

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： 二进制程序分析**

**院 系 ：计算机科学与技术**

**专业班级 ： 计算机202203班**

**学 号 ： U202215643**

**姓 名 ： 王国豪**

**指导教师 ： 张 勇**

**2024 年 4 月 12 日**

**一、实验目的与要求**

通过逆向分析一个二进制程序（称为“二进制炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示各方面知识点的理解，增强反汇编、跟踪、分析、调试等能力。

实验环境：Ubuntu，GCC，GDB等

**二、实验内容**

作为实验目标的二进制炸弹（binary bombs）可执行程序由多个“关”组成。每一个“关”（阶段）要求输入一个特定字符串，如果输入满足程序代码的要求，该阶段即通过，否则程序输出失败。实验的目标是设法得到得出解除尽可能多阶段的字符串。

为了完成二进制炸弹的拆除任务，需要通过反汇编和分析跟踪程序每一阶段的机器代码，从中定位和理解程序的主要执行逻辑，包括关键指令、控制结构和相关数据变量等等，进而推断拆除炸弹所需要的目标字符串。

实验源程序及相关文件 bomb.rar

bomb.c 主程序

phases.o 各个阶段的目标程序

support.c 完成辅助功能的目标程序

support.h 公共头文件

**阶段1： 串比较 phase\_1(char \*input);**

要求输出的字符串(input) 与程序中内置的某一特定字符串相同。提示：找到与input串相比较的特定串的地址，查看相应单元中的内容，从而确定input 应输入的串。

**阶段2：循环 phase\_2(char \*input);**

要求在一行上输入 6个整数数据，与程序自动产生的 6个数据进行比较，若一致，则过关。提示：将输入串input拆分成 6个数据由函数 read\_six\_numbers(input, numbers) 完成。之后是各个数据与自动产生的数据的比较，在比较中使用了循环语句。

**阶段3：条件分支 phase\_3(char \*input);**

要求输入一个整数数据，该数据与程序自动生成的 一个数据比较，相等则过关。提示：在自动生成数据时，使用了 switch … case 语句。

**阶段4：递归调用和栈 phase\_4(char \*input);**

要求在一行中输入两个数，第一个数表示在一个有序的数组（或者binary search tree）中需要搜索到的数，该数是在一定范围之内的；第二个数表示找到搜索数的路径（在树的左边搜索编码为二进制位0，在树的有边搜索编码为二进制位1）。

**阶段5：指针和数组访问 phase\_5(char \*input);**

要求在一行中输入一个串，该串与程序自动生成的串相同。在生成串和比较串时，使用了数组和指针。

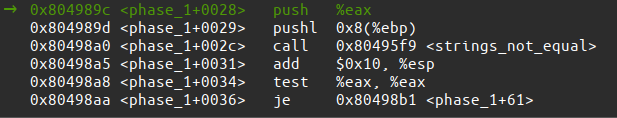
**阶段6：链表、结构、指针的访问 phase\_6(char \*input);**

要求在一行中输入6个数，这6个数是一个链表中结点的顺序号（从 1 到6）。按照输入的顺序号，将对应链表结点中的值形成一个数组。若该数组是按照降序排列的，则过关。

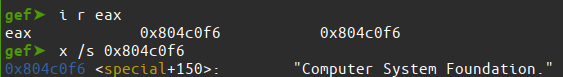
**三、实验记录及问题回答**

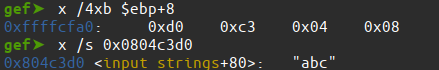
**（1）实验任务1 的实验记录**

**Phase\_1:**

我们先随便输入一个字符串“abc”，接着调试进入phase\_1,我们可以观察到  


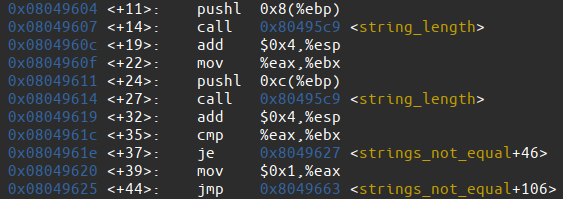
Phase\_1调用了strings\_not\_equal函数，并且调用前传入了两个参数，我们猜测其中与我们想要的答案有关。我们依次输出其中对应地址的内容：





