目　录

一、　拆弹密码第一关

二、　拆弹密码第二关

三、　拆弹密码第二关

四、　拆弹密码第四关

五、　拆弹密码第五关

六、　拆弹密码第六关

七、　拆弹密码隐藏关

一、拆弹密码第一关

1、任务描述

本关任务：在 phase_1 的汇编代码中找到第一关拆弹密码。

2、编程要求

根据提示，在汇编代码中找到本关密码，在 c 文件中将密码输出。

3、实验步骤

（1）进入 bomb 文件，在命令行利用反汇编得到 bomb.s，并且定位到 phase_1；

（2）分析汇编代码

```
0000000000400ee0 <phase_1>:
  400ee0: 48 83 ec 08          sub    $0x8,%rsp       //栈指针下移8个字节
  400ee4: be 00 24 40 00       mov    $0x402400,%esi  //将地址0x402400的内容复制到%esi
  400ee9: e8 4a 04 00 00       callq  401338 <strings_not_equal>  //调用<strings_not_equal>
  400eee: 85 c0                test   %eax,%eax
  400ef0: 74 05                je     400ef7 <phase_1+0x17>     //进行判断,为0即返回,否则爆炸
  400ef2: e8 43 05 00 00       callq  40143a <explode_bomb>
  400ef7: 48 83 c4 08          add    $0x8,%rsp    //栈指针上移8个字节,
  400efb: c3                   retq
```

其中<strings_not_equal>用来判断字符是否相等

（3）使用 gdb 进行调试

```
Breakpoint 1, 0x0000000000400ee9 in phase_1 ()
(gdb) x/s $esi
0x402400:       "Border relations with Canada have never been better."
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
(gdb)
```

在 0x400ee9 处设置断点

使用 *x/s $esi* 或者 *x/s 0x402400* 均可获得字符串，即第一关答案是

Border relations with Canada have never been better.

（4）运行

```
root@evassh-3151774:/data/workspace/myshixun/step1# ./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?
```

（5）代码文件

```c
#include<stdio.h>

void main(){
    /********** Begin **********/
    printf("Border relations with Canada have never been better.");
    /********** End **********/
}
```

二、拆弹密码第二关

1、任务描述

本关任务：在 phase_2 的汇编代码中找到本关拆弹密码。

2、编程要求

根据提示，在汇编代码中找到本关密码，在 c 文件中将密码输出。

3、实验步骤

（1）进入 bomb 文件，在命令行利用反汇编得到 bomb.s，并且定位到 phase_2；

```
000000000400efc <phase_2>:
  400efc: 55                    push   %rbp
  400efd: 53                    push   %rbx
  400efe: 48 83 ec 28           sub    $0x28,%rsp     //栈指针下移40个字节
  400f02: 48 89 e6              mov    %rsp,%rsi      //将栈指针保存在%rsi
  400f05: e8 52 05 00 00        callq  40145c <read_six_numbers>   //调用函数,读取六个数字
  400f0a: 83 3c 24 01           cmpl   $0x1,(%rsp)          //将栈指针的内存值与1进行比较
  400f0e: 74 20                 je     400f30 <phase_2+0x34>   //等于1, 跳转到0x400f30
  400f10: e8 25 05 00 00        callq  40143a <explode_bomb>   //不等于1, 爆炸
  400f15: eb 19                 jmp    400f30 <phase_2+0x34>
  400f17: 8b 43 fc              mov    -0x4(%rbx),%eax          //由400f3a的值调转到此处时, 将%rsp -4 的值赋给%eax
  400f1a: 01 c0                 add    %eax,%eax                //%eax值扩大两倍
  400f1c: 39 03     构成一个循环 cmp    %eax,(%rbx)             //将%rbx的值与%eax比较
  400f1e: 74 05                 je     400f25 <phase_2+0x29>   //相等, 调转到  0x400f25
  400f20: e8 15 05 00 00        callq  40143a <explode_bomb>    //不等, 爆炸
  400f25: 48 83 c3 04           add    $0x4,%rbx                //%rbx =%rbx+4
  400f29: 48 39 eb              cmp    %rbp,%rbx                //将%rbx的值与%rbp比较
  400f2c: 75 e9                 jne    400f17 <phase_2+0x1b>    //%rbx<=%rbp, 跳转到 0x400f3c
  400f2e: eb 0c                 jmp    400f3c <phase_2+0x40>     //大于跳转到0x400f3c
  400f30: 48 8d 5c 24 04        lea    0x4(%rsp),%rbx         //由400f0e跳转到此处时, 将%rsp +4的值赋给%rbx
  400f35: 48 8d 6c 24 18        lea    0x18(%rsp),%rbp        //将%rsp +24 的值赋给%rbp
  400f3a: eb db                 jmp    400f17 <phase_2+0x1b>   //跳转到0x400f17
  400f3c: 48 83 c4 28           add    $0x28,%rsp        //栈指针上移40个字节,
  400f40: 5b                    pop    %rbx
  400f41: 5d                    pop    %rbp
  400f42: c3                    retq
```

调用<read_six_numbers>函数，主要作用是读取并分配六个数字的位置

```
000000000040145c <read_six_numbers>:
  40145c: 48 83 ec 18           sub    $0x18,%rsp  //栈指针下移24个字节
  401460: 48 89 f2              mov    %rsi,%rdx
  401463: 48 8d 4e 04           lea    0x4(%rsi),%rcx
  401467: 48 8d 46 14           lea    0x14(%rsi),%rax
  40146b: 48 89 44 24 08        mov    %rax,0x8(%rsp)
  401470: 48 8d 46 10           lea    0x10(%rsi),%rax
  401474: 48 89 04 24           mov    %rax,(%rsp)
  401478: 4c 8d 4e 0c           lea    0xc(%rsi),%r9
  40147c: 4c 8d 46 08           lea    0x8(%rsi),%r8
  401480: be c3 25 40 00        mov    $0x4025c3,%esi
  401485: b8 00 00 00 00        mov    $0x0,%eax
  40148a: e8 61 f7 ff ff        callq  400bf0 <__isoc99_sscanf@plt>
  40148f: 83 f8 05              cmp    $0x5,%eax
  401492: 7f 05                 jg     401499 <read_six_numbers+0x3d>
  401494: e8 a1 ff ff ff        callq  40143a <explode_bomb>
  401499: 48 83 c4 18           add    $0x18,%rsp
  40149d: c3                    retq
```

查看 0x4025c3，可以得到输入六个数字的格式

```
(gdb) print (char *)0x4025c3
$8 = 0x4025c3 "%d %d %d %d %d %d"
```

