phase_1

在gdb中输入

```
disas phase_1
```

得到

```
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>: sub    $0x8,%rsp
   0x0000000000400ee4 <+4>: mov    $0x402400,%esi
   0x0000000000400ee9 <+9>: callq  0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   %eax,%eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    callq  0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    $0x8,%rsp
   0x0000000000400efb <+27>:    retq
End of assembler dump.
```

然后

```
printf "%s", 0x402400
```

即可得到所需字符串。

phase_2

根据 0x0000000000401480 <+36>: mov $0x4025c3,%esi，尝试打印 printf "%s", 0x4025c3 ， 发现改地址起是一个字符串“%d %d %d %d %d %d”，再加上

0x0000000000400f05 <+9>: callq 0x40145c <read_six_numbers>

可知phase_2需要读入以空格间隔的6个数字；

然后，由disas phase_2 得到的汇编代码：

```
Dump of assembler code for function phase_2:

=> 0x0000000000400efc <+0>:     push   %rbp
   0x0000000000400efd <+1>:     push   %rbx
   0x0000000000400efe <+2>:     sub    $0x28,%rsp                    // p = p - 40;
   0x0000000000400f02 <+6>:     mov    %rsp,%rsi                     // int * x = p;
   0x0000000000400f05 <+9>:     callq  0x40145c <read_six_numbers>
   0x0000000000400f0a <+14>:        cmpl   $0x1,(%rsp)                   // if ( 1 == *p )
   0x0000000000400f0e <+18>:        je     0x400f30 <phase_2+52>             //  goto: 52
   0x0000000000400f10 <+20>:        callq  0x40143a <explode_bomb>       // else {BOOM! ;

   0x0000000000400f15 <+25>:        jmp    0x400f30 <phase_2+52>             //  goto: 52 }
```

```
   0x0000000000400f17 <+27>:    mov    -0x4(%rbx),%eax        // t = y - 4;
   0x0000000000400f1a <+30>:    add    %eax,%eax                  // t = t*2;
   0x0000000000400f1c <+32>:    cmp    %eax,(%rbx)            // if (t == *y)
   0x0000000000400f1e <+34>:    je     0x400f25 <phase_2+41>      // goto: 41
   0x0000000000400f20 <+36>:    callq  0x40143a <explode_bomb>  // else BOOM!
   0x0000000000400f25 <+41>:    add    $0x4,%rbx                  // y = y + 4;
   0x0000000000400f29 <+45>:    cmp    %rbp,%rbx                  // if ( y != z )
   0x0000000000400f2c <+48>:    jne    0x400f17 <phase_2+27>      //  goto: 27
   0x0000000000400f2e <+50>:    jmp    0x400f3c <phase_2+64>      // goto: 64
   0x0000000000400f30 <+52>:    lea    0x4(%rsp),%rbx             // y = p+4;
   0x0000000000400f35 <+57>:    lea    0x18(%rsp),%rbp        // z = p+24;
   0x0000000000400f3a <+62>:    jmp    0x400f17 <phase_2+27>      // goto: 27
   0x0000000000400f3c <+64>:    add    $0x28,%rsp             // p = p + 40;
   0x0000000000400f40 <+68>:    pop    %rbx
   0x0000000000400f41 <+69>:    pop    %rbp
   0x0000000000400f42 <+70>:    retq
```

推出伪代码如下：

```
void fun() {
    char* p,* y,* z, t;
    if (1 == *p) {
        y = p + 4;
        z = p + 24;
        for (;y != z;) {
            t = (y - 4) * 2;
            if (t == *y) {
                y = y + 4;
            }
            else {
                boom();
            }
        }
    }
    else {
        boom();
    }
}
```

由此可知，其大意为，第一个数是1且后一个数是前一个数的两倍即可。

phase_3

根据

```
<+29>:  mov     $0x1,%eax
<+32>:  jg      0x400f6a <phase_3+39>
<+34>:  callq   0x40143a <explode_bomb>
```

可知，必须输入至少两个数。

又由

```
<+39>:  cmpl   $0x7,0x8(%rsp)
<+44>:  ja     0x400fad <phase_3+106>
<+106>: callq  0x40143a <explode_bomb>
```

可知，其中一个数要小于等于7。

当第一个数为6时，代码会执行到

```
<+92>:  mov    $0x2aa,%eax
<+97>:  jmp    0x400fbe <phase_3+123>
<+123>: cmp    0xc(%rsp),%eax
```

