bomblab 报告

姓名：赵梓名　　学号：2024201617

总分	phase_1	phase_2	phase_3	phase_4	phase_5	phase_6	secret_phase
3	1	1	1	1	1	0	0

scoreboard 截图：

解题报告

phase_1

```
    1439:   48 8d 35 40 1d 00 00    lea    0x1d40(%rip),%rsi
    1440:   e8 41 08 00 00          call   1c86 <strings_not_equal>
    1445:   85 c0                   test   %eax,%eax
    1447:   75 05                   jne    144e <phase_1+0x19>
    144e:   e8 98 0a 00 00          call   1eeb <explode_bomb>
```

为不跳转到 `144e` 行使得炸弹引爆，在 `1445` 行比较中，需输入的字符串与 `0x1d40(%rip)` 一致。访问该处：

```
   0x0000555555555439 <+4>:     lea    0x1d40(%rip),%rsi        # 0x555555557180
(gdb) x/s 0x555555557180
0x555555557180: "But I am so still here surrender the silence..."
```

可见 `phase_2` 答案为：**But I am so still here surrender the silence...**

---

phase_2

```
int matA[2][3] = {966, 649, 371, 194, 951, 493};
int matB[3][2] = {800, 617, 237, 278, 866, 223};    // 外部矩阵定义

void phase_2(char *input) {
    // 读取并检验
    int a, b, c, d;     int expected[4] = {a, b, c, d};
    if (sscanf(input, "%d %d %d %d", &a, &b, &c, &d) != 4)  explode_bomb();

    // 计算并验证结果
    int result_matrix[2][2] = matA \cdot matB
    if (result_flat[i] != expected[i])  explode_bomb();
}
```

程序实现了两个矩阵的乘法，要求我们正确输入结果。如何得知矩阵的初始值？注意到以下两行：

```
   0x000055555555548e <+57>:    lea    0x4c9b(%rip),%rdi        # 0x55555555a130 <matA.2>
   0x00005555555554bb <+102>:   lea    0x4c4e(%rip),%rsi        # 0x55555555a110 <matB.1>
```

于是：

```
(gdb) x/8wd 0x55555555a130
0x55555555a130 <matA.2>:        966     649     371     194
0x55555555a140 <matA.2+16>:     951     493     0       0
(gdb) x/8wd 0x55555555a110
0x55555555a110 <matB.1>:        800     617     237     278
0x55555555a120 <matB.1+16>:     866     223     0       0
```

通过计算，本题答案是：**1247899　859177　807525　494015**

---

phase_3

```
void phase_3(char *input) {
    // 读取并检验
    int a, b, result;
    if (sscanf(input, "%d %d", &a, &b) <= 1)    explode_bomb();
    if (b >= 0 || a > 7)    explode_bomb();

    // 计算并验证结果
    switch (a) {
        case 0: case 1: case 2: case 3: explode_bomb(); break;
        case 4: case 6: result = 0; break;
        case 5: case 7: result = -542;  break;
        default: explode_bomb();
    }
    if (a > 5 || b != result)   explode_bomb();
}
```

解为：**5　-542**。由于 `switch` 表，可行的 `a` 只能为 `4` 或 `5`，由于 `b` 负数的限制，只有以上唯一解。

这道题 `jump` 语句较多，需灵活判断怎样的取值是安全的。

---

phase_4

```
int func4_1(int n) {    // n 在 %rdi 中
    if (n <= 0) return 0;
    if (n == 1) return 1;
    return 2 * func4_1(n - 1) + 1;  // 以 %eax 返回
}
```

```
void func4_2(int n, int ebx, char r12, char r13, char r14, char *output) {
    // 基础情况：第一步
    if (n == 1) {output[0] = r12;    output[1] = r13;    output[2] = '\0';    return;}

    int steps = func4_1(n - 1);  // 2^(n-1) - 1
    // 递归处理 n-1
    if (steps >= ebx) func4_2(n-1, ebx, r14, r13, r12, output);
    // 刚好第 ebx 步
    else if (steps + 1 == ebx) {output[0] = r12;    output[1] = r13;    output[2] = '\0';}
    // 跳过前 steps+1 步
    else {ebx = ebx - steps - 1;    func4_2(n-1, ebx, r13, r12, r14, output);}
}
```

```
void phase2(char *input) {
    // 读取并检验
    int num;    char steps[3];  char expected[3];
    if (sscanf(input, "%d %s", &a, &b) != 2)    explode_bomb();
    if (a != func4_1(5) || strlen(b) != 2)      explode_bomb();

    // 计算并验证结果
    func4_2(5, 12, 'A', 'B', 'C', expected);
    if (strcmp(steps, expected) != 0)   explode_bomb();
}
```

