答案：

1、Houses will begat jobs, jobs will begat houses.

2、0 1 3 6 10 15（答案有无穷多个）

3、0 455(1 294;2 813;3 627;4 250;5 340;6 106;7 81)

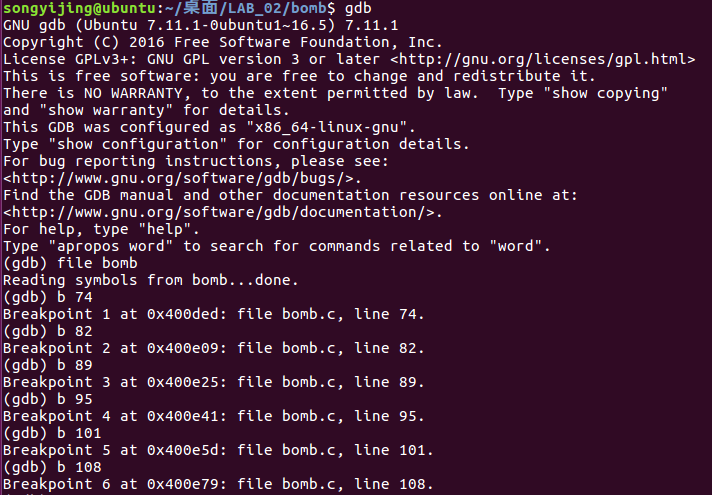
4、66 2(99 3;132 4)

5、5 115(or x = 16k-1, k = 1,2,3,4,...；y = )

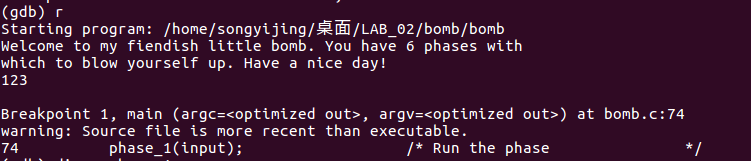
6、2 5 1 6 3 4

拆炸弹的过程：

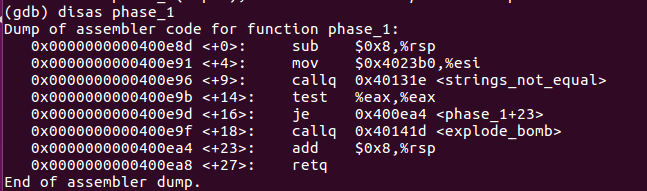
1. 进入gdb并在每一个函数设置断点：



1. 调试第一个函数：
2. 开始运行，随便输入字符串作为第一个函数的答案，此时会运行到第一个函数停止：



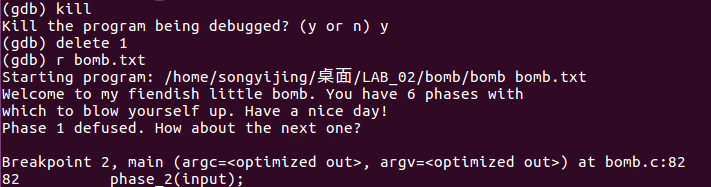
1. 查看第一个函数的汇编代码：



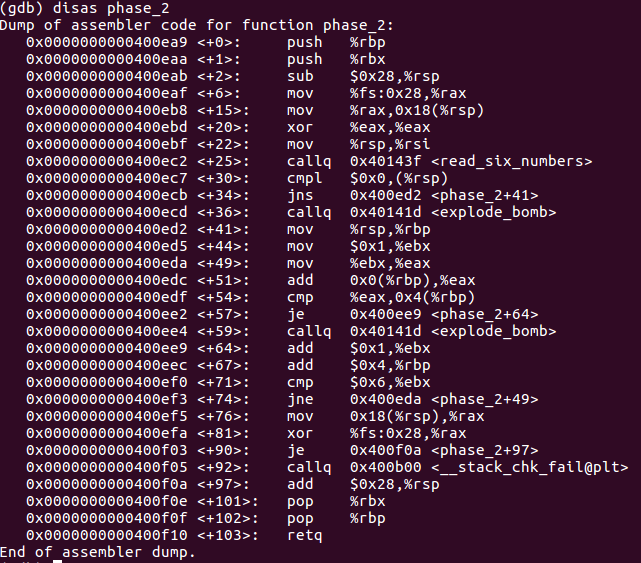
1. 由callq得知，phase\_1输入的为字符串，并且字符串储存在$0x4023b0，使用x/s命令查看：