此时，需要第二个数等于0x2aa即可。

phase_4

```
disas phase_4
```

得

```
Dump of assembler code for function phase_4:
   0x000000000040100c <+0>:     sub    $0x18,%rsp
   0x0000000000401010 <+4>:     lea    0xc(%rsp),%rcx
   0x0000000000401015 <+9>:     lea    0x8(%rsp),%rdx
   0x000000000040101a <+14>:    mov    $0x4025cf,%esi
   0x000000000040101f <+19>:    mov    $0x0,%eax
   0x0000000000401024 <+24>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x0000000000401029 <+29>:    cmp    $0x2,%eax
   0x000000000040102c <+32>:    jne    0x401035 <phase_4+41>
   0x000000000040102e <+34>:    cmpl   $0xe,0x8(%rsp)
   0x0000000000401033 <+39>:    jbe    0x40103a <phase_4+46>
   0x0000000000401035 <+41>:    callq  0x40143a <explode_bomb>
   0x000000000040103a <+46>:    mov    $0xe,%edx
   0x000000000040103f <+51>:    mov    $0x0,%esi
   0x0000000000401044 <+56>:    mov    0x8(%rsp),%edi
   0x0000000000401048 <+60>:    callq  0x400fce <func4>
   0x000000000040104d <+65>:    test   %eax,%eax
   0x000000000040104f <+67>:    jne    0x401058 <phase_4+76>
   0x0000000000401051 <+69>:    cmpl   $0x0,0xc(%rsp)

   0x0000000000401056 <+74>:    je     0x40105d <phase_4+81>
```

```
   0x0000000000401058 <+76>:    callq  0x40143a <explode_bomb>
   0x000000000040105d <+81>:    add    $0x18,%rsp
   0x0000000000401061 <+85>:    retq
End of assembler dump.
```

由

```
0x0000000000401029 <+29>:    cmp    $0x2,%eax
0x000000000040102c <+32>:    jne    0x401035 <phase_4+41
```

知，需输入恰好两个值。

发现，其中调用了一个函数func4()

```
disas func4
```

得到

```
Dump of assembler code for function func4:
   0x0000000000400fce <+0>:     sub    $0x8,%rsp               // x in %edx, y in %esi
   0x0000000000400fd2 <+4>:     mov    %edx,%eax               // ret = x;
   0x0000000000400fd4 <+6>:     sub    %esi,%eax               // ret = ret - y;
   0x0000000000400fd6 <+8>:     mov    %eax,%ecx               // z = ret;
   0x0000000000400fd8 <+10>:    shr    $0x1f,%ecx              // z = (z >> 31) & 0x00000001;
   0x0000000000400fdb <+13>:    add    %ecx,%eax               // ret = ret + z;
   0x0000000000400fdd <+15>:    sar    %eax                    // ret = ret/2;
   0x0000000000400fdf <+17>:    lea    (%rax,%rsi,1),%ecx      // z = ret + y;
   0x0000000000400fe2 <+20>:    cmp    %edi,%ecx               // if (z <= input1)
   0x0000000000400fe4 <+22>:    jle    0x400ff2 <func4+36>     // goto: +36;
   0x0000000000400fe6 <+24>:    lea    -0x1(%rcx),%edx         // x = z - 1;
   0x0000000000400fe9 <+27>:    callq  0x400fce <func4>        // ret = func4;
   0x0000000000400fee <+32>:    add    %eax,%eax               // ret = ret * 2;
   0x0000000000400ff0 <+34>:    jmp    0x401007 <func4+57>     // goto: +57;
   0x0000000000400ff2 <+36>:    mov    $0x0,%eax               // ret = 0;
   0x0000000000400ff7 <+41>:    cmp    %edi,%ecx               // if (z >= input1)
   0x0000000000400ff9 <+43>:    jge    0x401007 <func4+57>     // goto: +57;
   0x0000000000400ffb <+45>:    lea    0x1(%rcx),%esi          // y = z + 1;
   0x0000000000400ffe <+48>:    callq  0x400fce <func4>        // ret = func4;
   0x0000000000401003 <+53>:    lea    0x1(%rax,%rax,1),%eax   // ret = ret * 2 + 1;
   0x0000000000401007 <+57>:    add    $0x8,%rsp
   0x000000000040100b <+61>:    retq
End of assembler dump.
```

由

```
0x000000000040103a <+46>:    mov    $0xe,%edx
0x000000000040103f <+51>:    mov    $0x0,%esi
```

知，第一次调用func4()时，x = 14, y = 0.

观察发现当z == input1时，func4()函数不需要递归调用并可安全执行到尾，而z的值为7，故第一个输入值为7，

又由

```
0x0000000000401051 <+69>:    cmpl   $0x0,0xc(%rsp)
0x0000000000401056 <+74>:    je     0x40105d <phase_4+81
```

知，第二个输入值需为0.

phase_5

密码表：

a	b	c	d	e	f	g	h	i	j
a	d	u	i	e	r	s	n	f	o

k	l	m	n	o	p	q	r	s	t
t	v	b	y	l	m			u	i

u	v	w	x	y	z				

要匹配的字符串为flyers，由密码表可知ionefg为原字符串。

phase_6

太难。