简言之，`func4_1` 通过递归，计算 `n` 阶 `Hanoi Tower` 的最少步数；`func4_2` 同样使用递归，计算此时某一步的移动方法。而主函数负责检验两个输入参数，第一个必须是 `5` 阶 `Hanoi Tower` 的最少移动步数 `31`，第二个是第 `12` 步的移动方法 `CB`，这样才能避免爆炸。因此，答案为：**31 CB**

---

phase_5

```
void phase_5(char *input) {
    // 读取并检验
    int a, b;
    if (sscanf(input, "%d %d", &a, &b) <= 1)    explode_bomb();
    if (a >= 0) explode_bomb();
    int index = a & 0xf;
    if (index == 0xf)   explode_bomb();

    // 计算并验证结果
    int sum = 0, count = 0, cur_value = 0, array[16] = [<array>];
    do {
        cur_value = array[index];
        sum += cur_value;   count++;
        index = (unsigned char)cur_value;
    } while (cur_value != 0xf);
    if (count != 7 || b != sum) explode_bomb();
}
```

这题比较有趣，观察到编码了一个数列（更像链表），访问之：

```
   0x0000555555555800 <+80>:    lea    0x1a39(%rip),%rsi        # 0x555555557240 <array.0>
(gdb) x/16d 0x555555557240
0x555555557240 <array.0>:       10      2       14      7
0x555555557250 <array.0+16>:    8       12      15      11
0x555555557260 <array.0+32>:    0       4       1       13
0x555555557270 <array.0+48>:    3       9       6       5
```

得到 `array[16] = [10, 2, ... , 5]` 。接下来我们需要找到一个参数 `index` ，使得

```
起点：index = 8
├─→ array[8]  = 0    （值是0，下一个index = 0）
|
├─→ array[0]  = 10   （值是10，下一个index = 10）
|
├─→ array[10] = 1    （值是1，下一个index = 1）
|
├─→ array[1]  = 2    （值是2，下一个index = 2）
|
├─→ array[2]  = 14   （值是14，下一个index = 14）
|
├─→ array[14] = 6    （值是6，下一个index = 6）
|
├─→ array[6]  = 15   （值是15 = 0xf，**停止！**）
|
└─ 累加：0+10+1+2+14+6+15 = 48  ✅
   循环次数：7 次  ✅
```

