，再与%ecx 相加，。

此时再比较%eax 是否等于 0xf，如果不相等，则回到<+72>，令%edx 加一，重新循环。若相等，则跳出循环，在(%rsp)保存的地址中放入 0xf，比较%edx 是否等于 0xf，也就是之前的循环是否进行了 15 次。若不相等，则炸弹爆炸。否则比较%ecx 中存放的数与第二个输入的数字是否相等，若不相等，则炸弹爆炸。相等时，返回正确的值。

```
int i(%edx) = 0, res(%ecx) = 0, k(%eax) = m;
for ( ; ; ) {
    i++;
    k = array[k];
    res = res + k;
    if (k == 15) {
        break;
    }
}
if (i != 15) {
    explode_bomb();
}
```

可如前所述，在本题中，循环必须到%edx 中为 15 且%eax 中也为 15 时才停止，因此逆推 k 的初始值 m，依次是：15,6,14,2,1,10,0,8,4,9,13,11,7,3,12,5，也就是 m 应当等于 5。由于 m 是第一个输入的数的最小的一个十六进制位，因此第一个数应当是 16 * n + 5，其中 n > 0。

而在最终，%ecx 中存放的值为 0x402460 开始的数组中前 16 项的和，经计算，为 115。因此，输入的第二个数为 115。

因此，重新运行 bomb，输入“21 115”，显示：

```
So you got that one.  Try this one.
21 115
Good work!  On to the next...
```

6. Phase_6

```
(gdb) disas phase_6
Dump of assembler code for function phase_6:
   0x00000000004010ec <+0>:     push   %r13
   0x00000000004010ee <+2>:     push   %r12
   0x00000000004010f0 <+4>:     push   %rbp
   0x00000000004010f1 <+5>:     push   %rbx
   0x00000000004010f2 <+6>:     sub    $0x68,%rsp
   0x00000000004010f6 <+10>:    mov    %fs:0x28,%rax
   0x00000000004010ff <+19>:    mov    %rax,0x58(%rsp)
   0x0000000000401104 <+24>:    xor    %eax,%eax
   0x0000000000401106 <+26>:    mov    %rsp,%rsi
   0x0000000000401109 <+29>:    callq  0x40143f <read_six_numbers>
   0x000000000040110e <+34>:    mov    %rsp,%r12
   0x0000000000401111 <+37>:    mov    $0x0,%r13d
   0x0000000000401117 <+43>:    mov    %r12,%rbp
   0x000000000040111a <+46>:    mov    (%r12),%eax
   0x000000000040111e <+50>:    sub    $0x1,%eax
   0x0000000000401121 <+53>:    cmp    $0x5,%eax
   0x0000000000401124 <+56>:    jbe    0x40112b <phase_6+63>
   0x0000000000401126 <+58>:    callq  0x40141d <explode_bomb>
   0x000000000040112b <+63>:    add    $0x1,%r13d
   0x000000000040112f <+67>:    cmp    $0x6,%r13d
   0x0000000000401133 <+71>:    je     0x401172 <phase_6+134>
   0x0000000000401135 <+73>:    mov    %r13d,%ebx
   0x0000000000401138 <+76>:    movslq %ebx,%rax
   0x000000000040113b <+79>:    mov    (%rsp,%rax,4),%eax
   0x000000000040113e <+82>:    cmp    %eax,0x0(%rbp)
   0x0000000000401141 <+85>:    jne    0x401148 <phase_6+92>
   0x0000000000401143 <+87>:    callq  0x40141d <explode_bomb>
   0x0000000000401148 <+92>:    add    $0x1,%ebx
   0x000000000040114b <+95>:    cmp    $0x5,%ebx
   0x000000000040114e <+98>:    jle    0x401138 <phase_6+76>
   0x0000000000401150 <+100>:   add    $0x4,%r12
   0x0000000000401154 <+104>:   jmp    0x401117 <phase_6+43>
   0x0000000000401156 <+106>:   mov    0x8(%rdx),%rdx
   0x000000000040115a <+110>:   add    $0x1,%eax
   0x000000000040115d <+113>:   cmp    %ecx,%eax
   0x000000000040115f <+115>:   jne    0x401156 <phase_6+106>
   0x0000000000401161 <+117>:   mov    %rdx,0x20(%rsp,%rsi,2)
   0x0000000000401166 <+122>:   add    $0x4,%rsi
   0x000000000040116a <+126>:   cmp    $0x18,%rsi
   0x000000000040116e <+130>:   jne    0x401177 <phase_6+139>
   0x0000000000401170 <+132>:   jmp    0x40118b <phase_6+159>
```

