**温州大学计算机与人工智能学院**

**计算机系统综合实践 实验报告**

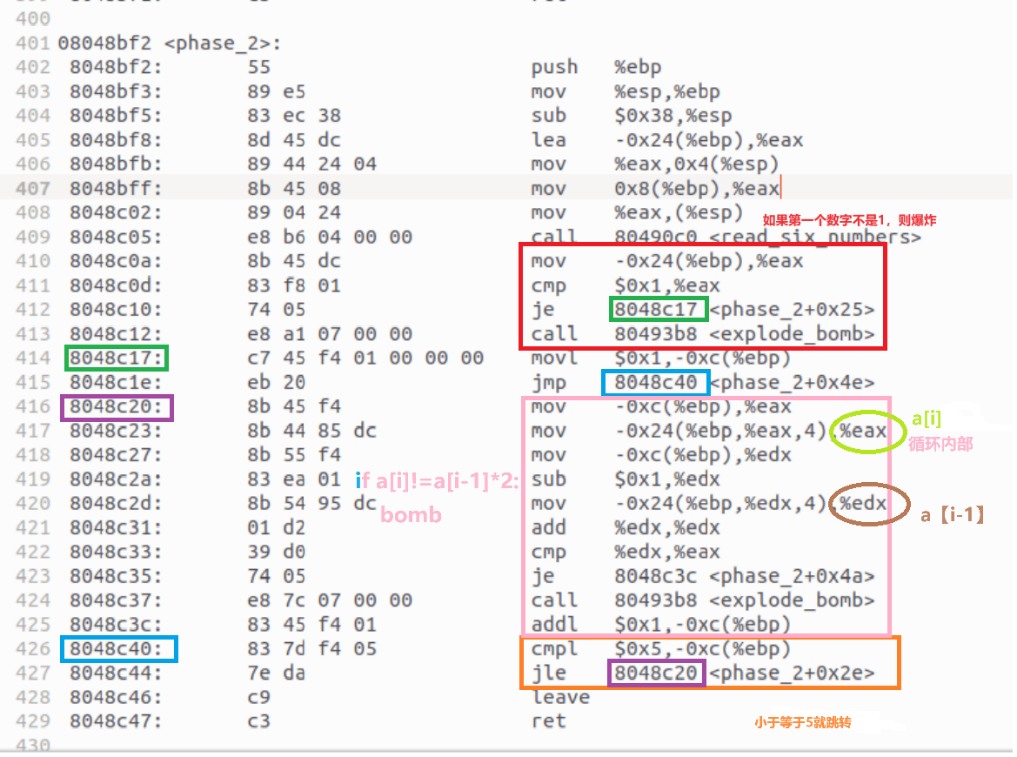
**（phase\_1）**

|  |  |  |  |
| --- | --- | --- | --- |
| phase\_1 |  |  |  |
|  | **地址** | **栈单元内容** |  |
|  | ebp+8 | **输入字符串** |  |
|  | ebp+4 | 返回地址 |  |
|  | ebp | 旧ebp |  |
|  | ebp-4 |  |  |
|  | ebp-8 |  |  |
|  | ebp-c |  |  |
|  | ebp-10 |  |  |
|  | ebp-14 | 0x804a354 | Houses will begat jobs, jobs will begat houses. |
| esp | ebp-18 | ebp+8 | 调用函数时用于传入参数 |

**思路：**

先查找0x804a354中的内容发现是一串字符串，输入该字符串就对了。代码的意思就是把输入的字符串传到strings\_not\_equal中判断和内部的字符串是否相等，不相等就爆炸。

