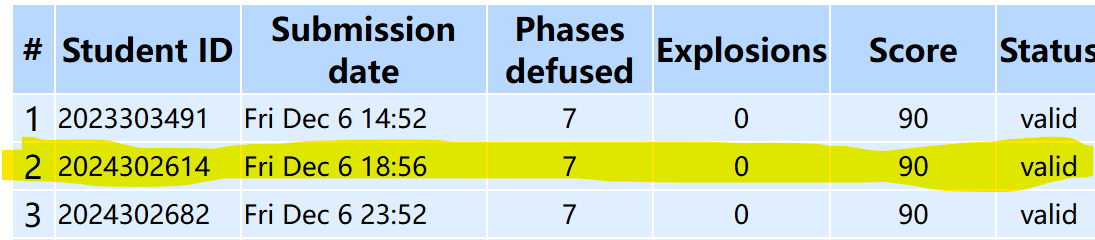
Lab2实验报告

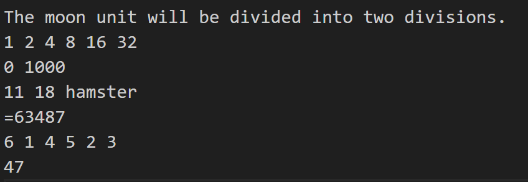
前情提要：

著名计算机教授褚伟波老师说过，看汇编代码要把自己的脑子当成个cpu。

可惜我的大脑不是cpu，我是如右图，那又怎么聪明地解决这类问题呢？

最终结果如下图：





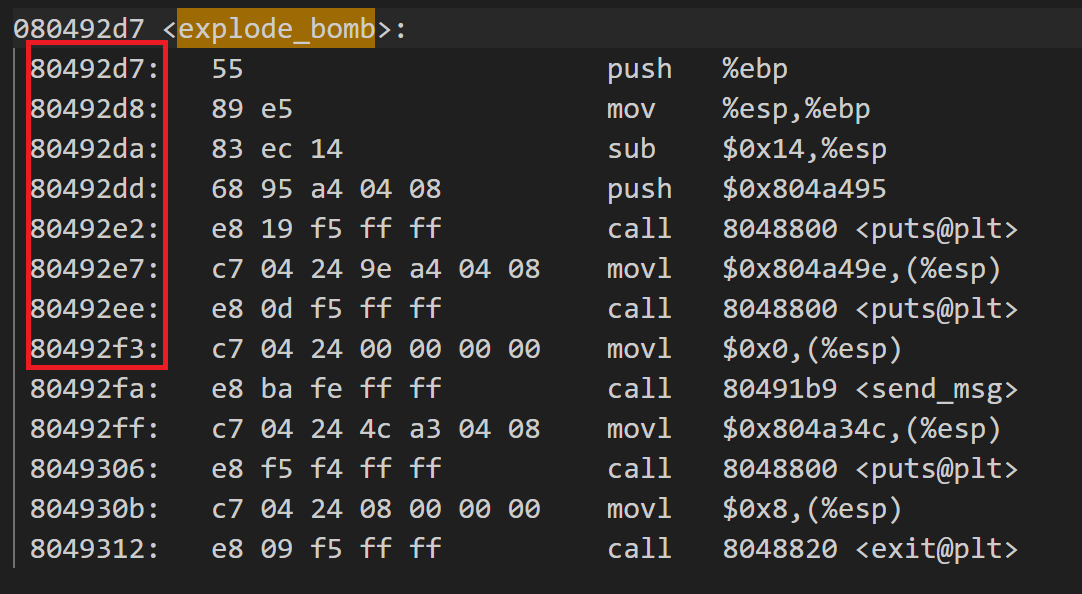
答案：

解题步骤：

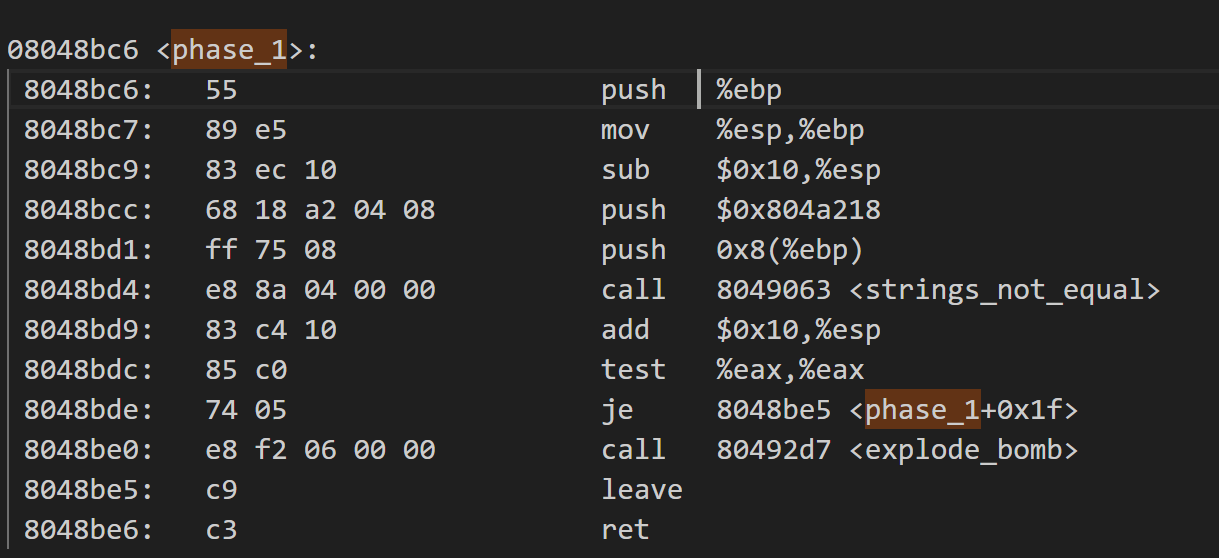
运用的一些小tricks：

合理运用gdb调试

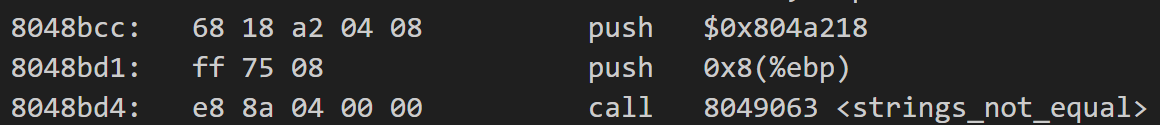
在如下几行设断点，防止手残按错然后爆了，跳到这个函数说明你已经做错了，可以r重新运行了



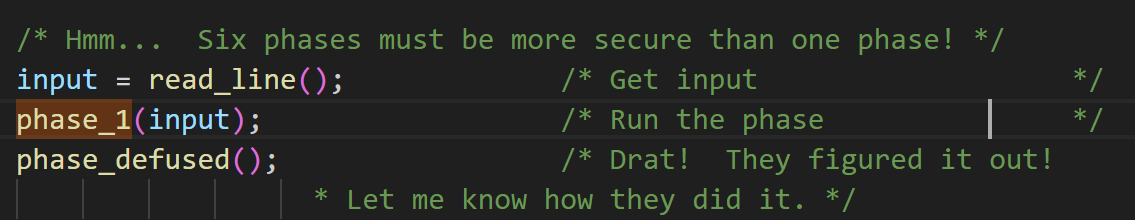
关于phase\_1：



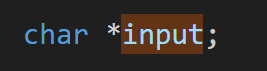
好，有了上图要怎么确定我们需要输进去什么呢？



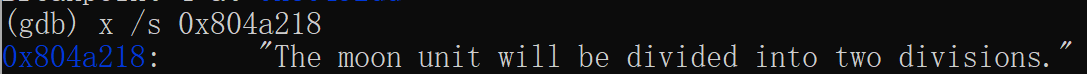
诶，发现有个字符串的比较函数，放了两个参数，第一个参数是phase\_1的第一个参数，第二个参数是一个立即数。根据所学知识，类比strcmp函数，传入的是两个字符串指针，也就是地址，然后我们翻翻bomb.c，发现传给phase\_1的参数不就是我们输的字符串吗？



然后我们又在main函数找到的其实是字符串存放的地址。



Ok，现在已经离成功比较近了，然后现在只要把第二个参数的地址放的字符串读出来就行了。（合理运用gdb）

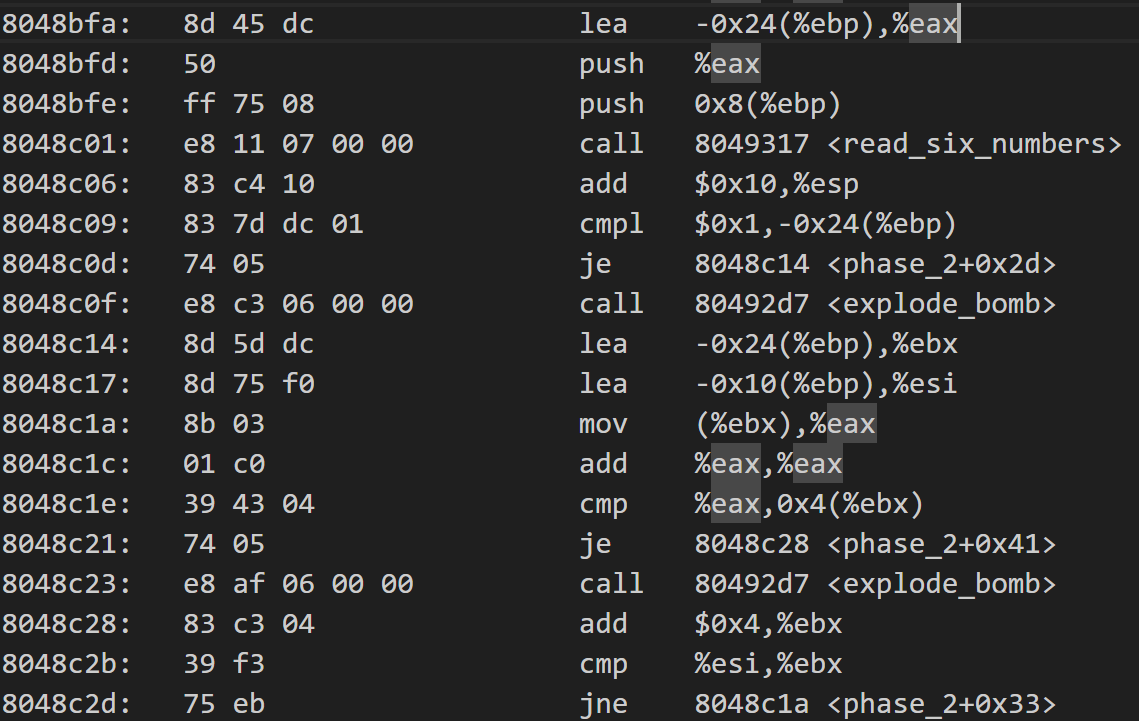


然后读出来就是应该我们要输入的字符串了。

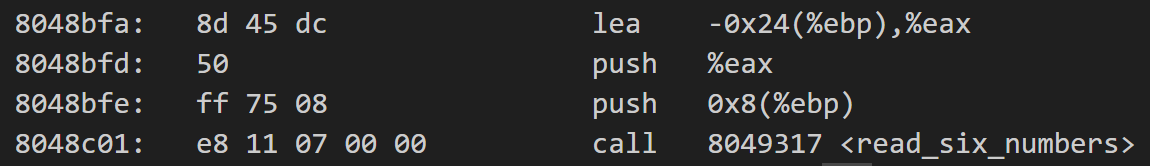
程序的最后几行就是判断比较的结果是不是相等了，相等就相当于完成了，不相等就爆了。

