计算机系统课程设计
考查报告

姓　　名：_____
学　　号：_____
班　　级：_____
学　　院：计算机学院
题　　目：二进制逆向工程
指导 教师：墙威

2024 年 1 月

目录

一、 二进制逆向工程

1.1 循环（phsae2）

1.1.1 程序代码

```
080494ee <phase_2>:
 80494ee:       55                      push   %ebp
 80494ef:       89 e5                   mov    %esp,%ebp
 80494f1:       53                      push   %ebx
 80494f2:       83 ec 34                sub    $0x34,%esp
 80494f5:       83 ec 04                sub    $0x4,%esp
 80494f8:       6a 09                   push   $0x9
 80494fa:       8d 45 d0                lea    -0x30(%ebp),%eax
 80494fd:       50                      push   %eax
 80494fe:       ff 75 08                pushl  0x8(%ebp)
 8049501:       e8 a1 06 00 00          call   8049ba7 <read_n_numbers>
 8049506:       83 c4 10                add    $0x10,%esp
 8049509:       85 c0                   test   %eax,%eax
 804950b:       75 07                   jne    8049514 <phase_2+0x26>
 804950d:       b8 00 00 00 00          mov    $0x0,%eax
 8049512:       eb 60                   jmp    8049574 <phase_2+0x86>
 8049514:       8b 45 d0                mov    -0x30(%ebp),%eax
 8049517:       3d 86 00 00 00          cmp    $0x86,%eax
 804951c:       74 0c                   je     804952a <phase_2+0x3c>
 804951e:       e8 a6 09 00 00          call   8049ec9 <explode_bomb>
 8049523:       b8 00 00 00 00          mov    $0x0,%eax
 8049528:       eb 4a                   jmp    8049574 <phase_2+0x86>
 804952a:       c7 45 f4 01 00 00 00    movl   $0x1,-0xc(%ebp)
 8049531:       eb 36                   jmp    8049569 <phase_2+0x7b>
 8049533:       8b 45 f4                mov    -0xc(%ebp),%eax
 8049536:       8b 54 85 d0             mov    -0x30(%ebp,%eax,4),%edx
 804953a:       8b 45 f4                mov    -0xc(%ebp),%eax
 804953d:       83 e8 01                sub    $0x1,%eax
 8049540:       8b 4c 85 d0             mov    -0x30(%ebp,%eax,4),%ecx
 8049544:       8b 5d f4                mov    -0xc(%ebp),%ebx
 8049547:       b8 00 00 00 00          mov    $0x0,%eax
 804954c:       29 d8                   sub    %ebx,%eax
 804954e:       01 c0                   add    %eax,%eax
 8049550:       01 c8                   add    %ecx,%eax
 8049552:       83 c0 01                add    $0x1,%eax
 8049555:       39 c2                   cmp    %eax,%edx
 8049557:       74 0c                   je     8049565 <phase_2+0x77>
 8049559:       e8 6b 09 00 00          call   8049ec9 <explode_bomb>
 804955e:       b8 00 00 00 00          mov    $0x0,%eax
 8049563:       eb 0f                   jmp    8049574 <phase_2+0x86>
 8049565:       83 45 f4 01             addl   $0x1,-0xc(%ebp)
 8049569:       83 7d f4 08             cmpl   $0x8,-0xc(%ebp)
 804956d:       7e c4                   jle    8049533 <phase_2+0x45>
 804956f:       b8 01 00 00 00          mov    $0x1,%eax
 8049574:       8b 5d fc                mov    -0x4(%ebp),%ebx
 8049577:       c9                      leave
 8049578:       c3                      ret
```

1.1.2 解题思路

首先看到在 0x8049501 处 call 指令调用 read_n_numbers 函数，猜测到要输入几个数，往前看，给局部变量分配栈空间，然后 push 了一个 0x9，不出意外 9 就是我们要输入的数的个数。

接下来启动 gdb 调试，单步运行。

```
(gdb) b phase_2
Breakpoint 1 at 0x80494f2
(gdb) r
Starting program: /home/sanfenbai/Desktop/计算机系统/课程设计/拆炸弹/bomb
Welcome to my fiendish little bomb. You have 7 phases with
which to blow yourself up. Have a nice day!
A text line is a sequence of ASCII characters.
Well done! You seem to have warmed up!
1107296256 1101196553
Phase 1 defused. How about the next one?
1 2 3 4 5 6 7 8 9

Breakpoint 1, 0x080494f2 in phase_2 ()
(gdb) ni
0x080494f5 in phase_2 ()
(gdb)
```

先随便输入 9 个数，然后 ni。

```
0x0804951e in phase_2 ()
(gdb)

BOOM!!!
The bomb has blown up.
0x08049523 in phase_2 ()
```

发现在 0x804951e 处引爆炸弹，查看此处的汇编代码

```
8049517:       3d 86 00 00 00          cmp    $0x86,%eax
804951c:       74 0c                   je     804952a <phase_2+0x3c>
804951e:       e8 a6 09 00 00          call   8049ec9 <explode_bomb>
```

发现将寄存器 eax 与 0x86 cmp，那第一个数就是 0x86，即 134。继续调试，发现炸弹在 0x08049559 处爆炸

```
(gdb)
0x08049557 in phase_2 ()
(gdb)
0x08049559 in phase_2 ()
(gdb)

BOOM!!!
The bomb has blown up.
```

查看此处汇编代码

```
8049555:       39 c2                   cmp    %eax,%edx
8049557:       74 0c                   je     8049565 <phase_2+0x77>
8049559:       e8 6b 09 00 00          call   8049ec9 <explode_bomb>
```

发现将寄存器 eax 与寄存器 edx 的内容作比较，调试查看此处寄存器 eax 与寄存器 edx 中存放的内容

```
0x08049555 in phase_2 ()
(gdb) i r eax edx
eax            0x85     133
edx            0x2      2
```