**答案：**Houses will begat jobs, jobs will begat houses.

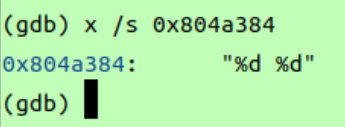
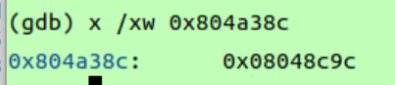
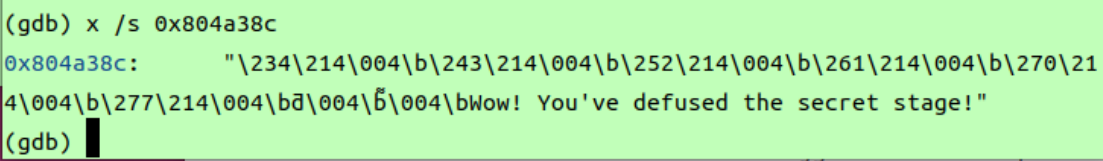
**（phase\_2）**

|  |  |  |  |
| --- | --- | --- | --- |
| phase\_2 |  |  |  |
|  | ebp+8 | **输入信息** |  |
|  | ebp+4 | 返回地址 |  |
|  | ebp | 旧ebp |  |
|  | ebp-4 |  |  |
|  | ebp-8 |  |  |
|  | ebp-c | 循环中的i | i=1,i<=5,i++ |
|  | ebp-10 | 第六个数 |  |
|  | ebp-14 | 第五个数 |  |
|  | ebp-18 | 第四个数 |  |
|  | ebp-1c | 第三个数 |  |
|  | ebp-20 | 第二个数 |  |
|  | ebp-24 | 第一个数 |  |
|  | ebp-28 |  |  |
|  | ebp-2c |  |  |
|  | ebp-30 |  |  |
|  | ebp-34 | ebp-24 |  |
| esp | ebp-38 | ebp+8 | 调用函数时用于传入参数 |

**思路：**

首先利用read\_six\_numbers检验是不是6个数字，不是就爆炸。然后检验第一个数字是不是1，不是1就爆炸。接着是一个**循环**，循环比较如果a[i-1]\*2!=a[i]就会爆炸，遍历六个数字。所以最后的结果是一串以1开头，比例为2的等比数列。

**答案：**1 2 4 8 16 32

******（phase\_3）**

|  |  |  |  |
| --- | --- | --- | --- |
| phase\_3 |  |  |  |
|  |  |  |  |
|  | ebp+8 | 输入信息 |  |
|  | ebp+4 | 返回地址 |  |
|  | ebp | 旧ebp |  |
|  | ebp-4 |  |  |
|  | ebp-8 |  |  |
|  | ebp-c | 初始化0 | switch 语句中的计算结果 |
|  | ebp-10 | 初始化0 | <\_\_isoc99\_sscanf@plt>的返回值 |
|  | ebp-14 | 第一个输入参数 |  |
|  | ebp-18 | 第二个输入参数 |  |
|  | ebp-1c | ebp-18 |  |
|  | ebp-20 | ebp-14 |  |
|  | ebp-24 | 0x804a384 | %d %d |
| esp | ebp-28 | ebp+8 | 调用函数时用于传入参数 |

**思路：**

第三题是**switch** **case**语句。

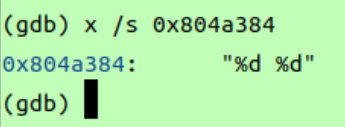
首先利用\_\_isoc99\_sscanf@plt判断输入参数个数，输入个数小于等于1就爆炸（所以其实可以传无数个参数，只要前两个参数符合要求就行）。