关于phase\_2：

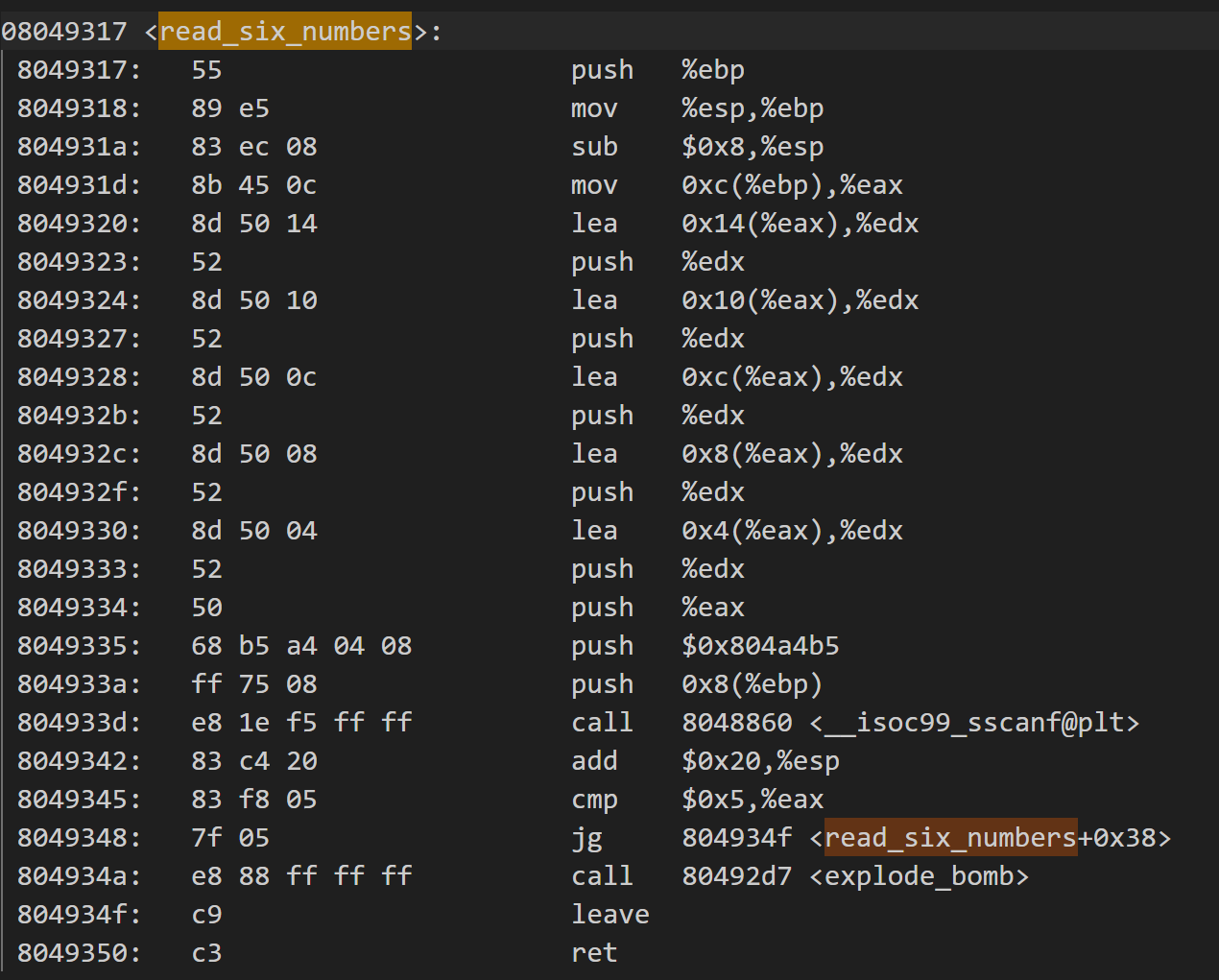
截重要部分



额，你会发现传了两个参数给read\_six\_numbers第一个参数是我们输入的字符串（上文有提到），第二个参数是栈的某一部分。



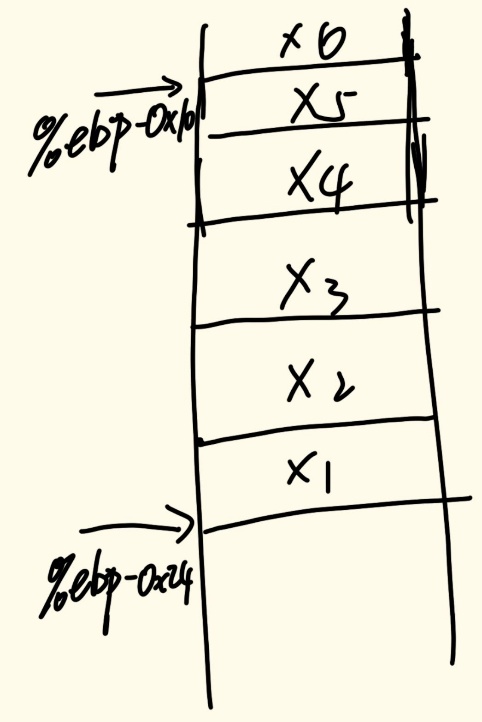
当然如果你自己猜是读入了六个数也行，然后存放顺序要不是正的要不就是反的也行。当然本文会分析read\_six\_numbers



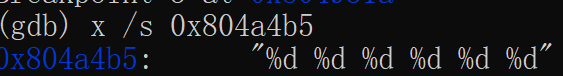
在这个read\_six\_nunbers中



先压了六个参数给sscaf。这六个参数的顺序是从传给read\_six\_numbers的第二个参数，也就是栈中的某一部分开始。即如下图。



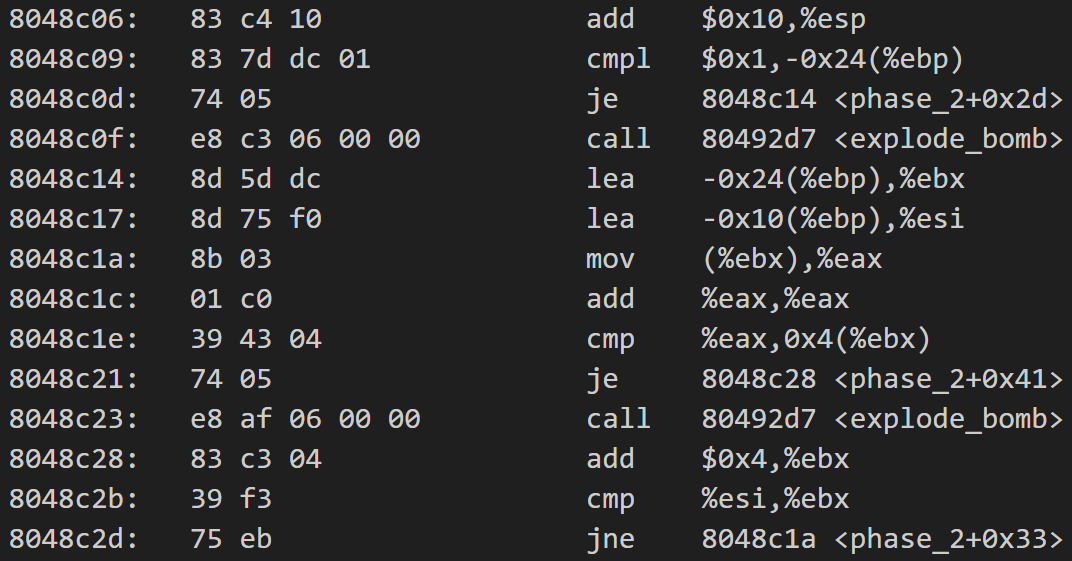
然后分析sscanf的前两个参数，第一个参数不就是传入read\_six\_numbers的我们输的字符串吗？然后翻翻第二的参数



