Bomb Lab 实验报告

Phase 1

汇编代码如下

```
0000000000001419 <phase_1>:
    1419:    48 83 ec 08              sub    $0x8,%rsp
    141d:    48 8d 35 2c 1d 00 00     lea    0x1d2c(%rip),%rsi        # 3150 <_IO_stdin_used+0x150>
    1424:    e8 02 05 00 00           call   192b <strings_not_equal>
    1429:    85 c0                    test   %eax,%eax
    142b:    75 05                    jne    1432 <phase_1+0x19>
    142d:    48 83 c4 08              add    $0x8,%rsp
    1431:    c3                       ret
    1432:    e8 f4 05 00 00           call   1a2b <explode_bomb>
    1437:    eb f4                    jmp    142d <phase_1+0x14>
```

阅读代码可知，1424行调用了strings_not_equal函数，猜测是将输入的字符串与某字符串相比较。strings_not_equal两个参数：%rdi在main中调用phase_1前就被赋值了，也就是我们的输入字符串；%rsi在141d行处被赋值，估计即为答案。gdb调试得到%rsi的值为 “I am the mayor. I can do anything I want.”，测试答案正确。

```
B+  0x555555555419 <phase_1>        sub    $0x8,%rsp
    0x55555555541d <phase_1+4>      lea    0x1d2c(%rip),%rsi        # 0x555555557150
  > 0x555555555424 <phase_1+11>     call   0x55555555592b <strings_not_equal>
    0x555555555429 <phase_1+16>     test   %eax,%eax
    0x55555555542b <phase_1+18>     jne    0x555555555432 <phase_1+25>
    0x55555555542d <phase_1+20>     add    $0x8,%rsp
    0x555555555431 <phase_1+24>     ret
    0x555555555432 <phase_1+25>     call   0x555555555a2b <explode_bomb>

multi-thre Thread 0x7ffff7d8b7 In: phase_1
(gdb) ni
0x0000555555555424 in phase_1 ()
(gdb) x/s $rsi
0x555555557150: "I am the mayor. I can do anything I want."
(gdb)
```

Phase 2

汇编代码如下

```
0000000000001439 <phase_2>:
    1439:    55                       push   %rbp
    143a:    53                       push   %rbx
    143b:    48 83 ec 28              sub    $0x28,%rsp
    143f:    48 89 e6                 mov    %rsp,%rsi
    1442:    e8 0a 06 00 00           call   1a51 <read_six_numbers>
    1447:    83 3c 24 01              cmpl   $0x1,(%rsp)
    144b:    75 0a                    jne    1457 <phase_2+0x1e>
    144d:    48 89 e3                 mov    %rsp,%rbx
    1450:    48 8d 6c 24 14           lea    0x14(%rsp),%rbp
    1455:    eb 10                    jmp    1467 <phase_2+0x2e>
    1457:    e8 cf 05 00 00           call   1a2b <explode_bomb>
    145c:    eb ef                    jmp    144d <phase_2+0x14>
    145e:    48 83 c3 04              add    $0x4,%rbx
    1462:    48 39 eb                 cmp    %rbp,%rbx
    1465:    74 10                    je     1477 <phase_2+0x3e>
    1467:    8b 03                    mov    (%rbx),%eax
    1469:    01 c0                    add    %eax,%eax
    146b:    39 43 04                 cmp    %eax,0x4(%rbx)
    146e:    74 ee                    je     145e <phase_2+0x25>
    1470:    e8 b6 05 00 00           call   1a2b <explode_bomb>
    1475:    eb e7                    jmp    145e <phase_2+0x25>
    1477:    48 83 c4 28              add    $0x28,%rsp
    147b:    5b                       pop    %rbx
    147c:    5d                       pop    %rbp
    147d:    c3                       ret
```

1442行调用read_six_numbers函数，可知本题答案为6个数，且读入的6个数存在%rsp开头的数组中。1447行显示第一个数应为1（否则会引爆炸弹），后145e-1470行为一个循环，遍历输入数组，其中1469行为将当前的数*2与数组中下一个数比较，若不相同则引爆炸弹。如此可知输入的每个数都应为前一个数2倍，即答案为“1 2 4 8 16 32”。

