湖南大学课程实验报告

课 程 名 称： 计算机组成与结构

实验项目名称： 二进制炸弹

专 业 班 级：

姓 名：

学 号：

指 导 教 师：

完 成 时 间： 2016 年 10 月 30 日

软件工程

|  |
| --- |
| 实验题目：  APP Bomb Lab |
| 实验目的：  程序运行中有6个关卡（6个phase），每个phase需要用户在终端上输入特定的字符或者数字才能通关，否则会引爆炸弹！那么如何才能知道输入什么内容呢？这需要你使用gdb工具反汇编出汇编代码，结合c语言文件找到每个关卡的入口函数。然后分析汇编代码，找到在每个phase程序段中，引导程序跳转到“explode\_bomb”程序段的地方，并分析其成功跳转的条件，以此为突破口寻找应该在命令行输入何种字符通关。 |
| 实验环境：  linux，终端，gdb工具 |
| 实验内容及操作步骤：  查看给出的bomb.c中的代码，得知控制检测密码正误的6个函数分别为：  phase\_1，phase\_2，phase\_3，phase\_4，phase\_5，phase\_6。  使用gbd工具对可执行文件bomb进行反汇编并将文件保存在temp.txt中，分析时直接看文件即可    得到六段汇编代码，将其复制成文本后，对其进行分析，**红色字体为分析**：  **phase\_1代码：**  08048f61 <phase\_1>:  8048f61: 55 push %ebp  8048f62: 89 e5 mov %esp,%ebp  8048f64: 83 ec 18 sub $0x18,%esp  8048f67: c7 44 24 04 5c a1 04 movl $0x804a15c,0x4(%esp) //将esp寄存器地址指向$0x804a15c  8048f6e: 08  8048f6f: 8b 45 08 mov 0x8(%ebp),%eax将参数1数据传送给eax寄存器  8048f72: 89 04 24 mov %eax,(%esp)  8048f75: e8 31 00 00 00 call 8048fab <strings\_not\_equal>  8048f7a: 85 c0 test %eax,%eax 比较用户输入和$0x804a15c中的值  8048f7c: 74 05 je 8048f83 <phase\_1+0x22>  8048f7e: e8 4e 01 00 00 call 80490d1 <explode\_bomb>  8048f83: c9 leave  8048f84: c3 ret  可以看出，phase\_1的密码是固定储存在$0x804a15c中的。令$0x804a15c中的数据与用户输入的数据比较，若相同则跳过explode\_bomb，避开炸弹。用命令x/s 0x804a15c查看其中的数据如下    因此知道第一个密码为：We have to stand with our North Korean allies.  **phase\_2代码：**  08048d6a <phase\_2>:  8048d6a: 55 push %ebp  8048d6b: 89 e5 mov %esp,%ebp  8048d6d: 56 push %esi  8048d6e: 53 push %ebx  8048d6f: 83 ec 30 sub $0x30,%esp //开辟当前帧空间  8048d72: 8d 45 e0 lea -0x20(%ebp),%eax eax指向ebp-0x20  8048d75: 89 44 24 04 mov %eax,0x4(%esp)  8048d79: 8b 45 08 mov 0x8(%ebp),%eax 参数一传给%eax  8048d7c: 89 04 24 mov %eax,(%esp)  8048d7f: e8 87 03 00 00 call 804910b <read\_six\_numbers> 读取第六个数字  8048d84: 83 7d e0 00 cmpl $0x0,-0x20(%ebp) 比较0和第一个数是否相等  8048d88: 75 06 jne 8048d90 <phase\_2+0x26>  8048d8a: 83 7d e4 01 cmpl $0x1,-0x1c(%ebp) 比较1和第二个数  8048d8e: 74 05 je 8048d95 <phase\_2+0x2b>  8048d90: e8 3c 03 00 00 call 80490d1 <explode\_bomb>  8048d95: 8d 5d e8 lea -0x18(%ebp),%ebx ebx指向第三个数字  8048d98: 8d 75 f8 lea -0x8(%ebp),%esi esi指向第六个数字再向上移一位的地址  8048d9b: 8b 43 fc mov -0x4(%ebx),%eax ebx下面一位的数字赋给eax  8048d9e: 03 43 f8 add -0x8(%ebx),%eax ebx eax再加上ebx向下两位的数字8048da1: 39 03 cmp %eax,(%ebx) 比较ebx前两位的和与ebx指向的数字  8048da3: 74 05 je 8048daa <phase\_2+0x40>  8048da5: e8 27 03 