发现它读的确实是六个整数，所以我们应该输入六个整数

最后这个read\_six\_numbers这个函数就判断是不是输入了六个数，然后就结束了

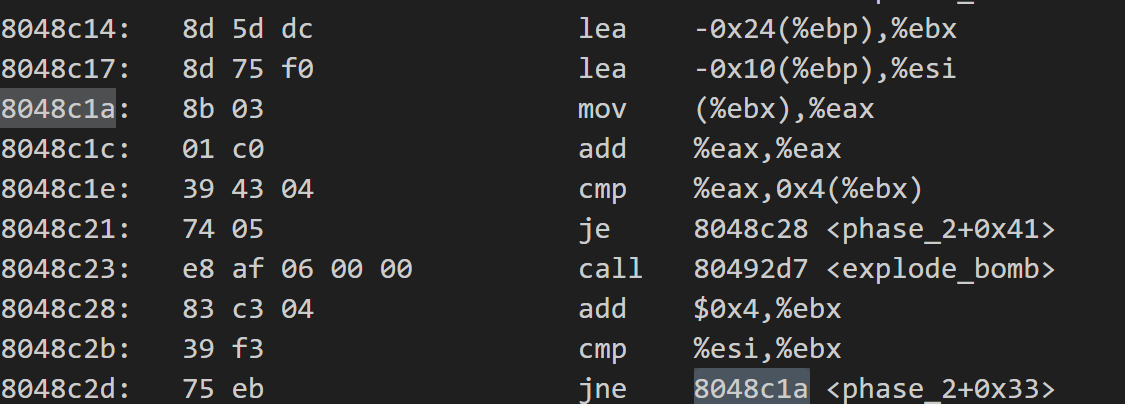
然后回到phase\_2中



然后下来第二句



你发现诶，那个地址存放的数不就是x1吗，那我们第一个数就应该是1吧，然后你看看下面，发现不等于1就爆了，可见就是1吧。

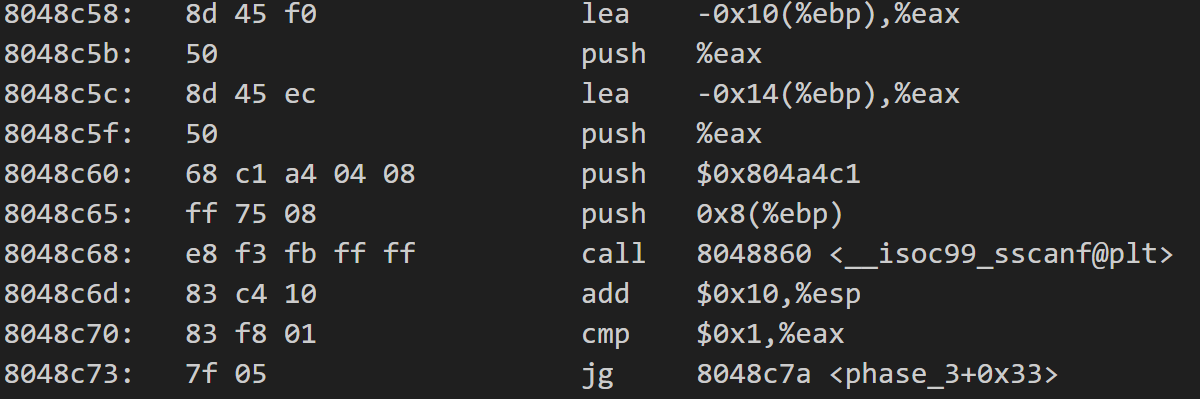


诶你发现在下面的指令还会往前跳，可见就是个循环吧，前面两句相当于初始化吧。

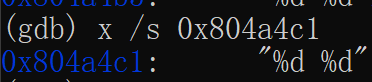
%ebx放的是x1的地址，%esi放的是x6的地址，然后开始进入循环。把当前地址上的数赋给%eax，然后比较当前数的两倍是不是下一个数（因为存放在栈中时连续的，偏移量是4就好了）。一直循环直到第5个数和第6个数比较完就结束了。所以本题的输入应该是1为首项，公比为2的6项等比数列

关于phase\_3：

首先sscanf读两个数



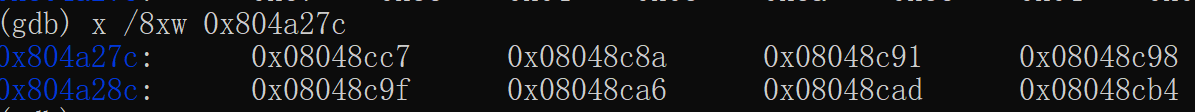
发现读的是两个整数



然后读完了两个数之后，发现第一个数要小于等于7要不然会爆。

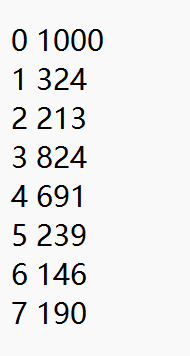
然后发现接下来会跳转到间接访问的地址

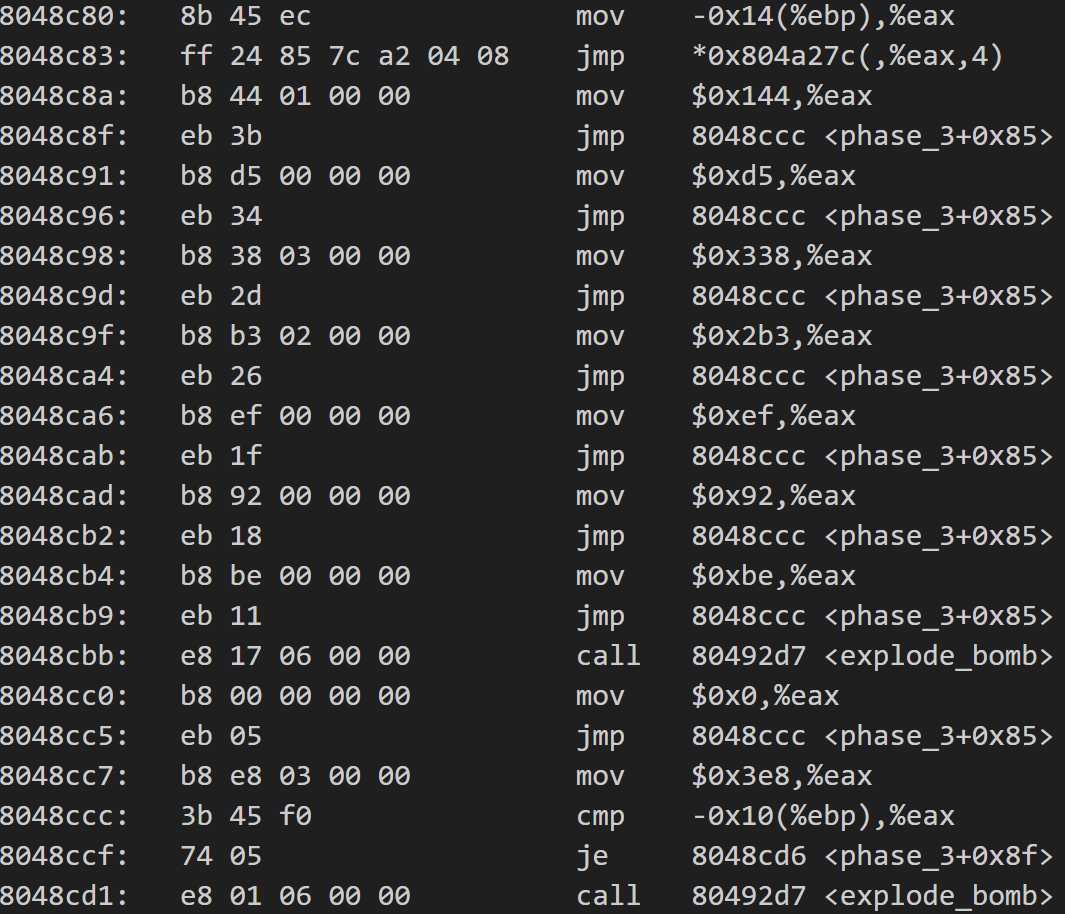
然后我们小手一查



发现跳的地址在这个函数中，然后会把一个值赋给%eax

把这些地址整理一下就是如下图了





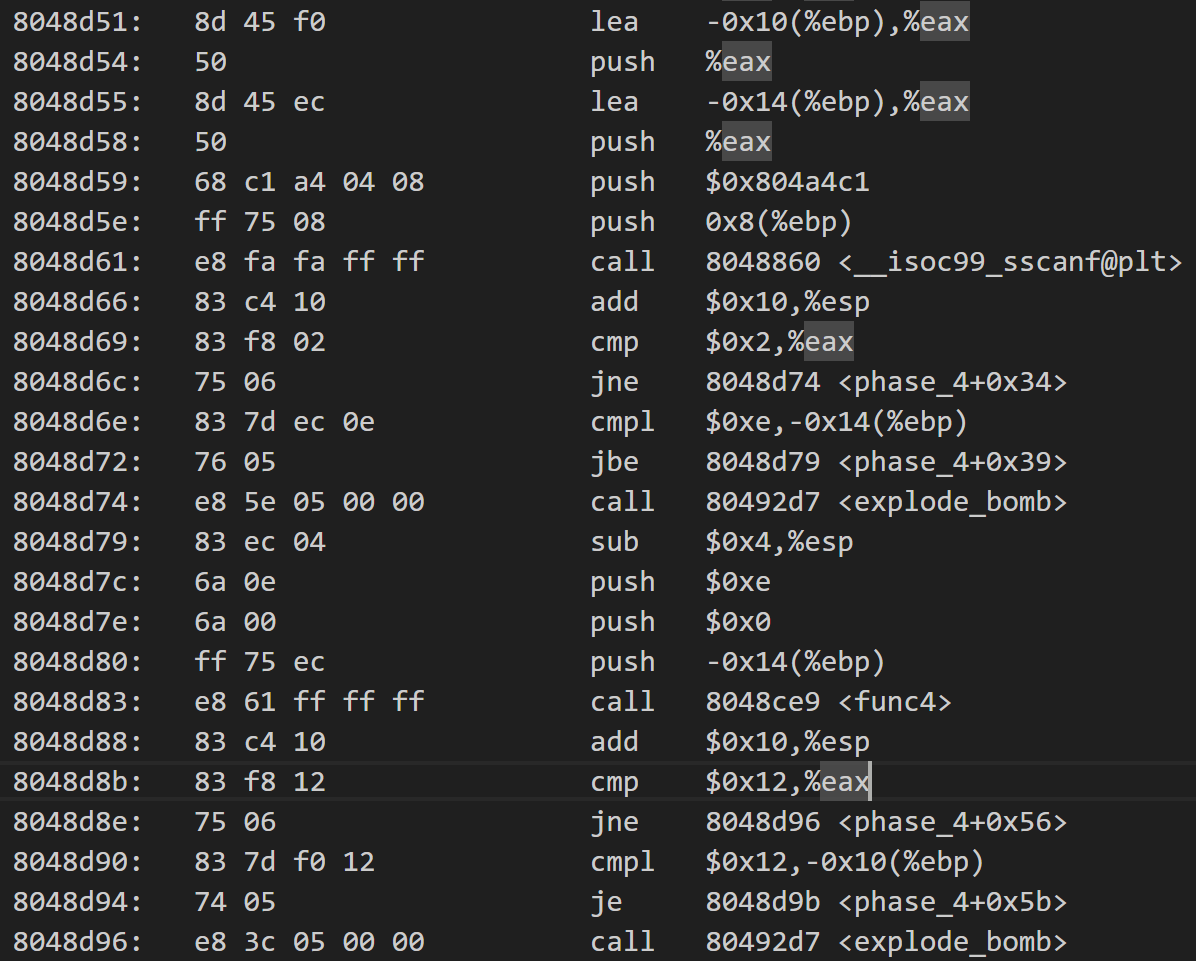
最后会把%eax和输入的第二个数比较，要求是相等。

所以这题我们上图的哪一组数据都能过

随便输一组就好啦。

关于phase\_4:

这题就是递归函数，当然不同人的题不一样，我的属于是“加强版的”（最后会说明为什么）



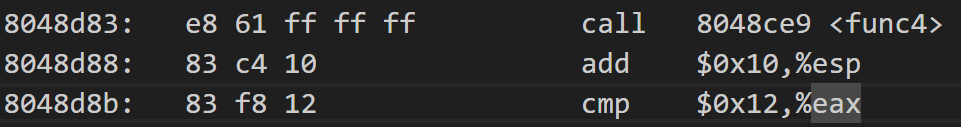
sscanf就不解析了吧，应该都会吧？

本题大意就是输入两个数，第一个数小于等于15。然后其中一个数作为函数的参数，得到函数的返回值和一个立即数比较，还有另外一个数和立即数比较，这两个满足相等就结束了。

