天津大学

计算机系统基础上机实验报告

实验题目 3：拆弹专家 bomb

学院名称 智能与计算学部

专　　业 计算机科学与技术（拔尖班）

学生姓名 牛天溳

学　　号 3024244288

年　　级 2024 级

班　　级 拔尖 1 班

时　　间 2024 年 5 月 18 日

实验 3：拆弹专家

Bomb

1. 实验目的

进一步掌握程序的机器级表示一章的知识。理解程序控制、过程调用的汇编级实现，熟练掌握 汇编语言程序的阅读。

2. 实验内容

程序 bomb 是一个电子炸弹，当该程序运行时，需要按照一定的顺序输入口令，才能阻 止炸弹的引爆。当输入错误的密码时，炸弹将会引爆。此时控制台将会产生如下输出，并结束 程序

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

在炸弹程序中，你需要输入多组口令，且每一组口令都正确才能够防止引爆。

目前已知的内容只有炸弹程序的二进制可执行文件 bomb（目标平台为：x86-64）和 bomb 的 main 函数框架代码，见main.c。其他的细节均不会以 c 语言的方式呈现。 你的任务是：利用现有的资源以及相关的工具，猜出炸弹的全部口令，并输入至炸弹程序中，以完成最终的拆弹工作。

3. 实验要求

1）在 Unbuntu18.04LTS 操作系统下，按照实验指导说明书，使用 gdb 和 objdump 等工具，以反向工程方式完成 Bomb 拆弹。

2）需提交：拆弹口令文本文件、电子版实验报告全文。

4. 实验结果

进入实验过程中，我先通过反汇编工具 objdump 并利用命令 objdump -d

bomb > bomb.s 获得了 bomb 的汇编代码。

在汇编代码中找到 phase_1 的部分：

```
0000000000400ee0 <phase_1>:
  400ee0: 48 83 ec 08           sub    $0x8,%rsp
  400ee4: be 00 24 40 00        mov    $0x402400,%esi
  400ee9: e8 4a 04 00 00        call   401338 <strings_not_equal>
  400eee: 85 c0                 test   %eax,%eax
  400ef0: 74 05                 je     400ef7 <phase_1+0x17>
  400ef2: e8 43 05 00 00        call   40143a <explode_bomb>
  400ef7: 48 83 c4 08           add    $0x8,%rsp
  400efb: c3                    ret
```

阅读这一段汇编代码，我们可以知道这是一个比较字符串的函数，具体是要和地址为 0x402400 的字符串进行比较，我们用 gdb 工具获取地址为 0x402400 的字符串是什么：

```
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
```

得到所要的字符串为“Border relations with Canada have never been better.”。测试一下：

```
root@ToddyN:~/lab3# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
```

通过！

接下来进行 phase_2。

phase_2 的汇编代码如图所示：

```
0000000000400efc <phase_2>:
  400efc: 55                    push   %rbp
  400efd: 53                    push   %rbx
  400efe: 48 83 ec 28           sub    $0x28,%rsp
  400f02: 48 89 e6              mov    %rsp,%rsi
  400f05: e8 52 05 00 00        call   40145c <read_six_numbers>
  400f0a: 83 3c 24 01           cmpl   $0x1,(%rsp)
  400f0e: 74 20                 je     400f30 <phase_2+0x34>
  400f10: e8 25 05 00 00        call   40143a <explode_bomb>
  400f15: eb 19                 jmp    400f30 <phase_2+0x34>
  400f17: 8b 43 fc              mov    -0x4(%rbx),%eax
  400f1a: 01 c0                 add    %eax,%eax
  400f1c: 39 03                 cmp    %eax,(%rbx)
  400f1e: 74 05                 je     400f25 <phase_2+0x29>
  400f20: e8 15 05 00 00        call   40143a <explode_bomb>
  400f25: 48 83 c3 04           add    $0x4,%rbx
  400f29: 48 39 eb              cmp    %rbp,%rbx
  400f2c: 75 e9                 jne    400f17 <phase_2+0x1b>
  400f2e: eb 0c                 jmp    400f3c <phase_2+0x40>
  400f30: 48 8d 5c 24 04        lea    0x4(%rsp),%rbx
  400f35: 48 8d 6c 24 18        lea    0x18(%rsp),%rbp
  400f3a: eb db                 jmp    400f17 <phase_2+0x1b>
  400f3c: 48 83 c4 28           add    $0x28,%rsp
  400f40: 5b                    pop    %rbx
  400f41: 5d                    pop    %rbp
  400f42: c3                    ret
```