Phase 3

注：汇编代码为节选

```
000000000000147e <phase_3>:
    147e:    48 83 ec 18              sub    $0x18,%rsp
    1482:    48 8d 4c 24 07           lea    0x7(%rsp),%rcx
    1487:    48 8d 54 24 0c           lea    0xc(%rsp),%rdx
    148c:    4c 8d 44 24 08           lea    0x8(%rsp),%r8
    1491:    48 8d 35 0e 1d 00 00     lea    0x1d0e(%rip),%rsi        # 31a6 <_IO_stdin_used+0x1a6>
    1498:    b8 00 00 00 00           mov    $0x0,%eax
    149d:    e8 9e fc ff ff           call   1140 <__isoc99_sscanf@plt>
    14a2:    83 f8 02                 cmp    $0x2,%eax
    14a5:    7e 1f                    jle    14c6 <phase_3+0x48>
    14a7:    83 7c 24 0c 07           cmpl   $0x7,0xc(%rsp)
    14ac:    0f 87 0c 01 00 00        ja     15be <phase_3+0x140>
    14b2:    8b 44 24 0c              mov    0xc(%rsp),%eax
    14b6:    48 8d 15 03 1d 00 00     lea    0x1d03(%rip),%rdx        # 31c0 <_IO_stdin_used+0x1c0>
    14bd:    48 63 04 82              movslq (%rdx,%rax,4),%rax
    14c1:    48 01 d0                 add    %rdx,%rax
    14c4:    ff e0                    jmp    *%rax
    14c6:    e8 60 05 00 00           call   1a2b <explode_bomb>
    14cb:    eb da                    jmp    14a7 <phase_3+0x29>
    14cd:    b8 6b 00 00 00           mov    $0x6b,%eax
    14d2:    81 7c 24 08 f6 02 00     cmpl   $0x2f6,0x8(%rsp)
    14d9:    00
    14da:    0f 84 e8 00 00 00        je     15c8 <phase_3+0x14a>
    14e0:    e8 46 05 00 00           call   1a2b <explode_bomb>
```

```
    15c8:    38 44 24 07              cmp    %al,0x7(%rsp)
    15cc:    75 05                    jne    15d3 <phase_3+0x155>
    15ce:    48 83 c4 18              add    $0x18,%rsp
    15d2:    c3                       ret
    15d3:    e8 53 04 00 00           call   1a2b <explode_bomb>
    15d8:    eb f4                    jmp    15ce <phase_3+0x150>
```

第149d行使用了sscanf，即读入数据，观察前面的指令，我尝试gdb输出了%rsi，发现结果为“%d %c %d”——即ssanf的占位符，那么要输入的数据即为"int char int"，且分别存在0xc(%rsp),0x7(%rsp),0x8(%rsp)。

```
(gdb) x/s $rsi
0x5555555571a6: "%d %c %d"
```

然后先判断读入数据个数（%eax）大于2，接着判断第一个数小于7。接下来14b2-14c4行进入switch控制部分，控制依据为第一个数字。我假设第一个数为0走了一下流程，发现需要满足第三个数据（int）为758（第14d2行），且第二个数据（char）为107（第15c8行），即'k'时，才能不引爆炸弹。第15c8行的%al通过gdb输出得知。故答案为"0 k 758"

```
(gdb) p $al
$1 = 107
```

其实走其他分支也能得到其他答案，但给出一个答案即可，这里就不赘述了。

Phase 4

```
0000000000001617 <phase_4>:
    1617:    48 83 ec 18             sub    $0x18,%rsp
    161b:    48 8d 4c 24 08          lea    0x8(%rsp),%rcx
    1620:    48 8d 54 24 0c          lea    0xc(%rsp),%rdx
    1625:    48 8d 35 03 1d 00 00    lea    0x1d03(%rip),%rsi        # 332f <array.0+0x14f>
    162c:    b8 00 00 00 00          mov    $0x0,%eax
    1631:    e8 0a fb ff ff          call   1140 <__isoc99_sscanf@plt>
    1636:    83 f8 02                cmp    $0x2,%eax
    1639:    75 07                   jne    1642 <phase_4+0x2b>
    163b:    83 7c 24 0c 0e          cmpl   $0xe,0xc(%rsp)
    1640:    76 05                   jbe    1647 <phase_4+0x30>
    1642:    e8 e4 03 00 00          call   1a2b <explode_bomb>
    1647:    ba 0e 00 00 00          mov    $0xe,%edx
    164c:    be 00 00 00 00          mov    $0x0,%esi
    1651:    8b 7c 24 0c             mov    0xc(%rsp),%edi
    1655:    e8 80 ff ff ff          call   15da <func4>
    165a:    83 f8 06                cmp    $0x6,%eax
    165d:    75 07                   jne    1666 <phase_4+0x4f>
    165f:    83 7c 24 08 06          cmpl   $0x6,0x8(%rsp)
    1664:    74 05                   je     166b <phase_4+0x54>
    1666:    e8 c0 03 00 00          call   1a2b <explode_bomb>
    166b:    48 83 c4 18             add    $0x18,%rsp
    166f:    c3                      ret
```