要注意的是push %eax 和 push 0x8(%ebp)的区别，一个是传入寄存器的值，一个是将ebp指针+8后指向的内容传入栈，我们猜测这两个参数都是字符串的地址，可以知道“abc”是我们自己传入的参数，那么另外一个就是我们想要的答案。

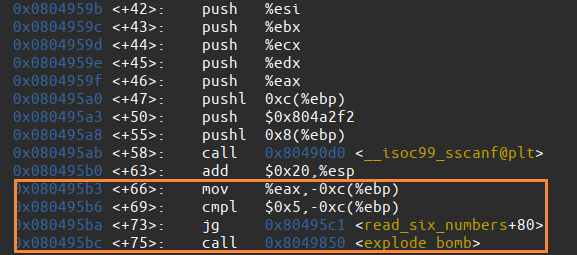
我们还可以进入strings\_not\_equal函数进行验证，我们反汇编strings\_not\_equal函数，分析可以知道调用了两次的string\_length函数



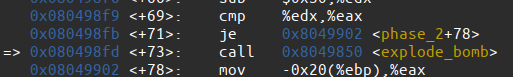
且分别将我们刚刚分析的两个参数传入，我们由此可知，是分别计算我们输入的字符串和答案字符串的长度，如果不相等的话直接跳转到strings\_not\_equal+106直接返回，不然接着进行下一步的判断。验证成功，答案为“Computer System Foundation.”。

**Phase\_2:**

我们还是随机传入六个数字1 2 3 4 5 6，进入phase\_2调试，我们进入read\_six\_numbers查看汇编代码，我们注意到这么一段代码

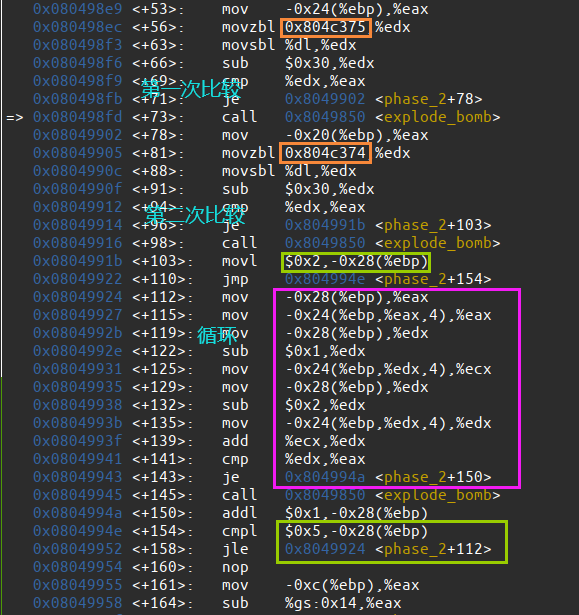


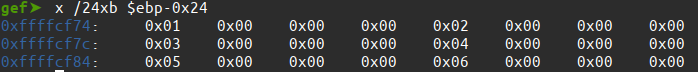
这里将0x5和-0xc(%ebp)中的值进行比较，只有-0xc(%ebp)中的值大于5才不会引发爆炸，我猜测是上面调用的函数将我们输入的六个数字和答案进行匹配，并将匹配的个数写入 -0xc(%ebp)中，后来发现不是，这个是读取我们输入了几个数字，不符合要求的话直接爆炸。

我们接着一步一步调试，着重关注比较的地方。  


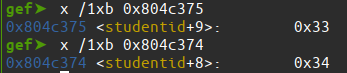
我们来到一个比较，且比较完发现炸弹要炸了，我们查看一下寄存器edx和eax的值  
  
 eax中是我们输入的第一个数字，所以对应的3应该为答案的第一个数字，我们这时候反汇编一下代码：

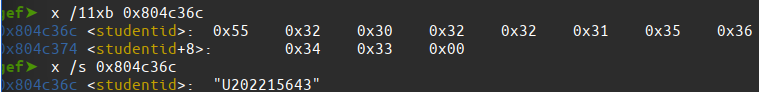
可以看到先进行了两次的比较，后循环进行比较，-0x28(%ebp)存放的是当前匹配数字个数，当这个个数小于等于5的时候一直执行循环，那么-0x24(%ebp)存储的是什么呢，我们可以试着去输出一下。