显而易见，第二个数为 133，接下来的步骤就是一样的了，9 次循环，每次查看此处寄存器 eax 存放的值，最终得到的结果为：134 133 130 125 118 109 98 85 70。

1. 1. 3 实验结果

运行 bomb，输入这 9 个数，拆弹成功。

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/拆炸弹$ ./bomb
Welcome to my fiendish little bomb. You have 7 phases with
which to blow yourself up. Have a nice day!
A text line is a sequence of ASCII characters.
Well done! You seem to have warmed up!
1107296256 1101196553
Phase 1 defused. How about the next one?
134 133 130 125 118 109 98 85 70
That's number 2.  Keep going!
```

1.2 条件/分支（phase3）

1.2.1 程序代码

```
08049579 <phase_3>:
 8049579:       55                      push   %ebp
 804957a:       89 e5                   mov    %esp,%ebp
 804957c:       83 ec 18                sub    $0x18,%esp
 804957f:       c7 45 f4 00 00 00 00    movl   $0x0,-0xc(%ebp)
 8049586:       c7 45 f0 00 00 00 00    movl   $0x0,-0x10(%ebp)
 804958d:       8d 45 e8                lea    -0x18(%ebp),%eax
 8049590:       50                      push   %eax
 8049591:       8d 45 ec                lea    -0x14(%ebp),%eax
 8049594:       50                      push   %eax
 8049595:       68 0f a2 04 08          push   $0x804a20f
 804959a:       ff 75 08                pushl  0x8(%ebp)
 804959d:       e8 2e fb ff ff          call   80490d0 <__isoc99_sscanf@plt>
 80495a2:       83 c4 10                add    $0x10,%esp
 80495a5:       89 45 f0                mov    %eax,-0x10(%ebp)
 80495a8:       83 7d f0 01             cmpl   $0x1,-0x10(%ebp)
 80495ac:       7f 0f                   jg     80495bd <phase_3+0x44>
 80495ae:       e8 16 09 00 00          call   8049ec9 <explode_bomb>
 80495b3:       b8 00 00 00 00          mov    $0x0,%eax
 80495b8:       e9 95 00 00 00          jmp    8049652 <phase_3+0xd9>
 80495bd:       8b 45 ec                mov    -0x14(%ebp),%eax
 80495c0:       2d fe 00 00 00          sub    $0xfe,%eax
 80495c5:       83 f8 09                cmp    $0x9,%eax
 80495c8:       77 63                   ja     804962d <phase_3+0xb4>
 80495ca:       8b 04 85 18 a2 04 08    mov    0x804a218(,%eax,4),%eax
 80495d1:       ff e0                   jmp    *%eax
 80495d3:       c7 45 f4 bd 00 00 00    movl   $0xbd,-0xc(%ebp)
 80495da:       eb 5d                   jmp    8049639 <phase_3+0xc0>
 80495dc:       c7 45 f4 bd 00 00 00    movl   $0xbd,-0xc(%ebp)
 80495e3:       eb 54                   jmp    8049639 <phase_3+0xc0>
 80495e5:       c7 45 f4 e5 03 00 00    movl   $0x3e5,-0xc(%ebp)
 80495ec:       eb 4b                   jmp    8049639 <phase_3+0xc0>
 80495ee:       c7 45 f4 bd 00 00 00    movl   $0xbd,-0xc(%ebp)
 80495f5:       eb 42                   jmp    8049639 <phase_3+0xc0>
 80495f7:       c7 45 f4 e5 03 00 00    movl   $0x3e5,-0xc(%ebp)
 80495fe:       eb 39                   jmp    8049639 <phase_3+0xc0>
 8049600:       c7 45 f4 bd 00 00 00    movl   $0xbd,-0xc(%ebp)
 8049607:       eb 30                   jmp    8049639 <phase_3+0xc0>
 8049609:       c7 45 f4 e5 03 00 00    movl   $0x3e5,-0xc(%ebp)
 8049610:       eb 27                   jmp    8049639 <phase_3+0xc0>
 8049612:       c7 45 f4 e5 03 00 00    movl   $0x3e5,-0xc(%ebp)
 8049619:       eb 1e                   jmp    8049639 <phase_3+0xc0>
 804961b:       c7 45 f4 bd 00 00 00    movl   $0xbd,-0xc(%ebp)
 8049622:       eb 15                   jmp    8049639 <phase_3+0xc0>
 8049624:       c7 45 f4 e5 03 00 00    movl   $0x3e5,-0xc(%ebp)
 804962b:       eb 0c                   jmp    8049639 <phase_3+0xc0>
 804962d:       e8 97 08 00 00          call   8049ec9 <explode_bomb>
 8049632:       b8 00 00 00 00          mov    $0x0,%eax
 8049637:       eb 19                   jmp    8049652 <phase_3+0xd9>
 8049639:       8b 45 e8                mov    -0x18(%ebp),%eax
 804963c:       39 45 f4                cmp    %eax,-0xc(%ebp)
 804963f:       74 0c                   je     804964d <phase_3+0xd4>
 8049641:       e8 83 08 00 00          call   8049ec9 <explode_bomb>
 8049646:       b8 00 00 00 00          mov    $0x0,%eax
 804964b:       eb 05                   jmp    8049652 <phase_3+0xd9>
 804964d:       b8 01 00 00 00          mov    $0x1,%eax
 8049652:       c9                      leave
 8049653:       c3                      ret
```

1.2.2 解题思路

首先看到熟悉的 0x804a20f 以及 call 指令调用 sscanf 函数，那么这题要输入两个整数。

```
8049595:       68 0f a2 04 08          push   $0x804a20f
804959a:       ff 75 08                pushl  0x8(%ebp)
804959d:       e8 2e fb ff ff          call   80490d0 <__isoc99_sscanf@plt>
```

继续往下看，有一个 cmp 比较，若不相等则跳转到 0x804962d 处，引发炸弹爆炸