有了phase 3的基础，依旧gdb查看%rsi，为"%d %d"，即输入两个int。然后调用了func4，三个参数依次为：输入的第一个数，0，以及14，要求func4的返回值为6。最后（第165f行），要求第二个数为6。查看func4的汇编代码：

```
00000000000015da <func4>:
    15da:    48 83 ec 08             sub    $0x8,%rsp
    15de:    89 d0                   mov    %edx,%eax
    15e0:    29 f0                   sub    %esi,%eax
    15e2:    89 c1                   mov    %eax,%ecx
    15e4:    c1 e9 1f                shr    $0x1f,%ecx
    15e7:    01 c1                   add    %eax,%ecx
    15e9:    d1 f9                   sar    %ecx
    15eb:    01 f1                   add    %esi,%ecx
    15ed:    39 f9                   cmp    %edi,%ecx
    15ef:    7f 0c                   jg     15fd <func4+0x23>
    15f1:    b8 00 00 00 00          mov    $0x0,%eax
    15f6:    7c 11                   jl     1609 <func4+0x2f>
    15f8:    48 83 c4 08             add    $0x8,%rsp
    15fc:    c3                      ret
    15fd:    8d 51 ff                lea    -0x1(%rcx),%edx
    1600:    e8 d5 ff ff ff          call   15da <func4>
    1605:    01 c0                   add    %eax,%eax
    1607:    eb ef                   jmp    15f8 <func4+0x1e>
    1609:    8d 71 01                lea    0x1(%rcx),%esi
    160c:    e8 c9 ff ff ff          call   15da <func4>
    1611:    8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
    1615:    eb e1                   jmp    15f8 <func4+0x1e>
```

这是一个递归函数，简单还原成c，大致如下：

```c
int func4(int a1, int a2, int a3)
{
  int t; // %ecx
  int result; // %rax

  t = a2 + (a3 - a2) / 2;
  if ( t > a1 )
    return 2 * func4(a1, a2, t - 1);
  result = 0;
  if ( t < a1 )
    return 2 * func4(a1, t + 1, a3) + 1;
  return result;
}
```

代入a2=0 a3=14，尝试一下就可以得到一个解a1=6。于是一个答案即为"6 6"。

Phase 5

```
0000000000001670 <phase_5>:
    1670:    48 83 ec 18              sub    $0x18,%rsp
    1674:    48 8d 4c 24 08           lea    0x8(%rsp),%rcx
    1679:    48 8d 54 24 0c           lea    0xc(%rsp),%rdx
    167e:    48 8d 35 aa 1c 00 00     lea    0x1caa(%rip),%rsi        # 332f <array.0+0x14f>
    1685:    b8 00 00 00 00           mov    $0x0,%eax
    168a:    e8 b1 fa ff ff           call   1140 <__isoc99_sscanf@plt>
    168f:    83 f8 01                 cmp    $0x1,%eax
    1692:    7e 4d                    jle    16e1 <phase_5+0x71>
    1694:    8b 44 24 0c              mov    0xc(%rsp),%eax
    1698:    83 e0 0f                 and    $0xf,%eax
    169b:    89 44 24 0c              mov    %eax,0xc(%rsp)
    169f:    83 f8 0f                 cmp    $0xf,%eax
    16a2:    74 33                    je     16d7 <phase_5+0x67>
    16a4:    b9 00 00 00 00           mov    $0x0,%ecx
    16a9:    ba 00 00 00 00           mov    $0x0,%edx
    16ae:    48 8d 35 2b 1b 00 00     lea    0x1b2b(%rip),%rsi        # 31e0 <array.0>
    16b5:    83 c2 01                 add    $0x1,%edx
    16b8:    48 98                    cltq
    16ba:    8b 04 86                 mov    (%rsi,%rax,4),%eax
    16bd:    01 c1                    add    %eax,%ecx
    16bf:    83 f8 0f                 cmp    $0xf,%eax
    16c2:    75 f1                    jne    16b5 <phase_5+0x45>
    16c4:    c7 44 24 0c 0f 00 00     movl   $0xf,0xc(%rsp)
    16cb:    00
    16cc:    83 fa 0f                 cmp    $0xf,%edx
    16cf:    75 06                    jne    16d7 <phase_5+0x67>
    16d1:    39 4c 24 08              cmp    %ecx,0x8(%rsp)
    16d5:    74 05                    je     16dc <phase_5+0x6c>
    16d7:    e8 4f 03 00 00           call   1a2b <explode_bomb>
    16dc:    48 83 c4 18              add    $0x18,%rsp
    16e0:    c3                       ret
    16e1:    e8 45 03 00 00           call   1a2b <explode_bomb>
    16e6:    eb ac                    jmp    1694 <phase_5+0x24>
```