尽力去理解代码，看到了代码中有调用 read_six_numbers 函数的步骤，即要输入 6 个数字。

我们看到代码中有这样的片段：

```
  400f0a: 83 3c 24 01           cmpl   $0x1,(%rsp)
  400f0e: 74 20                 je     400f30 <phase_2+0x34>
  400f10: e8 25 05 00 00        call   40143a <explode_bomb>
```

大致意思是，将 rsp 寄存器中导入的数与 1 进行比较，如果相等，则跳过，反之则引爆炸弹。

我们又注意到代码

```
  400f17: 8b 43 fc              mov    -0x4(%rbx),%eax
  400f1a: 01 c0                 add    %eax,%eax
  400f1c: 39 03                 cmp    %eax,(%rbx)
  400f1e: 74 05                 je     400f25 <phase_2+0x29>
  400f20: e8 15 05 00 00        call   40143a <explode_bomb>
```

大致意思是取前一个数，将其乘 2，比较当前数和前一个数的两倍，如果相等，就跳过炸弹，否则就引爆炸弹。

我们由此可知，这 6 个数是一个以 1 为首项，公比为 2 的等比数列，即“1，2，4，8，16，32”

在 bomb 程序中输入“1，2，4，8，16，32”，结果如下：

```
1 2 4 8 16 32
That's number 2.  Keep going!
```

通过！

接下来进行 phase_3，汇编代码如图所示：

```
0000000000400f43 <phase_3>:
  400f43: 48 83 ec 18           sub    $0x18,%rsp
  400f47: 48 8d 4c 24 0c        lea    0xc(%rsp),%rcx
  400f4c: 48 8d 54 24 08        lea    0x8(%rsp),%rdx
  400f51: be cf 25 40 00        mov    $0x4025cf,%esi
  400f56: b8 00 00 00 00        mov    $0x0,%eax
  400f5b: e8 90 fc ff ff        call   400bf0 <__isoc99_sscanf@plt>
  400f60: 83 f8 01              cmp    $0x1,%eax
  400f63: 7f 05                 jg     400f6a <phase_3+0x27>
  400f65: e8 d0 04 00 00        call   40143a <explode_bomb>
  400f6a: 83 7c 24 08 07        cmpl   $0x7,0x8(%rsp)
  400f6f: 77 3c                 ja     400fad <phase_3+0x6a>
  400f71: 8b 44 24 08           mov    0x8(%rsp),%eax
  400f75: ff 24 c5 70 24 40 00  jmp    *0x402470(,%rax,8)
  400f7c: b8 cf 00 00 00        mov    $0xcf,%eax
  400f81: eb 3b                 jmp    400fbe <phase_3+0x7b>
  400f83: b8 c3 02 00 00        mov    $0x2c3,%eax
  400f88: eb 34                 jmp    400fbe <phase_3+0x7b>
  400f8a: b8 00 01 00 00        mov    $0x100,%eax
  400f8f: eb 2d                 jmp    400fbe <phase_3+0x7b>
  400f91: b8 85 01 00 00        mov    $0x185,%eax
  400f96: eb 26                 jmp    400fbe <phase_3+0x7b>
  400f98: b8 ce 00 00 00        mov    $0xce,%eax
  400f9d: eb 1f                 jmp    400fbe <phase_3+0x7b>
  400f9f: b8 aa 02 00 00        mov    $0x2aa,%eax
  400fa4: eb 18                 jmp    400fbe <phase_3+0x7b>
  400fa6: b8 47 01 00 00        mov    $0x147,%eax
  400fab: eb 11                 jmp    400fbe <phase_3+0x7b>
  400fad: e8 88 04 00 00        call   40143a <explode_bomb>
  400fb2: b8 00 00 00 00        mov    $0x0,%eax
  400fb7: eb 05                 jmp    400fbe <phase_3+0x7b>
  400fb9: b8 37 01 00 00        mov    $0x137,%eax
  400fbe: 3b 44 24 0c           cmp    0xc(%rsp),%eax
  400fc2: 74 05                 je     400fc9 <phase_3+0x86>
  400fc4: e8 71 04 00 00        call   40143a <explode_bomb>
  400fc9: 48 83 c4 18           add    $0x18,%rsp
  400fcd: c3                    ret
```