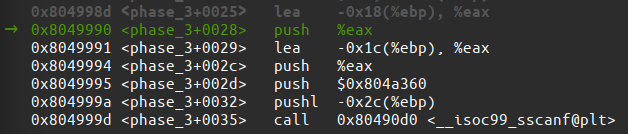


发现这个存储的就是我们输入的数字，所以循环体中的基址+变址寻址 -0x24(%ebp,%eax,4)都是在访问我输入的数字，我们再分析可以知道，从第三个数开始，答案都是由我们输入前两个数字相加得来，可以理解为类斐波那契数列，而答案的前两个都是一个定值，分别为是我用橙色方框括起来的两个地址指向的值再减去0x30，分别输出一下可以得知分别为3和4，那么最终的答案就是3 4 7 11 18 29。

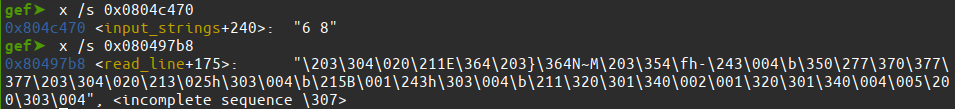


做完这个后我们还可以深入分析一下这个phase\_2的逻辑，发现0x804c375是我们输入的学号的一部分，我们的答案与我们输入的学号后两位有关系，我们输入的学号存储在一个叫做studentid的全局变量中。  
  
**Phase\_3:**

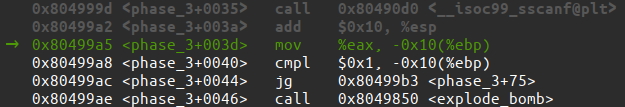
先按要求输入两个int类型数字，我输入了6 8，接着进入调试，密切关注传参，



我们依次取出两次push的eax的值，然后打印出来

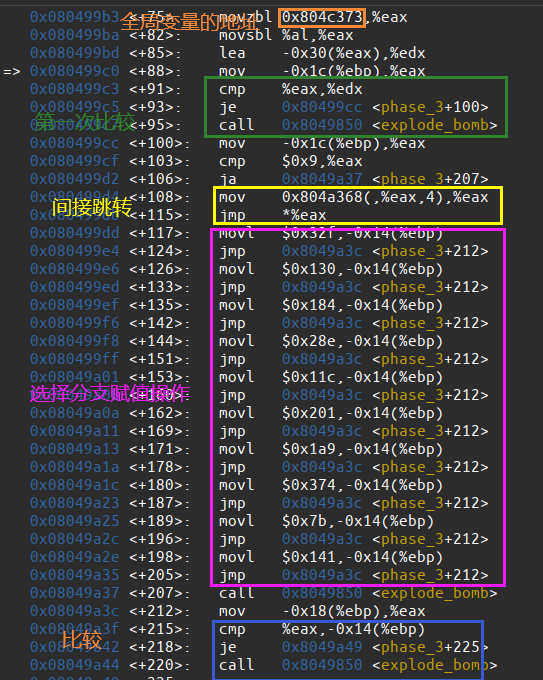


发现其中一个为我们输入的两个数字，都是以字符串存储，我猜测下面调用的sscanf就是将我们的字符串转换成数字。



接着往下调试可以验证我们的猜想，eax存储的是sscanf的返回值，如果无法正确识别两个数字，就会炸弹爆炸。且这个时候-0x10(%ebp)存储的是我们我们写入的数字的个数。