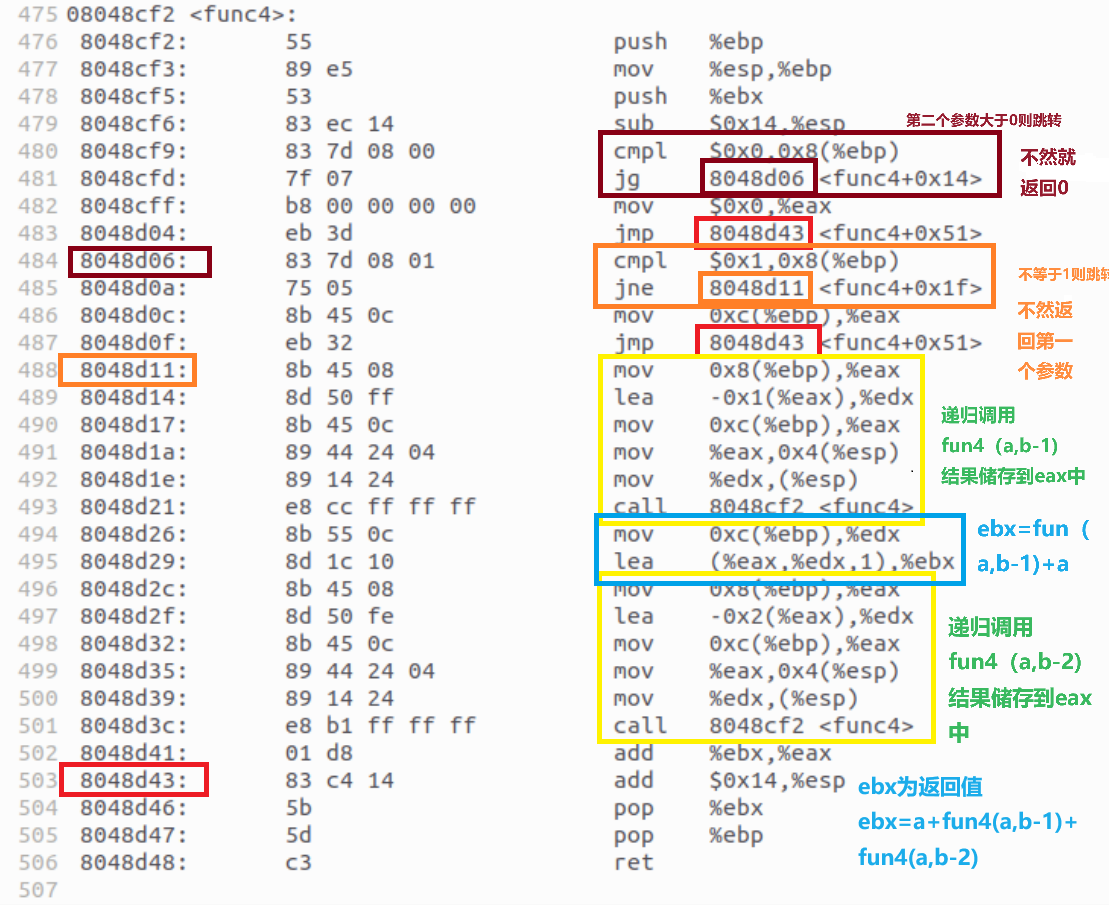
两个地方判断第一个数字大小，一个要求小于等于7，一个要求小于等于5，所以最终是小于等于5，按理说第一个参数可以是负数，但是负数会跳转出选择语句，不能正常进行程序。

进入switch中，会根据第一个参数进入到不同位置，在case中计算ebp-c的结果，ebp-c初始化为0，最终判断ebp-c的结果与输入的第二个参数相等即成功。

|  |  |
| --- | --- |
| 答案合集 |  |
| 0 | 260 |
| 1 | -519 |
| 2 | 98 |
| 3 | -848 |
| 4 | 0 |
| 5 | -848 |