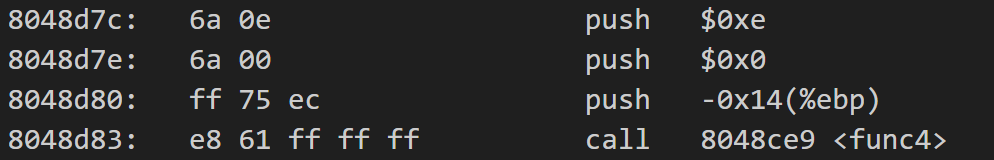
在上图很容易读出第二个参数是0x12即18



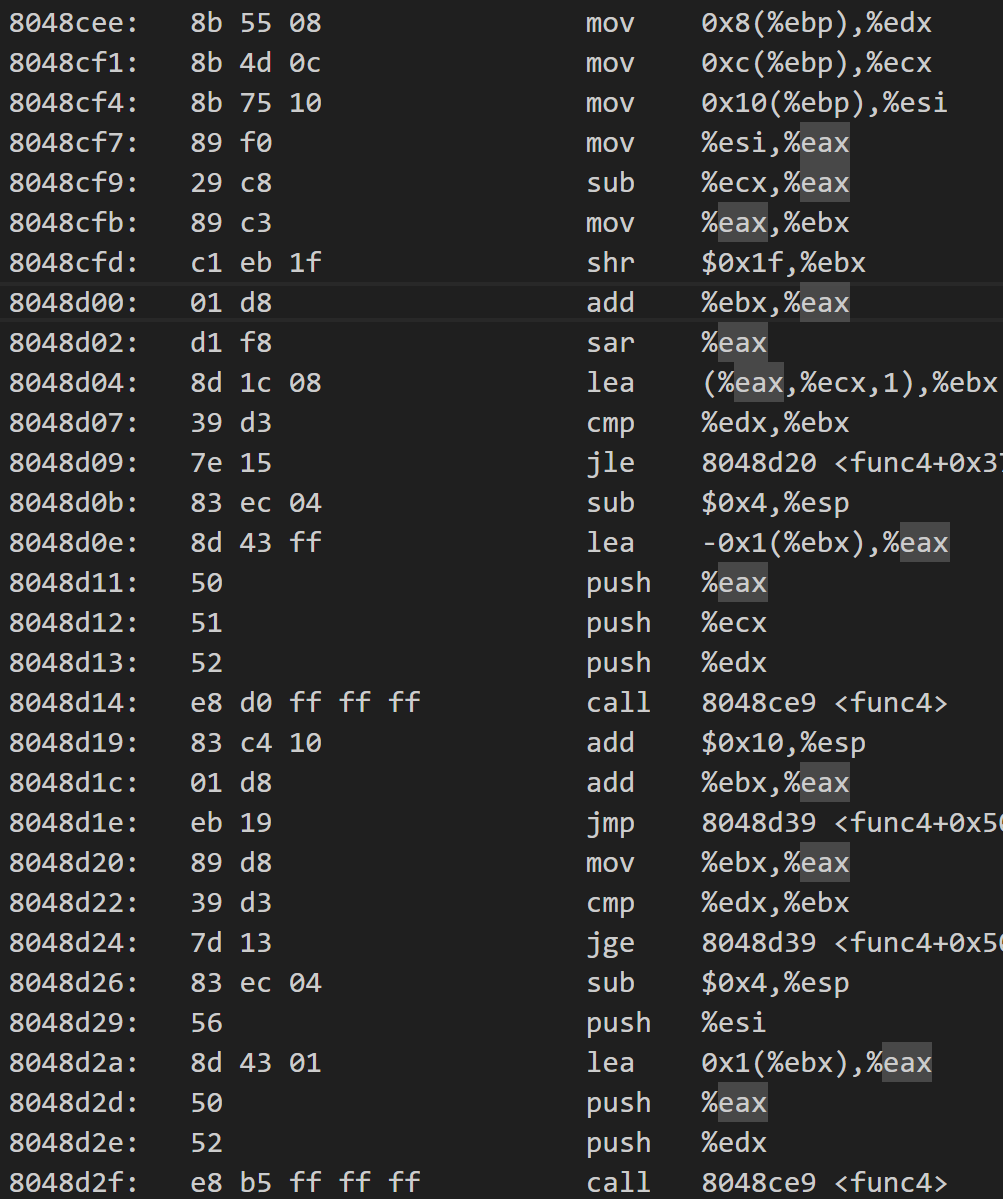
然后又发现函数的返回值应该是18



然后这个函数传入的参数是 x1,0,15 即func4(x1,0,15)



然后我们就要读func4，（说实话，我也没看懂在干嘛，反正我的大脑不是cpu，但是很“聪明”，那我们就要聪明地做完这题）





然后我就把这段汇编语言自己理解后用高级语言写出来了，具体理解过程比较复杂，简单说一下吧，

你会发现前面一段进行的是第二个参数和第三个参数除2，因为除2是负数还要加个偏移量

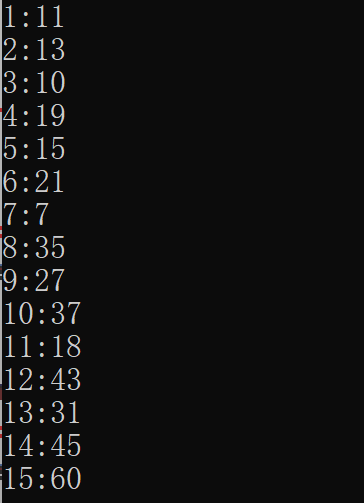
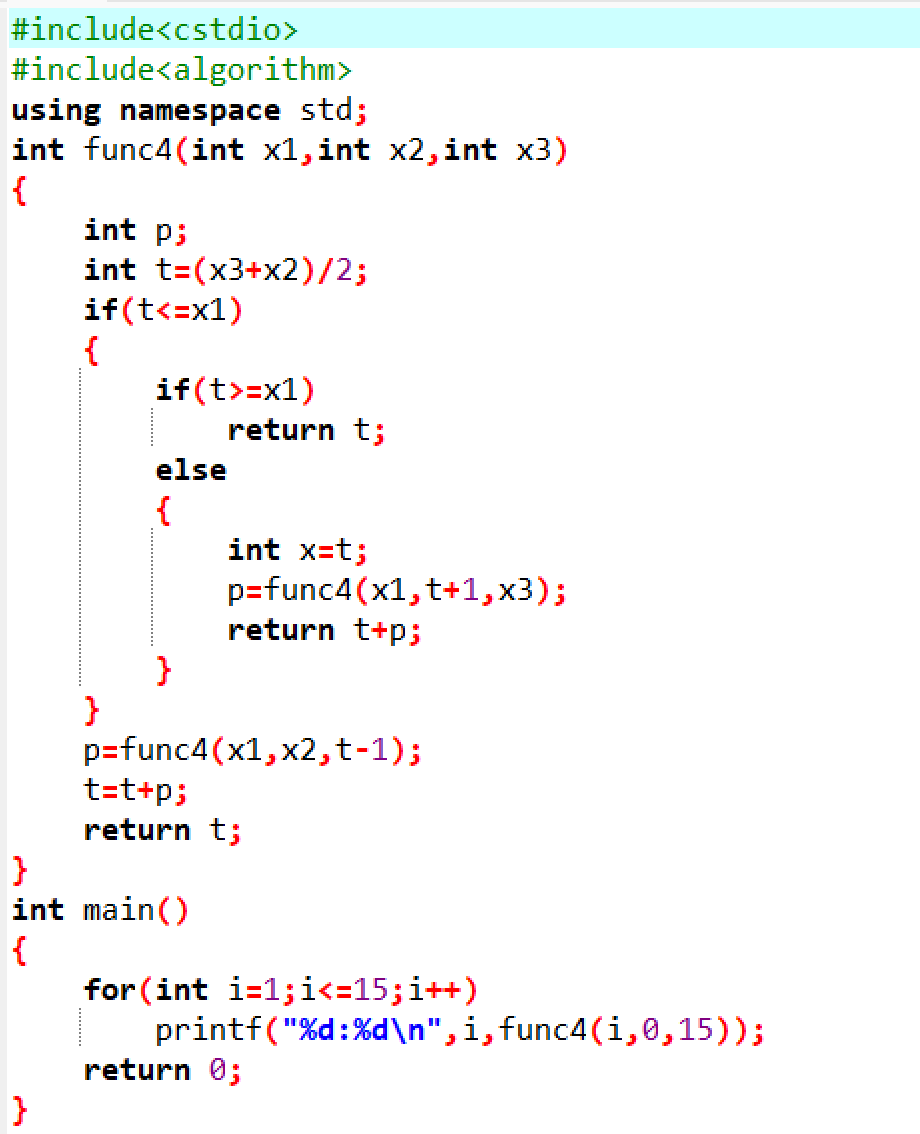
剩下就是各种判断了（然后我们运行程序去找就好了）