我们接着调试进入一处比较

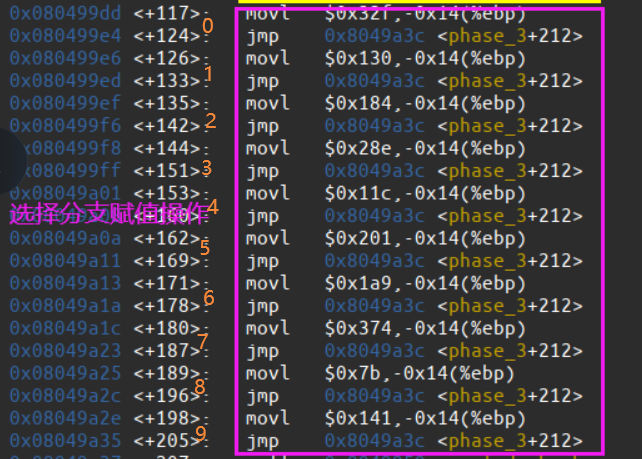


请注意，这里的0x804c373和我们phase\_2中验证的学号的存储地址接近，我们可以验证这就是我们学号的一部分。



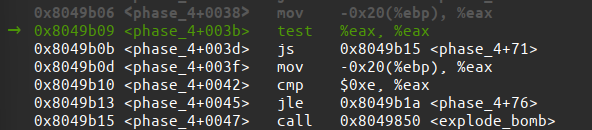
所以我们的第一个答案就是我们输入的学号的第8位，也就是6。比较完第一个数字以后，会根据再次根据我们输入的第一个数跳转到地址为0x804a368+第一个数\*4的值为地址的地方。记录此时要跳转的地址  


发现巧合跳转到了我们<phase3+171>的位置,将0x1a9赋值给了答案，再跳转到<phase3+212>与我们输入的第二个数字（存储在-0x18(%ebp)）进行比较。所以我们第二个答案就是0x1a9也就是425。

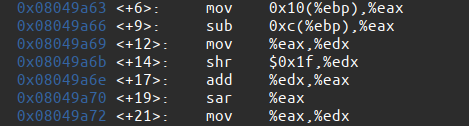
我们再回顾一下分支，根据我们学号的第八位选择我们的第二个答案，十个分支分别对应我们学号第八位的10中可能行。  


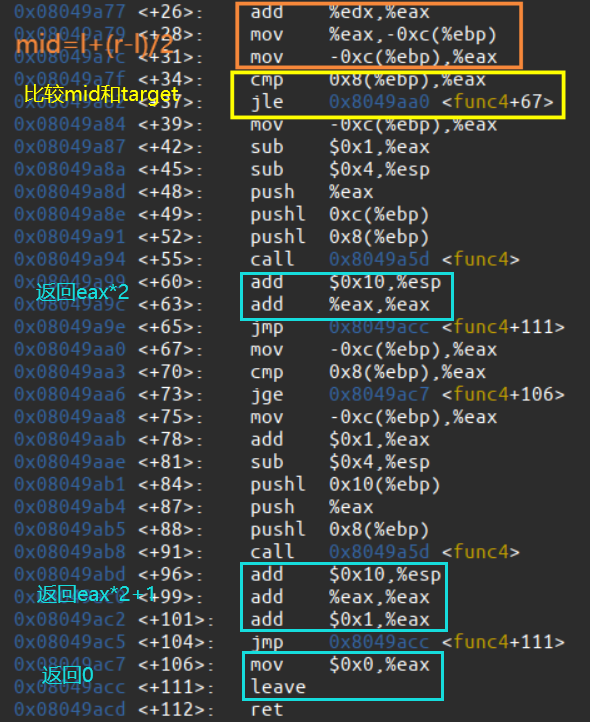
**Phase\_4:**

随机输入5 1 abc，进入调试。



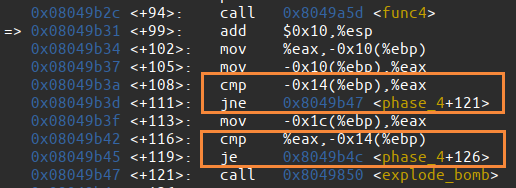
输入发现-0x20(%ebp)存储的是我们第一个数字，发现这个数字有范围显示，需要在[0,14]中。判断完数的范围后，传了三个参数分别是我们输入的第一个数，0和14，接着进入func4函数，我们要注意三个参数，分别是0x8(%ebp),0xc(%ebp),0x10(%ebp),分别对应我们传入的第一个数字，0,14,我们可以把func4定义为func4(target,l,r)。