调用<__isoc99_sscanf@plt>依次对输入的数字进行格式化

<read_six_numbers>函数将读取的六个数依次存放在（%rsp）-（%rsp+20），的位置；

（2）分析汇编代码，发现第一个数应该为 1，第二个数应该为 2，依次递增一倍，直达达到 6 个数字，可以输入 6 个以上的数字，但是只会读取前 6 个数字

（3）运行

```
root@evassh-3151774:/data/workspace/myshixun/step1# ./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?
1 2 4 8 16 32
That's number 2.  Keep going!
```

（4）代码文件

```c
#include<stdio.h>

void main(){
    /********** Begin **********/
    printf("1 2 4 8 16 32");
    /********** End **********/
}

```

三、拆弹密码第三关

1、任务描述

本关任务：在 phase_3 的汇编代码中找到本关拆弹密码。

2、编程要求

根据提示，在汇编代码中找到本关密码，在 c 文件中将密码输出。

3、实验步骤

（1）进入 bomb 文件，在命令行利用反汇编得到 bomb.s，并且定位到 phase_3；

```
0000000000400f43 <phase_3>:
  400f43: 48 83 ec 18          sub    $0x18,%rsp      //栈指针下移24个字节
  400f47: 48 8d 4c 24 0c       lea    0xc(%rsp),%rcx     //将%rsp+12 的值赋给%rcx
  400f4c: 48 8d 54 24 08       lea    0x8(%rsp),%rdx     //将%rsp+8  的值赋给%rdx
  400f51: be cf 25 40 00       mov    $0x4025cf,%esi      //将地址0x4025cf的内容复制到%esi,
  400f56: b8 00 00 00 00       mov    $0x0,%eax        //%eax=0
  400f5b: e8 90 fc ff ff       callq  400bf0 <__isoc99_sscanf@plt>     //调用函数输入
  400f60: 83 f8 01             cmp    $0x1,%eax                 //输入的值的数量与1进行比较
  400f63: 7f 05                jg     400f6a <phase_3+0x27>      //输入的值数量大于1, 跳转到 0x400f6a
  400f65: e8 d0 04 00 00       callq  40143a <explode_bomb>      //小于1, 爆炸
  400f6a: 83 7c 24 08 07       cmpl   $0x7,0x8(%rsp)             //%rsp+8 的值与7进行对比
  400f6f: 77 3c                ja     400fad <phase_3+0x6a>      //大于, 调到0x400fad,爆炸
  400f71: 8b 44 24 08          mov    0x8(%rsp),%eax             //将%rsp+8 的值赋给%eax
  400f75: ff 24 c5 70 24 40 00 jmpq   *0x402470(,%rax,8)        //跳转到0x4070+ 8*%rax 的地址
  400f7c: b8 cf 00 00 00       mov    $0xcf,%eax                 //%eax=207
  400f81: eb 3b                jmp    400fbe <phase_3+0x7b>
  400f83: b8 c3 02 00 00       mov    $0x2c3,%eax               //%eax=707
  400f88: eb 34                jmp    400fbe <phase_3+0x7b>       //跳转到0x400fbe
  400f8a: b8 00 01 00 00       mov    $0x100,%eax                //%eax=256
  400f8f: eb 2d                jmp    400fbe <phase_3+0x7b>       //跳转到0x400fbe
  400f91: b8 85 01 00 00       mov    $0x185,%eax               //%eax=389
  400f96: eb 26                jmp    400fbe <phase_3+0x7b>       //跳转到0x400fbe
  400f98: b8 ce 00 00 00       mov    $0xce,%eax                //%eax=206
  400f9d: eb 1f                jmp    400fbe <phase_3+0x7b>       //跳转到0x400fbe
  400f9f: b8 aa 02 00 00       mov    $0x2aa,%eax                 //%eax=682
  400fa4: eb 18                jmp    400fbe <phase_3+0x7b>       //跳转到0x400fbe
  400fa6: b8 47 01 00 00       mov    $0x147,%eax               //%eax=327
  400fab: eb 11                jmp    400fbe <phase_3+0x7b>     //跳转到0x400fbe
  400fad: e8 88 04 00 00       callq  40143a <explode_bomb>
```

```
  400fb2: b8 00 00 00 00       mov    $0x0,%eax                 //%eax=0
  400fb7: eb 05                jmp    400fbe <phase_3+0x7b>     //%eax=311
  400fb9: b8 37 01 00 00       mov    $0x137,%eax              //%eax=311
  400fbe: 3b 44 24 0c          cmp    0xc(%rsp),%eax            //将%rsp+12 的值赋给eax
  400fc2: 74 05                je     400fc9 <phase_3+0x86>      //等于的话, 返回
  400fc4: e8 71 04 00 00       callq  40143a <explode_bomb>      //不等, 爆炸
  400fc9: 48 83 c4 18          add    $0x18,%rsp
  400fcd: c3                   retq
```

分析汇编知，这是一个 **siwitch 语句**

（3）调试

[1]调试 *mov $0x4025cf,%esi* 语句，使用 gdb 进行查看，获取格式

```
(gdb) print (char*)0x4025cf
$3 = 0x4025cf "%d %d"
```

[2] *jmpq *0x402470(,%rax,8)* 存放的是 switch 语句索引的位置，使用 gdb 进行查看；

```
(gdb) x/x 0x402470
0x402470:       0x0000000000400f7c
(gdb) x/8x 0x402470
0x402470:       0x0000000000400f7c      0x0000000000400fb9
0x402480:       0x0000000000400f83      0x0000000000400f8a
0x402490:       0x0000000000400f91      0x0000000000400f98
0x4024a0:       0x0000000000400f9f      0x0000000000400fa6
```

[3]对应数字

0-207 1-311 2-707 3-256 4-389 5-206 6-682 7-327

[4]代码文件

```c
#include<stdio.h>

void main(){
    /********** Begin **********/
    //有多个答案，只需填写一个
   printf("1 311");
    /********** End **********/
}