同上，仍然gdb查看%rsi，为"%d %d"，则知本题答案为两个数。

1694-16a2行中，相当于把第一个数的后四位取出来（与15 and），若为15则引爆炸弹。

接下来进入一个循环，c代码大致如下：

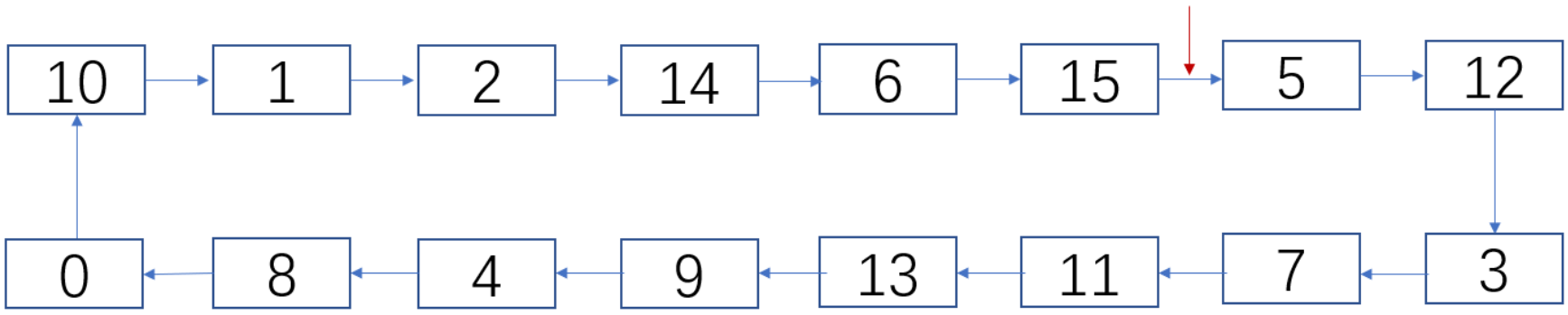
```
t=a1&15;// %eax
sum=0;// %ecx
while(t!=15){
    ++num;// %edx
    t=array[t];
    sum+=t;
}
if(num!=15 || sum!=a2)explode_bomb();
```

其中array的地址在16ae行传入%rsi，于是在gdb中打印出array：

```
   0x5555555556a4 <phase_5+52>     mov    $0x0,%ecx
   0x5555555556a9 <phase_5+57>     mov    $0x0,%edx
   0x5555555556ae <phase_5+62>     lea    0x1b2b(%rip),%rsi        # 0x5555555571e0 <array.0>
 > 0x5555555556b5 <phase_5+69>     add    $0x1,%edx
   0x5555555556b8 <phase_5+72>     cltq
   0x5555555556ba <phase_5+74>     mov    (%rsi,%rax,4),%eax
   0x5555555556bd <phase_5+77>     add    %eax,%ecx
   0x5555555556bf <phase_5+79>     cmp    $0xf,%eax
   0x5555555556c2 <phase_5+82>     jne    0x5555555556b5 <phase_5+69>
   0x5555555556c4 <phase_5+84>     movl   $0xf,0xc(%rsp)

multi-thre Thread 0x7ffff7d8b7 In: phase_5
(gdb) x/16wd $rsi
0x5555555571e0 <array.0>:       10      2       14      7
0x5555555571f0 <array.0+16>:    8       12      15      11
0x555555557200 <array.0+32>:    0       4       1       13
0x555555557210 <array.0+48>:    3       9       6       5
```

发现array刚好满足一个环状链表结构，那么要满足num==15时结束循环，t就应该从5处开始，累加可得a2=115。由此，一个答案为"5 115"（其实所有二进制后四位是0101的数都可以做第一个数）。