可以发现本题同样需要读入至少 6 个数。将读入的第一个数赋给%eax，再令%eax 减 1，比较%eax 是否小于等于 5。若不是，则炸弹爆炸。因为是无符号比较，所以第一个数既要大于等于 1，也要小于等于 6，只能取 1,2,3,4,5,6。从<+63>开始，令%r13d 加一，再比较%r13d 是否等于 6，若相等，跳转到<+134>，否则将其值赋给%eax，再令%eax 等于第(%rax + 1)个输入的数字，若其等于第一个数字，则炸弹爆炸，否则令%ebx 加 1，再比较其是否小于等于 5，若是，跳转回<+76>，否则<%r12>加 4，再跳转回<+43>。

可以发现，这是一个嵌套的循环。整理成 C 语言如下：

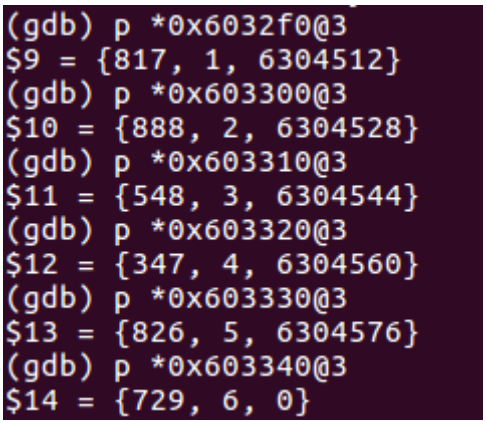
```
   0x0000000000401172 <+134>:   mov    $0x0,%esi
   0x0000000000401177 <+139>:   mov    (%rsp,%rsi,1),%ecx
   0x000000000040117a <+142>:   mov    $0x1,%eax
   0x000000000040117f <+147>:   mov    $0x6032f0,%edx
---Type <return> to continue, or q <return> to quit---
   0x0000000000401184 <+152>:   cmp    $0x1,%ecx
   0x0000000000401187 <+155>:   jg     0x401156 <phase_6+106>
   0x0000000000401189 <+157>:   jmp    0x401161 <phase_6+117>
   0x000000000040118b <+159>:   mov    0x20(%rsp),%rbx
   0x0000000000401190 <+164>:   lea    0x20(%rsp),%rax
   0x0000000000401195 <+169>:   lea    0x48(%rsp),%rsi
   0x000000000040119a <+174>:   mov    %rbx,%rcx
   0x000000000040119d <+177>:   mov    0x8(%rax),%rdx
   0x00000000004011a1 <+181>:   mov    %rdx,0x8(%rcx)
   0x00000000004011a5 <+185>:   add    $0x8,%rax
   0x00000000004011a9 <+189>:   mov    %rdx,%rcx
   0x00000000004011ac <+192>:   cmp    %rsi,%rax
   0x00000000004011af <+195>:   jne    0x40119d <phase_6+177>
   0x00000000004011b1 <+197>:   movq   $0x0,0x8(%rdx)
   0x00000000004011b9 <+205>:   mov    $0x5,%ebp
   0x00000000004011be <+210>:   mov    0x8(%rbx),%rax
   0x00000000004011c2 <+214>:   mov    (%rax),%eax
   0x00000000004011c4 <+216>:   cmp    %eax,(%rbx)
   0x00000000004011c6 <+218>:   jge    0x4011cd <phase_6+225>
   0x00000000004011c8 <+220>:   callq  0x40141d <explode_bomb>
   0x00000000004011cd <+225>:   mov    0x8(%rbx),%rbx
   0x00000000004011d1 <+229>:   sub    $0x1,%ebp
   0x00000000004011d4 <+232>:   jne    0x4011be <phase_6+210>
   0x00000000004011d6 <+234>:   mov    0x58(%rsp),%rax
   0x00000000004011db <+239>:   xor    %fs:0x28,%rax
   0x00000000004011e4 <+248>:   je     0x4011eb <phase_6+255>
   0x00000000004011e6 <+250>:   callq  0x400b00 <__stack_chk_fail@plt>
   0x00000000004011eb <+255>:   add    $0x68,%rsp
   0x00000000004011ef <+259>:   pop    %rbx
   0x00000000004011f0 <+260>:   pop    %rbp
   0x00000000004011f1 <+261>:   pop    %r12
   0x00000000004011f3 <+263>:   pop    %r13
   0x00000000004011f5 <+265>:   retq
End of assembler dump.
```

```
int i = 0;
for ( ; ; ) {
    i++;
    if (i == 6) {
        break;
    }
    if (array[i] <=6 && array[i] >= 1) {
        for (int k = i; ; ) {
            k++;
            if (array[k] == array[i]) {
                explode_bomb();
            } else {
                if (k > 5) {
                    break;
                }
            }
```

```
            }
        } else {
            explode_bomb();
        }
    }
```