```

四、拆弹密码第四关

1、任务描述

本关任务：在 phase_4 的汇编代码中找到本关拆弹密码。

2、编程要求

根据提示，在汇编代码中找到本关密码，在 c 文件中将密码输出。

3、实验步骤

(1)进入bomb文件，在命令行利用反汇编得到bomb.s，并且定位到phase_4；

```
0000000000400fce <func4>:                //func(%edi=%rsp+8,%esi=0,%edx=14)
  400fce: 48 83 ec 08          sub    $0x8,%rsp    //栈指针下移8个字节
  400fd2: 89 d0                mov    %edx,%eax     //%eax=%edx=14
  400fd4: 29 f0                sub    %esi,%eax    //%eax=%eax-%esi=14-0=14
  400fd6: 89 c1                mov    %eax,%ecx     //%ecx=%eax=14
  400fd8: c1 e9 1f             shr    $0x1f,%ecx    //逻辑右移31位，%ecx=0
  400fdb: 01 c8                add    %ecx,%eax     //%eax=%ecx+%eax=14
  400fdd: d1 f8                sar    %eax          //%eax算术右移一位，%eax=7
  400fdf: 8d 0c 30             lea    (%rax,%rsi,1),%ecx    //%ecx=%rax+%rsi=7+0=7
  400fe2: 39 f9                cmp    %edi,%ecx
  400fe4: 7e 0c                jle    400ff2 <func4+0x24> //%ecx (=7) <=%edi,跳转至0x400ff2,其中%edi=%rsp+8的值
  400fe6: 8d 51 ff             lea    -0x1(%rcx),%edx    //否则%edx=%rcx-1=6
  400fe9: e8 e0 ff ff ff       callq  400fce <func4>     //调用函数func4 ,func4(%edi=%rsp+8,%esi=0,%edx=6)
  400fee: 01 c0                add    %eax,%eax           //%eax=2*%eax
  400ff0: eb 15                jmp    401007 <func4+0x39>    //返回
  400ff2: b8 00 00 00 00       mov    $0x0,%eax        //%eax=0
  400ff7: 39 f9                cmp    %edi,%ecx
  400ff9: 7d 0c                jge    401007 <func4+0x39>  //%ecx>=%edi，跳转至0x401007,返回，其中%edi=%rsp+8的值
  400ffb: 8d 71 01             lea    0x1(%rcx),%esi     //否则，%esi=%rcx+1
  400ffe: e8 cb ff ff ff       callq  400fce <func4>      //调用函数func4,func4(%edi=%rsp+8,%esi=9,%edx=14)
  401003: 8d 44 00 01          lea    0x1(%rax,%rax,1),%eax  //%eax=2*%rax+1
  401007: 48 83 c4 08          add    $0x8,%rsp
  40100b: c3                   retq
```

```
000000000040100c <phase_4>:
  40100c: 48 83 ec 18          sub    $0x18,%rsp     //栈指针下移24的字节
  401010: 48 8d 4c 24 0c       lea    0xc(%rsp),%rcx    //将%rsp+12 的值赋给%rcx
  401015: 48 8d 54 24 08       lea    0x8(%rsp),%rdx    //将%rsp+8 的值赋给%rdx
  40101a: be cf 25 40 00       mov    $0x4025cf,%esi    //输入的值赋给%esi
  40101f: b8 00 00 00 00       mov    $0x0,%eax            //%eax=0
  401024: e8 c7 fb ff ff       callq  400bf0 <__isoc99_sscanf@plt>    //调用函数
  401029: 83 f8 02             cmp    $0x2,%eax            //将%eax的值与2进行比较
  40102c: 75 07                jne    401035 <phase_4+0x29>      //eax!=2,跳转到0x401035
  40102e: 83 7c 24 08 0e       cmpl   $0xe,0x8(%rsp)             //将%rsp+8的值与14进行比较
  401033: 76 05                jbe    40103a <phase_4+0x2e>      //<=14, 跳转到0x40103a
  401035: e8 00 04 00 00       callq  40143a <explode_bomb>      //否则爆炸
  40103a: ba 0e 00 00 00       mov    $0xe,%edx                  //%edx=14
  40103f: be 00 00 00 00       mov    $0x0,%esi               //%esi=0
  401044: 8b 7c 24 08          mov    0x8(%rsp),%edi          //%edi= %rsp+8 的值
  401048: e8 81 ff ff ff       callq  400fce <func4>          //调用函数func4,func(%edi,%esi=0,%edx=14)
  40104d: 85 c0                test   %eax,%eax
  40104f: 75 07                jne    401058 <phase_4+0x4c>      //%eax!=0,跳转爆炸
  401051: 83 7c 24 0c 00       cmpl   $0x0,0xc(%rsp)            //将%rsp+12的值与0进行比较
  401056: 74 05                je     40105d <phase_4+0x51>    //相等跳转返回
  401058: e8 dd 03 00 00       callq  40143a <explode_bomb>    //否则爆炸
  40105d: 48 83 c4 18          add    $0x18,%rsp
  401061: c3                   retq
```

通第三关一样，查看 0x4025cf，知第四关输入的是两个数字

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

```
rcx   0xc(%rsp)
rdx   0x8(%rsp)
```

（2）分析汇编，等效 C 代码

第一个数存入 0xc(%rsp)　第二个数存入 0x8(%rsp)，func4 等效 C 代码：

```
if(eax!=2)
{

    if(rdx<=15)
    {

        esi=0;
        edi=rdx;

        fun4(edx,esi,edi)   //  esi= 0  edx=14  %edi=%rsp+8
        {
            []    //sub    $0x8,%rsp

            eax=edx;
            eax=eax-esi;
             //eax=edx-esi
            ecx=eax;
            ecx=ecx>>31;  // logic  //0
            //ecx=edx-esi
            eax=eax+ecx;
            eax=eax>>1;//Ath
            //eax=(edx-esi)  +  (edx-esi)>>31
            //eax=eax>>1
            ecx=rax+rsi;
            //ecx=eax + esi;
            //ecx= ((edx-esi)  +  (edx-esi)>>31))/2  +esi
```

```
if(ecx>edi)
{
    edx=rcx-1;
    fun4(edx,esi,edi);
    eax=2*eax+1;
    return ;
}
else
{
    eax=0;
    if(ecx<edi)
    {
        esi=rcx+1;
        fun4(edx,esi,edi);
        eax=2*rax+1;
    }
    else
    {
        return ;
    }
}
}
```

func4的C代码

```
    if(eax=0)
    {
        explode_bomb;
    }

    if(rdx=0)    //rdx=0xc(%rsp)
      return;
    else
      explode_bomb;    //rdx=0;
  }
  else
  {
    explode_bomb;
  }
}
else
{
  explode_bomb;
}
```

判断出其中一个数为0

通过对 func4 的 C 代码分析，可以判断出 rdx 的取值范围为[0, 14]，所以代入进行判断，发现可以取值 0, 1, 3，7，所以这个数输入有四种可能 0, 1, 3, 7。