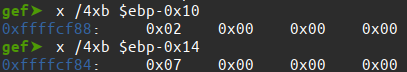
Func4的本质就是二分查找，首先我们需要计算偏移 (r-l)/2，下面这张图就是计算过程



我们接着计算mid=l+(r-l)/2,然后与我们的target进行比较。分三种情况，如果mid等于target，直接返回0，如果mid大于target，我们就要把mid-1，然后递归调用func(target,l,mid)，再返回2\*eax；如果mid小于target，我们就要mid加一,然后递归调用func(target,mid,r),再返回eax\*2+1；最后函数会找到一个mid等于target，然后层层返回eax。

回到phase\_4函数以后，第一次比较会将返回值和存储在-0x14(%ebp)中的值进行比较





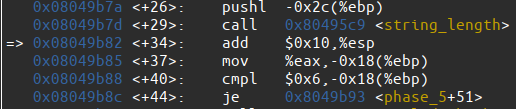
-0x14(%ebp)中的值为7，我们的返回值为2。所以我们需要设定我们第一个数，使得返回值为7，经过计算得到应该是14，第二个比较是把我们输入的第二个数字和-0x14(%ebp)中的值进行比较，我们可以找到这样一段代码：



可以知道第二个数字是一个定值，就是7。故最后的答案就是14 7。

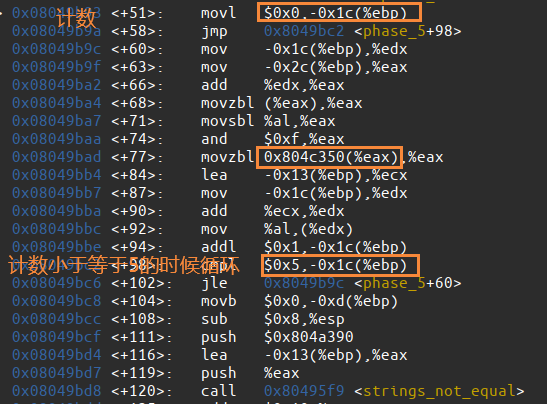
**Phase\_5:**

我们还是先输入一个字符串“abcdef”。然后进入函数查看大概的执行情况。



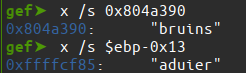
我看到传入了一个参数，然后调用了string\_length函数，经查看，我们传入的就是我们输入的字符串的地址。并且这个函数的返回值会和6进行对比，所以说答案字符串的长度是6。





这里的-0x1c(%ebp)用作计数，表示访问的数据到起始的偏移，-0x2c(%ebp)中的值是我们输入的字符串的首地址。那么这个0x804c350是什么东西呢，我们可以输出一下。



这是一个字符串的首地址。那么上面在进行什么操作呢？我们先往下面看。后面又调用了strings\_not\_equal函数，而且调用前传了两个参数，我们分别打印一下  


进入strings\_not\_equal函数后会对这两个字符串进行比较，如果不相等的话返回1，相等的话拆除炸弹。

那么我们怎么输入字符串才能经过运算得到bruins呢，看最上面那张图，每次都会根据计数的-0x1c(%ebp)选取我们输入的字符串的指定位数上的字符，通过无符号扩展为四个字节，再选取低字节进行有符号扩展为四个字节，然后通过和0xf与运算，只保留低四位保存在eax中，这说明答案只和我们输入的字符串的低四位有关。



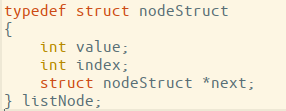
上面这个图表示根据字符串的低四位取出0x804c350对应偏移量的字符。

接着把$ebp-0x13赋值给ecx,偏移量给edx,接着让edx加上ecx,这时候edx中就是一个目标地址，接着我们要把al的值放到edx的值为地址的存储区。

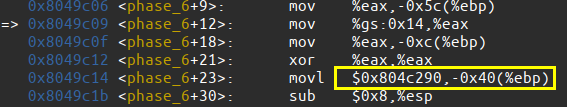
总的来说，答案和我们输入的六个字符串的低四位有关，答案并不唯一，只要我们能够根据我们输入的字符串的低四位去0x804c350指向字符串取出对应偏移的字符，最后6个字符组成的字符串是bruins的话就能过关。

