Lab2.bomblab

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
```

%edi 由 main 函数传入，指向输入的第一条 phase .

%esi 指向 mem[0x402400]，然后调用函数 <strings_not_equal> 判断所指 str，不相等则 bomb.

通过 gdb 获取字符串:

(gdb) x /1bs 0x402400

0x402400: "Border relations with Canada have never been better." (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)
  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>
  400f17: 8b 43 fc              mov    -0x4(%rbx),%eax
  400f1a: 01 c0                 add    %eax,%eax
  400f1c: 39 03                 cmp    %eax,(%rbx)
  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
  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
  400f3a: eb db                 jmp    400f17 <phase_2+0x1b>
  400f3c: 48 83 c4 28           add    $0x28,%rsp
  400f40: 5b                    pop    %rbx
  400f41: 5d                    pop    %rbp
```

```
  400f42: c3                    retq
```

%rsp 寄存器的值减 0x28，相当于在栈上开辟了 40 个字节，相当于开辟了一个数组
将此时的 %rsp 传给 %rsi 然后调用函数 <read_six_numbers>
比较数组第一个值是否等于 0x1 如果不相等就 bomb
如果相等跳转，将 %rbx 的值指向第二个元素，如果第二个元素是前一个元素的两倍则程序继续，否则 bomb
如此继续循环，直到 %rbx 的值与 %rbp (%rsp + 0x18) 相等，循环结束.说明六个数为首项为 1，公比为 2 的等比数列.
phase_2 = 1 2 4 8 16 32

phase_3:

```
0000000000400f43 <phase_3>:
  400f43: 48 83 ec 18           sub    $0x18,%rsp # 开辟栈空间
  400f47: 48 8d 4c 24 0c        lea    0xc(%rsp),%rcx # sscanf 的第四个参数
  400f4c: 48 8d 54 24 08        lea    0x8(%rsp),%rdx # sscanf 的第三个参数
  400f51: be cf 25 40 00        mov    $0x4025cf,%esi # sscanf 的第二个参数
  400f56: b8 00 00 00 00        mov    $0x0,%eax # 返回值先置为 0
  400f5b: e8 90 fc ff ff        callq  400bf0 <__isoc99_sscanf@plt> # 调用 sscanf 函数
  400f60: 83 f8 01              cmp    $0x1,%eax # 判断 返回值 和 1 的大小
  400f63: 7f 05                 jg     400f6a <phase_3+0x27> # 如果 返回值 > 1 则跳转, 否则 bomb
  400f65: e8 d0 04 00 00        callq  40143a <explode_bomb> # bomb !
  400f6a: 83 7c 24 08 07        cmpl   $0x7,0x8(%rsp) #判断第三个参数(输入的第一个数字)和 7 的大小
  400f6f: 77 3c                 ja     400fad <phase_3+0x6a> # 如果 输入的第一个数字 > 7 bomb
  400f71: 8b 44 24 08           mov    0x8(%rsp),%eax # 将输入的第一个数字传给 %eax
  400f75: ff 24 c5 70 24 40 00  jmpq   *0x402470(,%rax,8) # gdb x /8w 0x402470  查看 switch 表
  400f7c: b8 cf 00 00 00        mov    $0xcf,%eax # 输入的第一个数为 0 入口
  400f81: eb 3b                 jmp    400fbe <phase_3+0x7b>
  400f83: b8 c3 02 00 00        mov    $0x2c3,%eax # 输入的第一个数为 2 入口
  400f88: eb 34                 jmp    400fbe <phase_3+0x7b>
  400f8a: b8 00 01 00 00        mov    $0x100,%eax # 输入的第一个数为 3 入口
  400f8f: eb 2d                 jmp    400fbe <phase_3+0x7b>
  400f91: b8 85 01 00 00        mov    $0x185,%eax # 输入的第一个数为 4 入口
  400f96: eb 26                 jmp    400fbe <phase_3+0x7b>
  400f98: b8 ce 00 00 00        mov    $0xce,%eax # 输入的第一个数为 5 入口
  400f9d: eb 1f                 jmp    400fbe <phase_3+0x7b>
  400f9f: b8 aa 02 00 00        mov    $0x2aa,%eax # 输入的第一个数为 6 入口
  400fa4: eb 18                 jmp    400fbe <phase_3+0x7b>
  400fa6: b8 47 01 00 00        mov    $0x147,%eax # 输入的第一个数为 7 入口
  400fab: eb 11                 jmp    400fbe <phase_3+0x7b>
  400fad: e8 88 04 00 00        callq  40143a <explode_bomb> # 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 # 输入第一个数为 1 入口
  400fbe: 3b 44 24 0c           cmp    0xc(%rsp),%eax # 比较输入的第二个数和 %eax 大小
  400fc2: 74 05                 je     400fc9 <phase_3+0x86> # 如果相等就 ok, 否则 bomb
  400fc4: e8 71 04 00 00        callq  40143a <explode_bomb>
  400fc9: 48 83 c4 18           add    $0x18,%rsp
  400fcd: c3                    retq
```