```
80495bd:       8b 45 ec                mov    -0x14(%ebp),%eax
80495c0:       2d fe 00 00 00          sub    $0xfe,%eax
80495c5:       83 f8 09                cmp    $0x9,%eax
80495c8:       77 63                   ja     804962d <phase_3+0xb4>
804962d:       e8 97 08 00 00          call   8049ec9 <explode_bomb>
```

分析可知，这里将寄存器 eax 里的值减去 0xfe，再减去 0x9 判断是否为零，那寄存器 eax 里存放的内容应为 0xfe+0x9=0x107，即 263，大胆猜测第一个数就是 263，启动 gdb 调试，输入 263，第二个数随便输

```
That's number 2.  Keep going!
263 2

Breakpoint 1, 0x0804957f in phase_3 ()
(gdb) ni
```

单步调试，发现炸弹在 0x08049641 处爆炸

```
0x08049641 in phase_3 ()
(gdb)

BOOM!!!
The bomb has blown up.
0x08049646 in phase_3 ()
```

查看此处汇编代码

```
8049622:       eb 15                   jmp    8049639 <phase_3+0xc0>
8049624:       c7 45 f4 e5 03 00 00    movl   $0x3e5,-0xc(%ebp)
804962b:       eb 0c                   jmp    8049639 <phase_3+0xc0>
804962d:       e8 97 08 00 00          call   8049ec9 <explode_bomb>
8049632:       b8 00 00 00 00          mov    $0x0,%eax
8049637:       eb 19                   jmp    8049652 <phase_3+0xd9>
8049639:       8b 45 e8                mov    -0x18(%ebp),%eax
804963c:       39 45 f4                cmp    %eax,-0xc(%ebp)
804963f:       74 0c                   je     804964d <phase_3+0xd4>
8049641:       e8 83 08 00 00          call   8049ec9 <explode_bomb>
```

调用 explode_bomb 前有一个 cmp 比较，将寄存器 eax 的值与-0xc(%ebp)处的值比较是否相等，若不相等则引爆炸弹，继续往前看，-0xc(%ebp)里存放的值为 0x3e5，即 997，大胆猜测第二个数为 997。

1.2.3 实验结果

运行 bomb，输入 263 997，拆除成功。

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/拆炸弹$ ./bomb 000
Welcome to my fiendish little bomb. You have 7 phases with
which to blow yourself up. Have a nice day!
Well done! You seem to have warmed up!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
263 997
Halfway there!
```

1.3 递归（phase4）

1.3.1 程序代码

```
080496c7 <phase_4>:
 80496c7:       55                      push   %ebp
 80496c8:       89 e5                   mov    %esp,%ebp
 80496ca:       57                      push   %edi
 80496cb:       56                      push   %esi
 80496cc:       53                      push   %ebx
 80496cd:       81 ec ac 00 00 00       sub    $0xac,%esp
 80496d3:       8d 85 54 ff ff ff       lea    -0xac(%ebp),%eax
 80496d9:       bb 40 a2 04 08          mov    $0x804a240,%ebx
 80496de:       ba 24 00 00 00          mov    $0x24,%edx
 80496e3:       89 c7                   mov    %eax,%edi
 80496e5:       89 de                   mov    %ebx,%esi
 80496e7:       89 d1                   mov    %edx,%ecx
 80496e9:       f3 a5                   rep movsl %ds:(%esi),%es:(%edi)
 80496eb:       8d 85 4c ff ff ff       lea    -0xb4(%ebp),%eax
 80496f1:       50                      push   %eax
 80496f2:       8d 85 50 ff ff ff       lea    -0xb0(%ebp),%eax
 80496f8:       50                      push   %eax
 80496f9:       68 0f a2 04 08          push   $0x804a20f
 80496fe:       ff 75 08                pushl  0x8(%ebp)
 8049701:       e8 ca f9 ff ff          call   80490d0 <__isoc99_sscanf@plt>
 8049706:       83 c4 10                add    $0x10,%esp
 8049709:       89 45 e4                mov    %eax,-0x1c(%ebp)
 804970c:       83 7d e4 02             cmpl   $0x2,-0x1c(%ebp)
 8049710:       74 0f                   je     8049721 <phase_4+0x5a>
 8049712:       e8 b2 07 00 00          call   8049ec9 <explode_bomb>
 8049717:       b8 00 00 00 00          mov    $0x0,%eax
 804971c:       e9 bc 00 00 00          jmp    80497dd <phase_4+0x116>
```

```
8049757:       8b 85 50 ff ff ff       mov    -0xb0(%ebp),%eax
804975d:       85 c0                   test   %eax,%eax
804975f:       7e 36                   jle    8049797 <phase_4+0xd0>
8049761:       8b 85 4c ff ff ff       mov    -0xb4(%ebp),%eax
8049767:       8b 95 50 ff ff ff       mov    -0xb0(%ebp),%edx
804976d:       83 ea 01                sub    $0x1,%edx
8049770:       83 ec 04                sub    $0x4,%esp
8049773:       50                      push   %eax
8049774:       52                      push   %edx
8049775:       8d 85 54 ff ff ff       lea    -0xac(%ebp),%eax
804977b:       50                      push   %eax
804977c:       e8 d3 fe ff ff          call   8049654 <func4>
8049781:       83 c4 10                add    $0x10,%esp
8049784:       3d c8 01 00 00          cmp    $0x1c8,%eax
8049789:       75 0c                   jne    8049797 <phase_4+0xd0>
804978b:       e8 39 07 00 00          call   8049ec9 <explode_bomb>
8049790:       b8 00 00 00 00          mov    $0x0,%eax
8049795:       eb 46                   jmp    80497dd <phase_4+0x116>
8049797:       8b 85 4c ff ff ff       mov    -0xb4(%ebp),%eax
804979d:       83 f8 22                cmp    $0x22,%eax
80497a0:       7f 36                   jg     80497d8 <phase_4+0x111>
80497a2:       8b 85 4c ff ff ff       mov    -0xb4(%ebp),%eax
80497a8:       8d 50 01                lea    0x1(%eax),%edx
80497ab:       8b 85 50 ff ff ff       mov    -0xb0(%ebp),%eax
80497b1:       83 ec 04                sub    $0x4,%esp
80497b4:       52                      push   %edx
80497b5:       50                      push   %eax
80497b6:       8d 85 54 ff ff ff       lea    -0xac(%ebp),%eax
80497bc:       50                      push   %eax
80497bd:       e8 92 fe ff ff          call   8049654 <func4>
80497c2:       83 c4 10                add    $0x10,%esp
80497c5:       3d c8 01 00 00          cmp    $0x1c8,%eax
80497ca:       75 0c                   jne    80497d8 <phase_4+0x111>
80497cc:       e8 f8 06 00 00          call   8049ec9 <explode_bomb>
80497d1:       b8 00 00 00 00          mov    $0x0,%eax
80497d6:       eb 05                   jmp    80497dd <phase_4+0x116>
80497d8:       b8 01 00 00 00          mov    $0x1,%eax
80497dd:       8d 65 f4                lea    -0xc(%ebp),%esp
80497e0:       5b                      pop    %ebx
80497e1:       5e                      pop    %esi
80497e2:       5f                      pop    %edi
80497e3:       5d                      pop    %ebp
80497e4:       c3                      ret
```