对于另一个数，由语句 0x401051 分析知，（%rsp+12）为 0 时满足题意。

（4）结果为（0, 0），（1, 0），（3, 0），（7，0）

（5）代码文件（其中一种）

```
#include<stdio.h>

void main(){
    /********** Begin **********/
    //有多个答案，只需填写一个
    printf("3 0");
    /********** End **********/
}

```

五、拆弹密码第五关

1、任务描述

本关任务：在 phase_2 的汇编代码中找到本关拆弹密码。

2、编程要求

根据提示，在汇编代码中找到本关密码，在 c 文件中将密码输出。

3、实验步骤

（1）进入 bomb 文件，在命令行利用反汇编得到 bomb. s，并且定位到 phase_2；

```
0000000000401062 <phase_5>:
  401062: 53                    push   %rbx            //栈指针下移8个字节, %rsp的值等于%rbx
  401063: 48 83 ec 20           sub    $0x20,%rsp      //栈指针下移32个字节
  401067: 48 89 fb              mov    %rdi,%rbx        //%rbx=%rdi
  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           //%eax=0
  40107a: e8 9c 02 00 00        callq  40131b <string_length>   //判断字符串长度
  40107f: 83 f8 06              cmp    $0x6,%eax
  401082: 74 4e                 je     4010d2 <phase_5+0x70>       //=6,跳转到0x4010d2
  401084: e8 b1 03 00 00        callq  40143a <explode_bomb>     //否则爆炸
  401089: eb 47                 jmp    4010d2 <phase_5+0x70>     //跳转到0x4010d2
```

```
40108b: 0f b6 0c 03           movzbl (%rbx,%rax,1),%ecx    //将%rbx+%rax的值赋给%ecx,即输入
40108f: 88 0c 24              mov    %cl,(%rsp)           //%cl是8位,保存在%rsp的位置
                            /***将字符串的第一个字符压入栈***/

401092: 48 8b 14 24           mov    (%rsp),%rdx          //%将%rsp 的值赋给%rdx
401096: 83 e2 0f              and    $0xf,%edx           //%edx=%edx&1111,将字符与1111进行与运算

401099: 0f b6 92 b0 24 40 00  movzbl 0x4024b0(%rdx),%edx    //将%rdx+0x4024b0的值赋给%edx
4010a0: 88 54 04 10           mov    %dl,0x10(%rsp,%rax,1)  //%dl是8位,保存在%rsp+%rax+16的位置
4010a4: 48 83 c0 01           add    $0x1,%rax            //%rax+1
4010a8: 48 83 f8 06           cmp    $0x6,%rax           //与6进行对比
4010ac: 75 dd                 jne    40108b <phase_5+0x29>  //小于等于,跳转到0x40108b,进入循环
4010ae: c6 44 24 16 00        movb   $0x0,0x16(%rsp)    //否则将0赋给%rsp+22的位置
```

```
4010b3: be 5e 24 40 00        mov    $0x40245e,%esi     //将地址0x40245e的内容复制到%edi
4010b8: 48 8d 7c 24 10        lea    0x10(%rsp),%rdi    //将%rsp+8的值赋给%rdi
4010bd: e8 76 02 00 00        callq  401338 <strings_not_equal>   //调用函数判断是否相等
4010c2: 85 c0                 test   %eax,%eax
4010c4: 74 13                 je     4010d9 <phase_5+0x77>       //%eax=0,跳转到0x4010d9
4010c6: e8 6f 03 00 00        callq  40143a <explode_bomb>      //否则,爆炸
4010cb: 0f 1f 44 00 00        nopl   0x0(%rax,%rax,1)
4010d0: eb 07                 jmp    4010d9 <phase_5+0x77>    //跳转到0x4010d9
4010d2: b8 00 00 00 00        mov    $0x0,%eax            //%eax=0
4010d7: eb b2                 jmp    40108b <phase_5+0x29>    //跳转到0x40108b
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        callq  400b30 <__stack_chk_fail@plt>
4010ee: 48 83 c4 20           add    $0x20,%rsp
4010f2: 5b                    pop    %rbx
4010f3: c3                    retq
```

（2）使用 gdb 调试

使用 gdb 查看相应地址的字符串

```
(gdb) x/s 0x4024b0
0x4024b0 <array.3449>:  "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
```

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

输入的字符串取每个字符的 ASCII 表的后四位与 1111 进行与运算，分析语句 *movzbl 0x4024b0(%rdx),%edx*，**知取出（0x4024b0+edx）处对应位置的字符，将结果依次存在%rsp+16+%rax 的地址上；**

之后处理结果和 0x40245e 处的字符，调用<strings_not_equal>判断是否相等。比较字符串，知道

f-9-1001，l-15-1111，y-14-1110，e-5-0101，r-6-0110，s-7-0111；

所以在 ASCII 表中只有后四位对应相等即可，通过查表

f-9-1001 ——) 9 I Y i y

l-15-1111 —— / ? O _ o

y-14-1110 —— . > N ^ n

e-5-0101 —— 5 E U e u %

r-6-0110 —— 6 & F f v V

s-7-0111 —— 7 G g W w 其中一个答案是 9 o n 5 6 7

(3)代码文件

```
#include<stdio.h>

void main(){
    /********** Begin **********/
    //有多个答案，只需填写一个
    printf("9 o n 5 6 7");
    /********** End **********/
}

```

六、拆弹密码第六关

1、任务描述

本关任务：在 phase_2 的汇编代码中找到本关拆弹密码。

2、编程要求

根据提示，在汇编代码中找到本关密码，在 c 文件中将密码输出。

3、实验步骤

(1)进入 bomb 文件，在命令行利用反汇编得到 bomb.s，并且定位到 phase_2；

```
00000000004010f4 <phase_6>:
  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    //栈指针下移80的字节
  401100: 49 89 e5             mov    %rsp,%r13
  401103: 48 89 e6             mov    %rsp,%rsi
  401106: e8 51 03 00 00       callq  40145c <read_six_numbers>  //读取六个数字,从%rsi开始存入
  40110b: 49 89 e6             mov    %rsp,%r14       //%r14=%rsp,第一个数赋给%r14
  40110e: 41 bc 00 00 00 00    mov    $0x0,%r12d  //%r12=0
```

读取六个数

```
401114: 4c 89 ed            mov    %r13,%rbp   //%rbp=%r13
401117: 41 8b 45 00         mov    0x0(%r13),%eax   //将%r13的值赋给%eax
40111b: 83 e8 01            sub    $0x1,%eax        //%eax=%eax-1
40111e: 83 f8 05            cmp    $0x5,%eax
401121: 76 05               jbe    401128 <phase_6+0x34>   //%eax<=5,跳转到0x401128
401123: e8 12 03 00 00      callq  40143a <explode_bomb>   //>5,爆炸
```