---

phase_6

```
Dump of assembler code for function read_six_numbers:
   0x00005555555555fab <+0>:     sub    $0x8,%rsp
   0x00005555555555faf <+4>:     mov    %rsi,%rdx
   0x00005555555555fb2 <+7>:     lea    0x4(%rsi),%rcx
   0x00005555555555fb6 <+11>:    lea    0x14(%rsi),%rax
   0x00005555555555fba <+15>:    push   %rax
   0x00005555555555fbb <+16>:    lea    0x10(%rsi),%rax
   0x00005555555555fbf <+20>:    push   %rax
   0x00005555555555fc0 <+21>:    lea    0xc(%rsi),%r9
   0x00005555555555fc4 <+25>:    lea    0x8(%rsi),%r8
   0x00005555555555fc8 <+29>:    lea    0x15fc(%rip),%rsi        # 0x5555555575cb
   0x00005555555555fcf <+36>:    mov    $0x0,%eax
   0x00005555555555fd4 <+41>:    call   0x555555555150 <__isoc99_sscanf@plt>
   0x00005555555555fd9 <+46>:    add    $0x10,%rsp
```

```
   0x0000555555555fdd <+50>:    cmp    $0x5,%eax
   0x0000555555555fe0 <+53>:    jle    0x555555555fe7 <read_six_numbers+60>
   0x0000555555555fe2 <+55>:    add    $0x8,%rsp
   0x0000555555555fe6 <+59>:    ret
   0x0000555555555fe7 <+60>:    call   0x555555555eeb <explode_bomb>
End of assembler dump.
```

```
Dump of assembler code for function phase_6:
   0x0000555555555847 <+0>:     push   %r14
   0x0000555555555849 <+2>:     push   %r13
   0x000055555555584b <+4>:     push   %r12
   0x000055555555584d <+6>:     push   %rbp
   0x000055555555584e <+7>:     push   %rbx
   0x000055555555584f <+8>:     sub    $0x60,%rsp
   0x0000555555555853 <+12>:    mov    %fs:0x28,%rax
   0x000055555555585c <+21>:    mov    %rax,0x58(%rsp)
   0x0000555555555861 <+26>:    xor    %eax,%eax
   0x0000555555555863 <+28>:    mov    %rsp,%r13
   0x0000555555555866 <+31>:    mov    %r13,%rsi
*  0x0000555555555869 <+34>:    call   0x555555555fab <read_six_numbers>
   0x000055555555586e <+39>:    mov    $0x1,%r14d
   0x0000555555555874 <+45>:    mov    %rsp,%r12
y  0x0000555555555877 <+48>:    jmp    0x5555555558a1 <phase_6+90>
!  0x0000555555555879 <+50>:    call   0x555555555eeb <explode_bomb>
   0x000055555555587e <+55>:    jmp    0x5555555558b0 <phase_6+105>
   0x0000555555555880 <+57>:    add    $0x1,%rbx
   0x0000555555555884 <+61>:    cmp    $0x5,%ebx
   0x0000555555555887 <+64>:    jg     0x555555555899 <phase_6+82>

   0x0000555555555889 <+66>:    mov    (%r12,%rbx,4),%eax
   0x000055555555588d <+70>:    cmp    %eax,0x0(%rbp)
y  0x0000555555555890 <+73>:    jne    0x555555555880 <phase_6+57>
!  0x0000555555555892 <+75>:    call   0x555555555eeb <explode_bomb>
   0x0000555555555897 <+80>:    jmp    0x555555555880 <phase_6+57>
   0x0000555555555899 <+82>:    add    $0x1,%r14
   0x000055555555589d <+86>:    add    $0x4,%r13
   0x00005555555558a1 <+90>:    mov    %r13,%rbp
   0x00005555555558a4 <+93>:    mov    0x0(%r13),%eax
   0x00005555555558a8 <+97>:    sub    $0x1,%eax
   0x00005555555558ab <+100>:   cmp    $0x5,%eax
n  0x00005555555558ae <+103>:   ja     0x555555555879 <phase_6+50>
   0x00005555555558b0 <+105>:   cmp    $0x5,%r14d
e  0x00005555555558b4 <+109>:   jg     0x5555555558bb <phase_6+116>
   0x00005555555558b6 <+111>:   mov    %r14,%rbx
   0x00005555555558b9 <+114>:   jmp    0x555555555889 <phase_6+66>

   0x00005555555558bb <+116>:   mov    $0x0,%esi
   0x00005555555558c0 <+121>:   mov    (%rsp,%rsi,4),%ecx
   0x00005555555558c3 <+124>:   mov    $0x1,%eax
   0x00005555555558c8 <+129>:   lea    0x4941(%rip),%rdx        # 0x55555555a210 <node1>