我们注意到’a’的ascii码是0x61，所以我们可以巧妙利用小写字母的ascii码来得到我们想要的偏移，我的答案是mfcdhg，恰好可以从0x804c350指向的字符串中取出bruins。

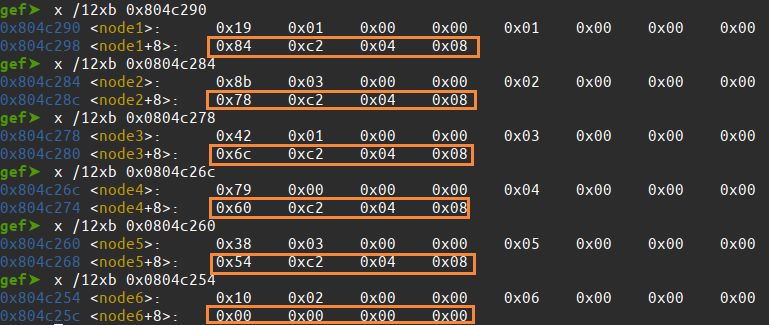
**Phase\_6:**



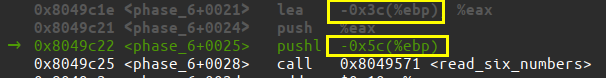
我们先查看一下节点的定义

我们还是试探性输入1 2 3 4 5 6，我们进入调试后，要关注一下这个地址  


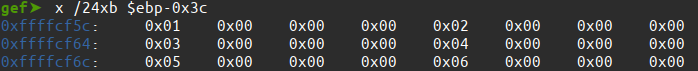
我们试着去打印出来，猜测是一个节点的地址，输出打印以后，我猜测第一个数字是value，第二个是index，第三个是指针，事实也如此，这样我就找到了存储数据的链表。



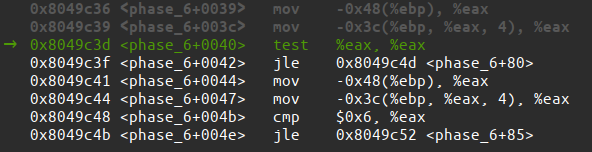
注意我们链表中的第一个元素的地址存储到了$ebp-0x40中。接着我们准备进入read\_six\_numbers,我们还先传入了两个参数。



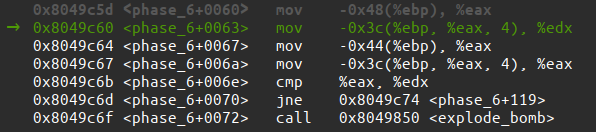
执行完函数以后我们打印刚刚的两个参数，发现其中一个存储我们输入的字符串，一个是存储我们字符串中的数字。



接着把0放入$ebp-0x48,然后这个值会和5进行比较，所以我们猜测这个存储的就是到数组开头的偏移。



接着我们采用基址+变址寻址将第一个值取出来做比较，这个值要小于等于6，不然爆炸。接着我们把这个偏移量（当前是0）加上1存储到$ebp-0x44中。

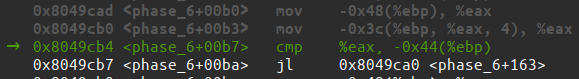


接着我们将第一个数和第二个数取出来比较，相等的话爆炸。比较完以后又把$ebp-0x44中的值加一，重复上面的过程，完成将第一个数和后面五个数进行比较，第一个数不能和后面五个数相等。比较这一轮以后，再将$ebp-0x48中的值加一，然后再取出这个偏移量的数，判断这个数和0和6的大小，要大于0小于等于6. 接着循环上面的比较，也就是说要确保我们输入的六个数互不相等。

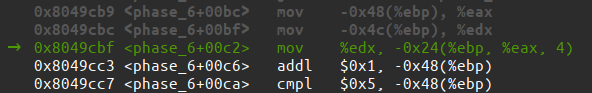
接着把0放入$ebp-0x48,把我们链表第一个node地址放在$ebp-0x4c,把1放在$ebp-0x44。