数字要小于等于6

```
401128: 41 83 c4 01         add    $0x1,%r12d       //%r12d=%r12d+1
40112c: 41 83 fc 06         cmp    $0x6,%r12d
401130: 74 21               je     401153 <phase_6+0x5f>  //% r12=6,跳转到0x401153
401132: 44 89 e3            mov    %r12d,%ebx            //%ebx=%r12d+1
```

进入循环，
%r12d计数，
等于6时结束循环

```
401132: 44 89 e3            mov    %r12d,%ebx           //%ebx=%r12d+1
401135: 48 63 c3            movslq %ebx,%rax            //%rax=%ebx=%r12d+1
401138: 8b 04 84            mov    (%rsp,%rax,4),%eax    //%eax=%rsp+4*%rax=%rsp+4*(%r12d+1)的值
40113b: 39 45 00            cmp    %eax,0x0(%rbp)        //相邻两个数比较
40113e: 75 05               jne    401145 <phase_6+0x51>    //%rbp 的值!=%eax,跳转0x401145
401140: e8 f5 02 00 00      callq  40143a <explode_bomb>
401145: 83 c3 01            add    $0x1,%ebx            //%ebx=%ebx+1
401148: 83 fb 05            cmp    $0x5,%ebx
40114b: 7e e8               jle    401135 <phase_6+0x41>   //%ebx<=5 跳转到0x401135,循环
```

前后两个数字不相等

由（1）第三张图知，当%r12d=6 时，跳转到 0x401153

```
401153: 48 8d 74 24 18      lea    0x18(%rsp),%rsi      //%rsi=%rsp+24, 第六个数地址
401158: 4c 89 f0            mov    %r14,%rax            //%rax=%r14, 第一个数地址   (40110b: 49 89 e6  mov    %rsp,%r14  )
40115b: b9 07 00 00 00      mov    $0x7,%ecx           //%ecx=7

401160: 89 ca               mov    %ecx,%edx           //%edx=%ecx=7
401162: 2b 10               sub    (%rax),%edx            //将%edx-%rax的值, 7-第一个数
401164: 89 10               mov    %edx,(%rax)          //将%edx赋给%rax的值, 第一个数=7-第一个数
401166: 48 83 c0 04         add    $0x4,%rax              //%rax=%rax+4 ,下一个数
40116a: 48 39 f0            cmp    %rsi,%rax              //%rsi是第六个数地址,比较地址,判断是否读完
40116d: 75 f1               jne    401160 <phase_6+0x6c>   //%rax!=%rsi ,跳转到0x401160,循环,
40116f: be 00 00 00 00      mov    $0x0,%esi           //%esi=0
401174: eb 21               jmp    401197 <phase_6+0xa3>
****************************  /*将所有的数变为 7-原来的数*/ ************************************
```

跳转到 0x401197

这一部分的主要作用是读取输入的六个数字，输入的值应该是 1，2，3，4，5，6 这六个数字，将这六个数字依次按输入顺序存在（%rsp）-（%rsp+20）的位置，并且将每个数作为被减数与 7 相减，将结果存储在原位置；

```
401176: 48 8b 52 08          mov    0x8(%rdx),%rdx       //将%rdx+8 的值赋给%rdx
                                                         0x6032d0+8
40117a: 83 c0 01             add    $0x1,%eax            //%eax+1
40117d: 39 c8                cmp    %ecx,%eax
40117f: 75 f5                jne    401176 <phase_6+0x82>     //%eax!=%ecx,跳转至0x401176
401181: eb 05                jmp    401188 <phase_6+0x94>     //跳转到0x401188

401183: ba d0 32 60 00       mov    $0x6032d0,%edx          //将0x6032d0赋给%edx
401188: 48 89 54 74 20       mov    %rdx,0x20(%rsp,%rsi,2)   //将%rdx赋给%rsp+2%rsi+32=%rsp+32的值   将0x6032d0的值赋给这个位置
40118d: 48 83 c6 04          add    $0x4,%rsi              //%rsi+4
401191: 48 83 fe 18          cmp    $0x18,%rsi             //判断%rsi是否为24
401195: 74 14                je     4011ab <phase_6+0xb7>      //若%rsi=24,跳转到0x4011ab
                 循环入口
401197: 8b 0c 34             mov    (%rsp,%rsi,1),%ecx       //将%rsp+%rsi = %rsp+0 = %rsp 的值赋给%ecx,%ecx第一次为第一个数
40119a: 83 f9 01             cmp    $0x1,%ecx
40119d: 7e e4                jle    401183 <phase_6+0x8f>   //%ecx<=1,跳转到0x401183

40119f: b8 01 00 00 00       mov    $0x1,%eax            //%eax=1
4011a4: ba d0 32 60 00       mov    $0x6032d0,%edx       //将0x6032d0赋给%edx
4011a9: eb cb                jmp    401176 <phase_6+0x82>
```

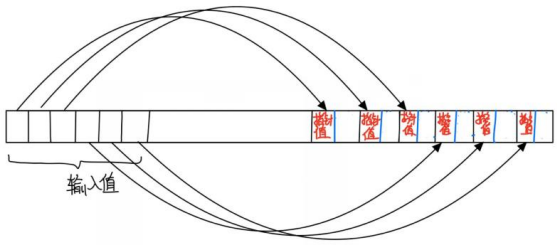
使用 gdb 查看 0x6032d0 处左右的值

```
(gdb) x/24x 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
```

可以发现每个结点有 16 个字节，四个部分，第一部分存放的是结点的值，第二个存放的是结点对应的编号，第三个结点存放的是下一个结点的位置，第四个部分则全为 0；

这部分第一步，%rsp 处地址存放的值（第一个数）与 1 比，小于等于则将 0x6032d0 地址处的值放在%rsp+2%rsi+32 的地方，即存放第一个结点的值；否则，%eax=1，将 0x6032d0 地址处的值赋给 edx，跳转到 0x401176，将 0x6032d0+8 的地址，赋给%rdx，获得第二个结点指针的，eax+1，%eax!=%ecx，跳转至 0x401176，继续循环，直到 eax=ecx，其中 ecx 就是（7-单个输入值数字），然后将%rdx 赋给%rsp+2%rsi+32=%rsp+32 所指向的值，此时%rdx 存放的是 ecx 对应结点顺序的地址，然后进入循环。这一部分简而言之，就是根据按照（%rsp）-（%rsp+20）的数字顺序把指向对应结点的指针值依次存入（%rsp+32）-（%rsp+72）的位置，