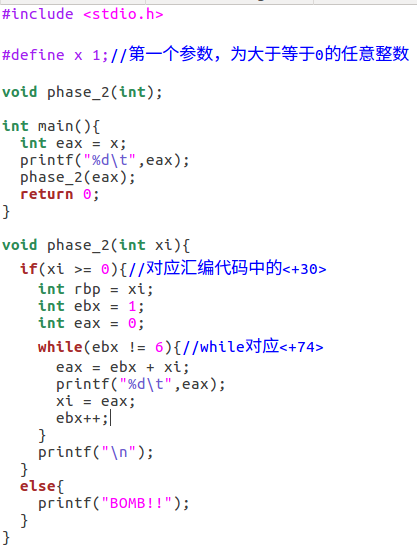
1. 因此第一个函数的答案为“Houses will begat jobs, jobs will begat houses.”，将该答案存入bomb.txt文件中。
2. 调试第二个函数：
3. 结束之前的调试，删除第一个断点，并重新运行，此时会在第二个函数处停止：



1. 查看第二个函数的和汇编代码：



1. 通过callq read\_six\_numbers可以得知需要输入的为六个数字。
2. 该函数的重点在于循环，C语言代码：



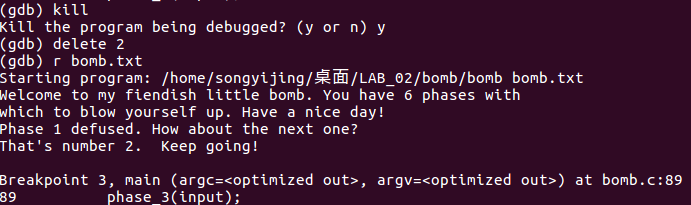
1. 当第一个参数为0时：



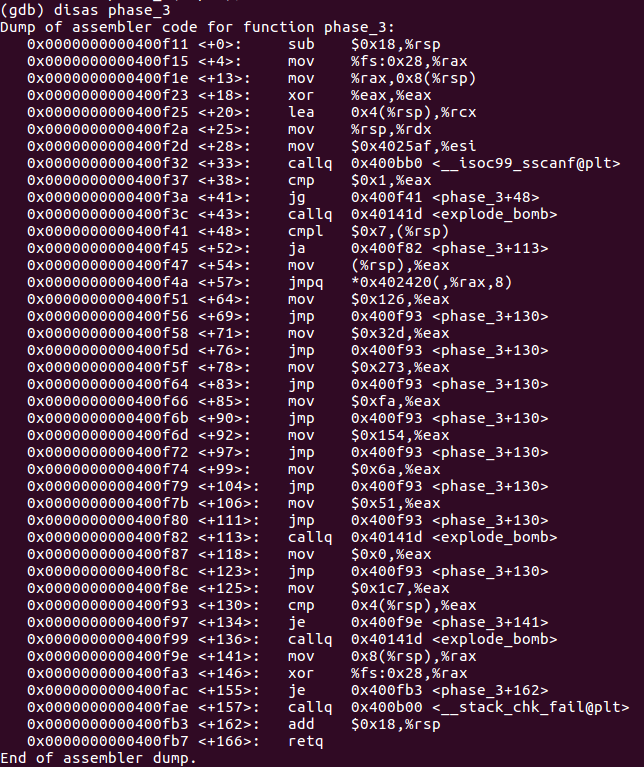
当第一个参数为1时：



1. 因此第二个函数的答案不唯一，任选一个“0 1 3 6 10 15”存入bomb.txt中
2. 调试第三个函数：
3. 结束之前的调试，删除第二个断点，并重新运行，此时会在第三个函数处停止：



