0x00000000004010f4 <+0>: push %r14

0x00000000004010f6 <+2>: push %r13

0x00000000004010f8 <+4>: push %r12

0x00000000004010fa <+6>: push %rbp

0x00000000004010fb <+7>: push %rbx

0x00000000004010fc <+8>: sub $0x50,%rsp

0x0000000000401100 <+12>: mov %rsp,%r13

0x0000000000401103 <+15>: mov %rsp,%rsi

0x0000000000401106 <+18>: call 0x40145c <read\_six\_numbers> 读数据

0x000000000040110b <+23>: mov %rsp, %r14 r14 记录栈顶

0x000000000040110e <+26>: mov $0x0,%r12d r12 记录轮次

0x0000000000401114 <+32>: mov %r13,%rbp r13 rbp 都记外圈次的栈顶指针

0x0000000000401117 <+35>: mov 0x0(%r13),%eax 栈顶数据给到rax

0x000000000040111b <+39>: sub $0x1,%eax 数据-1

0x000000000040111e <+42>: cmp $0x5,%eax -1数据

0x0000000000401121 <+45>: jbe 0x401128 <phase\_6+52> 数据-1≤5

0x0000000000401123 <+47>: call 0x40143a <explode\_bomb>不然爆炸

0x0000000000401128 <+52>: add $0x1,%r12d r12记录轮次，轮次+1

0x000000000040112c <+56>: cmp $0x6,%r12d 外圈六轮过去后才跳转

0x0000000000401130 <+60>: je 0x401153 <phase\_6+95> 第七轮 跳转到95

0x0000000000401132 <+62>: mov %r12d,%ebx r12外圈轮次，内圈轮次ebx

0x0000000000401135 <+65>: movslq %ebx,%rax 以外圈起始，作为为内圈

0x0000000000401138 <+68>: mov (%rsp,%rax,4),%eax ，取内圈当前栈顶

0x000000000040113b <+71>: cmp %eax,0x0(%rbp) 比较内圈数据和外圈栈顶数据

0x000000000040113e <+74>: jne 0x401145 <phase\_6+81> rbp就是外层栈顶

0x0000000000401140 <+76>: call 0x40143a <explode\_bomb> 不等于外圈栈顶，后五个数字≠a[0],即可

0x0000000000401145 <+81>: add $0x1,%ebx 更改ebx内圈

0x0000000000401148 <+84>: cmp $0x5,%ebx

0x000000000040114b <+87>: jle 0x401135 <phase\_6+65> 只要没比较完成后五个数字

0x000000000040114d <+89>: add $0x4,%r13 r13是栈顶指针，代表跳到下一个数据项，更改r13的顶端 ，r13=&a[2]

0x0000000000401151 <+93>: jmp 0x401114 <phase\_6+32>

这以上是在处理每一数字都-1后<5且，不重复，没要求顺序

0x0000000000401153 <+95>: lea 0x18(%rsp),%rsi rsp没动过，就是最下栈顶指针，rsi更新为栈地底指针，0x18=+24

0x0000000000401158 <+100>: mov %r14,%rax rax更新为栈顶

0x000000000040115b <+103>: mov $0x7,%ecx 7->ecx 有新数据 保存在ecx

0x0000000000401160 <+108>: mov %ecx,%edx 保存的数据 edx拿来用

0x0000000000401162 <+110>: sub (%rax),%edx edx-栈顶数据

0x0000000000401164 <+112>: mov %edx,(%rax) 又把减完剩下的复制给栈顶， 7-1=6->栈顶数据

0x0000000000401166 <+114>: add $0x4,%rax 栈顶指针+4转移下一个栈帧

0x000000000040116a <+118>: cmp %rsi,%rax 判断rax没到栈底

0x000000000040116d <+121>: jne 0x401160 <phase\_6+108> 不相等就循环更新数据，更新完后 堆栈变成654321

0x000000000040116f <+123>: mov $0x0,%esi esi不再记录栈底，更新为0

0x0000000000401174 <+128>: jmp 0x401197 <phase\_6+163>

第二段更新完之后的循环判断

0x0000000000401176 <+130>: mov 0x8(%rdx),%rdx 栈顶＞1，指针加8字节，访问内存、1，0，0，1，0更新数据 rdx 0  
  
这段mov是将 内存地址（rdx+8）的内存地址给到rdx,根据底下图片输出信息的推理得到，这个是在后移node指针，将指针抓一道栈顶数据的node，栈顶是2最后他就指向node 几

0x000000000040117a <+134>: add $0x1,%eax 更新eax=2

0x000000000040117d <+137>: cmp %ecx,%eax 比较eax和栈顶

0x000000000040117f <+139>: jne 0x401176 <phase\_6+130> eax和栈顶不相等，就加到相等为止，最终rax=ecx栈顶，更新的次数是栈>1的栈顶的次数，更新栈顶次数