00 00 call 80490d1 <explode\_bomb>  8048daa: 83 c3 04 add $0x4,%ebx ebx地址向上移动一位  8048dad: 39 f3 cmp %esi,%ebx 判断是否到第六位数字  8048daf: 75 ea jne 8048d9b <phase\_2+0x31> 未超过则跳转到8048d9b  8048db1: 83 c4 30 add $0x30,%esp  8048db4: 5b pop %ebx  8048db5: 5e pop %esi  8048db6: 5d pop %ebp  8048db7: c3 ret  从上面的代码分析中可以知道，phase\_2的密码有六个数字，第一个是0，第二个是1，然后之后的每位都是前两位的和，即斐波拉契数列的前6位。所以密码是:0 1 1 2 3 5。  **phase\_3代码：**  08048ea1 <phase\_3>:  8048ea1: 55 push %ebp  8048ea2: 89 e5 mov %esp,%ebp  8048ea4: 83 ec 28 sub $0x28,%esp  8048ea7: 8d 45 f0 lea -0x10(%ebp),%eax  8048eaa: 89 44 24 0c mov %eax,0xc(%esp) 此处为第二个数字  8048eae: 8d 45 f4 lea -0xc(%ebp),%eax  8048eb1: 89 44 24 08 mov %eax,0x8(%esp) 此处为第一个数字  8048eb5: c7 44 24 04 3e a2 04 movl $0x804a23e,0x4(%esp) 用x/s 0x804a23e命令查看$0x804a23e为  %d %d，即要输入两个整数，上面已经指出  8048ebc: 08  8048ebd: 8b 45 08 mov 0x8(%ebp),%eax  8048ec0: 89 04 24 mov %eax,(%esp)  8048ec3: e8 78 f9 ff ff call 8048840 <\_\_isoc99\_sscanf@plt>  8048ec8: 83 f8 01 cmp $0x1,%eax以上两行即要求输入至少两组数据，否则引爆  8048ecb: 7f 05 jg 8048ed2 <phase\_3+0x31>  8048ecd: e8 ff 01 00 00 call 80490d1 <explode\_bomb>  8048ed2: 83 7d f4 07 cmpl $0x7,-0xc(%ebp) 第一个数大于7引爆，即第一个数小于等于7  8048ed6: 77 6b ja 8048f43 <phase\_3+0xa2>  8048ed8: 8b 45 f4 mov -0xc(%ebp),%eax  8048edb: ff 24 85 a0 a1 04 08 jmp \*0x804a1a0(,%eax,4) 跳转至0x804a1a0+eax\*4(第一个数)内数  据所指的行数  8048ee2: b8 00 00 00 00 mov $0x0,%eax  8048ee7: eb 53 jmp 8048f3c <phase\_3+0x9b>  8048ee9: b8 00 00 00 00 mov $0x0,%eax  8048eee: 66 90 xchg %ax,%ax  8048ef0: eb 45 jmp 8048f37 <phase\_3+0x96>  8048ef2: b8 00 00 00 00 mov $0x0,%eax x=0  8048ef7: eb 39 jmp 8048f32 <phase\_3+0x91>  8048ef9: b8 00 00 00 00 mov $0x0,%eax x=0  8048efe: 66 90 xchg %ax,%ax  8048f00: eb 2b jmp 8048f2d <phase\_3+0x8c>  8048f02: b8 00 00 00 00 mov $0x0,%eax x=0  8048f07: eb 1f jmp 8048f28 <phase\_3+0x87>  8048f09: b8 00 00 00 00 mov $0x0,%eax x=0  8048f0e: 66 90 xchg %ax,%ax  8048f10: eb 11 jmp 8048f23 <phase\_3+0x82>  8048f12: b8 14 03 00 00 mov $0x314,%eax当第一个数为0时跳转到此处，第二个数x=788  8048f17: eb 05 jmp 8048f1e <phase\_3+0x7d>  8048f19: b8 00 00 00 00 mov $0x0,%eax x=0  8048f1e: 2d 5a 03 00 00 sub $0x35a,%eax x=x-858  8048f23: 05 ef 02 00 00 add $0x2ef,%eax x=x+751  8048f28: 2d 16 02 00 00 sub $0x216,%eax x=x-534  8048f2d: 05 16 02 00 00 add $0x216,%eax x=x+534  8048f32: 2d 16 02 00 00 sub $0x216,%eax