可以得知传给 sscanf 函数的模式串指针为 0x4024cf，通过 gdb 获取可得:

(gdb) x /1bs 0x4025cf

0x4025cf: "%d %d"

而 sscanf 函数的第一个参数就是所输入的 phase_3，可以推测其为两个整数

如果输入的不是两个整数则直接 bomb

得到第一个整数如果 大于 7 直接 bomb

通过第一个整数查找 switch 跳转表，通过 gdb 可以得到该表:

(gdb) x /8g 0x402470

0x402470: 0x0000000000400f7c 0x0000000000400fb9

0x402480: 0x0000000000400f83 0x0000000000400f8a

0x402490: 0x0000000000400f91 0x0000000000400f98

0x4024a0: 0x0000000000400f9f 0x0000000000400fa6

通过第一个整数决定跳转的地址.

最终可以获得答案:

phase_3 = 0 207 or 1 311 or 2 707 or 3 256 or 4 389 or 5 206 or 6 682 or 7 327.

phase_4

```
0000000000400fce <func4>:
  400fce: 48 83 ec 08           sub    $0x8,%rsp
  400fd2: 89 d0                 mov    %edx,%eax # 第三个参数
  400fd4: 29 f0                 sub    %esi,%eax # 第三个参数 - 第二个参数
  400fd6: 89 c1                 mov    %eax,%ecx # 上行值给 %ecx
  400fd8: c1 e9 1f              shr    $0x1f,%ecx # 保留差值符号位
  400fdb: 01 c8                 add    %ecx,%eax # 差值 += 符号位
  400fdd: d1 f8                 sar    %eax # 差值 /= 2
  400fdf: 8d 0c 30              lea    (%rax,%rsi,1),%ecx # 差值 + 第二个参数
  400fe2: 39 f9                 cmp    %edi,%ecx # 第一个参数 和 %ecx
  400fe4: 7e 0c                 jle    400ff2 <func4+0x24> # 第一个参数 >= mid 跳转
  400fe6: 8d 51 ff              lea    -0x1(%rcx),%edx
  400fe9: e8 e0 ff ff ff        callq  400fce <func4>
  400fee: 01 c0                 add    %eax,%eax
  400ff0: eb 15                 jmp    401007 <func4+0x39>
  400ff2: b8 00 00 00 00        mov    $0x0,%eax
  400ff7: 39 f9                 cmp    %edi,%ecx
  400ff9: 7d 0c                 jge    401007 <func4+0x39> # 第一个参数 <= mid 跳转
  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
```

```
000000000040100c <phase_4>:
  40100c: 48 83 ec 18           sub    $0x18,%rsp # 开辟栈空间
  401010: 48 8d 4c 24 0c        lea    0xc(%rsp),%rcx # 第四个参数
  401015: 48 8d 54 24 08        lea    0x8(%rsp),%rdx # 第三个参数
  40101a: be cf 25 40 00        mov    $0x4025cf,%esi # 第二个参数
  40101f: b8 00 00 00 00        mov    $0x0,%eax # 返回值置 0
  401024: e8 c7 fb ff ff        callq  400bf0 <__isoc99_sscanf@plt> # 调用 sscanf
  401029: 83 f8 02              cmp    $0x2,%eax # 判断是否读入两个值
  40102c: 75 07                 jne    401035 <phase_4+0x29> # 为正常读入就 bomb
  40102e: 83 7c 24 08 0e        cmpl   $0xe,0x8(%rsp) # cmp 第一个输入 和 0xe
  401033: 76 05                 jbe    40103a <phase_4+0x2e> # 第一个输入 <= 0xe 跳过 bomb
  401035: e8 00 04 00 00        callq  40143a <explode_bomb> # bomb
  40103a: ba 0e 00 00 00        mov    $0xe,%edx # 第三个参数
  40103f: be 00 00 00 00        mov    $0x0,%esi # 第二个参数
  401044: 8b 7c 24 08           mov    0x8(%rsp),%edi # 第一个参数(第一个输入)
  401048: e8 81 ff ff ff        callq  400fce <func4> # 调用函数 func4
  40104d: 85 c0                 test   %eax,%eax # 判断返回值
  40104f: 75 07                 jne    401058 <phase_4+0x4c> # 返回不为 0 就 bomb
  401051: 83 7c 24 0c 00        cmpl   $0x0,0xc(%rsp) # cmp 第二个输入 和 0
  401056: 74 05                 je     40105d <phase_4+0x51> # 第二个输入为 0 跳过 bomb
  401058: e8 dd 03 00 00        callq  40143a <explode_bomb>
  40105d: 48 83 c4 18           add    $0x18,%rsp
  401061: c3                    retq
```