0x0000000000401181 <+141>: jmp 0x401188 <phase\_6+148>

x/d ($rsp+0x20+2\*($rsi))

栈顶＜=1，就到这一段，

0x0000000000401183 <+143>: mov $0x6032d0,%edx

0x0000000000401188 <+148>: mov %rdx,0x20(%rsp,%rsi,2) 理解这个是关键

0x000000000040118d <+153>: add $0x4,%rsi 每一路你都会对rsi+4 记录每一段数据，用来更新栈顶的

分析 148 偏移量是0x20 =32， 对rsi\*2，rdx是node指针， rsi移动的标准是4 4\*2=8，因此，rsp堆栈里面移动的空间的比例是8个B字节为标准的，是在记录指针，，所于原堆栈20个B偏置网上是在记录node，将转化出来的654321，指向相对应的node！！！

完成指针的赋值转化 这个逻辑循环是在完成指针转化

0x0000000000401191 <+157>: cmp $0x18,%rsi 判断6轮过去没有，rsi

0x0000000000401195 <+161>: je 0x4011ab <phase\_6+183> 第六轮才跳转

0x0000000000401197 <+163>: mov (%rsp,%rsi,1),%ecx rsp+0 栈顶数据给到ecx 654321 ecx记录每轮栈顶

0x000000000040119a <+166>: cmp $0x1,%ecx 判断栈顶和1的比较

0x000000000040119d <+169>: jle 0x401183 <phase\_6+143> 栈顶数据≤1跳转

0x000000000040119f <+171>: mov $0x1, %eax 栈顶＞1，初始化eax=1

0x00000000004011a4 <+176>: mov $0x6032d0,%edx 初始话edx 76 这个内存里面放着 edx这个赋值，是给了指针

0x00000000004011a9 <+181>: jmp 0x401176 <phase\_6+130> 栈顶＞1，跳转130 更新rdx

以上是用来更新rsp+rsi的位置 mov %rdx,0x20(%rsp,%rsi,2) 更新的结果是将高20字节

0x00000000004011ab <+183>: mov 0x20(%rsp),%rbx rbx再次存结点地址

0x00000000004011b0 <+188>: lea 0x28(%rsp),%rax rax 指针堆栈第二个空间

0x00000000004011b5 <+193>: lea 0x50(%rsp),%rsi rsi指向栈底

0x00000000004011ba <+198>: mov %rbx,%rcx rcx初始化为第一个结点地址

上面是循环初始化，

0x00000000004011bd <+201>: mov (%rax),%rdx rdx记录第二个结点地址 √

0x00000000004011c0 <+204>: mov %rdx,0x8(%rcx) 更新node内部 形成了链 node6->node

5

0x00000000004011c4 <+208>: add $0x8,%rax

0x00000000004011c8 <+212>: cmp %rsi,%rax

0x00000000004011cb <+215>: je 0x4011d2 <phase\_6+222>

0x00000000004011cd <+217>: mov %rdx,%rcx

0x00000000004011d0 <+220>: jmp 0x4011bd <phase\_6+201>

0x00000000004011d2 <+222>: movq $0x0,0x8(%rdx) 结束后置零 node1指针指控

第三步骤关键 ，来观察出上面这个循环又更新了什么，stepi 看处理啊，是更新了node

Node内部， node 6 里面，第一个数字443，6， node5的地址 0  
这是在指针反指，node6内部的指针指向node5，一开始是顺序的，然现在反之

0x00000000004011da <+230>: mov $0x5,%ebp

0x00000000004011df <+235>: mov 0x8(%rbx),%rax

0x00000000004011e3 <+239>: mov (%rax),%eax

0x00000000004011e5 <+241>: cmp %eax,(%rbx) 取了下一个节点的第一个数字，还是要大于等于才可以，不然爆炸

0x00000000004011e7 <+243>: jge 0x4011ee <phase\_6+250> rbx一定要大于

看完了，反置链之后，依旧是要

0x00000000004011e9 <+245>: call 0x40143a <explode\_bomb>

0x00000000004011ee <+250>: mov 0x8(%rbx),%rbx

0x00000000004011f2 <+254>: sub $0x1,%ebp

0x00000000004011f5 <+257>: jne 0x4011df <phase\_6+235>

0x00000000004011f7 <+259>: add $0x50,%rsp

0x00000000004011fb <+263>: pop %rbx

0x00000000004011fc <+264>: pop %rbp

0x00000000004011fd <+265>: pop %r12

0x00000000004011ff <+267>: pop %r13

0x0000000000401201 <+269>: pop %r14

0x0000000000401203 <+271>: ret   
  