还调用了 func4 函数：

```
08049654 <func4>:
 8049654:       55                      push   %ebp
 8049655:       89 e5                   mov    %esp,%ebp
 8049657:       83 ec 18                sub    $0x18,%esp
 804965a:       8b 55 0c                mov    0xc(%ebp),%edx
 804965d:       8b 45 10                mov    0x10(%ebp),%eax
 8049660:       01 d0                   add    %edx,%eax
 8049662:       89 c2                   mov    %eax,%edx
 8049664:       c1 ea 1f                shr    $0x1f,%edx
 8049667:       01 d0                   add    %edx,%eax
 8049669:       d1 f8                   sar    %eax
 804966b:       89 45 f4                mov    %eax,-0xc(%ebp)
 804966e:       8b 45 0c                mov    0xc(%ebp),%eax
 8049671:       3b 45 10                cmp    0x10(%ebp),%eax
 8049674:       7c 13                   jl     8049689 <func4+0x35>
 8049676:       8b 45 10                mov    0x10(%ebp),%eax
 8049679:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
 8049680:       8b 45 08                mov    0x8(%ebp),%eax
 8049683:       01 d0                   add    %edx,%eax
 8049685:       8b 00                   mov    (%eax),%eax
 8049687:       eb 3c                   jmp    80496c5 <func4+0x71>
 8049689:       83 ec 04                sub    $0x4,%esp
 804968c:       ff 75 f4                pushl  -0xc(%ebp)
 804968f:       ff 75 0c                pushl  0xc(%ebp)
 8049692:       ff 75 08                pushl  0x8(%ebp)
 8049695:       e8 ba ff ff ff          call   8049654 <func4>
 804969a:       83 c4 10                add    $0x10,%esp
 804969d:       89 45 f0                mov    %eax,-0x10(%ebp)
 80496a0:       8b 45 f4                mov    -0xc(%ebp),%eax
 80496a3:       83 c0 01                add    $0x1,%eax
 80496a6:       83 ec 04                sub    $0x4,%esp
 80496a9:       ff 75 10                pushl  0x10(%ebp)
 80496ac:       50                      push   %eax
 80496ad:       ff 75 08                pushl  0x8(%ebp)
 80496b0:       e8 9f ff ff ff          call   8049654 <func4>
 80496b5:       83 c4 10                add    $0x10,%esp
 80496b8:       89 45 ec                mov    %eax,-0x14(%ebp)
 80496bb:       8b 45 ec                mov    -0x14(%ebp),%eax
 80496be:       39 45 f0                cmp    %eax,-0x10(%ebp)
 80496c1:       0f 4d 45 f0             cmovge -0x10(%ebp),%eax
 80496c5:       c9                      leave
 80496c6:       c3                      ret
```

1.3.2 解题思路

首先看到熟悉的 0x804a20f 和 sscanf 函数，那么本题也要输入两个数。

看开头，分配了栈空间，然后在栈上缓冲区内存放了一个数组，数组的起始地址为 0x804a240，长度为 0x24，即 36。

```
80496d3:       8d 85 54 ff ff ff       lea    -0xac(%ebp),%eax
80496d9:       bb 40 a2 04 08          mov    $0x804a240,%ebx
80496de:       ba 24 00 00 00          mov    $0x24,%edx
80496e3:       89 c7                   mov    %eax,%edi
80496e5:       89 de                   mov    %ebx,%esi
80496e7:       89 d1                   mov    %edx,%ecx
80496e9:       f3 a5                   rep movsl %ds:(%esi),%es:(%edi)
```

启动 gdb 调试查看数组的值

```
Halfway there!
1 2

Breakpoint 1, 0x080496cd in phase_4 ()
(gdb) ni
0x080496d3 in phase_4 ()
(gdb)
0x080496d9 in phase_4 ()
(gdb)
0x080496de in phase_4 ()
(gdb) x/36dw 0x804a240
0x804a240:      498     387     204     456
0x804a250:      96      300     141     231
0x804a260:      125     7       409     269
0x804a270:      158     37      490     508
0x804a280:      13      337     401     502
0x804a290:      270     183     407     48
0x804a2a0:      474     375     224     193
0x804a2b0:      208     93      282     95
0x804a2c0:      323     452     9       170
```