相当长的一段代码。

首先进行了输入操作。注意到出现了$0x4025cf，用 gbd 查看一下地址为 0x4025cf 的部分是什么。

```
(gdb) x/s 0x4025cf
0x4025cf:       "%d %d"
```

所以要输入两个数。

传入的两个参数分别在 0x8(%rsp)和 0xc(%rsp)。

```
400f6a: 83 7c 24 08 07        cmpl   $0x7,0x8(%rsp)
400f6f: 77 3c                 ja     400fad <phase_3+0x6a>
```

将第一个参数与 7 比较，如果比 7 大，直接跳转到 0x400fad，即引爆炸弹。所以传入的第一个数不可比 7 大。

接下来是一个 switch 操作。首先跳转到*(0x402470 + eax*8)的地方。0x402470 显然是一个数组的地址，我们来用 gdb 工具看一下这个数组里有什么。

```
(gdb) p /x *0x402470@16
$1 = {0x400f7c, 0x0, 0x400fb9, 0x0, 0x400f83, 0x0, 0x400f8a, 0x0, 0x400f91, 0x0, 0x400f98, 0x0, 0x400f9f, 0x0,
  0x400fa6, 0x0}
```

这是一个 switch 跳转表。我令第一个参数为 0，则跳转到 0x400f7c。而在原汇编代码中，0x400f7c 处，eax 是 0xcf，转化为十进制是 207。所以传入的两个参数可以是 0 和 207。

试一下：

```
0 207
Halfway there!
```

通过！

接下来是 phase_4，原汇编代码如下：

```
000000000040100c <phase_4>:
  40100c: 48 83 ec 18           sub    $0x18,%rsp
  401010: 48 8d 4c 24 0c        lea    0xc(%rsp),%rcx
  401015: 48 8d 54 24 08        lea    0x8(%rsp),%rdx
  40101a: be cf 25 40 00        mov    $0x4025cf,%esi
  40101f: b8 00 00 00 00        mov    $0x0,%eax
  401024: e8 c7 fb ff ff        call   400bf0 <__isoc99_sscanf@plt>
  401029: 83 f8 02              cmp    $0x2,%eax
  40102c: 75 07                 jne    401035 <phase_4+0x29>
  40102e: 83 7c 24 08 0e        cmpl   $0xe,0x8(%rsp)
  401033: 76 05                 jbe    40103a <phase_4+0x2e>
  401035: e8 00 04 00 00        call   40143a <explode_bomb>
  40103a: ba 0e 00 00 00        mov    $0xe,%edx
  40103f: be 00 00 00 00        mov    $0x0,%esi
  401044: 8b 7c 24 08           mov    0x8(%rsp),%edi
  401048: e8 81 ff ff ff        call   400fce <func4>
  40104d: 85 c0                 test   %eax,%eax
  40104f: 75 07                 jne    401058 <phase_4+0x4c>
  401051: 83 7c 24 0c 00        cmpl   $0x0,0xc(%rsp)
  401056: 74 05                 je     40105d <phase_4+0x51>
  401058: e8 dd 03 00 00        call   40143a <explode_bomb>
  40105d: 48 83 c4 18           add    $0x18,%rsp
  401061: c3                    ret
```

前面的部分和 phase_3 的基本一样。传入两个参数，第一个参数放入 0x8(%rsp)，第二个参数放入 0xc(%rsp)。

将第一个参数与 0xe（14）比较，如果小于等于就跳转，不然就引爆炸弹。

接着，将 edx 赋值为 14，esi 赋值为 0。然后再将第一个输入的参数存入寄存器，调用 func4(edi,0,14)函数。接着检查 func4(edi,0,14)是否为 0。如果