最后你会发现当参数是11时返回值是18

所以我们第一个数是11,

那又根据上面所描述，第二个数就是18了。

（当然本题一个个试第一个数，看哪个能过就好了，反正只要试16次就行）



当然本题到这里就结束了，那现在要回答上文的问题，为什么我的题是加强版？那说明就有弱化版

以下是弱化版

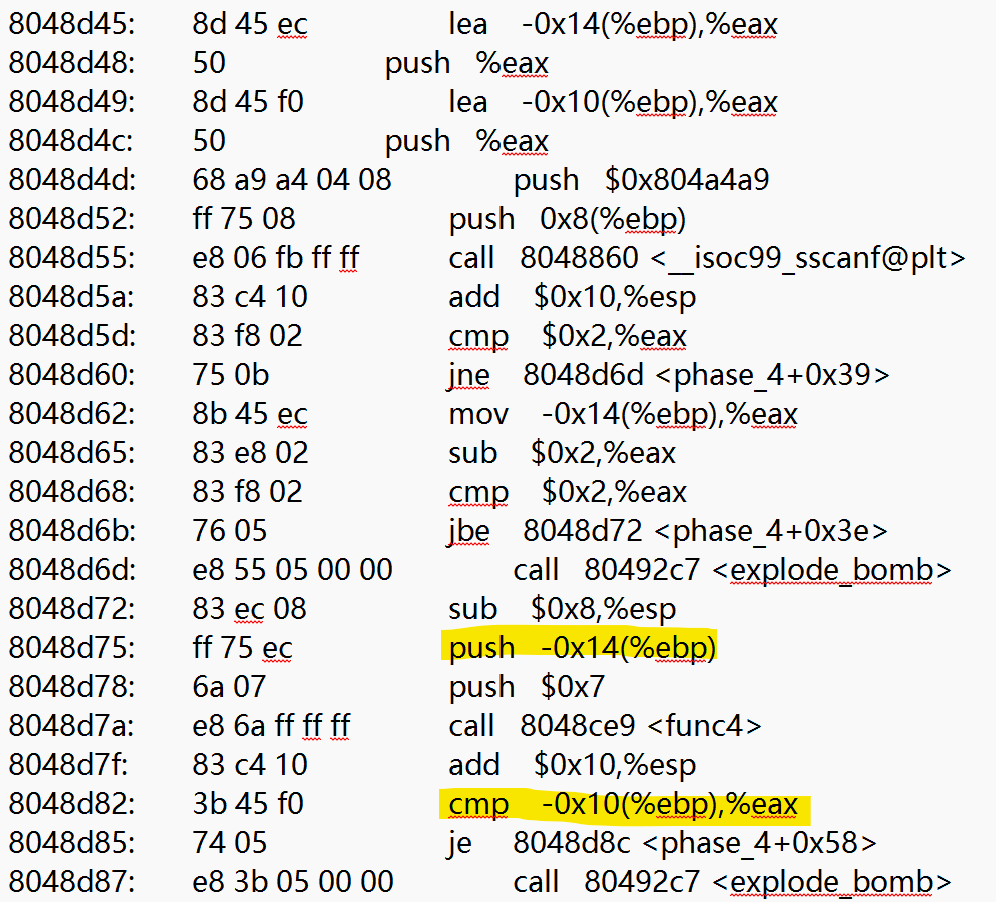
为什么是弱化版呢？

诶，你会发现，它是一个数作为函数的参数，第二个数与函数的返回值进行比较

然后就会有一种很妙的做法

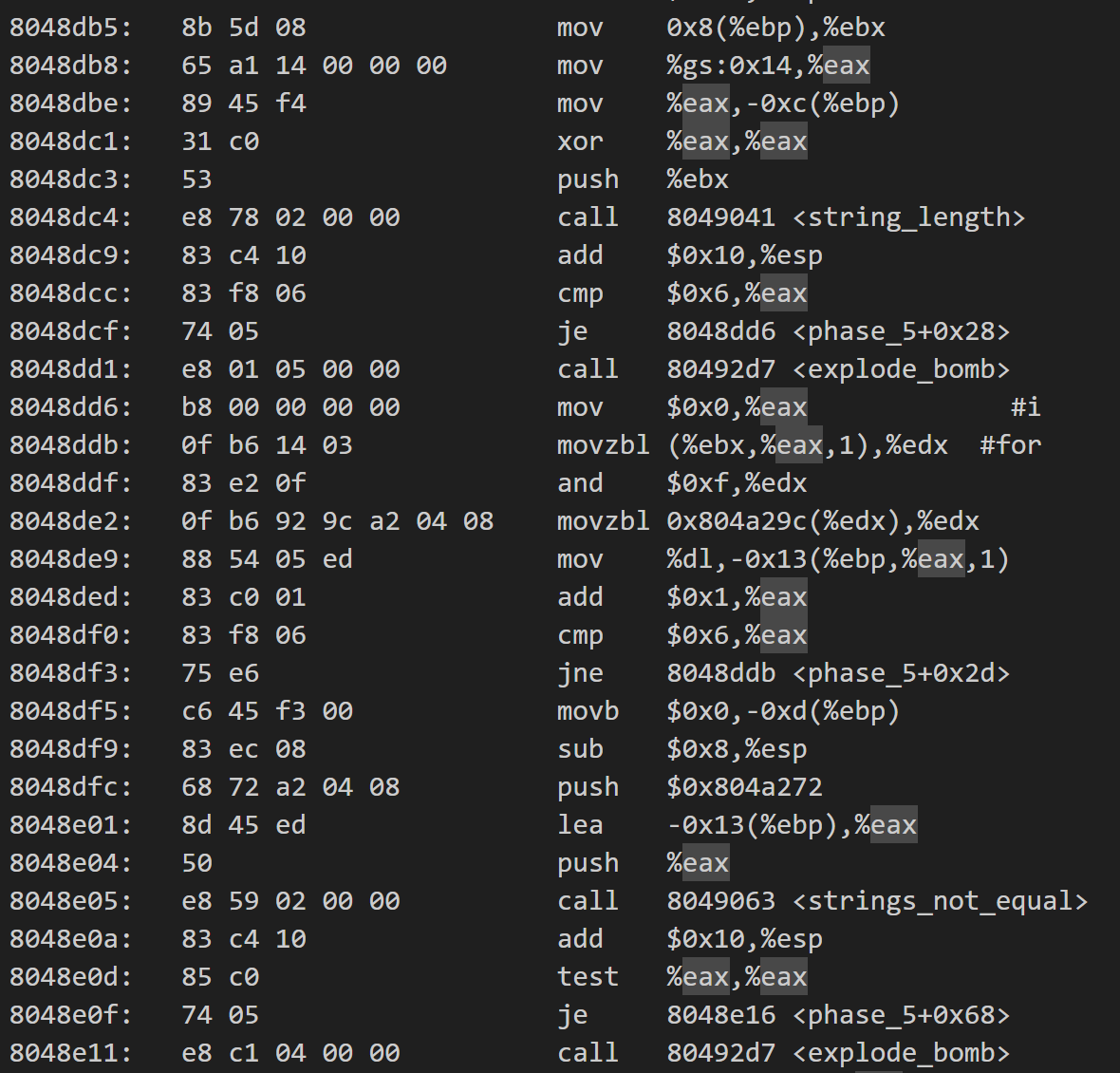
先输入两个数，然后其中一个数是作为函数的参数，要满足题中的要求，另一个数随便输。

然后合理利用gdb读出函数的返回值，其实就是我们要的第二个数了。



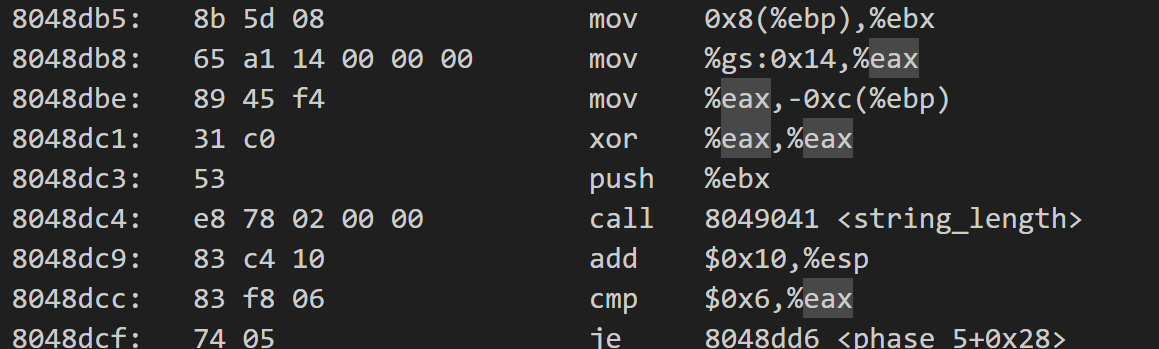
所以弱化版根本就不用看函数，连试都不用试就能过。。。。。

关于phase\_5:

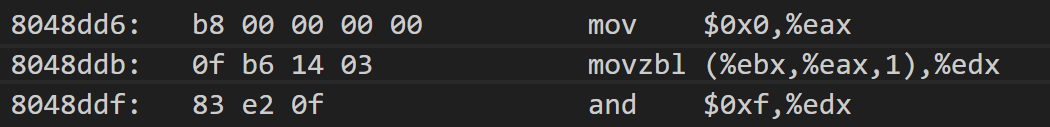


关于phase\_5内容大意

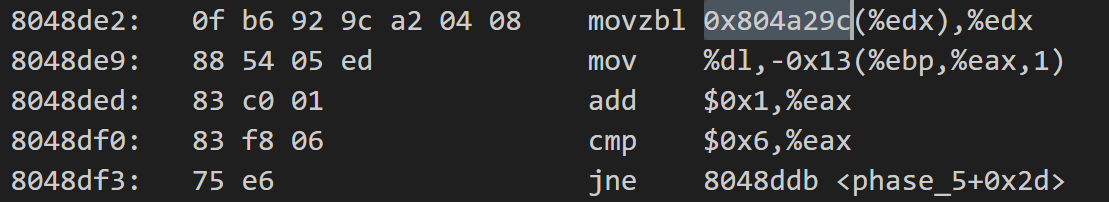
它把我们输入的字符串拿去求长度，要求长度一定要是6



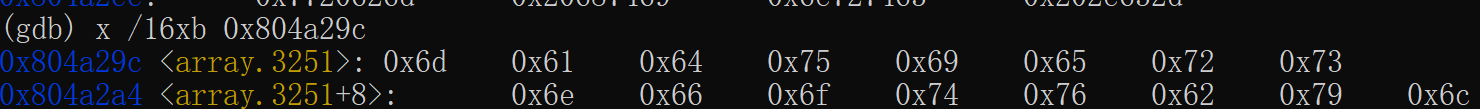
然后把每个字符取出来，并取ascii码的低4位



然后在这个地址找第几个，并把它放到栈中

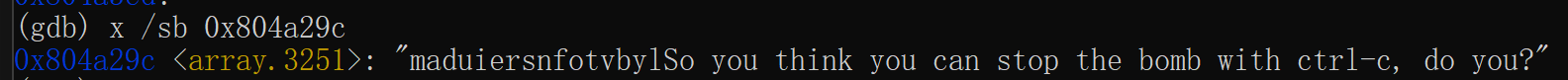


诶，你一读这部分的地址，发现诶，怎么是奇怪的东西



然后你又看了下下文的字符串比较，你猜应该存的是字符串数组吧

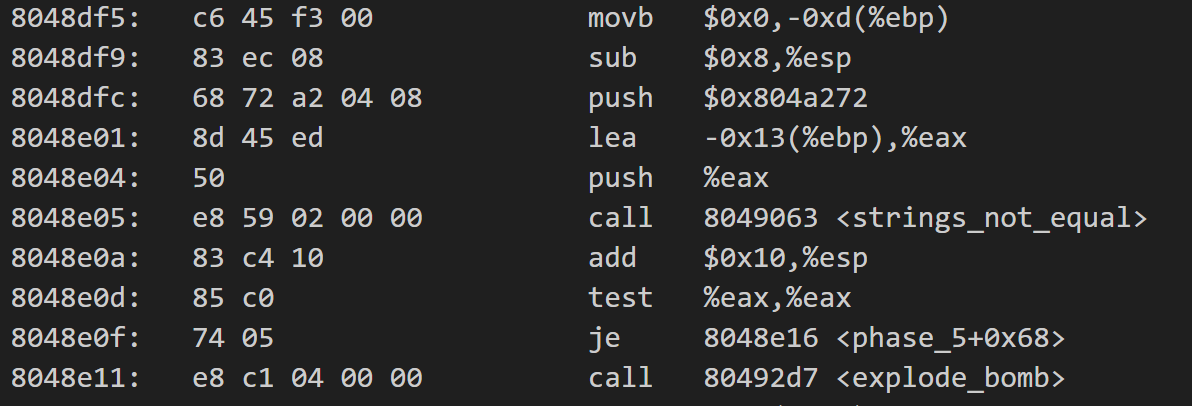
嘿，结果一试，真是



但是我们只要前16个，即0到15



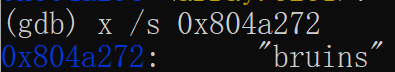
OK，接下去看



它最后在栈中放了个0，根据字符串的特点，结束是’\0’，ascii码值为0

然后就比较字符串是否相等了

然后我们看看这个字符串是哪个

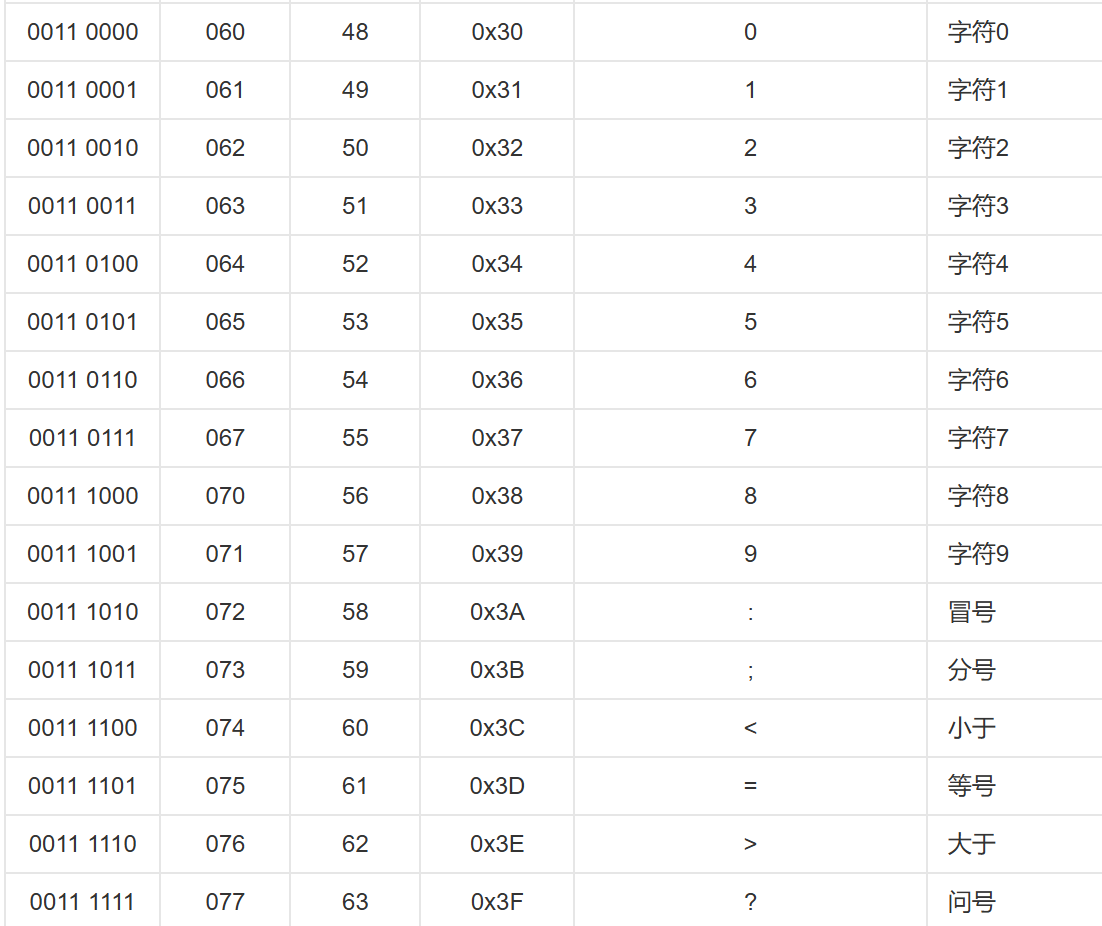


发现要上述字符串

bruins分别位于13，6，3，4，8，7,

所以你要输入的字符ascii低4位应为上述

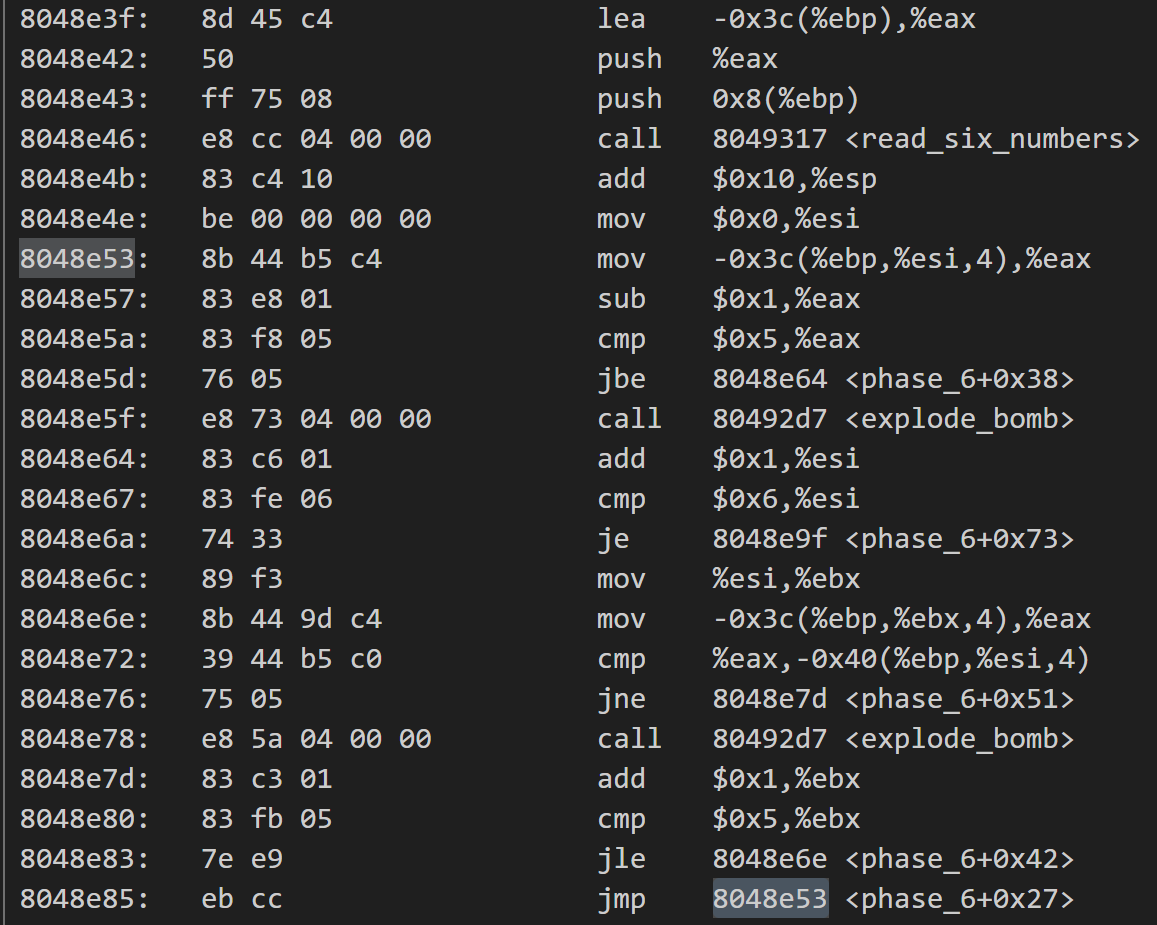
因为懒，又因为’0’的ascii码后4位是0，所以直接找0开始的16个字符是哪个就行了



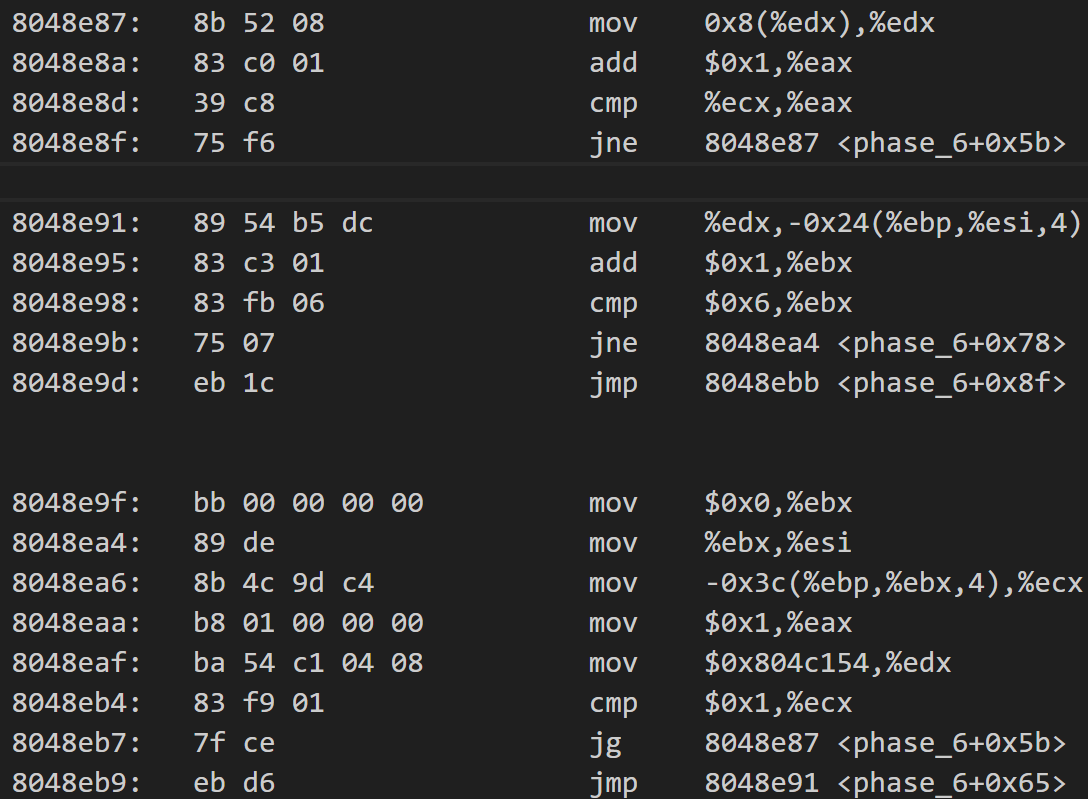
关于phase\_6:

因为phase\_6真的好麻烦就大概说说吧

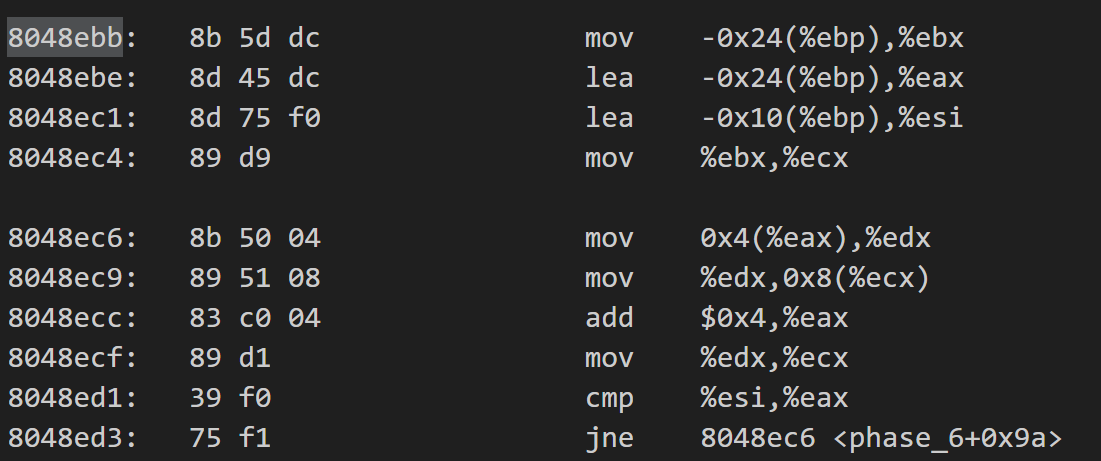
首先读入了6个数，然后用两重循环判断有没有重复（大概是这样吧？）



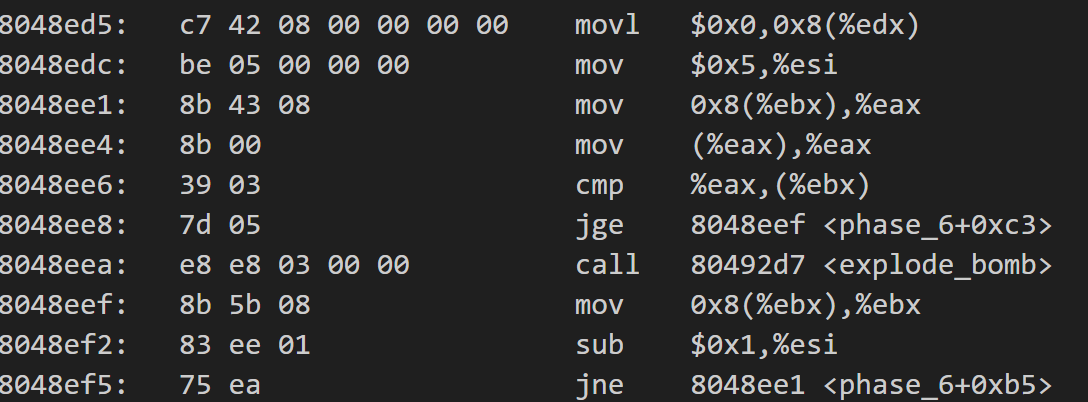
然后他就要把你输入这六个数用作下标，取出对应链表的节点放在栈中



然后他就是要把这些链表的节点按你给的数据重新连接



最后遍历链表，要使得前一个点的值大于等于下一个点的值



所以我们可以先把链表的值读出来，从大到小排序

读出来有两种方法

一种是直接读地址，然后算地址的偏移量，存的就是下一个节点的地址

这样就可以一个个列出来了

还有一种直接输入1 2 3 4 5 6

然后在栈中把节点的地址读出来

然后就可以比较顺利地把值读出来了

我是采用第二种方法

最后读出来的值排序一下就好了

把相应下标输入就是答案了

最后讲讲secret\_phase吧

做完前6题之后

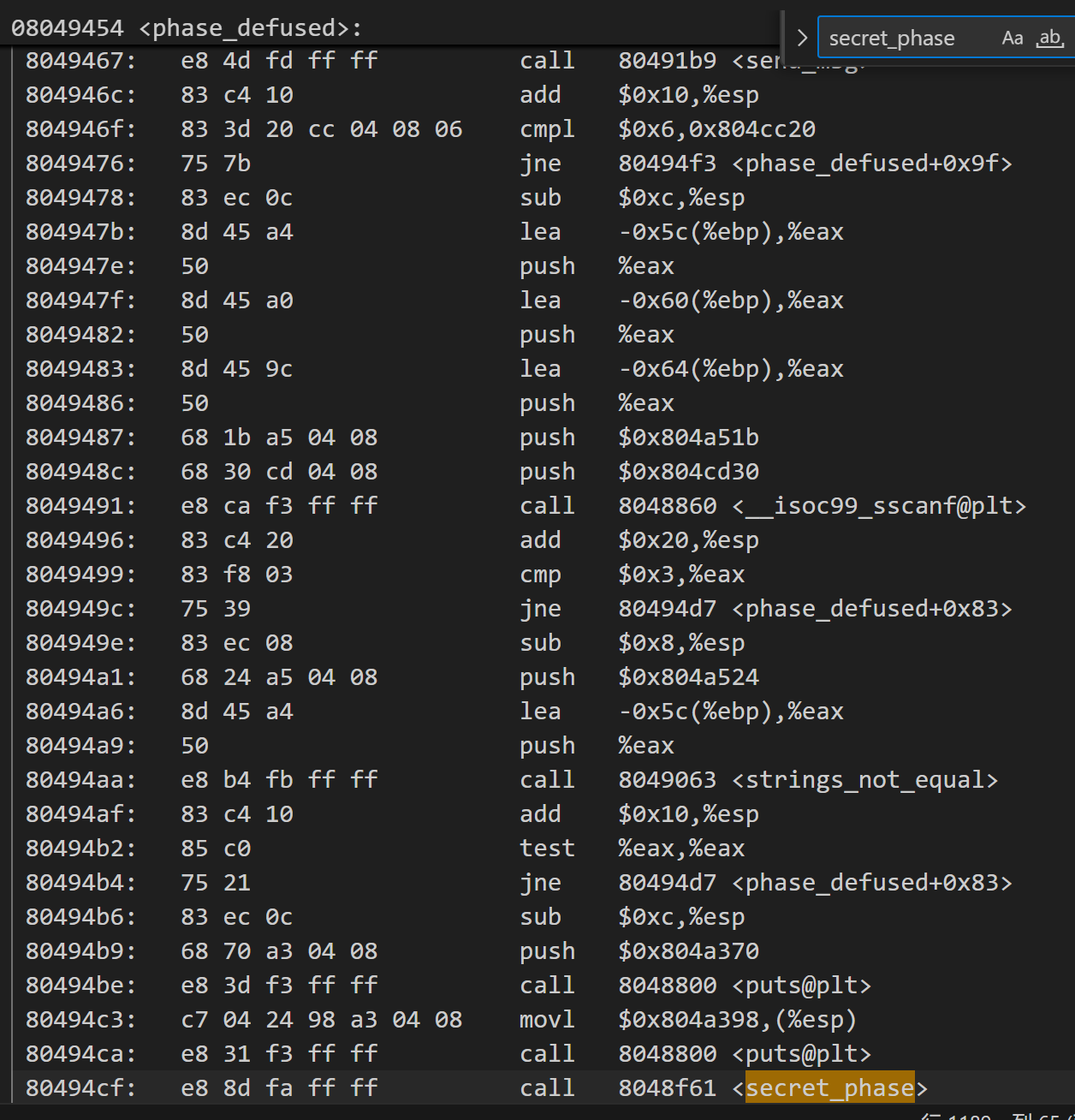
你发现直接结束了

bomb.c也没有直接调用

所以一想就是在哪调用了

结果你ctrl+f一找

发现在phase\_defused部分



怎么到那里呢

你发现它用sscanf读了个字符串