Phase 6

本题汇编代码较长，故分开处理。

```
00000000000016e8 <phase_6>:
    16e8:       41 56                   push   %r14
    16ea:       41 55                   push   %r13
    16ec:       41 54                   push   %r12
    16ee:       55                      push   %rbp
    16ef:       53                      push   %rbx
    16f0:       48 83 ec 50             sub    $0x50,%rsp
    16f4:       4c 8d 74 24 30          lea    0x30(%rsp),%r14
    16f9:       4c 89 f6                mov    %r14,%rsi
    16fc:       e8 50 03 00 00          call   1a51 <read_six_numbers>
```

首先是read_six_numbers，可知本题答案为六个int。

然后进入一个双重循环，汇编代码：

```
    1701:       41 bd 01 00 00 00       mov    $0x1,%r13d
    1707:       4d 89 f4                mov    %r14,%r12
    170a:       e9 ad 00 00 00          jmp    17bc <phase_6+0xd4>
    ......
    1791:       48 83 c3 01             add    $0x1,%rbx
    1795:       83 fb 05                cmp    $0x5,%ebx
    1798:       7f 10                   jg     17aa <phase_6+0xc2>
    179a:       41 8b 04 9c             mov    (%r12,%rbx,4),%eax
    179e:       39 45 00                cmp    %eax,0x0(%rbp)
    17a1:       75 ee                   jne    1791 <phase_6+0xa9>
    17a3:       e8 83 02 00 00          call   1a2b <explode_bomb>
    17a8:       eb e7                   jmp    1791 <phase_6+0xa9>
    17aa:       49 83 c6 04             add    $0x4,%r14
    17ae:       49 83 c5 01             add    $0x1,%r13
    17b2:       49 83 fd 07             cmp    $0x7,%r13
    17b6:       0f 84 62 ff ff ff       je     171e <phase_6+0x36>
    17bc:       4c 89 f5                mov    %r14,%rbp
    17bf:       41 8b 06                mov    (%r14),%eax
    17c2:       83 e8 01                sub    $0x1,%eax
    17c5:       83 f8 05                cmp    $0x5,%eax
    17c8:       0f 87 41 ff ff ff       ja     170f <phase_6+0x27>
    17ce:       41 83 fd 05             cmp    $0x5,%r13d
    17d2:       7f d6                   jg     17aa <phase_6+0xc2>
    17d4:       4c 89 eb                mov    %r13,%rbx
    17d7:       eb c1                   jmp    179a <phase_6+0xb2>
```

翻译一下：

```
int *p=a;// %rbp or %r14，初始为a的头指针（a即输入数组，从0x30(%rsp)开始）
for(int i=1;i!=7;++i,++p){//i为%r13
    if((unsigned int)*p-1>5)explode_bomb();//汇编代码为ja(170f行)
    else if(i<=5){
        for(int j=i;j<=5;j++)//j为%rbx
            if(a[j]==*p)explode_bomb();
    }
}
```

很显然，这段代码意在保证输入数据不重复且均<=6，那么即为1-6的某种排列。其中，++p对应17aa行的 `add $0x4,%r14`

然后又是一个双重循环，形如：

```
    171e:    be 00 00 00 00          mov    $0x0,%esi
    1723:    8b 4c b4 30             mov    0x30(%rsp,%rsi,4),%ecx
    1727:    b8 01 00 00 00          mov    $0x1,%eax
    172c:    48 8d 15 bd 3b 00 00    lea    0x3bbd(%rip),%rdx        # 52f0 <node1>
    1733:    83 f9 01                cmp    $0x1,%ecx
    1736:    7e 0b                   jle    1743 <phase_6+0x5b>
    1738:    48 8b 52 08             mov    0x8(%rdx),%rdx
    173c:    83 c0 01                add    $0x1,%eax
    173f:    39 c8                   cmp    %ecx,%eax
    1741:    75 f5                   jne    1738 <phase_6+0x50>
    1743:    48 89 14 f4             mov    %rdx,(%rsp,%rsi,8)
    1747:    48 83 c6 01             add    $0x1,%rsi
    174b:    48 83 fe 06             cmp    $0x6,%rsi
    174f:    75 d2                   jne    1723 <phase_6+0x3b>
```

写成c代码：

```c
for(int i=0;i!=6;++i){//i为%rsi
    p=a[i];// %ecx
    q=1;// %eax
    r=&node1;// %rdx
    if(p>1)
        for(;q!=p;++q)
            r=r->next;
    b[i]=r;//b从%rsp开始
}
```