例如若第一个数字输入 3，则处理后数字为 7-3=4，那再此次处理中（%rsp+32）存放的就是 0x6032f0，直至%rsi=24，跳转到 0x4011ab，再次进入循环



```
4011ab: 48 8b 5c 24 20        mov    0x20(%rsp),%rbx     //将%rsp+32 的值赋给%rbx   上一步获得的第一个值
4011b0: 48 8d 44 24 28        lea    0x28(%rsp),%rax     //%rax= %rsp+40[redacted]   上一步获得的第二个值地址
4011b5: 48 8d 74 24 50        lea    0x50(%rsp),%rsi     //%rsi= %rsp+80[redacted]
4011ba: 48 89 d9              mov    %rbx,%rcx           //%rcx=%rbx

4011bd: 48 8b 10              mov    (%rax),%rdx        //将%rax的值赋给%rdx
4011c0: 48 89 51 08           mov    %rdx,0x8(%rcx)     //将%rdx 赋给%rcx+8的值
4011c4: 48 83 c0 08           add    $0x8,%rax         //%rax+8                    循环，第三个，第四个值，第五个值
4011c8: 48 39 f0              cmp    %rsi,%rax    %rsi=24
4011cb: 74 05                 je     4011d2 <phase_6+0xde>  //%rax=%rsi,跳转到0x400d2
4011cd: 48 89 d1              mov    %rdx,%rcx          //%rcx=%rdx
4011d0: eb eb                 jmp    4011bd <phase_6+0xc9>  //跳转到0x4011bd
```

按顺序将下一个值放在上一个值+8 得到的地址中，

跳转到 0x4011d2

```
4011d2: 48 c7 42 08 00 00 00  movq   $0x0,0x8(%rdx)        //将0赋给%rdx+8的值
4011d9: 00                                                                  最后一个链表的next应该为null
4011da: bd 05 00 00 00        mov    $0x5,%ebp          //%ebp=5

                                                   +8
4011df: 48 8b 43 08           mov    0x8(%rbx),%rax     //将%rbx 的值 赋给%rax [redacted]
4011e3: 8b 00                 mov    (%rax),%eax       //将%rax 的值 赋给%eax
4011e5: 39 03                 cmp    %eax,(%rbx)                                 前一个值要大于后一个值，
4011e7: 7d 05                 jge    4011ee <phase_6+0xfa>   //%rbx 的值大于%eax,则跳转到0x4011ee                 否则爆炸
  4011e9: e8 4c 02 00 00        callq  40143a <explode_bomb>   //否则,爆炸

4011ee: 48 8b 5b 08           mov    0x8(%rbx),%rbx      //将%rbx+8的值赋给%rbx
4011f2: 83 ed 01              sub    $0x1,%ebp       //%ebp-1
4011f5: 75 e8                 jne    4011df <phase_6+0xeb>  //非零,循环
4011f7: 48 83 c4 50           add    $0x50,%rsp
4011fb: 5b                    pop    %rbx
4011fc: 5d                    pop    %rbp
4011fd: 41 5c                 pop    %r12
4011ff: 41 5d                 pop    %r13
401201: 41 5e                 pop    %r14
401203: c3                    retq
```

结点值必须成递减序列，否则爆炸。

（3）分析

第一步令输入的数字=7-输入的数字，存放在（%rsp）-（%rsp+20）的位置；

第二步将对应结点值按照第一步处理后数字与对应结点编号将结点指针值存放在%rsp+32-%rsp+72 的位置；

如图所示：

第三步，根据爆炸条件，结点值必须按从大到小排列，否则爆炸；

```
(gdb) x/x 0x6032d0
0x6032d0 <node1>:       0x0000014c
(gdb) x/24x 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
```

所以按照从大到小排列，顺序依次为 3 4 5 6 1 2，与 7 相减，

则为 4 3 2 1 6 5；

（4）代码文件

```c
#include<stdio.h>

void main(){
    /********** Begin **********/
    printf("4 3 2 1 6 5");
    /********** End **********/
}

```

七、拆弹密码隐藏关

1、任务描述

本关任务：先找到隐藏关，再找到本关拆弹密码。

2、编程要求

找出隐藏关进入方式，在汇编代码中找到相应密码，在 c 文件中将密码输出。

3、实验步骤

（1），先找出进入隐藏关的方法，在汇编代码发现存在<secret_phase>,分析 C 代码发现每次拆除炸弹成功后，都会调用 phase_defused（）；

```
00000000004015c4 <phase_defused>:
  4015c4: 48 83 ec 78          sub    $0x78,%rsp
  4015c8: 64 48 8b 04 25 28 00 mov    %fs:0x28,%rax
  4015cf: 00 00
  4015d1: 48 89 44 24 68       mov    %rax,0x68(%rsp)
  4015d6: 31 c0                xor    %eax,%eax
  4015d8: 83 3d 81 21 20 00 06 cmpl   $0x6,0x202181(%rip)        # 603760 <num_input_strings>
  4015df: 75 5e                jne    40163f <phase_defused+0x7b>
  4015e1: 4c 8d 44 24 10       lea    0x10(%rsp),%r8
  4015e6: 48 8d 4c 24 0c       lea    0xc(%rsp),%rcx
  4015eb: 48 8d 54 24 08       lea    0x8(%rsp),%rdx
  4015f0: be 19 26 40 00       mov    $0x402619,%esi
  4015f5: bf 70 38 60 00       mov    $0x603870,%edi
  4015fa: e8 f1 f5 ff ff       callq  400bf0 <__isoc99_sscanf@plt>
  4015ff: 83 f8 03             cmp    $0x3,%eax        //输入的应该包含3个部分
  401602: 75 31                jne    401635 <phase_defused+0x71>
  401604: be 22 26 40 00       mov    $0x402622,%esi
  401609: 48 8d 7c 24 10       lea    0x10(%rsp),%rdi
  40160e: e8 25 fd ff ff       callq  401338 <strings_not_equal>     //判断是否是否相等
  401613: 85 c0                test   %eax,%eax
  401615: 75 1e                jne    401635 <phase_defused+0x71>
  401617: bf f8 24 40 00       mov    $0x4024f8,%edi
  40161c: e8 ef f4 ff ff       callq  400b10 <puts@plt>
  401621: bf 20 25 40 00       mov    $0x402520,%edi
  401626: e8 e5 f4 ff ff       callq  400b10 <puts@plt>
  40162b: b8 00 00 00 00       mov    $0x0,%eax
  401630: e8 0d fc ff ff       callq  401242 <secret_phase>
  401635: bf 58 25 40 00       mov    $0x402558,%edi
  40163a: e8 d1 f4 ff ff       callq  400b10 <puts@plt>
  40163f: 48 8b 44 24 68       mov    0x68(%rsp),%rax
  401644: 64 48 33 04 25 28 00 xor    %fs:0x28,%rax
```