x=x-534  8048f37: 05 16 02 00 00 add $0x216,%eax x=x+534  8048f3c: 2d 16 02 00 00 sub $0x216,%eax x=x-534  8048f41: eb 0a jmp 8048f4d <phase\_3+0xac>跳转到0x8048f4d行  8048f43: e8 89 01 00 00 call 80490d1 <explode\_bomb>  8048f48: b8 00 00 00 00 mov $0x0,%eax  8048f4d: 83 7d f4 05 cmpl $0x5,-0xc(%ebp) 第一个数大于5引爆，即第一个数小于等于5  8048f51: 7f 05 jg 8048f58 <phase\_3+0xb7>  8048f53: 3b 45 f0 cmp -0x10(%ebp),%eax第一个数是0时，算得x=147，即第二个数  8048f56: 74 05 je 8048f5d <phase\_3+0xbc>  8048f58: e8 74 01 00 00 call 80490d1 <explode\_bomb>  8048f5d: c9 leave  8048f5e: 66 90 xchg %ax,%ax  8048f60: c3 ret  从以上的代码分析中知道，第一个数要小于等于5，即可以取0、1、2、3、4、5。  则第一个数取值对应的情况是：  0：  跳转到0x8048f12。以上已经做过分析，得到第二个数为788-858+751-534=147。  1：  跳转到0x8048f19。x=0-858+751-534=-641。  2：  跳转到0x8048f09。x=0+751-534=217。  3：  跳转到0x8048f02。x=534。  4:  跳转到0x8048ef9。x=0。  5：  跳转到0x8048ef2。x=-534。  因此得到五组正确密码：0 147，1 -641，2 217，3 534，4 0，5 -534。  **phase\_4代码：**  08048e2e <phase\_4>:  8048e2e: 55 push %ebp  8048e2f: 89 e5 mov %esp,%ebp  8048e31: 83 ec 28 sub $0x28,%esp  8048e34: 8d 45 f0 lea -0x10(%ebp),%eax  8048e37: 89 44 24 0c mov %eax,0xc(%esp) 第二个数字  8048e3b: 8d 45 f4 lea -0xc(%ebp),%eax  8048e3e: 89 44 24 08 mov %eax,0x8(%esp) 第一个数字  8048e42: c7 44 24 04 3e a2 04 movl $0x804a23e,0x4(%esp) $0x804a23e内为%d %d，即输入两个整数  8048e49: 08  8048e4a: 8b 45 08 mov 0x8(%ebp),%eax  8048e4d: 89 04 24 mov %eax,(%esp)  8048e50: e8 eb f9 ff ff call 8048840 <\_\_isoc99\_sscanf@plt>  8048e55: 83 f8 02 cmp $0x2,%eax要求输入两个数据否则爆炸  8048e58: 75 0c jne 8048e66 <phase\_4+0x38>  8048e5a: 8b 45 f4 mov -0xc(%ebp),%eax  8048e5d: 85 c0 test %eax,%eax 第一个数字>=0，否则爆炸  8048e5f: 78 05 js 8048e66 <phase\_4+0x38>  8048e61: 83 f8 0e cmp $0xe,%eax 第一个数字<=14跳过爆炸  8048e64: 7e 05 jle 8048e6b <phase\_4+0x3d>  8048e66: e8 66 02 00 00 call 80490d1 <explode\_bomb>  8048e6b: c7 44 24 08 0e 00 00 movl $0xe,0x8(%esp)  8048e72: 00  8048e73: c7 44 24 04 00 00 00 movl $0x0,0x4(%esp)  8048e7a: 00  8048e7b: 8b 45 f4 mov -0xc(%ebp),%eax  8048e7e: 89 04 24 mov %eax,(%esp)  8048e81: e8 da fc ff ff call 8048b60 <func4> 调用递归函数func4确定第一个数的值  8048e86: 83 f8 01 cmp $0x1,%eax  8048e89: 75 06 jne 8048e91 <phase\_4+0x63>  8048e8b: 83 7d f0 01 cmpl $0x1,-0x10(%ebp) 第二个数等于1则跳过爆炸  8048e8f: 74 0c je 8048e9d <phase\_4+0x6f>  8048e91: 8d b4 26 00 00 00 00 lea 0x0(%esi,%eiz,1),%esi  8048e98: e8 34 02 00 00 call 80490d1 <explode\_bomb>  8048e9d: c9 leave  8048e9e: 66 90 xchg %ax,%ax  8048ea0: c3 ret  由以上对代码的分析可以看出，第二个数据确定为1。