继续往下看，在调用 func4 函数的地方都有 eax 与 0x1c8 进行 cmp，并且若不相等则引爆炸弹，说明 0x1c8 很关键。

```
8049738:       50                      push   %eax
8049739:       e8 16 ff ff ff          call   8049654 <func4>
804973e:       83 c4 10                add    $0x10,%esp
8049741:       3d c8 01 00 00          cmp    $0x1c8,%eax
8049746:       74 0f                   je     8049757 <phase_4+0x90>
8049748:       e8 7c 07 00 00          call   8049ec9 <explode_bomb>

804977b:       50                      push   %eax
804977c:       e8 d3 fe ff ff          call   8049654 <func4>
8049781:       83 c4 10                add    $0x10,%esp
8049784:       3d c8 01 00 00          cmp    $0x1c8,%eax
8049789:       75 0c                   jne    8049797 <phase_4+0xd0>
804978b:       e8 39 07 00 00          call   8049ec9 <explode_bomb>
```

0x1c8，即 456，发现为数组中第四个元素。查看 func4 函数的汇编代码，分析可知，func4 函数的作用是比较两次递归调用的结果，取较大者，然后函数返回。那么可以猜想 func4 函数递归比较数组中的一个数与 456 的大小，最后返回最大值 456，这样才不会引发炸弹爆炸，那么这个数组元素里的最大值得是 456，进而这个用于比较的数组就是来自存放于栈缓冲区中，以 456 为最大值的一个区间，即 387 204 456 96 300 141 231 125 7 409 269 158 37。输入的两个数即为这个数组在原数组中所处位置的左右下标，即 1 和 13。

1.3.3 实验结果

运行 bomb，输入 1 13，拆除成功。

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/拆炸弹$ ./bomb 000
Welcome to my fiendish little bomb. You have 7 phases with
which to blow yourself up. Have a nice day!
Well done! You seem to have warmed up!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
1 13
So you got that one.  Try this one.
```

二、 链接与 ELF

2.1 符号解析（phase3）

2.1.1 程序代码

首先对程序进行链接并执行，查看结果：

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接$ gcc -no-pie -o lb3 main.o phase3.o
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接$ ./lb3

sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接$
```

默认输出为空白。

反汇编 phase3.o，查看 do_phase 汇编代码：

```
phase3.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <do_phase>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 18                sub    $0x18,%esp
   6:   c7 45 ea 79 7a 67 69    movl   $0x69677a79,-0x16(%ebp)
   d:   c7 45 ee 75 68 6e 62    movl   $0x626e6875,-0x12(%ebp)
  14:   66 c7 45 f2 65 00       movw   $0x65,-0xe(%ebp)
  1a:   c7 45 f4 00 00 00 00    movl   $0x0,-0xc(%ebp)
  21:   eb 28                   jmp    4b <do_phase+0x4b>
  23:   8d 55 ea                lea    -0x16(%ebp),%edx
  26:   8b 45 f4                mov    -0xc(%ebp),%eax
  29:   01 d0                   add    %edx,%eax
  2b:   0f b6 00                movzbl (%eax),%eax
  2e:   0f b6 c0                movzbl %al,%eax
  31:   0f b6 80 00 00 00 00    movzbl 0x0(%eax),%eax
  38:   0f be c0                movsbl %al,%eax
  3b:   83 ec 0c                sub    $0xc,%esp
  3e:   50                      push   %eax
  3f:   e8 fc ff ff ff          call   40 <do_phase+0x40>
  44:   83 c4 10                add    $0x10,%esp
  47:   83 45 f4 01             addl   $0x1,-0xc(%ebp)
  4b:   8b 45 f4                mov    -0xc(%ebp),%eax
  4e:   83 f8 08                cmp    $0x8,%eax
  51:   76 d0                   jbe    23 <do_phase+0x23>
  53:   83 ec 0c                sub    $0xc,%esp
  56:   6a 0a                   push   $0xa
  58:   e8 fc ff ff ff          call   59 <do_phase+0x59>
  5d:   83 c4 10                add    $0x10,%esp
  60:   90                      nop
  61:   c9                      leave
  62:   c3                      ret
```

这个函数先是把 9 个字节的信息放入了栈里，然后进入一个循环次数为 9 的循环。对于刚刚那 9 个字节的信息，每次循环会从中取出 1 个字节的信息，第 x 次循环就是取第 x 个字节的信息。然后将这个第 x 个字节的信息作为一个 char

数组的索引，得到一个 char 字符，并调用 putchar 函数将其打印出来。

2.1.2 求解思路

使用 gdb 调试查看 char 数组的内容：

```
(gdb) b *0x804849e
Breakpoint 1 at 0x804849e
(gdb) r
Starting program: /home/sanfenbai/Desktop/计算机系统/课程设计/链接/lb3

Breakpoint 1, 0x0804849e in do_phase ()
(gdb) i r ebp
ebp            0xbfffeeb8       0xbfffeeb8
(gdb) x/9bx 0xbfffeeaf
0xbfffeeaf:     0x00    0x01    0x00    0x00    0x00    0x74    0xef    0xff
0xbfffeeb7:     0xbf
(gdb) x/9bx 0xbfffeea2
0xbfffeea2:     0x79    0x7a    0x67    0x69    0x75    0x68    0x6e    0x62
0xbfffeeaa:     0x65
```

%ebp 的值是 0xbfffeeb8，所以 9 个字节的信息的起始位置是%ebp-0x16=0xbfffeea2，用 x 命令打印如上。

这个 char 数组的首地址是 0x804a060。用 gdb 查看一下这个数组：

```
(gdb) x/72xw 0x804a060
0x804a060 <NQqPQyqUth>: 0x00000000      0x00000000      0x00000000      0x00000000
0x804a070 <NQqPQyqUth+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a080 <NQqPQyqUth+32>:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a090 <NQqPQyqUth+48>:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0a0 <NQqPQyqUth+64>:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0b0 <NQqPQyqUth+80>:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0c0 <NQqPQyqUth+96>:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0d0 <NQqPQyqUth+112>:     0x00000000      0x00000000      0x00000000      0x00000000
0x804a0e0 <NQqPQyqUth+128>:     0x00000000      0x00000000      0x00000000      0x00000000
0x804a0f0 <NQqPQyqUth+144>:     0x00000000      0x00000000      0x00000000      0x00000000
0x804a100 <NQqPQyqUth+160>:     0x00000000      0x00000000      0x00000000      0x00000000
0x804a110 <NQqPQyqUth+176>:     0x00000000      0x00000000      0x00000000      0x00000000
0x804a120 <NQqPQyqUth+192>:     0x00000000      0x00000000      0x00000000      0x00000000
0x804a130 <NQqPQyqUth+208>:     0x00000000      0x00000000      0x00000000      0x00000000
0x804a140 <NQqPQyqUth+224>:     0x00000000      0x00000000      0x00000000      0x00000000
0x804a150 <NQqPQyqUth+240>:     0x00000000      0x00000000      0x00000000      0x00000000
0x804a160:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a170:      0x00000000      0x00000000      0x00000000      0x00000000
```