不为 0，直接引爆炸弹。再检查第二个参数是否为 0。如果不为 0，也会引爆炸弹。

接下来，我们来看一下 func4 函数的代码：

```
0000000000400fce <func4>:
  400fce: 48 83 ec 08           sub    $0x8,%rsp
  400fd2: 89 d0                 mov    %edx,%eax
  400fd4: 29 f0                 sub    %esi,%eax
  400fd6: 89 c1                 mov    %eax,%ecx
  400fd8: c1 e9 1f              shr    $0x1f,%ecx
  400fdb: 01 c8                 add    %ecx,%eax
  400fdd: d1 f8                 sar    %eax
  400fdf: 8d 0c 30              lea    (%rax,%rsi,1),%ecx
  400fe2: 39 f9                 cmp    %edi,%ecx
  400fe4: 7e 0c                 jle    400ff2 <func4+0x24>
  400fe6: 8d 51 ff              lea    -0x1(%rcx),%edx
  400fe9: e8 e0 ff ff ff        call   400fce <func4>
  400fee: 01 c0                 add    %eax,%eax
  400ff0: eb 15                 jmp    401007 <func4+0x39>
  400ff2: b8 00 00 00 00        mov    $0x0,%eax
  400ff7: 39 f9                 cmp    %edi,%ecx
  400ff9: 7d 0c                 jge    401007 <func4+0x39>
  400ffb: 8d 71 01              lea    0x1(%rcx),%esi
  400ffe: e8 cb ff ff ff        call   400fce <func4>
  401003: 8d 44 00 01           lea    0x1(%rax,%rax,1),%eax
  401007: 48 83 c4 08           add    $0x8,%rsp
  40100b: c3                    ret
```

首先第一行的作用是调用函数栈空间。接着

```
400fce: 48 83 ec 08           sub    $0x8,%rsp
400fd2: 89 d0                 mov    %edx,%eax
400fd4: 29 f0                 sub    %esi,%eax
400fd6: 89 c1                 mov    %eax,%ecx
400fd8: c1 e9 1f              shr    $0x1f,%ecx
400fdb: 01 c8                 add    %ecx,%eax
400fdd: d1 f8                 sar    %eax
```

将 edx 的值调入到 eax 中，再将 eax 的值减去 esi 的值，即起到了 eax=edx-esi 的作用。再将 eax 的值调入 ecx 中，并右移 0x1f(31)位，取出符号位。接下来，将符号位与 eax 相加存入 eax。最后 eax 除以 2，即 eax=(edx-esi)/2。这一系列操作，是为了实现中位数下取整，避免负数除法的问题。

```
400fdf: 8d 0c 30              lea    (%rax,%rsi,1),%ecx
400fe2: 39 f9                 cmp    %edi,%ecx
400fe4: 7e 0c                 jle    400ff2 <func4+0x24>
```

接着，令 ecx = rax + rsi = (edx-esi)/2，ecx = (edx+esi)/2，比较其和 edi。如果 edi<ecx，则会跳转到 0x400ff2。否则：

```
400fe6: 8d 51 ff              lea    -0x1(%rcx),%edx
400fe9: e8 e0 ff ff ff        call   400fce <func4>
400fee: 01 c0                 add    %eax,%eax
```

即递归调用函数 func4（edi，esi，(edx+esi)/2 -1），并返回 2*func4。

如果 edi>ecx：

```
400ff2: b8 00 00 00 00        mov    $0x0,%eax
400ff7: 39 f9                 cmp    %edi,%ecx
400ff9: 7d 0c                 jge    401007 <func4+0x39>
```

先将 0 放入 eax，默认返回 0，接着判断 edi 和(edx+esi)/2 是否相等。如果相等，直接返回 0。

如果 edi>(edx+esi)/2：

```
400ffb: 8d 71 01              lea    0x1(%rcx),%esi
400ffe: e8 cb ff ff ff        call   400fce <func4>
401003: 8d 44 00 01           lea    0x1(%rax,%rax,1),%eax
```