首先开辟栈空间，将两个地址传入给 sscanf 函数，通过 gdb 得出 sscanf 的第二个参数模式串

(gdb) x /s 0x4025cf

0x4025cf: "%d %d"

得知将我们输入的两个参数放到指定的局部变量中.

如果输入的不是两个整型变量就 bomb

如果第一个输入的整数大于 0xe 就 bomb

调用 func4(第一个输入的整数, 0, 14). 观察 func4 可以知道如果传入的整数落在了每次二分查找的右半边就不会返回 0 导致 bomb，所以可以推出第一个输入的整型变量为 7 or 3 or 1 or 0.

然后检查第二个输入的整型变量，如果不是 0 就会 bomb，由此得出答案:

phase_4 = 7 0 or 3 0 or 1 0 or 0 0

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 # 清空 %eax
  40107a: e8 9c 02 00 00        callq  40131b <string_length>
  40107f: 83 f8 06              cmp    $0x6,%eax # 比较输入字符串长度和 6
```

```
  401082: 74 4e                je     4010d2 <phase_5+0x70> # 字符串长度 != 6 就 bomb
  401084: e8 b1 03 00 00        callq  40143a <explode_bomb>
  401089: eb 47                jmp    4010d2 <phase_5+0x70> # 跳去置 %rax 为 0 继续执行
  40108b: 0f b6 0c 03          movzbl (%rbx,%rax,1),%ecx # 将输入的 6 个字符依次传出
  40108f: 88 0c 24             mov    %cl,(%rsp)
  401092: 48 8b 14 24          mov    (%rsp),%rdx # 取字符低 8 位
  401096: 83 e2 0f             and    $0xf,%edx # 取字符低 4 位
  401099: 0f b6 92 b0 24 40 00  movzbl 0x4024b0(%rdx),%edx # 字符低 4 位在指定内存寻字符给 %edx
  4010a0: 88 54 04 10          mov    %dl,0x10(%rsp,%rax,1) # 将找到的字符依次传
给 %rsp[10+%rax]处
  4010a4: 48 83 c0 01          add    $0x1,%rax # 更新 %rax 以循环
  4010a8: 48 83 f8 06          cmp    $0x6,%rax # 循环 6 次结束
  4010ac: 75 dd                jne    40108b <phase_5+0x29>
  4010ae: c6 44 24 16 00        movb   $0x0,0x16(%rsp) # 给字符串结尾加 '\0'
  4010b3: be 5e 24 40 00        mov    $0x40245e,%esi # 第二个参数 (指定内存的字符串)
  4010b8: 48 8d 7c 24 10        lea    0x10(%rsp),%rdi # 第一个参数 (栈上的刚刚写入的字符串)
  4010bd: e8 76 02 00 00        callq  401338 <strings_not_equal> # 比较字符串是否相等
  4010c2: 85 c0                test   %eax,%eax
  4010c4: 74 13                je     4010d9 <phase_5+0x77> # 不相等就 bomb
  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>
  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
```

首先调用函数 string_length 查看所输入字符串的长度，如果不等于 6 就 bomb

然后依次取出输入的 6 个字符取其末 4 位加上 0x4024b0 作为地址取出相应字符放到栈上

最后比较栈上的字符串和地址 0x40245e 的字符串比较，如果相等就通过，否则 bomb

(gdb) x /1s 0x4024b0

0x4024b0 <array.3449>: "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"

(gdb) x /1s 0x40245e

0x40245e: "flyers"

通过对比 ascii 表，“flyers” 在上述字符串的索引依次为: 9, f, e, 5, 6, 7，可以找到一组低四位与之对应的字符串为”IONEFG”.

phase_5 = IONEFG

phase_6