第一个数据范围为大于等于0，小于14，下面为粗糙简陋的fun4代码分析，得出x1=8,9,11。因此密码有三组：8 1，9 1,11 1。  int fun4(x1,x2,x3){  ecx=x3;  ecx=x3-x2;  esi=ecx>>31;  ecx=esi+ecx;  ecx>>1;  ecx=ecx+x2;  if(ecx<=x1){  if(ecx>=x1)  return 0;  else  return 1+2\*fun4(x1,ecx+1,x3);  }  else  return 2\*fun4(x1,x2,ecx-1);  }  **phase\_5代码：**  08048db8 <phase\_5>:  8048db8: 55 push %ebp  8048db9: 89 e5 mov %esp,%ebp  8048dbb: 56 push %esi  8048dbc: 53 push %ebx  8048dbd: 83 ec 20 sub $0x20,%esp  8048dc0: 8d 45 f0 lea -0x10(%ebp),%eax  8048dc3: 89 44 24 0c mov %eax,0xc(%esp) 此处为第二个数  8048dc7: 8d 45 f4 lea -0xc(%ebp),%eax  8048dca: 89 44 24 08 mov %eax,0x8(%esp) 此处为第一个数  8048dce: c7 44 24 04 3e a2 04 movl $0x804a23e,0x4(%esp) $0x804a23e内为%d %d，即输入两个整数  8048dd5: 08  8048dd6: 8b 45 08 mov 0x8(%ebp),%eax  8048dd9: 89 04 24 mov %eax,(%esp)  8048ddc: e8 5f fa ff ff call 8048840 <\_\_isoc99\_sscanf@plt>  8048de1: 83 f8 01 cmp $0x1,%eax以上两行保证输入数据多于1组，否则引爆  8048de4: 7f 05 jg 8048deb <phase\_5+0x33>  8048de6: e8 e6 02 00 00 call 80490d1 <explode\_bomb>  8048deb: 8b 45 f4 mov -0xc(%ebp),%eax  8048dee: 83 e0 0f and $0xf,%eax 按位与，即保留第一个数的后四位  8048df1: 89 45 f4 mov %eax,-0xc(%ebp)  8048df4: 83 f8 0f cmp $0xf,%eax 第一个数二进制后四位不能为(1111)2，否则引爆  8048df7: 74 29 je 8048e22 <phase\_5+0x6a>  8048df9: b9 00 00 00 00 mov $0x0,%ecx sum初始为0  8048dfe: ba 00 00 00 00 mov $0x0,%edx i初始为0  8048e03: bb c0 a1 04 08 mov $0x804a1c0,%ebx $0x804a1c0指向的是一个数组  8048e08: 83 c2 01 add $0x1,%edx i++  8048e0b: 8b 04 83 mov (%ebx,%eax,4),%eax a+4t,获取a[t]放在eax中，暂记为t  8048e0e: 01 c1 add %eax,%ecx sum+=t  8048e10: 83 f8 0f cmp $0xf,%eax 判断t是否等于15  8048e13: 75 f3 jne 8048e08 <phase\_5+0x50>以上五行为一个for循环语句  8048e15: 89 45 f4 mov %eax,-0xc(%ebp)  8048e18: 83 fa 0f cmp $0xf,%edx上述的for循环要循环15次  8048e1b: 75 05 jne 8048e22 <phase\_5+0x6a>  8048e1d: 39 4d f0 cmp %ecx,-0x10(%ebp) 确定第二个数的值为sum  8048e20: 74 05 je 8048e27 <phase\_5+0x6f>  8048e22: e8 aa 02 00 00 call 80490d1 <explode\_bomb>  8048e27: 83 c4 20 add $0x20,%esp  8048e2a: 5b pop %ebx  8048e2b: 5e pop %esi  8048e2c: 5d pop %ebp  8048e2d: c3 ret  根据上述的代码分析可以知道，第一个数二进制后四位不能为(1111)2。密码主要由上述的一个for循环确定。首先$0x804a1c0指向了一个数组，在for循环中每循环一次调用一次，因为必须循环15次，且最开始i=0还要调用一次，因此需要16个元素，使用 p \*0x804a1c0@16命令来查看这个数组为：  将这个for循环写出：  设i在%edx中，数组a[16]在%ebx中，t在%eax中，sum在%ecx中。  