也就是说，输入的前 6 个数必须满足：这些数互不相等，且都大于等于 1 小于等于 6。所以输入的六个数应当是 1,2,3,4,5,6 的一个全排列。

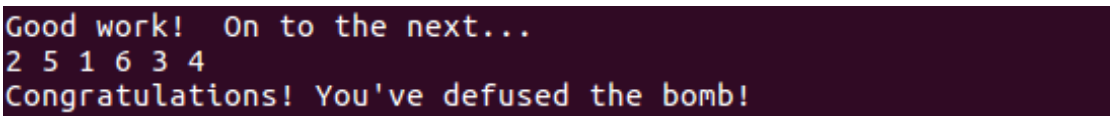
继续向后观察，可以发现在<+147>中将内存 0x6032f0 的内容赋给了%edx。经过检验可以发现，从 0x6032f0 开始存放了一个链表：



地址	0x6032f0	0x603300	0x603310	0x603320	0x603330	0x603340
数字 1	817	888	548	347	826	729
数字 2	1	2	3	4	5	6
后一节点	6304512	6304528	6304544	6304560	6304576	0
十六进制	0x603300	0x603310	0x603320	0x603330	0x603340	0x0

而在此之后，所做的便是改变了各个节点的指针域，使各个节点按照输入的数字 2 的顺序连接。并要求按此顺序连接时，数字 1 中的数字应当是单调递减的。因此顺序即为：2(888), 5(826), 1(817), 6(729), 3(548), 4(347)。

因此，重新运行 bomb，输入“2 5 1 6 3 4”，显示：



综上:

phase_1 的解为：“Houses will begat jobs, jobs will begat houses.”;
phase_2 的解为：“n　n+1　n+3　n+6　n+10　n+15”，其中 n 为大于等于 0 的整数;
phase_3 的解为：“0 455”/“1 294”/“2 813”/“3 627”/“4 250”/“5 340”/“6 106”/“7 81”;
phase_4 的解为：“132 4”/“99 3”/“66 2”;
phase_5 的解为：“16*n+5　115”，其中 n 为大于等于 1 的整数;
phase_6 的解为：“2 5 1 6 3 4”。