和小于的情况一样，递归调用函数 func4(edi，(edx+esi)/2+1，edx)，并返回 2*func4+1。

这个函数实际上就实现了二叉搜索树中的路径编码，它的结果就是从根到目标节点的路径，左走 = 0，右走 = 1，组成的二进制路径值。

所以我们只需要找出使得 func4 返回 0 的值，即(0+14)/2=7，也就是第一个参数应该为 7。

当然，经过检验，第一个参数为 3，1，0 均可以满足。

因此，最终的答案为“7 0”，“3 0”，“1 0”，“0 0”。

挑一个测试一下：

```
7 0
So you got that one.  Try this one.
```

通过！

其他三组数据我也测试过了，均能通过，就不一一展示了。

接下来是 phase_5，原汇编代码如下：

```
0000000000401062 <phase_5>:
  401062: 53                    push   %rbx
  401063: 48 83 ec 20           sub    $0x20,%rsp
  401067: 48 89 fb              mov    %rdi,%rbx
  40106a: 64 48 8b 04 25 28 00  mov    %fs:0x28,%rax
  401071: 00 00
  401073: 48 89 44 24 18        mov    %rax,0x18(%rsp)
  401078: 31 c0                 xor    %eax,%eax
  40107a: e8 9c 02 00 00        call   40131b <string_length>
  40107f: 83 f8 06              cmp    $0x6,%eax
  401082: 74 4e                 je     4010d2 <phase_5+0x70>
  401084: e8 b1 03 00 00        call   40143a <explode_bomb>
  401089: eb 47                 jmp    4010d2 <phase_5+0x70>
  40108b: 0f b6 0c 03           movzbl (%rbx,%rax,1),%ecx
  40108f: 88 0c 24              mov    %cl,(%rsp)
  401092: 48 8b 14 24           mov    (%rsp),%rdx
  401096: 83 e2 0f              and    $0xf,%edx
  401099: 0f b6 92 b0 24 40 00  movzbl 0x4024b0(%rdx),%edx
  4010a0: 88 54 04 10           mov    %dl,0x10(%rsp,%rax,1)
  4010a4: 48 83 c0 01           add    $0x1,%rax
  4010a8: 48 83 f8 06           cmp    $0x6,%rax
  4010ac: 75 dd                 jne    40108b <phase_5+0x29>
  4010ae: c6 44 24 16 00        movb   $0x0,0x16(%rsp)
  4010b3: be 5e 24 40 00        mov    $0x40245e,%esi
  4010b8: 48 8d 7c 24 10        lea    0x10(%rsp),%rdi
  4010bd: e8 76 02 00 00        call   401338 <strings_not_equal>
  4010c2: 85 c0                 test   %eax,%eax
  4010c4: 74 13                 je     4010d9 <phase_5+0x77>
  4010c6: e8 6f 03 00 00        call   40143a <explode_bomb>
  4010cb: 0f 1f 44 00 00        nopl   0x0(%rax,%rax,1)
  4010d0: eb 07                 jmp    4010d9 <phase_5+0x77>
  4010d2: b8 00 00 00 00        mov    $0x0,%eax
  4010d7: eb b2                 jmp    40108b <phase_5+0x29>
  4010d9: 48 8b 44 24 18        mov    0x18(%rsp),%rax
  4010de: 64 48 33 04 25 28 00  xor    %fs:0x28,%rax
  4010e5: 00 00
  4010e7: 74 05                 je     4010ee <phase_5+0x8c>
  4010e9: e8 42 fa ff ff        call   400b30 <__stack_chk_fail@plt>
  4010ee: 48 83 c4 20           add    $0x20,%rsp
  4010f2: 5b                    pop    %rbx
  4010f3: c3                    ret
```

又超级长……

一开始还是最基础的栈指针的调用。

```
  40107a: e8 9c 02 00 00        call   40131b <string_length>
  40107f: 83 f8 06              cmp    $0x6,%eax
  401082: 74 4e                 je     4010d2 <phase_5+0x70>
```

这一段代码可以看出，要求传入一段字符串，如果长度不为 6，则会引爆炸弹。所以输入的字符串的长度一定要为 6。

```
40108f: 88 0c 24              mov    %cl,(%rsp)
401092: 48 8b 14 24           mov    (%rsp),%rdx
401096: 83 e2 0f              and    $0xf,%edx
401099: 0f b6 92 b0 24 40 00  movzbl 0x4024b0(%rdx),%edx
4010a0: 88 54 04 10           mov    %dl,0x10(%rsp,%rax,1)
4010a4: 48 83 c0 01           add    $0x1,%rax
```

接下来，将取到的第一个字节与 0xf 进行与运算。因为在 c 语言中，字符类型通常占据 1 字节，所以就是将字符串的第一个字符的 ASCII 值与 0xf 进行与运算存储到 edx 寄存器中。然后将 0x4024b0+rdx=edx，接着 10+rsp+rax*1=dl，然后再进行下一个字节的操作。最后判断与 0x40245e 地址的字符串是否相等，相等即是正解。

我们用 gdb 工具看一下 0x40245e 地址的字符串是什么。

```
(gdb) x/s 0x40245e
0x40245e:       "flyers"
```