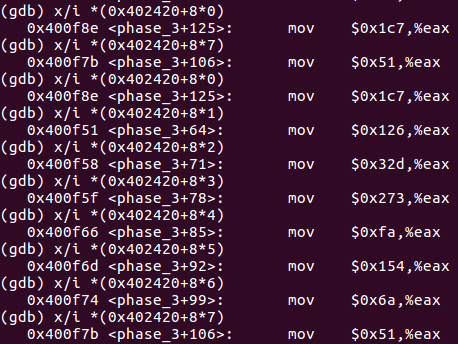
1. 查看第三个函数的汇编代码：



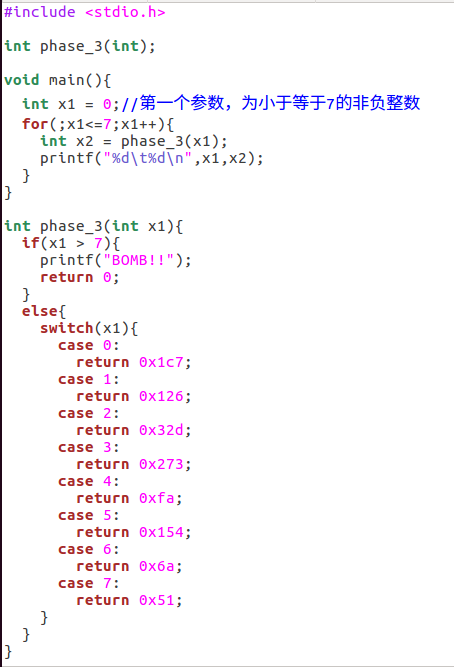
1. 根据<+28>可以得知该函数需要输入的参数为两个十进制的数：



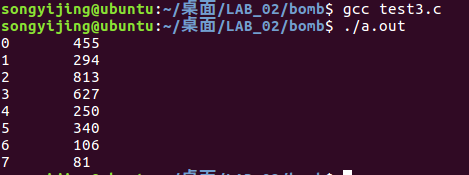
1. 该函数考察的是switch语句，x/i 得到跳转位置：



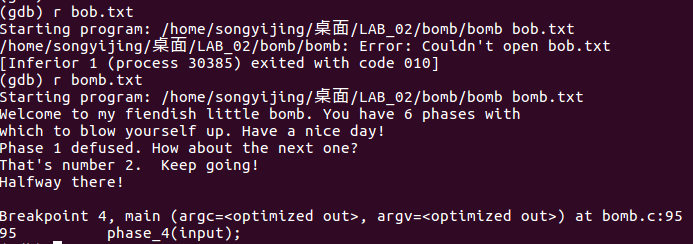
1. 对应的C语言代码：



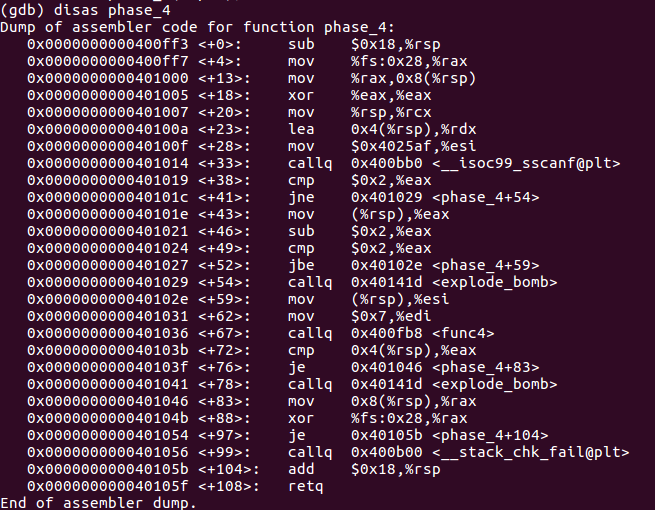
运行结果：



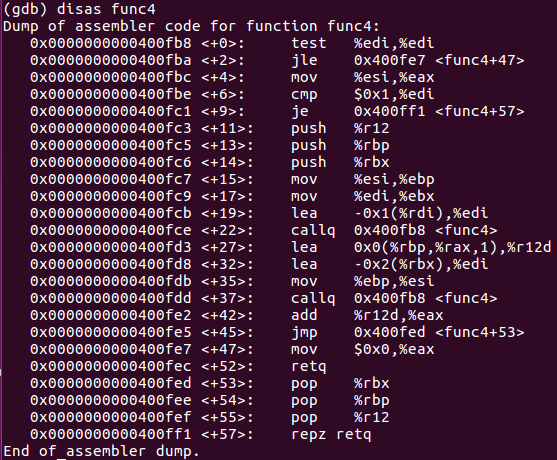
1. 因此第三个函数的答案也不唯一，将其中一个答案“0 455”存入bomb.txt文件中
2. 调试第四个函数：