```
00000000004010f4 <phase_6>:
  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 # 开辟栈空间
  401100: 49 89 e5             mov    %rsp,%r13 # %rsp -> %r13
  401103: 48 89 e6             mov    %rsp,%rsi # 第二个参数
  401106: e8 51 03 00 00       callq  40145c <read_six_numbers> # ----------第一部分:保证六个
数字为 1,2,3,4,5,6 的一种排序
  40110b: 49 89 e6             mov    %rsp,%r14 # %rsp -> %r14
  40110e: 41 bc 00 00 00 00    mov    $0x0,%r12d # 0 -> %r12
  401114: 4c 89 ed             mov    %r13,%rbp # %r13 一开始为 %rsp 然后在循环中一直在 +4
  401117: 41 8b 45 00          mov    0x0(%r13),%eax # 将数组中的数字依次给 %eax
  40111b: 83 e8 01             sub    $0x1,%eax # if %eax > 6: bomb
  40111e: 83 f8 05             cmp    $0x5,%eax # if %eax > 6: bomb
  401121: 76 05                jbe    401128 <phase_6+0x34> # if %eax > 6: bomb
  401123: e8 12 03 00 00       callq  40143a <explode_bomb> # if %eax > 6: bomb
  401128: 41 83 c4 01          add    $0x1,%r12d # r12 在循环中 +1
  40112c: 41 83 fc 06          cmp    $0x6,%r12d # r12 和 6 比较
  401130: 74 21                je     401153 <phase_6+0x5f> # if r12 == 6: 跳出循环
  401132: 44 89 e3             mov    %r12d,%ebx # 将数组下一个数字下标给到 %ebx
  401135: 48 63 c3             movslq %ebx,%rax
  401138: 8b 04 84             mov    (%rsp,%rax,4),%eax # 将数组下一个数字给 %eax
  40113b: 39 45 00             cmp    %eax,0x0(%rbp) # 将下一个数字和当前数字比较
  40113e: 75 05                jne    401145 <phase_6+0x51> # if a[i] == a[i + 1]: bomb
  401140: e8 f5 02 00 00       callq  40143a <explode_bomb>
  401145: 83 c3 01             add    $0x1,%ebx # 嵌套小循环遍历 i+1, i+2, ...
  401148: 83 fb 05             cmp    $0x5,%ebx
  40114b: 7e e8                jle    401135 <phase_6+0x41>
  40114d: 49 83 c5 04          add    $0x4,%r13
  401151: eb c1                jmp    401114 <phase_6+0x20> #--------------------------------
-------------第一部分结束
  401153: 48 8d 74 24 18       lea    0x18(%rsp),%rsi # 数组尾地址 + 4 -> %rsi ------------第
二部分:将数组的每个数用 7 减去
  401158: 4c 89 f0             mov    %r14,%rax # 数组首地址 -> %rax
  40115b: b9 07 00 00 00       mov    $0x7,%ecx # 0x7 -> %ecx
  401160: 89 ca                mov    %ecx,%edx
  401162: 2b 10                sub    (%rax),%edx
  401164: 89 10                mov    %edx,(%rax) # (%rax) = %ecx - (%rax)
  401166: 48 83 c0 04          add    $0x4,%rax # %rax 指向数组下一个元素
  40116a: 48 39 f0             cmp    %rsi,%rax # 看 %rax 是否指到最后
  40116d: 75 f1                jne    401160 <phase_6+0x6c> # 循环直到遍历完整个数组 ----------
-------------第二部分结束
  40116f: be 00 00 00 00       mov    $0x0,%esi # %rsi 一开始为 0, 循环中一直在 +4 ------------
第三部分:按数组的顺序排列链表
```