   0x00005555555558cf <+136>:   cmp    $0x1,%ecx
```

```
   0x00005555555558d2 <+139>:   jle    0x5555555558df <phase_6+152>

   0x00005555555558d4 <+141>:   mov    0x8(%rdx),%rdx
   0x00005555555558d8 <+145>:   add    $0x1,%eax
   0x00005555555558db <+148>:   cmp    %ecx,%eax
   0x00005555555558dd <+150>:   jne    0x5555555558d4 <phase_6+141>

   0x00005555555558df <+152>:   mov    %rdx,0x20(%rsp,%rsi,8)
   0x00005555555558e4 <+157>:   add    $0x1,%rsi
   0x00005555555558e8 <+161>:   cmp    $0x6,%rsi
   0x00005555555558ec <+165>:   jne    0x5555555558c0 <phase_6+121>

   0x00005555555558ee <+167>:   mov    0x20(%rsp),%rbx
   0x00005555555558f3 <+172>:   mov    0x28(%rsp),%rax
   0x00005555555558f8 <+177>:   mov    %rax,0x8(%rbx)
   0x00005555555558fc <+181>:   mov    0x30(%rsp),%rdx
   0x0000555555555901 <+186>:   mov    %rdx,0x8(%rax)
   0x0000555555555905 <+190>:   mov    0x38(%rsp),%rax
   0x000055555555590a <+195>:   mov    %rax,0x8(%rdx)
   0x000055555555590e <+199>:   mov    0x40(%rsp),%rdx
   0x0000555555555913 <+204>:   mov    %rdx,0x8(%rax)
   0x0000555555555917 <+208>:   mov    0x48(%rsp),%rax
   0x000055555555591c <+213>:   mov    %rax,0x8(%rdx)
   0x0000555555555920 <+217>:   movq   $0x0,0x8(%rax)
   0x0000555555555928 <+225>:   mov    $0x5,%ebp
   0x000055555555592d <+230>:   jmp    0x555555555938 <phase_6+241>
   0x000055555555592f <+232>:   mov    0x8(%rbx),%rbx
   0x0000555555555933 <+236>:   sub    $0x1,%ebp
f  0x0000555555555936 <+239>:    je     0x555555555949 <phase_6+258>

   0x0000555555555938 <+241>:   mov    0x8(%rbx),%rax
   0x000055555555593c <+245>:   mov    (%rax),%eax
   0x000055555555593e <+247>:   cmp    %eax,(%rbx)
r  0x0000555555555940 <+249>:    jle    0x55555555592f <phase_6+232>
!  0x0000555555555942 <+251>:    call   0x555555555eeb <explode_bomb>
   0x0000555555555947 <+256>:   jmp    0x55555555592f <phase_6+232>
   0x0000555555555949 <+258>:   mov    0x58(%rsp),%rax
   0x000055555555594e <+263>:   sub    %fs:0x28,%rax
f  0x0000555555555957 <+272>:    jne    0x555555555966 <phase_6+287>
   0x0000555555555959 <+274>:   add    $0x60,%rsp
   0x000055555555595d <+278>:   pop    %rbx
   0x000055555555595e <+279>:   pop    %rbp
   0x000055555555595f <+280>:   pop    %r12
   0x0000555555555961 <+282>:   pop    %r13
   0x0000555555555963 <+284>:   pop    %r14
   0x0000555555555965 <+286>:   ret
   0x0000555555555966 <+287>:   call   0x5555555550a0 <__stack_chk_fail@plt>
End of assembler dump.
```

```
struct Node {
    int data;
```

```
    int padding;
    struct Node *next;
};

extern struct Node node1, node2, node3, node4, node5, node6;

void phase_6(char *input) {
    // 读取并检验
    int nums[6];
    read_six_numbers(input, nums);
    for (int i = 0; i < 6; i++) {
        if ((unsigned)(nums[i] - 1) > 5)    explode_bomb();
        for (int j = i + 1; j < 6; j++) {if (nums[i] == nums[j])    explode_bomb();}
    }

    // 计算并验证结果
    struct Node *nodes[6], cur;
    for (int i = 0; i < 6; i++) {
        cur = &node1;
        for (int j = 1; j < nums[i]; j++)   cur = cur->next;
        nodes[i] = cur;
    }

    for (int i = 0; i < 5; i++) nodes[i]->next = nodes[i + 1];
    nodes[5]->next = NULL;

    cur = nodes[0];
    for (int i = 0; i < 5; i++) {
        if (cur->data < cur->next->data)    explode_bomb();
        cur = cur->next;
    }
}
```

```
```