3. 结束之前的调试，删除第三个断点，并重新运行，此时会在第四个函数处停止：



1. 查看第四个函数：



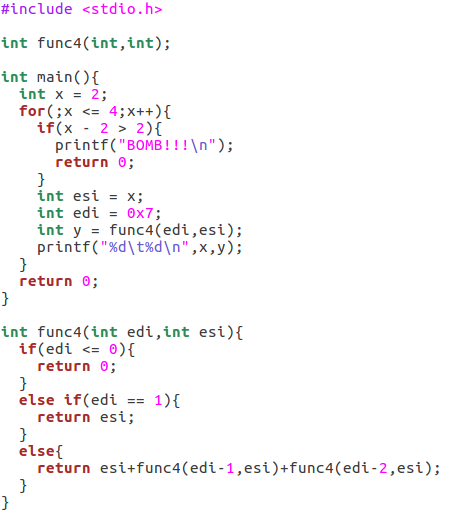
1. 其中调用了一个func4，查看func4：



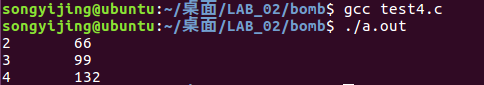
1. 由phase\_4中<+28>以及<+38>，都可以确定该函数需要两个参数，并且还可以确定这两个参数都是十进制的数：



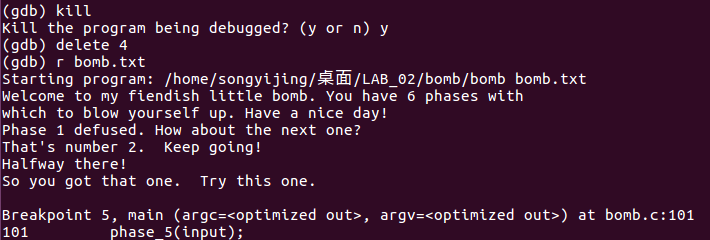
1. 由phase\_4<+43>至<+52>可以确定第一个参数只能为2 3 4其中之一。
2. 由phase\_4中<+72><+76>可以得知，第二个参数的值等于func4（第一个参数）。
3. 观察func4，发现是一个简单的递归函数，边界条件为edi≤0或edi == 1，对应的C语言代码：