**答案：**

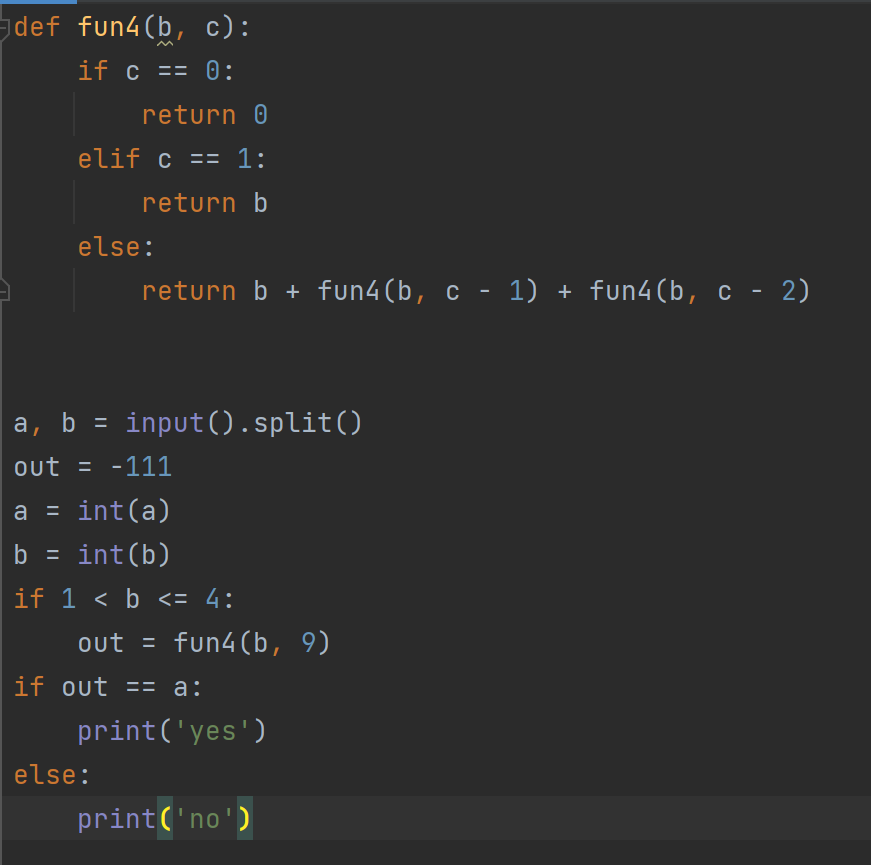
**（phase\_4）**



|  |  |  |
| --- | --- | --- |
| fun4 | ebp+c | 第1个参数，a |
|  | ebp+8 | 第2个参数，b |
|  | ebp+4 |  |
|  | ebp |  |
|  | ebp-4 |  |
|  | ebp-8 |  |
|  | ebp-c |  |
|  | ebp-10 |  |
| Esp | ebp-14 |  |

****

|  |  |  |
| --- | --- | --- |
| phase\_4 | ebp+8 | 输入信息 |
|  | ebp+4 | 返回地址 |
|  | ebp | 旧ebp |
|  | ebp-4 |  |
|  | ebp-8 |  |
|  | ebp-c |  |
|  | ebp-10 | 9 |
|  | ebp-14 | fun4的返回值 |
|  | ebp-18 | 第2个参数 |
|  | ebp-1c | 第1个参数 |
|  | ebp-20 |  |
|  | ebp-24 |  |
|  | ebp-28 |  |
|  | ebp-2c |  |
|  | ebp-30 |  |
|  | ebp-34 | 第2个参数 |
| esp | ebp-38 | ebp-10 |

**第四题用python语言改写后的代码表示：**

**思路：**第四题属于**递归调用**

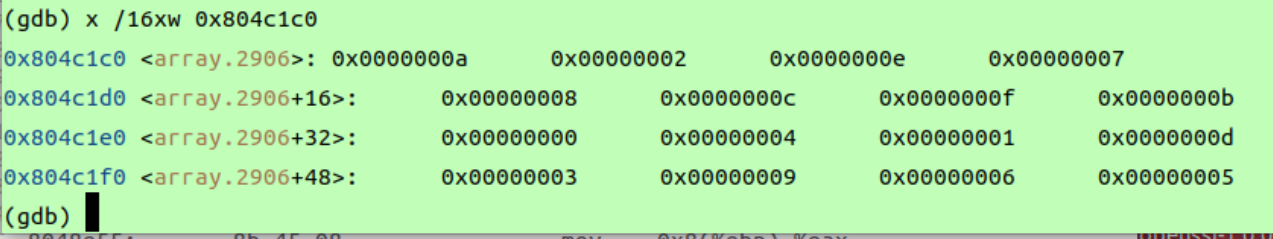
在主函数中，要求第二个数字大于1且小于等于4，传入参数个数不等于2就会爆炸，然后将9作为第二个参数，输入的第二个参数作为第一个参数传入fun4函数中。