使用 gdb 查看对应地址的值

```
(gdb) b phase_defused
Breakpoint 1 at 0x4015c4
(gdb) run answer.txt
Starting program: /data/workspace/myshixun/step1/bomb answer.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!

Breakpoint 1, 0x00000000004015c4 in phase_defused () 函数起始处设置断点
(gdb) x/s 0x402619
0x402619:       "%d %d %s"   格式分别为十进制数 十进制数 字符串
(gdb) x/s 0x603870
0x603870 <input_strings+240>:   ""   输入值
(gdb) x/s 0x402622
0x402622:       "DrEvil"         判断的字符串是否相等
(gdb)
```

再将断点设置在 0x4015fa ,读取其中的值

```
(gdb) b *0x4015fa
Breakpoint 2 at 0x4015fa
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /data/workspace/myshixun/step1/bomb answer.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...

Breakpoint 2, 0x00000000004015fa in phase_defused ()
(gdb) x/s 0x603870
0x603870 <input_strings+240>:   "7 0 DrEvil"
(gdb)
```

发现如果是 7 0 DrEvil;

结合之前六关分析知，7 0 正是第四关的一个答案，所以如果第四关输入的是 7 0 DrEvil，即可进入隐藏关

```
40160e: e8 25 fd ff ff        callq  401338 <strings_not_equal>    //判断是否是否相等
401613: 85 c0                 test   %eax,%eax
401615: 75 1e                 jne    401635 <phase_defused+0x71>
401617: bf f8 24 40 00        mov    $0x4024f8,%edi
40161c: e8 ef f4 ff ff        callq  400b10 <puts@plt>
401621: bf 20 25 40 00        mov    $0x402520,%edi
401626: e8 e5 f4 ff ff        callq  400b10 <puts@plt>
40162b: b8 00 00 00 00        mov    $0x0,%eax
401630: e8 0d fc ff ff        callq  401242 <secret_phase>
401635: bf 58 25 40 00        mov    $0x402558,%edi
40163a: e8 d1 f4 ff ff        callq  400b10 <puts@plt>
40163f: 48 8b 44 24 68        mov    0x68(%rsp),%rax
```

使用 gdb 调试，

```
(gdb) run
Starting program: /data/workspace/myshixun/step1/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?
1 2 4 8 16 32
That's number 2.  Keep going!
0 207
Halfway there!
7 0 DrEvil
So you got that one.  Try this one.
9on567
Good work!  On to the next...
4 3 2 1 6 5
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```

进入隐藏关

所以进入隐藏关的方式是**在第四关输入 7 0 DrEvil**;

（2）分析<secret_phase>

```
0000000000401242 <secret_phase>:
  401242: 53                    push   %rbx
  401243: e8 56 02 00 00        callq  40149e <read_line>   //读取输入行
  401248: ba 0a 00 00 00        mov    $0xa,%edx       //%edx=10
  40124d: be 00 00 00 00        mov    $0x0,%esi       //%esi=0
  401252: 48 89 c7              mov    %rax,%rdi       //%rdi=%rax
  401255: e8 76 f9 ff ff        callq  400bd0 <strtol@plt>
  40125a: 48 89 c3              mov    %rax,%rbx       //%rbx输入值
  40125d: 8d 40 ff              lea    -0x1(%rax),%eax   //%rax-1
  401260: 3d e8 03 00 00        cmp    $0x3e8,%eax       //%eax 与 1000比较
  401265: 76 05                 jbe    40126c <secret_phase+0x2a>  //输入值-1 小于等于1000时，跳转到0x40126c
  401267: e8 ce 01 00 00        callq  40143a <explode_bomb>     //否则爆炸
  40126c: 89 de                 mov    %ebx,%esi       //%esi=%ebx=输入值
  40126e: bf f0 30 60 00        mov    $0x6030f0,%edi    //%edi=0x6030f0
  401273: e8 8c ff ff ff        callq  401204 <fun7>   //调用函数fun7
  401278: 83 f8 02              cmp    $0x2,%eax      //返回值与2比，相等就返回
  40127b: 74 05                 je     401282 <secret_phase+0x40>
  40127d: e8 b8 01 00 00        callq  40143a <explode_bomb>
  401282: bf 38 24 40 00        mov    $0x402438,%edi
  401287: e8 84 f8 ff ff        callq  400b10 <puts@plt>
  40128c: e8 33 03 00 00        callq  4015c4 <phase_defused>
  401291: 5b                    pop    %rbx
  401292: c3                    retq
  401293: 90                    nop
  401294: 90                    nop
  401295: 90                    nop
  401296: 90                    nop
```

输入值应该小于1001

调用fun7

fun7的返回值是2时，成功解除炸弹

使用 gdb 查看 0x6030f0 的地址

```
(gdb) x/x 0x6030f0
0x6030f0 <n1>:  0x00000024
```

仅根据地址 0x6030f0 无法发现什么，先放置一边，查看 fun7 函数

查看 fun7 的汇编代码

```
0000000000401204 <fun7>:
  401204: 48 83 ec 08          sub    $0x8,%rsp     //栈指针下移8个字节
  401208: 48 85 ff             test   %rdi,%rdi
  40120b: 74 2b                je     401238 <fun7+0x34>   //判断是否为0, =0跳转0x401238
  40120d: 8b 17                mov    (%rdi),%edx       //将%rdi的地址存放值赋给%edx
  40120f: 39 f2                cmp    %esi,%edx         //与输入值进行比较
  401211: 7e 0d                jle    401220 <fun7+0x1c>   //小于等于时, 跳转0x401220
  401213: 48 8b 7f 08          mov    0x8(%rdi),%rdi
  401217: e8 e8 ff ff ff       callq  401204 <fun7>
  40121c: 01 c0                add    %eax,%eax
  40121e: eb 1d                jmp    40123d <fun7+0x39>