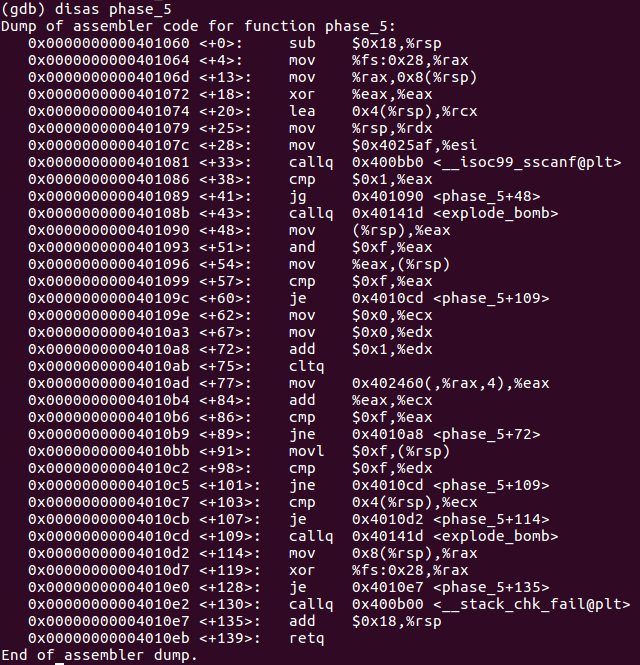
得到结果：



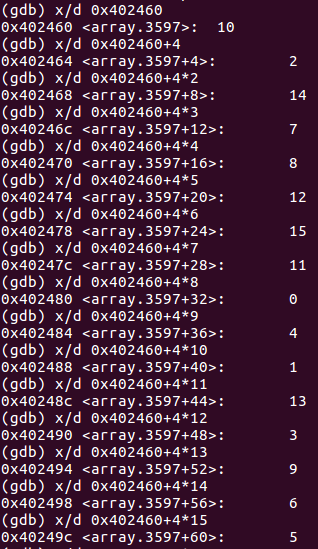
1. 因此第四个函数共有三个可选答案，但是当测试三个答案时出现错误，尝试了将两个参数对调位置，发现成功拆解炸弹，此处没有搞懂。
2. 因此将其中一个“66 2”存入bomb.txt文件。
3. 调试第五个函数：
4. 结束之前的调试，删除第四个断点，并重新运行，此时会在第五个函数处停止：



1. 查看第五个函数：



1. 该函数使用了一个比较复杂的循环，其中包含一个长度为16的int类型的数组，存在0x402460处
2. x/d得到数组的元素;



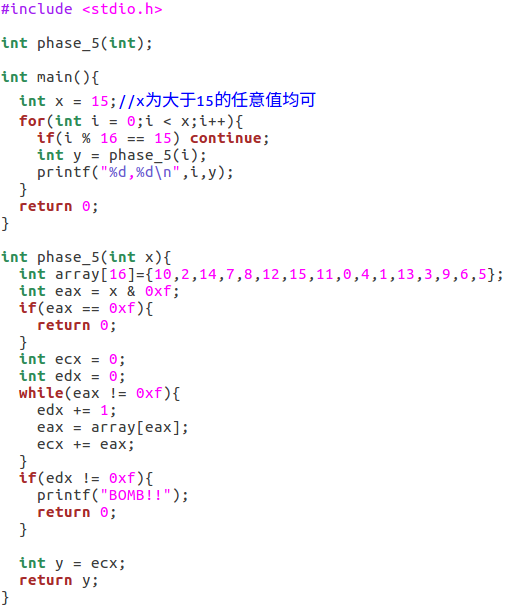
Array[16] = {10,2,14,7,8,12,15,11,0,4,1,13,3,9,6,5}