```
  401174: eb 21                jmp    401197 <phase_6+0xa3>
  401176: 48 8b 52 08          mov    0x8(%rdx),%rdx # (%rdx) + 8 -> %rdx ------------------
--]-------}
  40117a: 83 c0 01             add    $0x1,%eax # %eax += 1
  40117d: 39 c8                cmp    %ecx,%eax # 拿数组中的数和 从 0x1 开始不断增加的 %eax 比较
  40117f: 75 f5                jne    401176 <phase_6+0x82> # ------------------------------
--[
  401181: eb 05                jmp    401188 <phase_6+0x94> # 从上一行下来说明找到了第 %ecx 个节
点
  401183: ba d0 32 60 00       mov    $0x6032d0,%edx # -------------------------------------
----------------)
  401188: 48 89 54 74 20       mov    %rdx,0x20(%rsp,%rsi,2) #将当前节点的指针放到栈中指定位置
  40118d: 48 83 c6 04          add    $0x4,%rsi
  401191: 48 83 fe 18          cmp    $0x18,%rsi # 不断循环直到所有的节点的指针都移到栈
中
  401195: 74 14                je     4011ab <phase_6+0xb7> #
  401197: 8b 0c 34             mov    (%rsp,%rsi,1),%ecx # 将数组中的数依次给 %ecx
  40119a: 83 f9 01             cmp    $0x1,%ecx #
  40119d: 7e e4                jle    401183 <phase_6+0x8f> # 数组中的数 == 1 说明不需要向后遍历
链表,跳转------(
  40119f: b8 01 00 00 00       mov    $0x1,%eax # 1 -> %eax
  4011a4: ba d0 32 60 00       mov    $0x6032d0,%edx # 0x6032d0 -> %edx
  4011a9: eb cb                jmp    401176 <phase_6+0x82> # ------------------------------
----------{---第三部分结束
  4011ab: 48 8b 5c 24 20       mov    0x20(%rsp),%rbx # 指针数组首值-----------第四部分开始:将
链表的顺序修改为我们数组中的顺序
  4011b0: 48 8d 44 24 28       lea    0x28(%rsp),%rax # 指针数组次地址
  4011b5: 48 8d 74 24 50       lea    0x50(%rsp),%rsi # 指针数组尾地址 + 8
  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> # ------------------------------
------------第四部分结束
  4011d2: 48 c7 42 08 00 00 00  movq   $0x0,0x8(%rdx) # 最后一个节点的 next 指向 NULL
  4011d9: 00
  4011da: bd 05 00 00 00       mov    $0x5,%ebp # 计数器
  4011df: 48 8b 43 08          mov    0x8(%rbx),%rax # 将 node->next 给 %rax
  4011e3: 8b 00                mov    (%rax),%eax # 将 node->next->val 给 %eax
  4011e5: 39 03                cmp    %eax,(%rbx) # 将 node->next->val 和 node->val 比大小
  4011e7: 7d 05                jge    4011ee <phase_6+0xfa> # 后一个节点的数应当小于前面的
  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>
  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
```

首先检查所输入的数是不是 6 个
再检查输入的 6 个数是不是 1, 2, 3, 4, 5, 6 的一个排列
不妨设数组为 a, b, c, d, e, f
然后用 7 减去数组中的每个数，得到新的数组为
7-a, 7-b, 7-c, 7-d, 7-e, 7-f
依次找到上述数组所指的链表节点排成新的链表
所得的链表的值需要是降序排列的，以下是链表在内存中的展示:

(gdb) x /12gx 0x6032d0

0x6032d0 <node1>: 0x000000010000014c 0x00000000006032e0
0x6032e0 <node2>: 0x00000002000000a8 0x00000000006032f0
0x6032f0 <node3>: 0x000000030000039c 0x0000000000603300
0x603300 <node4>: 0x00000004000002b3 0x0000000000603310
0x603310 <node5>: 0x00000005000001dd 0x0000000000603320
0x603320 <node6>: 0x00000006000001bb 0x0000000000000000

降序排列索引顺序应当为: 3->4->5->6->1->2
可以反推出数组为: 4, 3, 2, 1, 6, 5
Phase_6 = 4 3 2 1 6 5

Result:

```
sh-4.4# cat ans.txt
Border relations with Canada have never been better.
1 2 4 8 16 32
0 207
0 0
IONEFG
4 3 2 1 6 5

sh-4.4# ./bomb < ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Congratulations! You've defused the bomb!
```