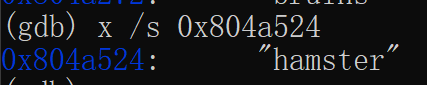
你一查

诶，这不就是第4题输的字符串吗

但是它多了一个参数%s

所以我们要在第四题后面输个字符串

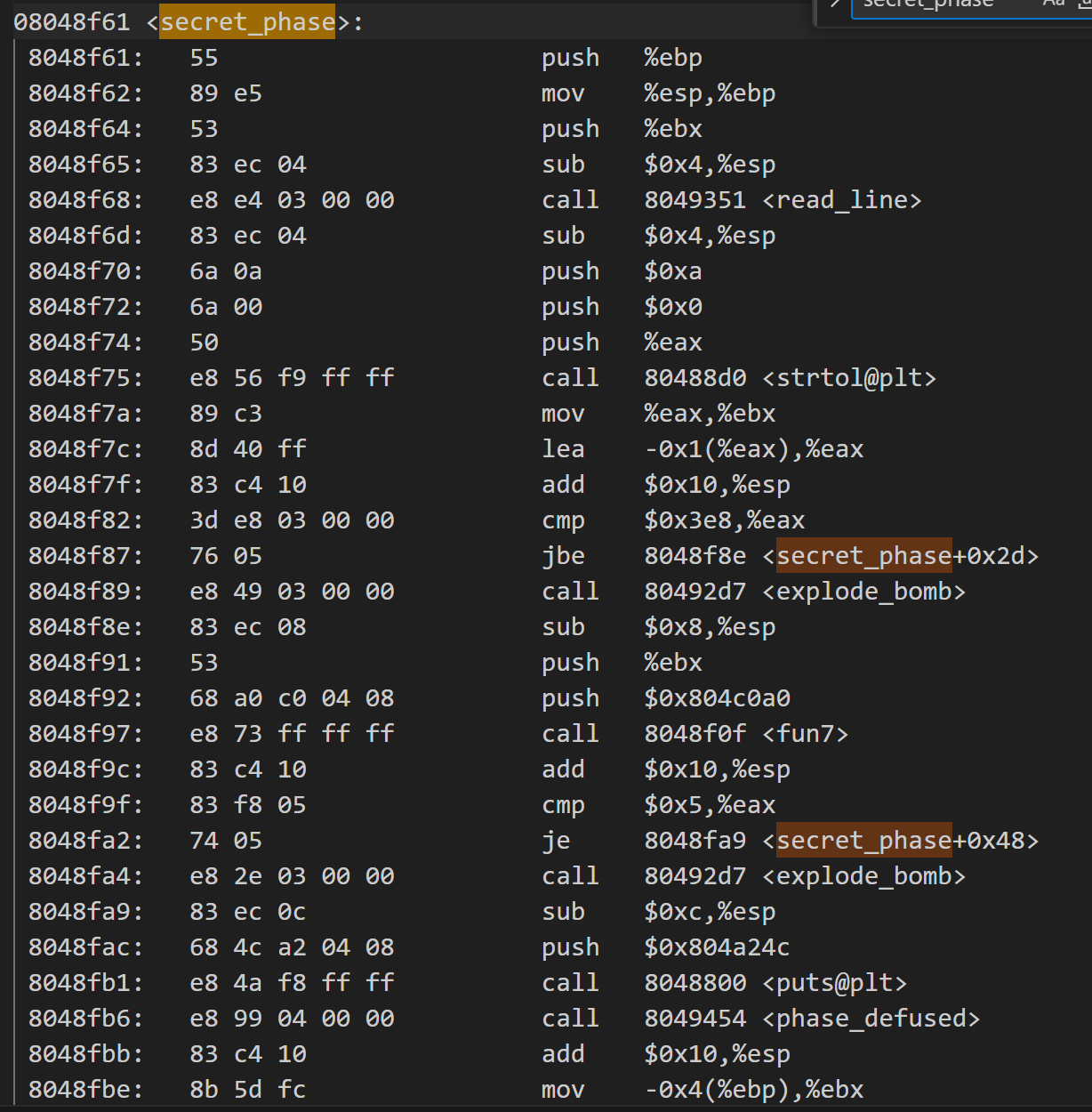
而且发现它要把这个字符串与其他字符串比较



说明我们应该追加的是上述字符串

然后重新运行后

你发现进来secret\_phase了

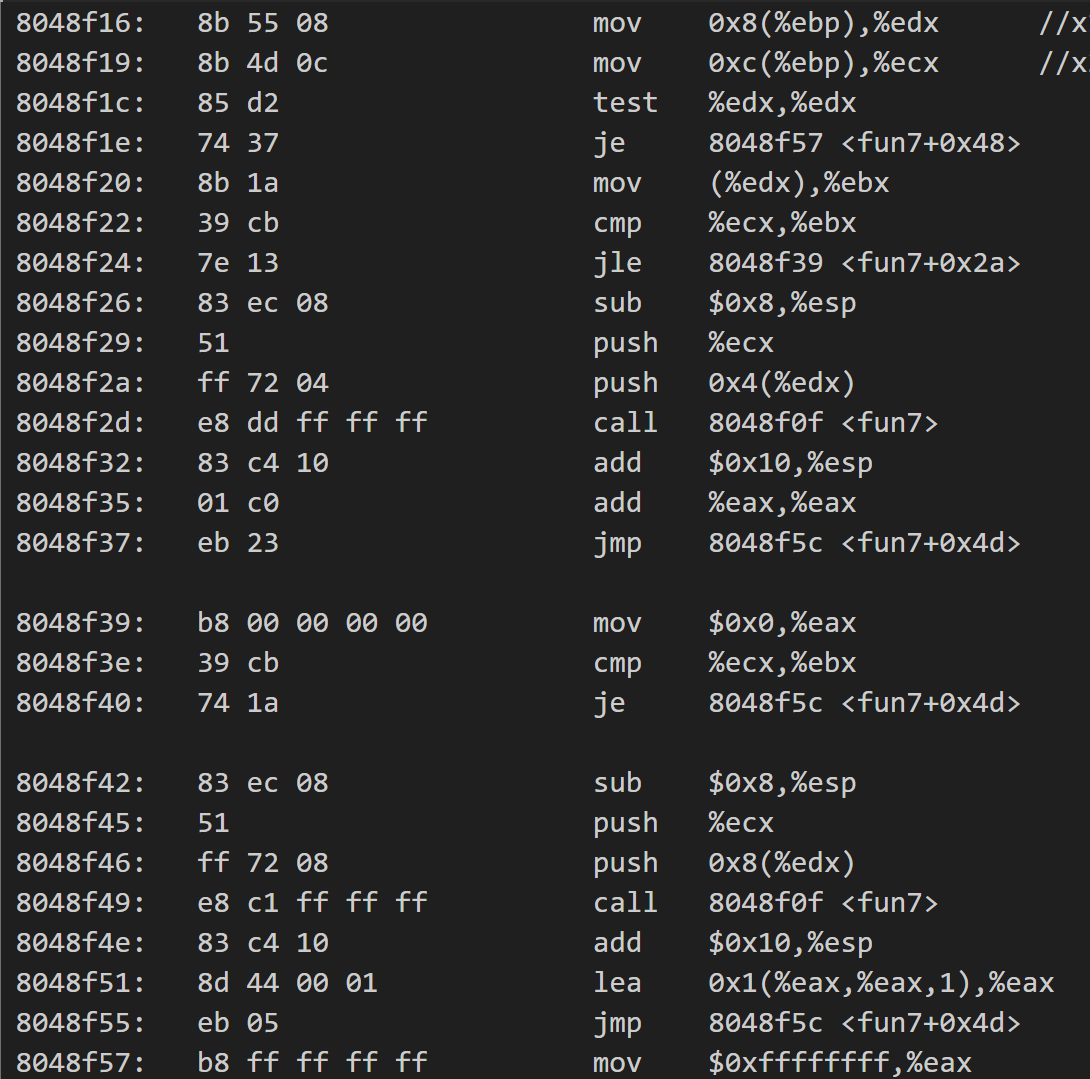


简单概述上述内容就是输入一个数，作为函数的参数，要使得返回值是5，同时该数的大小小于等于1001

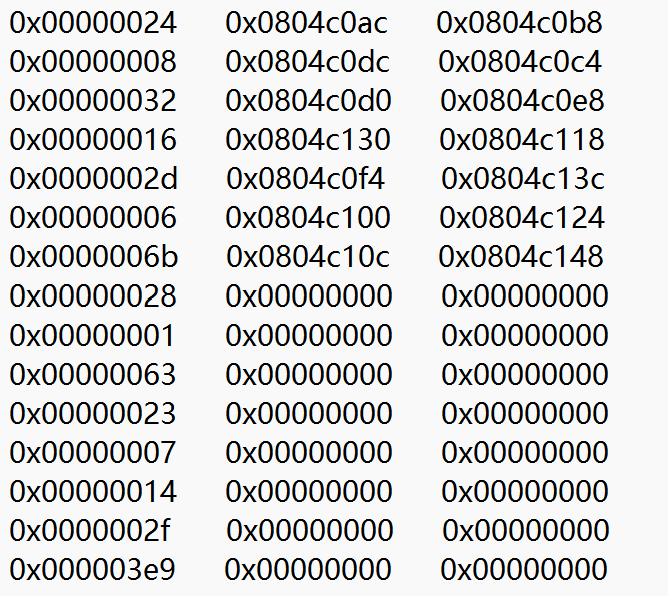
然后我们翻翻func7

发现又是一个递归函数

但是这次好像是个结构体，有一个值，两个指针



然后我们把内存中的值读出来整理一下

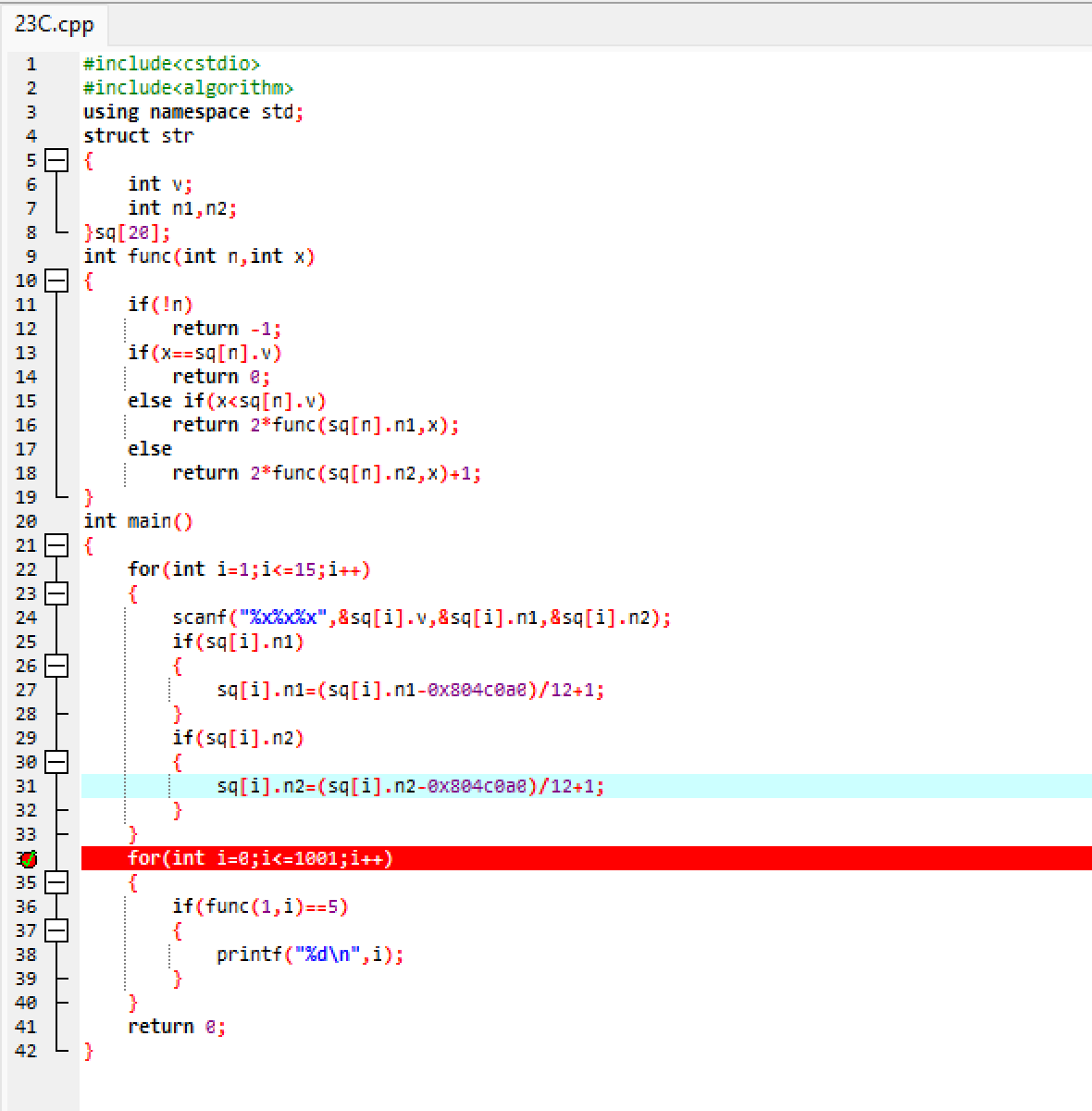


然后我们大概看看这程序在讲什么

然后我们又可以用高级语言写出来了

同时枚举一下参数，就可以发现它的输出是47

即我们的答案是47了~



补：有人发现这其实是一棵二叉搜索树，即bst，返回的是节点的下标

(y1s1确实是，当时做的比较直接也没有去想就是了)