  401220: b8 00 00 00 00       mov    $0x0,%eax     //%eax=0
  401225: 39 f2                cmp    %esi,%edx     //与输入值进行比较
  401227: 74 14                je     40123d <fun7+0x39>   //=时, 跳转0x40123d ,直接返回eax=0
  401229: 48 8b 7f 10          mov    0x10(%rdi),%rdi      //<时, 将%rdi+16的地址 存放值赋给%rdi
  40122d: e8 d2 ff ff ff       callq  401204 <fun7>     //调用fun7函数
  401232: 8d 44 00 01          lea    0x1(%rax,%rax,1),%eax   //%eax=2*%rax+1
  401236: eb 05                jmp    40123d <fun7+0x39>

  401238: b8 ff ff ff ff       mov    $0xffffffff,%eax   //%eax=0xffffffff
  40123d: 48 83 c4 08          add    $0x8,%rsp
  401241: c3                   retq
```

将其转化为相应的 C 代码

```
fun7(int x, *p)  //*p起始值为 (0x6030f0)
{
    if(p=null)
       return -1; //0xffffffff=-1
   int y=p->data;
    if(y<=x)
    {
        result=0;
        if(y=x)
        {
            return 0;
        }
        if(y<x)
        {
           p=p+16;
            return 1+2*fun7(x,*p);
        }
    }
    p=p+8;
    if(y>x)
    {
       return 2*fun7(x,*p);
    }
}

返回值为2时符合题意;
```

通过对 c 代码的分析，发现 fun7 是一个递归函数，递归出口是*p=0，指针为空所以就要结合 fun7 来查看 0x6030f0 及其之后的值

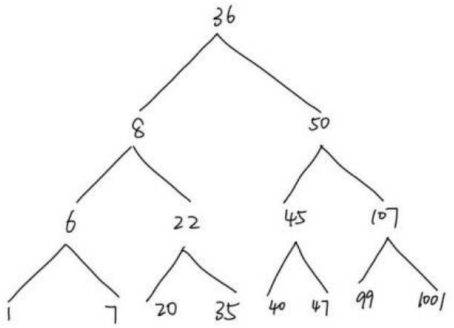
```
(gdb) x/x 0x6030f0
0x6030f0 <n1>:  0x00000024
(gdb) x/24x 0x6030f0
0x6030f0 <n1>:  0x00000024      0x00000000      0x00603110      0x00000000
0x603100 <n1+16>:       0x00603130      0x00000000      0x00000000      0x00000000
0x603110 <n21>: 0x00000008      0x00000000      0x00603190      0x00000000
0x603120 <n21+16>:      0x00603150      0x00000000      0x00000000      0x00000000
0x603130 <n22>: 0x00000032      0x00000000      0x00603170      0x00000000
0x603140 <n22+16>:      0x006031b0      0x00000000      0x00000000      0x00000000
```

```
(gdb) x/128x 0x6030f0
0x6030f0 <n1>:  0x00000024      0x00000000      0x00603110      0x00000000
0x603100 <n1+16>:       0x00603130      0x00000000      0x00000000      0x00000000
0x603110 <n21>: 0x00000008      0x00000000      0x00603190      0x00000000
0x603120 <n21+16>:      0x00603150      0x00000000      0x00000000      0x00000000
0x603130 <n22>: 0x00000032      0x00000000      0x00603170      0x00000000
0x603140 <n22+16>:      0x006031b0      0x00000000      0x00000000      0x00000000
0x603150 <n32>: 0x00000016      0x00000000      0x00603270      0x00000000
0x603160 <n32+16>:      0x00603230      0x00000000      0x00000000      0x00000000
0x603170 <n33>: 0x0000002d      0x00000000      0x006031d0      0x00000000
0x603180 <n33+16>:      0x00603290      0x00000000      0x00000000      0x00000000
0x603190 <n31>: 0x00000006      0x00000000      0x006031f0      0x00000000
0x6031a0 <n31+16>:      0x00603250      0x00000000      0x00000000      0x00000000
0x6031b0 <n34>: 0x0000006b      0x00000000      0x00603210      0x00000000
0x6031c0 <n34+16>:      0x006032b0      0x00000000      0x00000000      0x00000000
0x6031d0 <n45>: 0x00000028      0x00000000      0x00000000      0x00000000
0x6031e0 <n45+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x6031f0 <n41>: 0x00000001      0x00000000      0x00000000      0x00000000
0x603200 <n41+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x603210 <n47>: 0x00000063      0x00000000      0x00000000      0x00000000
0x603220 <n47+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x603230 <n44>: 0x00000023      0x00000000      0x00000000      0x00000000
0x603240 <n44+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x603250 <n42>: 0x00000007      0x00000000      0x00000000      0x00000000
0x603260 <n42+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x603270 <n43>: 0x00000014      0x00000000      0x00000000      0x00000000
0x603280 <n43+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x603290 <n46>: 0x0000002f      0x00000000      0x00000000      0x00000000
```

```
0x6032a0 <n46+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x6032b0 <n48>: 0x000003e9      0x00000000      0x00000000      0x00000000
0x6032c0 <n48+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x6032d0 <node1>:       0x0000014c      0x00000001      0x006032e0      0x00000000
---Type <return> to continue, or q <return> to quit---c
0x6032e0 <node2>:       0x000000a8      0x00000002      0x006032f0      0x00000000
```

发现从 0x6030f0-0x6032c0 是相同的结构

在 fun7 中指针*p 的变化有+8，+16，结合相关知识发现这是一个二叉树



所以指针初始值为 36，若输入值等于 36，则返回 0，若小于 36，则指针+8，结合上图指向 0x603110,0x603110 存储的是 8，若大于 36，则指针+16，结合上图指向 0x603100,0x603100 存放的值是 0x603130，此时调用 fun7，递归，则 p=0x603130，*p=50，

所以结合图示可以理解为 p+8 指向该结点的左子女,p+16 指向该结点的右子女，递归结束条件是指针为空，或者输入值恰好等于该结点；

分析 fun7 时发现，如果输入值不为结点值时，返回的值肯定不为 2；

所以代入已有的 15 个结点值，发现 20，22 返回值为 2，符合题意；

所以输入值为 20 或者 22

（3）运行

```
Curses, you've found the secret phase!
But finding it and solving it are quite different...
20
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
root@evassh-3151774:/data/workspace/myshixun/step1#
```

```
Curses, you've found the secret phase!
But finding it and solving it are quite different...
22
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
root@evassh-3151774:/data/workspace/myshixun/step1#
```

（在命令行 20，22 都能通过，提交代码文件时只有 20 能通过）

（4）代码文件

```c
#include<stdio.h>

void main(){
    /********** Begin **********/
    printf("20");
    /********** End **********/
}

```