这个数组大小是 256（0x100），里面的元素都是 0。所以打印出来的东西是一片空白。

分析可知，需要一个赋有初始值的强符号 NQqPQyqUth，使得 phase3.o 中的弱符号 NQqPQyqUth 能够引用这个强符号。这样就能使在执行 do_phase 函数时根据索引值打印字符数组 NQqPQyqUth 中的某些元素，所有打印出来的内容连起来就是我们的学号。

新建一个 phase3_patch.c 文件，在其中填入以下内容：

```
1. char NQqPQyqUth[256] = "1";
```

然后编译生成.o 文件，查看这个文件的节头表和符号表：

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ readelf -s phase3_patch.o

Symbol table '.symtab' contains 8 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS phase3_patch.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    2
     4: 00000000     0 SECTION LOCAL  DEFAULT    3
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000000     0 SECTION LOCAL  DEFAULT    4
     7: 00000000   256 OBJECT  GLOBAL DEFAULT    2 NQqPQyqUth
```

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ readelf -S phase3_patch.o
共有 9 个节头，从偏移量 0x258 开始：

节头：
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000000 00  AX  0   0  1
  [ 2] .data             PROGBITS        00000000 000040 000100 00  WA  0   0 32
  [ 3] .bss              NOBITS          00000000 000140 000000 00  WA  0   0  1
  [ 4] .comment          PROGBITS        00000000 000140 000036 01  MS  0   0  1
  [ 5] .note.GNU-stack   PROGBITS        00000000 000176 000000 00      0   0  1
  [ 6] .shstrtab         STRTAB          00000000 000213 000045 00      0   0  1
  [ 7] .symtab           SYMTAB          00000000 000178 000080 10      8   7  4
  [ 8] .strtab           STRTAB          00000000 0001f8 00001b 00      0   0  1
```

可知，.data 节在 phase3_patch.o 中的偏移量为 0x40，符号 NQqPQyqUth 在.data 节中的偏移量为 0x0。所以符号 NQqPQyqUth 在 phase3_patch.o 中的偏移量为 0x40+0x0=0x40。又因为符号 NQqPQyqUth 的大小为 256（0x100），所以它的内容在位置 0x40~0x13f 上。

根据 do_phase 函数的功能，想要把学号打印出来，那么就需要把数组{0x79,0x7a,0x67,0x69,0x75,0x68,0x6e,0x62,0x65}中的每一个元素加上 0x40 之后对应的位置上的值依次改为我们的学号：

```
00000000   7F 45 4C 46  01 01 01 00  00 00 00 00  00 00 00 00  .ELF............
00000010   01 00 03 00  01 00 00 00  00 00 00 00  00 00 00 00  ................
00000020   58 02 00 00  00 00 00 00  34 00 00 00  00 00 28 00  X.......4.....(.
00000030   09 00 06 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000040   31 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  1...............
00000050   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000060   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000070   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000080   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000090   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000A0   00 00 33 00  00 36 00 32  30 31 00 00  00 00 30 00  ..3..6.201....0.
000000B0   00 00 00 00  00 31 00 00  00 32 30 00  00 00 00 00  .....1...20.....
000000C0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000D0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000E0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000F0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000100   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000110   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000120   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000130   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000140   00 47 43 43  3A 20 28 55  62 75 6E 74  75 20 35 2E  .GCC: (Ubuntu 5.
00000150   34 2E 30 2D  36 75 62 75  6E 74 75 31  7E 31 36 2E  4.0-6ubuntu1~16.
00000160   30 34 2E 31  31 29 20 35  2E 34 2E 30  20 32 30 31  04.11) 5.4.0 201
```

2.1.3 实验结果

重新链接并执行，成功输出学号：

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ hexedit phase3_patch.o
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ gcc -no-pie -o lb3 mai
n.o phase3.o phase3_patch.o
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ ./lb3
202110036
```

2.2 switch 语句与重定位（phase4）

2.2.1 程序代码

首先对程序进行链接并执行，查看结果：

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ gcc -no-pie -o lb4 mai
n.o phase4.o
```

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ objdump -d lb4 > lb4.s

sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ ./lb4
B9[wL:Nec
```

默认输出为 B9[wL:Nec。查看 do_phase 汇编代码：