---

serect_phase

```
0000000000001b92 <secret_phase>:
```

阅读代码找到了 `serect_phase` 的位置，我们查找得到

```
```

```
0000000000002126 <phase_defused>:
    2126:   48 83 ec 08             sub    $0x8,%rsp
    212a:   bf 01 00 00 00          mov    $0x1,%edi
    212f:   e8 ea fc ff ff          call   1e1e <send_msg>
    2134:   83 3d dd 45 00 00 06    cmpl   $0x6,0x45dd(%rip)        # 6718
<num_input_strings>
    213b:   74 05                   je     2142 <phase_defused+0x1c>
    213d:   48 83 c4 08             add    $0x8,%rsp
    2141:   c3                      ret
    2142:   0f b6 0d 2f 48 00 00    movzbl 0x482f(%rip),%ecx        # 6978
<input_strings+0x258>
    2149:   84 c9                   test   %cl,%cl
    214b:   74 34                   je     2181 <phase_defused+0x5b>
    214d:   b8 01 00 00 00          mov    $0x1,%eax
    2152:   ba 00 00 00 00          mov    $0x0,%edx
    2157:   48 8d 3d 1a 48 00 00    lea    0x481a(%rip),%rdi        # 6978
<input_strings+0x258>
    215e:   80 f9 20                cmp    $0x20,%cl
    2161:   0f 94 c1                sete   %cl
    2164:   0f b6 c9                movzbl %cl,%ecx
    2167:   01 ca                   add    %ecx,%edx
    2169:   89 c6                   mov    %eax,%esi
    216b:   0f b6 0c 07             movzbl (%rdi,%rax,1),%ecx
    216f:   48 83 c0 01             add    $0x1,%rax
    2173:   83 fa 05                cmp    $0x5,%edx
    2176:   7f 04                   jg     217c <phase_defused+0x56>
    2178:   84 c9                   test   %cl,%cl
    217a:   75 e2                   jne    215e <phase_defused+0x38>
    217c:   83 fa 06                cmp    $0x6,%edx
    217f:   74 1a                   je     219b <phase_defused+0x75>
    2181:   48 8d 3d 00 12 00 00    lea    0x1200(%rip),%rdi        # 3388 <array.0+0x148>
    2188:   e8 e3 ee ff ff          call   1070 <puts@plt>
    218d:   48 8d 3d 24 12 00 00    lea    0x1224(%rip),%rdi        # 33b8 <array.0+0x178>
    2194:   e8 d7 ee ff ff          call   1070 <puts@plt>
    2199:   eb a2                   jmp    213d <phase_defused+0x17>
    219b:   48 63 f6                movslq %esi,%rsi
    219e:   48 8d 05 d3 47 00 00    lea    0x47d3(%rip),%rax        # 6978
<input_strings+0x258>
    21a5:   48 8d 3c 06             lea    (%rsi,%rax,1),%rdi
    21a9:   48 8d 35 71 14 00 00    lea    0x1471(%rip),%rsi        # 3621 <array.0+0x3e1>
    21b0:   e8 d1 fa ff ff          call   1c86 <strings_not_equal>
    21b5:   85 c0                   test   %eax,%eax
    21b7:   75 c8                   jne    2181 <phase_defused+0x5b>
    21b9:   48 8d 3d 68 11 00 00    lea    0x1168(%rip),%rdi        # 3328 <array.0+0xe8>
    21c0:   e8 ab ee ff ff          call   1070 <puts@plt>
    21c5:   48 8d 3d 84 11 00 00    lea    0x1184(%rip),%rdi        # 3350 <array.0+0x110>
    21cc:   e8 9f ee ff ff          call   1070 <puts@plt>
    21d1:   b8 00 00 00 00          mov    $0x0,%eax
    21d6:   e8 b7 f9 ff ff          call   1b92 <secret_phase>
    21db:   eb a4                   jmp    2181 <phase_defused+0x5b>
```

反馈/收获/感悟/总结

参考的重要资料