sum=0;  i=0;  for(i=1;t=15;i++)  {  t=a[t-1];  sum+=t;  }  t第一个值为用户输入，后面15个为循环中从a数组得到，根据t=a[t],共16个值. for循环当t为15时终止，因此最后一个t一定为15，根据15=a[6]知上一个t=6,以此类推得到t的16个值。并得到第一个值二进制后4位为5。如下表   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 序号 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | | a[i] | 10 | 2 | 14 | 7 | 8 | 12 | 15 | 11 | 0 | 4 | 1 | 13 | 3 | 9 | 6 | 5 | | t | 5 | 12 | 3 | 7 | 11 | 13 | 9 | 4 | 8 | 0 | 10 | 1 | 2 | 14 | 6 | 15 |   sum为除去第一个值之外所有t取值的和。  sum=115。即用户输入的第二个数为115。用户输入第一个数二进制下的后四位为0101(即第一个n=5)，  而之前位可以随意取值。因此有无数多的密码，如：5 115，21 115。  **phase\_6代码：**  08048c89 <phase\_6>:  8048c89: 55 push %ebp  8048c8a: 89 e5 mov %esp,%ebp  8048c8c: 57 push %edi  8048c8d: 56 push %esi  8048c8e: 53 push %ebx  8048c8f: 83 ec 5c sub $0x5c,%esp  8048c92: 8d 45 d0 lea -0x30(%ebp),%eax  8048c95: 89 44 24 04 mov %eax,0x4(%esp)  8048c99: 8b 45 08 mov 0x8(%ebp),%eax  8048c9c: 89 04 24 mov %eax,(%esp)  8048c9f: e8 67 04 00 00 call 804910b <read\_six\_numbers>  8048ca4: be 00 00 00 00 mov $0x0,%esi  8048ca9: 8d 7d d0 lea -0x30(%ebp),%edi  8048cac: 8b 04 b7 mov (%edi,%esi,4),%eax a[i]=>eax。开始外层for循环,i=0  8048caf: 83 e8 01 sub $0x1,%eax eax--  8048cb2: 83 f8 05 cmp $0x5,%eax eax小于等于5跳过炸弹，即a[i]<=6  8048cb5: 76 05 jbe 8048cbc <phase\_6+0x33>  8048cb7: e8 15 04 00 00 call 80490d1 <explode\_bomb>  8048cbc: 83 c6 01 add $0x1,%esi esi++  8048cbf: 83 fe 06 cmp $0x6,%esi esi等于6时跳转至0x8048ce6  8048cc2: 74 22 je 8048ce6 <phase\_6+0x5d>  8048cc4: 8d 1c b7 lea (%edi,%esi,4),%ebx &a[i]=> ebx  8048cc7: 89 75 b4 mov %esi,-0x4c(%ebp)  8048cca: 8b 44 b7 fc mov -0x4(%edi,%esi,4),%eax a[i-1]=>eax。开始内层for循环 j=i  8048cce: 3b 03 cmp (%ebx),%eax若eax和ebx不相等则跳过炸弹  8048cd0: 75 05 jne 8048cd7 <phase\_6+0x4e>  8048cd2: e8 fa 03 00 00 call 80490d1 <explode\_bomb>  8048cd7: 83 45 b4 01 addl $0x1,-0x4c(%ebp) 每次加1  8048cdb: 83 c3 04 add $0x4,%ebx ebx+=4  8048cde: 83 7d b4 05 cmpl $0x5,-0x4c(%ebp) 小于等于5时跳转，即能循环6-esi次  8048ce2: 7e e6 jle 8048cca <phase\_6+0x41>  8048ce4: eb c6 jmp 8048cac <phase\_6+0x23>  8048ce6: bb 00 00 00 00 mov $0x0,%ebx  8048ceb: 8d 7d d0 lea -0x30(%ebp),%edi  8048cee: eb 16 jmp 8048d06 <phase\_6+0x7d>跳转到第0x8048d06行  8048cf0: 8b 52 08 mov 0x8(%edx),%edx此处为两层for循环语句。