1. 该函数的参数的限制条件有两个：

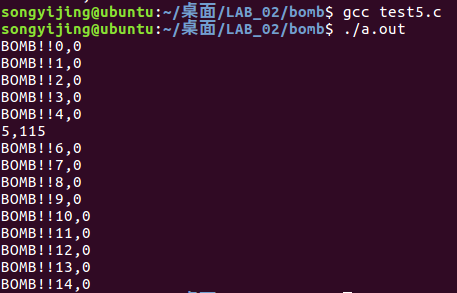
y = x & 0xf ≠ 0xf

每次以array[y]作为新的y，循环15次之后，y = 0xf

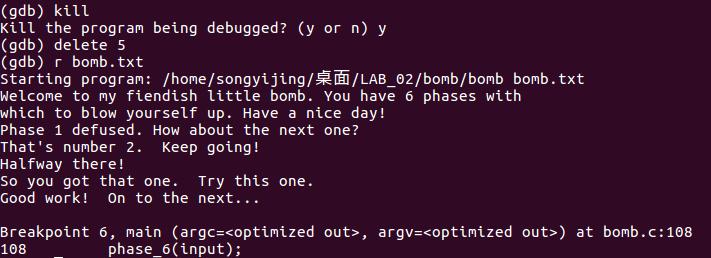
1. C程序：



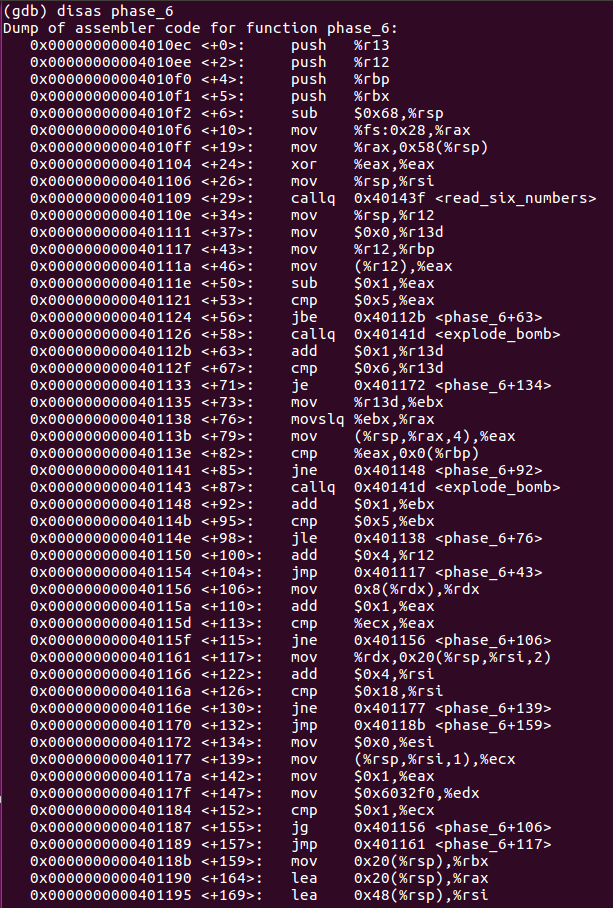
结果：

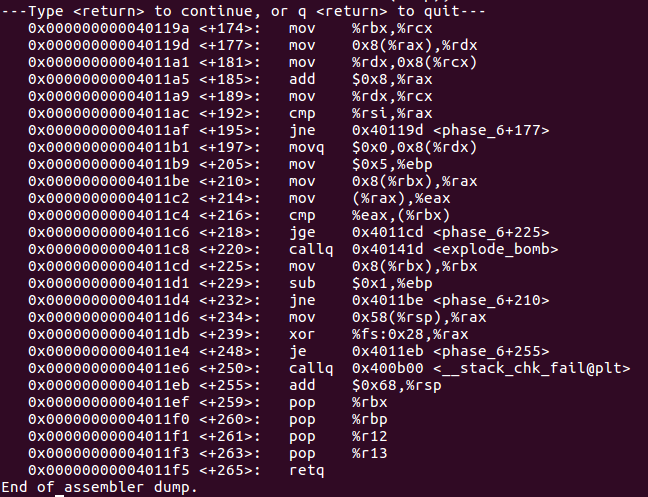


1. 因此答案为“5 115”或者（5+k\*16 115），k=1，2，3，…，将“5 115”存入bomb.txt
2. 调试第六个函数：
3. 结束之前的调试，删除第五个断点，并重新运行，此时会在第六个函数处停止：