在fun4函数中，解读代码就会发现是递归调用，具体内容如**python代码**所示。

最后，将结果返回主函数，比较返回值和传入的第一个参数，如果不一样就爆炸

|  |  |
| --- | --- |
| **答案合集** |  |
| 176 | 2 |
| 264 | 3 |
| 352 | 4 |

**（phase\_5）**

****Map表**

**循环过程展示：**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 12 |  | 3 | 7 | 11 | 13 | 9 | 4 | 8 | 0 | 10 | 1 | 2 | 14 | 6 | 15 |

**

|  |  |  |  |
| --- | --- | --- | --- |
| phase\_5 | ebp+8 | 输入信息 |  |
|  | ebp+4 | 返回地址 |  |
|  | ebp | 旧ebp |  |
|  | ebp-4 |  |  |
|  | ebp-8 |  |  |
| 循环次数标记 | ebp-c | 初始化0 |  |
| 循环累加求和 | ebp-10 | 初始化0 |  |
|  | ebp-14 | 输入参数个数 | <\_\_isoc99\_sscanf@plt>的返回值 |
|  | ebp-18 | 按位与之后的参数1 |  |
|  | ebp-1c | 第1个参数 | 和0xf按位与 |
|  | ebp-20 | 第2个参数 |  |
|  | ebp-24 |  |  |
|  | ebp-28 |  |  |
|  | ebp-2c | 第一个参数 |  |
|  | ebp-30 | 第二个参数 |  |
|  | ebp-34 |  |  |
| esp | ebp-38 |  |  |

**思路：**第五题属于**列表循环**

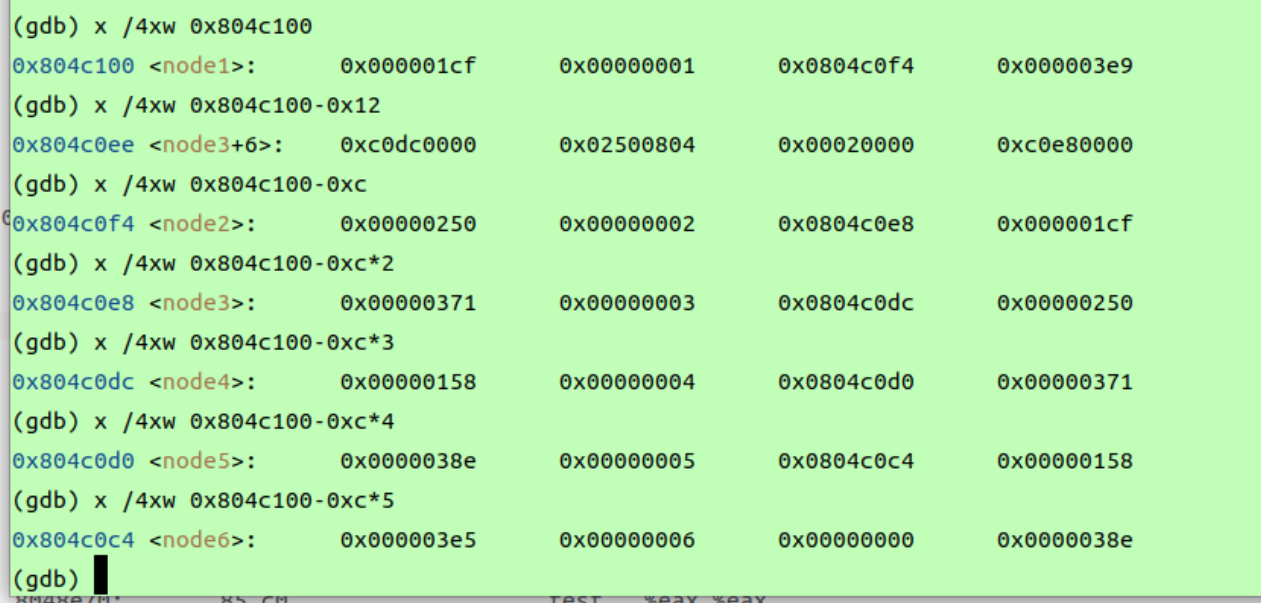
遇到绝对地址就先查询，查询后发现0x804c1c0中存的是一个1到16之间的列表，第一、二两个框中判断输入参数的个数，如果小于等于1就爆炸。