通过用户输入的值找  8048cf3: 83 c0 01 add $0x1,%eax 到相应位置并排序好链表的值  8048cf6: 39 c8 cmp %ecx,%eax  8048cf8: 75 f6 jne 8048cf0 <phase\_6+0x67>  8048cfa: 89 54 b5 b8 mov %edx,-0x48(%ebp,%esi,4)  8048cfe: 83 c3 01 add $0x1,%ebx  8048d01: 83 fb 06 cmp $0x6,%ebx  8048d04: 74 16 je 8048d1c <phase\_6+0x93>  8048d06: 89 de mov %ebx,%esi  8048d08: 8b 0c 9f mov (%edi,%ebx,4),%ecx  8048d0b: ba c4 c0 04 08 mov $0x804c0c4,%edx  8048d10: b8 01 00 00 00 mov $0x1,%eax  8048d15: 83 f9 01 cmp $0x1,%ecx  8048d18: 7f d6 jg 8048cf0 <phase\_6+0x67>  8048d1a: eb de jmp 8048cfa <phase\_6+0x71>  8048d1c: 8b 5d b8 mov -0x48(%ebp),%ebx 关联链表使按排序好的顺序将链表每个  8048d1f: 8b 45 bc mov -0x44(%ebp),%eax 值的最后4个字节指向下一个数的地址（next）  8048d22: 89 43 08 mov %eax,0x8(%ebx)  8048d25: 8b 55 c0 mov -0x40(%ebp),%edx  8048d28: 89 50 08 mov %edx,0x8(%eax)  8048d2b: 8b 45 c4 mov -0x3c(%ebp),%eax  8048d2e: 89 42 08 mov %eax,0x8(%edx)  8048d31: 8b 55 c8 mov -0x38(%ebp),%edx  8048d34: 89 50 08 mov %edx,0x8(%eax)  8048d37: 8b 45 cc mov -0x34(%ebp),%eax  8048d3a: 89 42 08 mov %eax,0x8(%edx) 对排序好的链表的值检验是否排序正确  8048d3d: c7 40 08 00 00 00 00 movl $0x0,0x8(%eax)  8048d44: be 00 00 00 00 mov $0x0,%esi  8048d49: 8b 43 08 mov 0x8(%ebx),%eax  8048d4c: 8b 13 mov (%ebx),%edx  8048d4e: 3b 10 cmp (%eax),%edx  8048d50: 7d 05 jge 8048d57 <phase\_6+0xce>  8048d52: e8 7a 03 00 00 call 80490d1 <explode\_bomb>  8048d57: 8b 5b 08 mov 0x8(%ebx),%ebx  8048d5a: 83 c6 01 add $0x1,%esi  8048d5d: 83 fe 05 cmp $0x5,%esi  8048d60: 75 e7 jne 8048d49 <phase\_6+0xc0>  8048d62: 83 c4 5c add $0x5c,%esp  8048d65: 5b pop %ebx  8048d66: 5e pop %esi  8048d67: 5f pop %edi  8048d68: 5d pop %ebp  8048d69: c3 ret  由上述代码分析，代码大致分为4块。第一块两个for循环外循环得出输入的数小于等于6，内循环得出输入的数两两不能相同。第二块为根据输入的数组值找到相应位置并为链表值按从大到小排序。第三块把排序好的链表值关联起来 。第四块是检验链表的排序是否正确z  第二个嵌套for语句可以写成如下代码：  for (i = 0; i < 6; i++)  {  p = start;  for (j = 1; j < a[i]; j++)  p = p -> next;  pointers[i] = p;  }  也就是说根据用户输入的值来对链表的值进行排序，比如给出值 5 6 1 4 3 2，则将原来链表第  5个位置的值放到第1位，第6个位置的值放到第二位，依次类推。  而下面检测链表值时，要求链表值按从大到小排列。由上图可以看出，链表原本各个位置的值  按顺序排列为(十六进制)：1a7 6c 155 187 3bd 255。  而以上值按从大到小的顺序排列应该为：3bd 255 1a7 187 155 6c。  因此可以得到，用户需要输入的数为：5 6 1 4 3 2。  实验结果及分析：    运行bomb程序，上图可知，六组密码正确。  收获与体会：  做完第一反应就是终于可以交了这份实验了。做了好久啊。这次实验主要是学会看懂汇编语言从而找出拆炸弹的输入数字。第一关和最后一关难度相差真的很大。在第四关的fun4和第六关花了太多太多的时间。其实这些汇编还有很多值得考究的东西。最大的体会就是你以为你看懂了但是其实可能不是你想的那样。就拿简单的lea指令和mov指令，如果草草看过也能知道这个寄存器等于哪个值，但是这两个指令本身一个传送的是地址一个传送的是值。往往是这些细节的东西决定你学的扎实还是肤浅。彻底搞懂还是需要一定的时间。希望计组实验不要和小班课作业堆在一块，紧迫的时间大家交不出作业那么作业质量当然会下降 |