这里面涉及到了名叫node的数据结构，gdb打印出172c行给出的地址，观察发现每个node有三个值：id（自己的编号）,val和next，其中next存了下一个node的地址

如下图，前五个node刚好挨着，打印出node5.next发现即为node6

```
(gdb) x/20w $rdx
0x5555555592f0 <node1>: 0x00000168      0x00000001      0x55559300      0x00005555
0x555555559300 <node2>: 0x000003dc      0x00000002      0x55559310      0x00005555
0x555555559310 <node3>: 0x000002fb      0x00000003      0x55559320      0x00005555
0x555555559320 <node4>: 0x000002ce      0x00000004      0x55559330      0x00005555
0x555555559330 <node5>: 0x0000020b      0x00000005      0x555591f0      0x00005555
```

```
(gdb) x/4x 0x5555555591f0
0x5555555591f0 <node6>: 0x000003d6      0x00000006      0x00000000      0x00000000
```

那么上述二重循环的作用就是按照输入的顺序在b数组内存放node对应顺序的指针。

接下来

```
    1751:    48 8b 1c 24             mov    (%rsp),%rbx
    1755:    48 8b 44 24 08          mov    0x8(%rsp),%rax
    175a:    48 89 43 08             mov    %rax,0x8(%rbx)
    175e:    48 8b 54 24 10          mov    0x10(%rsp),%rdx
    1763:    48 89 50 08             mov    %rdx,0x8(%rax)
    1767:    48 8b 44 24 18          mov    0x18(%rsp),%rax
    176c:    48 89 42 08             mov    %rax,0x8(%rdx)
    1770:    48 8b 54 24 20          mov    0x20(%rsp),%rdx
    1775:    48 89 50 08             mov    %rdx,0x8(%rax)
    1779:    48 8b 44 24 28          mov    0x28(%rsp),%rax
    177e:    48 89 42 08             mov    %rax,0x8(%rdx)
    1782:    48 c7 40 08 00 00 00    movq   $0x0,0x8(%rax)
```

这段作用是也按照b数组顺序，调整node的next项，让他们的链接关系也与输入顺序相同。

最后一段

```
   178a:    bd 05 00 00 00          mov    $0x5,%ebp
   178f:    eb 51                   jmp    17e2 <phase_6+0xfa>
   ......
   17d9:    48 8b 5b 08             mov    0x8(%rbx),%rbx
   17dd:    83 ed 01                sub    $0x1,%ebp
   17e0:    74 11                   je     17f3 <phase_6+0x10b>
   17e2:    48 8b 43 08             mov    0x8(%rbx),%rax
   17e6:    8b 00                   mov    (%rax),%eax
   17e8:    39 03                   cmp    %eax,(%rbx)
   17ea:    7e ed                   jle    17d9 <phase_6+0xf1>
   17ec:    e8 3a 02 00 00          call   1a2b <explode_bomb>
   17f1:    eb e6                   jmp    17d9 <phase_6+0xf1>
```

依旧是一个循环，翻译一下：

```
node *p=b[0];// %rbx
for(int i=5;i>0;--i){
    p1=p->next;
    x=p1->val;
    if(p->val<=x)p=p->next;
    else explode_bomb();
}
```

这段代码也是很易懂的，就是要保证按照输入顺序重新排序后的node按val递增。

那么我们的答案就是node编号按val大小从小到大的顺序，即"1 5 4 3 6 2"（node.val在上面gdb截图中）。

感想

感觉做这个lab最锻炼我的是耐心orz...其实只要耐心看进去，即使是最难的phase 6也不是说无迹可寻，但是一上来看见那么长的一段汇编真的是头都大了。

然后就是gdb的能力了吧，其实之前一直用的vscode图形化调试，这次第一次自己上手。开始的时候因为懒，就直接用了实验指导的操作（x/s），也没管x是什么意思，s是什么意思。后来忘了是哪个phase了，一直输出不出来想要的结果，才不得不去看了gdb教程。

不得不说，这个lab真的是环环相扣、引人深入，难度梯度很好而且不会觉得难以上手（还要感谢助教写的实验指导，真的很有帮助）。

整个实验给我印象最深的应该是phase 6的node，虽然它名字提示已经很足够了，但是我当时不知道怎么回事完全没往结构体方向想。意识到了之后，发现next就水到渠成了，验证node5.next就是node6的时候成就感确实很强。