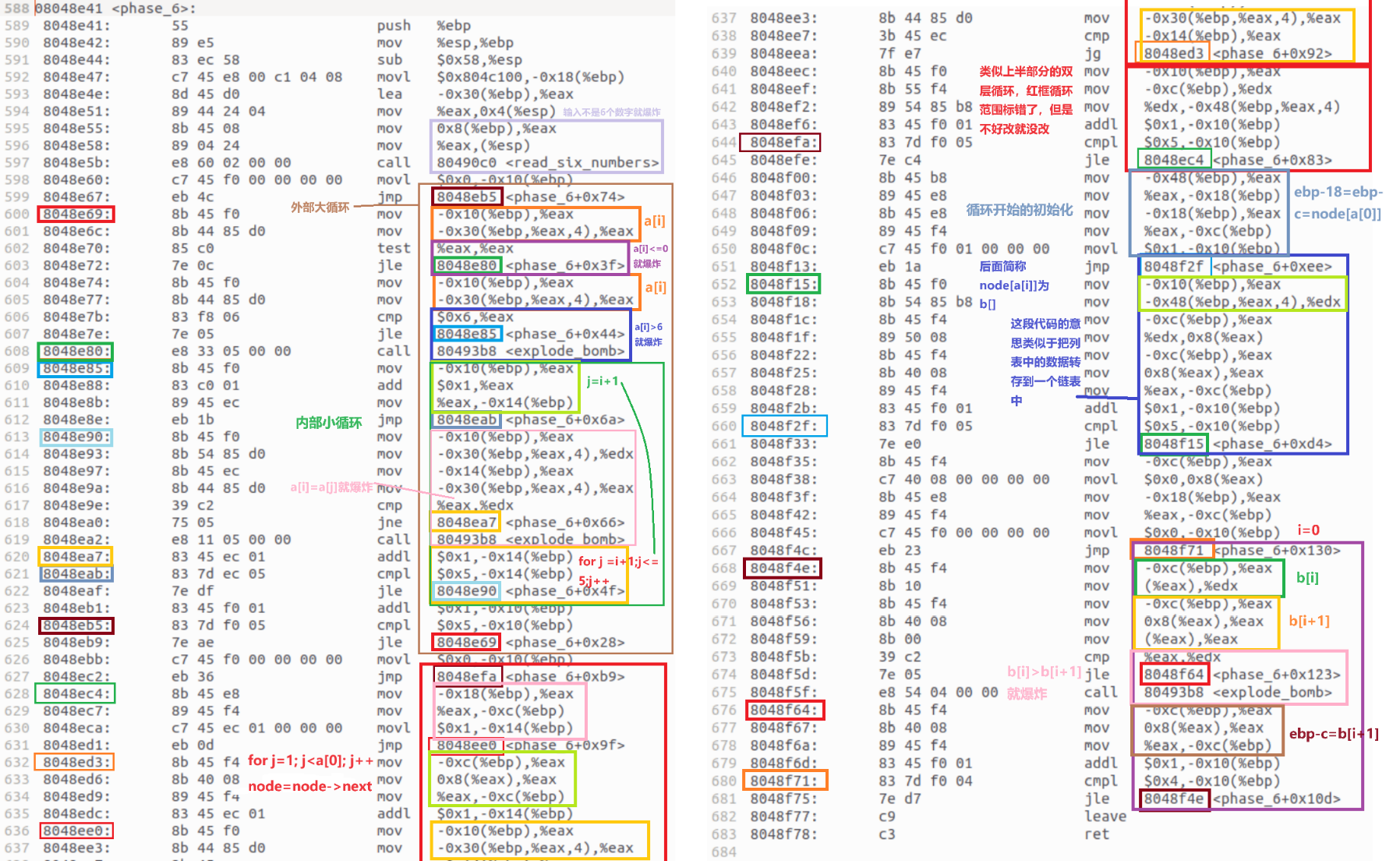
接着对信息初始化以进入循环ebp-c和ebp-10初始化为0，并且把第一个参数与0xf按位与，保留后四位。ebp-10用于累加a，ebp-c用于标记循环次数。初始a为第一个输入参数，a=m[a],即以a为下标，表中的数字。如果a=15就退出循环。