```
08048474 <do_phase>:
 8048474:       55                      push   %ebp
 8048475:       89 e5                   mov    %esp,%ebp
 8048477:       83 ec 28                sub    $0x28,%esp
 804847a:       c7 45 e6 53 4e 58 47    movl   $0x47584e53,-0x1a(%ebp)
 8048481:       c7 45 ea 4a 54 43 46    movl   $0x4643544a,-0x16(%ebp)
 8048488:       66 c7 45 ee 50 00       movw   $0x50,-0x12(%ebp)
 804848e:       c7 45 f0 00 00 00 00    movl   $0x0,-0x10(%ebp)
 8048495:       e9 e0 00 00 00          jmp    804857a <do_phase+0x106>
 804849a:       8d 55 e6                lea    -0x1a(%ebp),%edx
 804849d:       8b 45 f0                mov    -0x10(%ebp),%eax
 80484a0:       01 d0                   add    %edx,%eax
 80484a2:       0f b6 00                movzbl (%eax),%eax
 80484a5:       88 45 f7                mov    %al,-0x9(%ebp)
 80484a8:       0f be 45 f7             movsbl -0x9(%ebp),%eax
 80484ac:       83 e8 41                sub    $0x41,%eax
 80484af:       83 f8 19                cmp    $0x19,%eax
 80484b2:       0f 87 b0 00 00 00       ja     8048568 <do_phase+0xf4>
 80484b8:       8b 04 85 c8 87 04 08    mov    0x80487c8(,%eax,4),%eax
 80484bf:       ff e0                   jmp    *%eax
 80484c1:       c6 45 f7 38             movb   $0x38,-0x9(%ebp)
 80484c5:       e9 9e 00 00 00          jmp    8048568 <do_phase+0xf4>
 80484ca:       c6 45 f7 65             movb   $0x65,-0x9(%ebp)
```

中间省略 20 多个跳转…

```
 804855b:       eb 0b                   jmp    8048568 <do_phase+0xf4>
 804855d:       c6 45 f7 34             movb   $0x34,-0x9(%ebp)
 8048561:       eb 05                   jmp    8048568 <do_phase+0xf4>
 8048563:       c6 45 f7 67             movb   $0x67,-0x9(%ebp)
 8048567:       90                      nop
 8048568:       8d 55 dc                lea    -0x24(%ebp),%edx
 804856b:       8b 45 f0                mov    -0x10(%ebp),%eax
 804856e:       01 c2                   add    %eax,%edx
 8048570:       0f b6 45 f7             movzbl -0x9(%ebp),%eax
 8048574:       88 02                   mov    %al,(%edx)
 8048576:       83 45 f0 01             addl   $0x1,-0x10(%ebp)
 804857a:       8b 45 f0                mov    -0x10(%ebp),%eax
 804857d:       83 f8 08                cmp    $0x8,%eax
 8048580:       0f 86 14 ff ff ff       jbe    804849a <do_phase+0x26>
 8048586:       8d 55 dc                lea    -0x24(%ebp),%edx
 8048589:       8b 45 f0                mov    -0x10(%ebp),%eax
 804858c:       01 d0                   add    %edx,%eax
 804858e:       c6 00 00                movb   $0x0,(%eax)
 8048591:       83 ec 0c                sub    $0xc,%esp
 8048594:       8d 45 dc                lea    -0x24(%ebp),%eax
 8048597:       50                      push   %eax
 8048598:       e8 43 fd ff ff          call   80482e0 <puts@plt>
 804859d:       83 c4 10                add    $0x10,%esp
 80485a0:       90                      nop
 80485a1:       c9                      leave
 80485a2:       c3                      ret
 80485a3:       66 90                   xchg   %ax,%ax
 80485a5:       66 90                   xchg   %ax,%ax
 80485a7:       66 90                   xchg   %ax,%ax
 80485a9:       66 90                   xchg   %ax,%ax
 80485ab:       66 90                   xchg   %ax,%ax
 80485ad:       66 90                   xchg   %ax,%ax
 80485af:       90                      nop
```

2.2.2 求解思路

phase4 的 do_phase 函数和 phase3 的 do_phase 函数有相同之处，不同之处 phase4 还有一个 char 数组，首先将第一个 char 数组 cookie 打印出来：

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/sanfenbai/Desktop/计算机系统/课程设计/链接与ELF/lb4

Breakpoint 1, 0x0804849a in do_phase ()
(gdb) i r ebp
ebp            0xbfffeea8       0xbfffeea8
(gdb) x/9bx 0xbfffee8e
0xbfffee8e:     0x53    0x4e    0x58    0x47    0x4a    0x54    0x43    0x46
0xbfffee96:     0x50
```

%ebp 的值是 0xbfffeea8，所以 9 个字节的信息的起始位置是%ebp-0x1a=0xbfffee8e，另一个 char 数组 output 的首地址为%ebp-0x24。

和 switch 语句相关的汇编代码中，%eax 存放的是 cookie[i]的值，它先减去 0x41，然后和 0x19 比较。如果它大于 0x19，那么就执行 default 语句的内容，否则就跳转到对应的表项执行相应的 case 语句。

```
80484ac:      83 e8 41                sub    $0x41,%eax
80484af:      83 f8 19                cmp    $0x19,%eax
80484b2:      0f 87 b0 00 00 00       ja     8048568 <do_phase+0xf4>
80484b8:      8b 04 85 c8 87 04 08    mov    0x80487c8(,%eax,4),%eax
80484bf:      ff e0                   jmp    *%eax
80484c1:      c6 45 f7 38             movb   $0x38,-0x9(%ebp)
80484c5:      e9 9e 00 00 00          jmp    8048568 <do_phase+0xf4>
80484ca:      c6 45 f7 65             movb   $0x65,-0x9(%ebp)
80484ce:      e9 95 00 00 00          jmp    8048568 <do_phase+0xf4>
80484d3:      c6 45 f7 35             movb   $0x35,-0x9(%ebp)
80484d7:      e9 8c 00 00 00          jmp    8048568 <do_phase+0xf4>
80484dc:      c6 45 f7 39             movb   $0x39,-0x9(%ebp)
80484e0:      e9 83 00 00 00          jmp    8048568 <do_phase+0xf4>
80484e5:      c6 45 f7 6d             movb   $0x6d,-0x9(%ebp)
80484e9:      eb 7d                   jmp    8048568 <do_phase+0xf4>
80484eb:      c6 45 f7 6d             movb   $0x6d,-0x9(%ebp)
80484ef:      eb 77                   jmp    8048568 <do_phase+0xf4>
80484f1:      c6 45 f7 4c             movb   $0x4c,-0x9(%ebp)
```