接着把我们输入的第一个数1放到eax和我们刚刚放到$ebp-0x44的1进行比较。小于的时候跳转不然进行下一步。

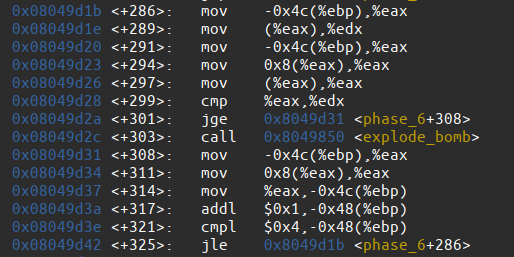


把第一个节点的地址放在-0x24(%ebp,%eax,4)的地方。



接着进行循环操作，按照我们输入的顺序把对应节点的地址依次存储到  
-0x24(%ebp,%eax,4)中。接着还要进行循环操作，我们就要根据我们排的这个序列对链表进行重新排序，每一个节点的next指向下一个节点的地址，这里有个细节，退出循环后，单独将最后一个结点的下一个结点置为0，即NULL。

最后一步我们就要进行比较，-0x4c(%ebp)就是取出我们当前节点的地址，0x8(%eax)就是取出下一个节点的地址，然后二者比较，如果符合要求就让

-0x4c(%ebp)等于next，循环操作，-0x48(%ebp)这个存储我们循环的次数。  


所以我们就按照我们输入的数字序列排序后的链表进行比较大小，得到正确的序列。经过计算，我们的答案是2 5 6 3 1 4。

**（2）拆除炸弹的过程中关键操作**

拆炸弹的时候看着一堆的汇编代码，没有c语言的直观，如果没有一些敏锐性的话，会陷入混乱中，我们需要掌握一些关键的地方。

1. 注意0x30,’0’的ascii码是0x30, 我们输入的数字和字符串都会被当作字符串存储起来，后续处理提取成数字的时候如果出现了-0x30(%寄存器)这个就是应该将我们输入的字符串转换为数字，方便后续操作。
2. 进入一个新的关的时候，要先总体全局看一下大致的函数调用情况， push了几个参数，每次我们都可以输出这几个参数，提前对接下来要运行的函数有个大概的了解。
3. 一定要边看汇编代码，边记录，由于局部变量都是通过类似-0x1c(%ebp)等方式去访问，这样如果多几个变量就会不直观，我们如果不记录下来这个地方存储了什么，接下来根本不知道对什么内容进行了操作，最好是get到一个点就记录一个点。

**四、体会**

在进行二进制炸弹拆除实验的过程中，我深刻领悟到了对计算机系统底层运作的理解和操作技巧的重要性。以下是我的一些心得体会：

首先，我重新从新的方面认识到了汇编代码的重要性。在面对没有源代码只有可执行文件的情况下，充分利用反汇编功能并理解汇编代码的细节，让我能够反向推断出源代码的思路和想法。这次实验让我更加明白了汇编代码在理解计算机系统中的关键作用。

其次，理解程序逻辑的关键性也是我在实验中的重要收获。仔细分析程序的逻辑结构，理解每一步操作的含义和作用，尤其是对于树形结构等复杂逻辑的理解，直接影响了我解决问题的效率和准确性。

另外，注重细节和耐心也是我在实验中不断强调的重点。汇编代码繁杂，很容易漏看一些关键节点，因此在研究代码时，我始终保持心平气和，一步步深入分析，以免遗漏重要信息。

灵活运用调试工具如GDB是解决问题的关键。通过GDB，我能够快速定位问题所在，查看地址对应值，观察寄存器状态等，大大简化了解决问题的过程，提高了效率。

与同学交流学习也是我在实验中的重要策略。通过积极交流，我学到了新的知识和技巧，提升了团队合作能力，解决问题的效率也得到了提升。

最后，总结经验教训是巩固学习成果的关键一环。通过总结实验中遇到的问题和解决方法，我能够更好地应对类似的挑战，并不断提升自己的技能水平。

通过这次二进制炸弹拆除实验，我不仅深入理解了汇编代码的工作原理，反汇编的重要性和调试工具的使用方法，还提升了自己的问题解决能力和团队合作意识。这些经验对我今后的学习和工作都将产生积极的影响。