字符串是“flyers”。

我们接着来看一下 0x4024b0 地址的数组（即映射表）是什么。

```
(gdb) x/16cb 0x4024b0
0x4024b0 <array.3449>:  109 'm' 97 'a'  100 'd' 117 'u' 105 'i' 101 'e' 114 'r' 115 's'
0x4024b8 <array.3449+8>:        110 'n' 102 'f' 111 'o' 116 't' 118 'v' 98 'b'  121 'y' 108 'l'
```

通过映射表来找字符串“flyers”各字符在数组中的索引，分别是“9，15，14，5，6，7”。

我们接下来可以我选满足&0xf 条件的字符。

那我们只需取 ASCII 值分别为“0x29，0x2f，0x2e，0x25，0x26，0x27”的字符即可。（因为第二位是 1 的数所对应的符号均为控制符，无法显示）

通过查 ASCII 表，所得的字符串为“)/.%&’”。

测试一下：

```
)/.%&'
Good work!  On to the next...
```

通过！

最后是 phase_6，原汇编代码过长，就不完整展示了。

```
4010f4: 41 56                push   %r14
4010f6: 41 55                push   %r13
4010f8: 41 54                push   %r12
4010fa: 55                   push   %rbp
4010fb: 53                   push   %rbx
4010fc: 48 83 ec 50          sub    $0x50,%rsp
401100: 49 89 e5             mov    %rsp,%r13
401103: 48 89 e6             mov    %rsp,%rsi
401106: e8 51 03 00 00       call   40145c <read_six_numbers>
40110b: 49 89 e6             mov    %rsp,%r14
40110e: 41 bc 00 00 00 00    mov    $0x0,%r12d
401114: 4c 89 ed             mov    %r13,%rbp
401117: 41 8b 45 00          mov    0x0(%r13),%eax
40111b: 83 e8 01             sub    $0x1,%eax
40111e: 83 f8 05             cmp    $0x5,%eax
401121: 76 05                jbe    401128 <phase_6+0x34>
401123: e8 12 03 00 00       call   40143a <explode_bomb>
401128: 41 83 c4 01          add    $0x1,%r12d
40112c: 41 83 fc 06          cmp    $0x6,%r12d
401130: 74 21                je     401153 <phase_6+0x5f>
401132: 44 89 e3             mov    %r12d,%ebx
401135: 48 63 c3             movslq %ebx,%rax
401138: 8b 04 84             mov    (%rsp,%rax,4),%eax
40113b: 39 45 00             cmp    %eax,0x0(%rbp)
40113e: 75 05                jne    401145 <phase_6+0x51>
401140: e8 f5 02 00 00       call   40143a <explode_bomb>
401145: 83 c3 01             add    $0x1,%ebx
401148: 83 fb 05             cmp    $0x5,%ebx
40114b: 7e e8                jle    401135 <phase_6+0x41>
40114d: 49 83 c5 04          add    $0x4,%r13
401151: eb c1                jmp    401114 <phase_6+0x20>
```

代码的第一步是进行大小检测。因为有 sub $0x1,%eax 这一行代码，所以应该检查第一个参数减 1 后是否大于 5，也就是大于 6 就会引爆炸弹。并进行一个很长的循环检测，确保输入的 6 个数没有重复。

接着，开始了数据处理环节：

```
401153: 48 8d 74 24 18       lea    0x18(%rsp),%rsi
401158: 4c 89 f0             mov    %r14,%rax
40115b: b9 07 00 00 00       mov    $0x7,%ecx
401160: 89 ca                mov    %ecx,%edx
401162: 2b 10                sub    (%rax),%edx
401164: 89 10                mov    %edx,(%rax)
401166: 48 83 c0 04          add    $0x4,%rax
40116a: 48 39 f0             cmp    %rsi,%rax
40116d: 75 f1                jne    401160 <phase_6+0x6c>
40116f: be 00 00 00 00       mov    $0x0,%esi
401174: eb 21                jmp    401197 <phase_6+0xa3>
```