如果没有正好进行15次循环就爆炸，如果第二个参数和循环的累加结果相同就推出，否则爆炸。上面给出了循环的推导过程。因为第一个参数和0xf按位与了，只保留第四位，所以加16不影响最终结果

**答案合集：**

5+Z\*16 115 ,Z为正整数

**（phase\_6）**

****

|  |  |  |
| --- | --- | --- |
| phase\_6 | ebp+8 | 输入信息 |
|  | ebp+4 | 返回地址 |
|  | ebp | 旧ebp |
|  | ebp-4 |  |
|  | ebp-8 |  |
|  | ebp-c | =ebp-18 |
| 循环标记i | ebp-10 | 0 |
| 循环标记j | ebp-14 | =i+1 |
|  | ebp-18 | 0x804c100 |
|  | ebp-1c | 参数6 |
|  | ebp-20 | 参数5 |
|  | ebp-24 | 参数4 |
|  | ebp-28 | 参数3 |
|  | ebp-2c | 参数2 |
|  | ebp-30 | 参数1 |
|  | ebp-34 | node[参数6] |
|  | ebp-38 | node[参数5] |
|  | ebp-3c | node[参数4] |
|  | ebp-40 | node[参数3] |
|  | ebp-44 | node[参数2] |
|  | ebp-48 | node[参数1] |
|  | ebp-4c |  |
|  | ebp-50 |  |
|  | ebp-54 | ebp-30 |
| Esp | ebp-58 |  |



**思路：**第六题属于**指针**

一个开始用read\_six\_number函数判断输入是不是6个整型，接着的第一个大循环是一个冒泡排序，用来判断输入的6个数字都在1到6之前且不重复。

后一个大循环是把输入数字对应的节点中的内容保存到一个列表中，如栈帧表所示。

第三个循环没有很明白，感觉是把列表中的数值按数序存到一个指针数组中

最后一个循环是判断该**指针数组**是不是**升序**的，不是升序就爆炸。

所以输入的字符串是指针数组中数值按升序排序的顺序结果

**答案合集：**

4 1 2 3 5 6

**函数（写出这几个函数的原型）**

**<read\_six\_number>**

void read\_six\_numbers(int \*input\_numbers, const char \*input\_string) {  
 int numbers[6];  
 int result = sscanf(input\_string, "%d %d %d %d %d %d",  
 &numbers[0], &numbers[1], &numbers[2],  
 &numbers[3], &numbers[4], &numbers[5]);  
 if (result != 6) {  
 explode\_bomb();  
 }  
}

**<string\_length>**

size\_t string\_length(const char \*str) {

size\_t length = 0;

while (\*str != '\0') {

length++;

str++;

}

return length;

}

**<string\_not\_equal>**

bool strings\_not\_equal(const char \*str1, const char \*str2) {

size\_t len1 = string\_length(str1);

size\_t len2 = string\_length(str2);

if (len1 != len2) {

return true;

}

for (size\_t i = 0; i < len1; i++) {

if (str1[i] != str2[i]) {

return true;

}

}

return false;

}