查看跳转表的内容：

```
(gdb) x/26wx 0x80487c8
0x80487c8:      0x08048539      0x080484e5      0x0804854b      0x0804852d
0x80487d8:      0x080484fd      0x080484ca      0x0804853f      0x080484c1
0x80487e8:      0x08048563      0x080484f1      0x08048503      0x080484f7
0x80487f8:      0x08048551      0x080484dc      0x080484eb      0x08048527
0x8048808:      0x08048515      0x08048557      0x08048509      0x0804851b
0x8048818:      0x0804855d      0x08048533      0x08048545      0x08048521
0x8048828:      0x0804850f      0x080484d3
```

分析可知，函数的执行过程为：当第 7 次循环时，取 cookie[6]=0x4c，那么就执行 0x4c-0x41=0xb，即 case11 这条语句，此时跳转的地址为 cookie 数组首地址 0x80487c8+4*0xb=0x80487f4，查看跳转表，对应的值为 0x080484f7。

switch 语句的跳转表是存放在.rodata 节（只读数据节）的，因此查看节头表，发现.rodata 节的偏移量是 0x1d8。

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ readelf -S phase4.o
共有 15 个节头，从偏移量 0x548 开始：

节头：
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000158 00  AX  0   0  1
  [ 2] .rel.text         REL             00000000 0003e0 000010 08   I 12   1  4
  [ 3] .data             PROGBITS        00000000 0001a0 000038 00  WA  0   0 32
  [ 4] .rel.data         REL             00000000 0003f0 000010 08   I 12   3  4
  [ 5] .bss              NOBITS          00000000 0001d8 000000 00  WA  0   0  1
  [ 6] .rodata           PROGBITS        00000000 0001d8 00006c 00   A  0   0  4
  [ 7] .rel.rodata       REL             00000000 000400 0000d0 08   I 12   6  4
  [ 8] .comment          PROGBITS        00000000 000244 00001d 01  MS  0   0  1
  [ 9] .note.GNU-stack   PROGBITS        00000000 000261 000000 00      0   0  1
  [10] .eh_frame         PROGBITS        00000000 000264 000058 00   A  0   0  4
  [11] .rel.eh_frame     REL             00000000 0004d0 000010 08   I 12  10  4
  [12] .symtab           SYMTAB          00000000 0002bc 0000f0 10     13  11  4
  [13] .strtab           STRTAB          00000000 0003ac 000033 00      0   0  1
  [14] .shstrtab         STRTAB          00000000 0004e0 000067 00      0   0  1
```

查看 phase4.o 的反汇编代码，从偏移量为 0x6d 的那一行可知跳转表相对于.rodata 节的偏移量为 0x4：

```
00000029 <do_phase>:
  29:   55                      push   %ebp
  2a:   89 e5                   mov    %esp,%ebp
  2c:   83 ec 28                sub    $0x28,%esp
  2f:   c7 45 e6 53 4e 58 47    movl   $0x47584e53,-0x1a(%ebp)
  36:   c7 45 ea 4a 54 43 46    movl   $0x4643544a,-0x16(%ebp)
  3d:   66 c7 45 ee 50 00       movw   $0x50,-0x12(%ebp)
  43:   c7 45 f0 00 00 00 00    movl   $0x0,-0x10(%ebp)
  4a:   e9 e0 00 00 00          jmp    12f <do_phase+0x106>
  4f:   8d 55 e6                lea    -0x1a(%ebp),%edx
  52:   8b 45 f0                mov    -0x10(%ebp),%eax
  55:   01 d0                   add    %edx,%eax
  57:   0f b6 00                movzbl (%eax),%eax
  5a:   88 45 f7                mov    %al,-0x9(%ebp)
  5d:   0f be 45 f7             movsbl -0x9(%ebp),%eax
  61:   83 e8 41                sub    $0x41,%eax
  64:   83 f8 19                cmp    $0x19,%eax
  67:   0f 87 b0 00 00 00       ja     11d <do_phase+0xf4>
  6d:   8b 04 85 04 00 00 00    mov    0x4(,%eax,4),%eax
```

因此，跳转表的起始位置为 0x1d8+0x4=0x1dc。

接下来针对 cookie 数组{0x53,0x4e,0x58,0x47,0x4a,0x54,0x43,0x46,0x50}的每一个元素，追踪到跳转表对应的表项，进行修改，使得最终打印出来的是我们的学号。

跳转的地址为：{0x224, 0x210, 0x238, 0x1f4, 0x200, 0x228, 0x1e4, 0x1f01, 0x218}，

对应的值为：{BE, 91, D6, F4, A6, D0, 00, 7F, DC}，

找到它们在 phase4.s 中对应的指令，使用 hexedit 修改跳转表：

```
00000198   00 00 00 00  00 00 00 00  EE 0C C4 49  B2 3C 41 DA  E9 50 B3 FC  17 C8 B6 70
000001B0   C0 18 F4 7C  AC 6B 27 82  52 9A 03 16  09 DC E0 3E  BC 80 6A 5A  D8 6E 74 FD
000001C8   29 37 81 8D  FC 40 00 00  00 00 00 00  00 00 00 00  34 00 00 00  EE 00 00 00
000001E0   9A 00 00 00  06 01 00 00  E2 00 00 00  B2 00 00 00  E8 00 00 00  0C 00 00 00
000001F8   76 00 00 00  18 01 00 00  0C 00 00 00  B8 00 00 00  AC 00 00 00  06 01 00 00
00000210   06 00 00 00  A0 00 00 00  CA 00 00 00  CA 00 00 00  0C 01 00 00  FA 00 00 00
00000228   06 00 00 00  12 01 00 00  E8 00 00 00  FA 00 00 00  FA 00 00 00  C4 00 00 00
00000240   88 00 00 00  00 47 43 43  3A 20 28 44  65 62 69 61  6E 20 38 2E  33 2E 30 2D
```

2.2.3 实验结果

重新链接并执行，输出段错误，不知道什么情况。

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ gcc -no-pie -o lb4 mai
n.o phase4.o
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/链接与ELF$ ./lb4
段错误（核心已转储）
```