整段代码起到的效果就是让每一个数都被 7 减了。

接下来，要根据索引构造新链表。

```
401176: 48 8b 52 08           mov    0x8(%rdx),%rdx
40117a: 83 c0 01              add    $0x1,%eax
40117d: 39 c8                 cmp    %ecx,%eax
40117f: 75 f5                 jne    401176 <phase_6+0x82>
401181: eb 05                 jmp    401188 <phase_6+0x94>
401183: ba d0 32 60 00        mov    $0x6032d0,%edx
401188: 48 89 54 74 20        mov    %rdx,0x20(%rsp,%rsi,2)
40118d: 48 83 c6 04           add    $0x4,%rsi
401191: 48 83 fe 18           cmp    $0x18,%rsi
401195: 74 14                 je     4011ab <phase_6+0xb7>
401197: 8b 0c 34              mov    (%rsp,%rsi,1),%ecx
40119a: 83 f9 01              cmp    $0x1,%ecx
40119d: 7e e4                 jle    401183 <phase_6+0x8f>
40119f: b8 01 00 00 00        mov    $0x1,%eax
4011a4: ba d0 32 60 00        mov    $0x6032d0,%edx
4011a9: eb cb                 jmp    401176 <phase_6+0x82>
4011ab: 48 8b 5c 24 20        mov    0x20(%rsp),%rbx
4011b0: 48 8d 44 24 28        lea    0x28(%rsp),%rax
4011b5: 48 8d 74 24 50        lea    0x50(%rsp),%rsi
4011ba: 48 89 d9              mov    %rbx,%rcx
4011bd: 48 8b 10              mov    (%rax),%rdx
4011c0: 48 89 51 08           mov    %rdx,0x8(%rcx)
4011c4: 48 83 c0 08           add    $0x8,%rax
4011c8: 48 39 f0              cmp    %rsi,%rax
4011cb: 74 05                 je     4011d2 <phase_6+0xde>
4011cd: 48 89 d1              mov    %rdx,%rcx
4011d0: eb eb                 jmp    4011bd <phase_6+0xc9>
4011d2: 48 c7 42 08 00 00 00  movq   $0x0,0x8(%rdx)
```

通过 gdb 工具，可以知道地址为 0x6032d0 的链表

```
(gdb) x /48x 0x6032d0
0x6032d0 <node1>:       0x0000014c      0x00000001      0x006032e0      0x00000000
0x6032e0 <node2>:       0x000000a8      0x00000002      0x006032f0      0x00000000
0x6032f0 <node3>:       0x0000039c      0x00000003      0x00603300      0x00000000
0x603300 <node4>:       0x000002b3      0x00000004      0x00603310      0x00000000
0x603310 <node5>:       0x000001dd      0x00000005      0x00603320      0x00000000
0x603320 <node6>:       0x000001bb      0x00000006      0x00000000      0x00000000
0x603330:       0x00000000      0x00000000      0x00000000      0x00000000
0x603340 <host_table>:  0x00402629      0x00000000      0x00402643      0x00000000
0x603350 <host_table+16>:       0x0040265d      0x00000000      0x00000000      0x00000000
0x603360 <host_table+32>:       0x00000000      0x00000000      0x00000000      0x00000000
0x603370 <host_table+48>:       0x00000000      0x00000000      0x00000000      0x00000000
0x603380 <host_table+64>:       0x00000000      0x00000000      0x00000000      0x00000000
```

这个过程是把输入映射后的索引依次找到对应的链表节点，然后存在栈上的一个数组中。每次循环的时候按照一个逻辑取数，然后走链表。

```
4011e3: 8b 00                 mov    (%rax),%eax
4011e5: 39 03                 cmp    %eax,(%rbx)
4011e7: 7d 05                 jge    4011ee <phase_6+0xfa>
4011e9: e8 4c 02 00 00        call   40143a <explode_bomb>
```

最后，检查一下节点值是否按照降序排列。

最终的答案是：“4 3 2 1 6 5”。

测试一下：

```
4 3 2 1 6 5
Congratulations! You've defused the bomb!
```

通过！

5. 实验总结及心得体会

这个实验既充满挑战，又十分有趣。从最初的陌生与摸索，到逐步深入地理解和掌握，我在整个实验过程中不断提升了自己对汇编语言的编程能力，也更加深刻地认识了 C 和 C++ 语言在底层运行机制中的细节与原理。这种从理论到实践的过程，不仅加深了我对课堂知识的理解，也让我真正体会到了知识应用所带来的成就感。

在实验的推进中，我逐渐学会了如何使用各种调试和开发工具，提升了动手能力和问题解决能力。同时，也锻炼了我沉下心来思考和反复推敲问题的耐心与韧性。我相信，这些技能和品质不仅有助于我今后的学习，更将在将来从事科研工作时发挥重要作用。