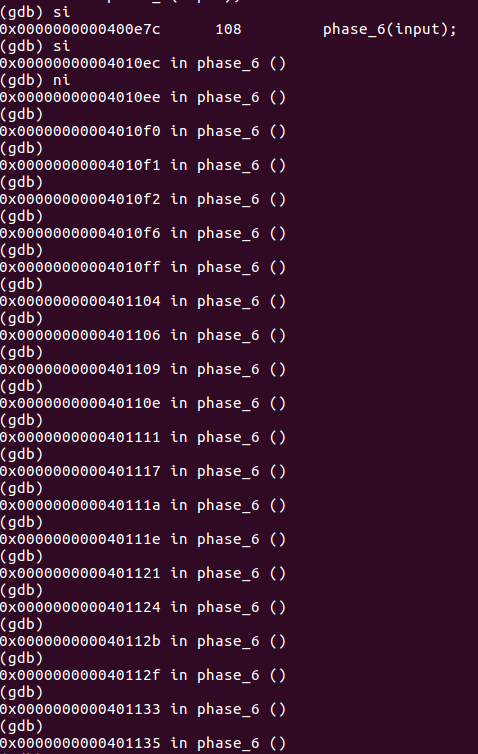


1. 查看第六个函数：

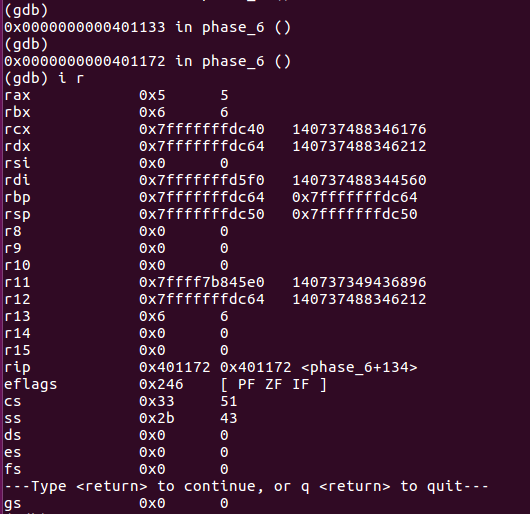




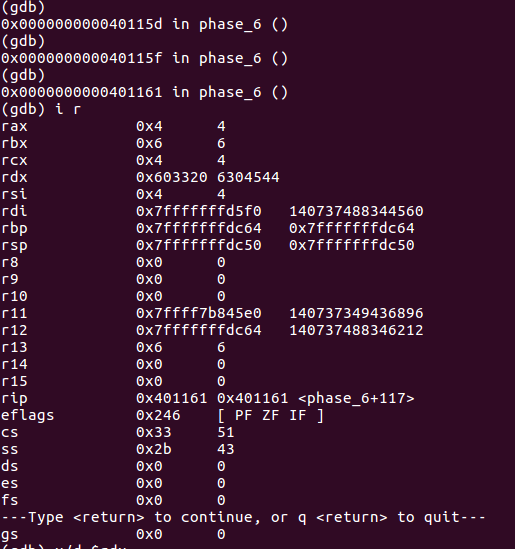
1. 根据callq read\_six\_numbers可以得知需要输入的为六个数
2. 然后输入六个数字，si ni进入函数内部进行一步步调试，可以得知，该函数主要有三次循环，第一次循环，保证输入的六个数字必须属于{1，2，3，4，5，6}，第二次循环，保证六个数互不相等，并且将每个数对应的节点按顺序存入0x20（%rsp，x\*2）中，第三次循环判断是否前面一个节点（地址）对应的值是否大于后者，即按照输入的参数顺序，对应节点处的值是否为从大到小的排列：



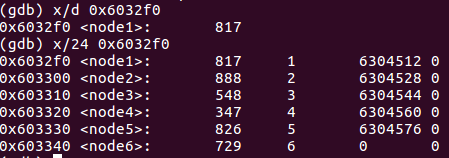
第一次循环结束后，寄存器信息如下：



第二次循环结束后，寄存器信息如下：



接下来第三次循环，得知该循环是为了降序排列，我们可以直接通过得到每个节点对应的值进而得到答案，由<+147>得知该列表位于0x6032f0处：



降序排列得到2 5 1 6 3 4

1. 因此第六个函数的答案为“2 1 6 3 4”，测试通过。

实验总结：

* 1. 部分较为简单的函数可以直接通过汇编代码的得到C程序代码
  2. 但是比较复杂的函数需要进入函数一步步调试，看每一步执行完毕之后，寄存器中的内容，得知每一步的目的。