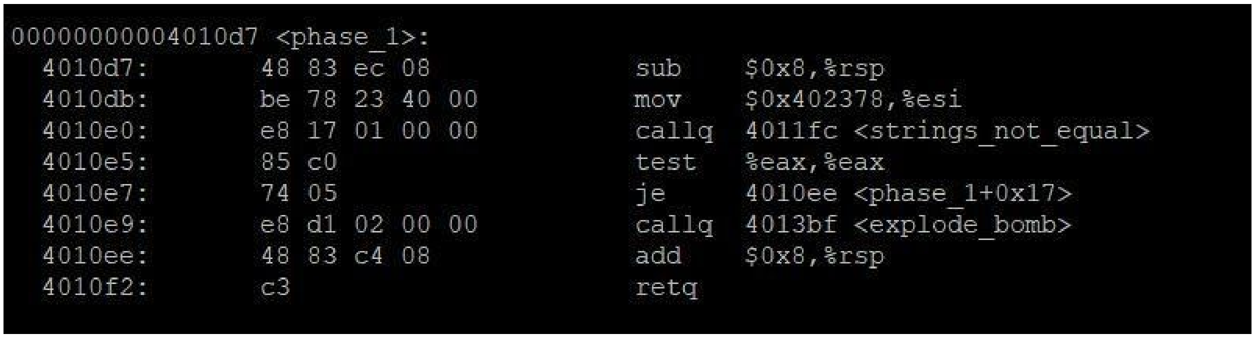
LAB REPORT: BOMB LAB

OSMAN ALI
CMPSC 311
OZA5@PSU.EDU
977810411

● PHASE 1:

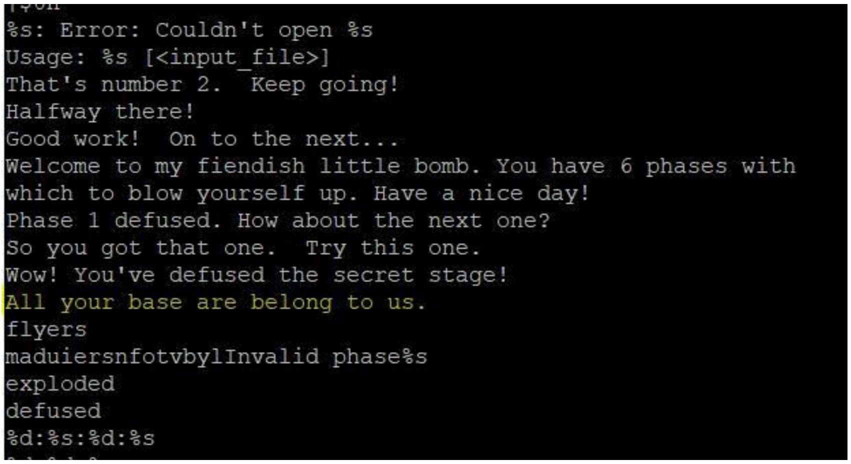
An overview of phase 1 is that it required us to input a randomly generated sentence into the input to diffuse the bomb. Most commonly used set of commands for this phase were **"x/s *address*"** and **"break"**. The gdb debugger **"disas"** command was used to review all assembly code.

The process to solve phase 1 required us to initially fire up the gdb and load the bomb assembly file as to it. Upon analyzing the assembly file pictured below:



```
00000000004010d7 <phase_1>:
  4010d7:       48 83 ec 08             sub    $0x8,%rsp
  4010db:       be 78 23 40 00          mov    $0x402378,%esi
  4010e0:       e8 17 01 00 00          callq  4011fc <strings_not_equal>
  4010e5:       85 c0                   test   %eax,%eax
  4010e7:       74 05                   je     4010ee <phase_1+0x17>
  4010e9:       e8 d1 02 00 00          callq  4013bf <explode_bomb>
  4010ee:       48 83 c4 08             add    $0x8,%rsp
  4010f2:       c3                      retq
```

We are able to observe that our user input is being put into **"$esi",** we can confirm this fact by using the command **"x/s $esi"** which would return our test string. Furthermore, we are able to observe that once we enter our string the function makes a call to **"<strings_not_equal>"**, which literally is a function comparing two strings to be equivalent. Knowing these facts, we know our answer is a hidden string within the **"$eax"** register. Upon printing this register I was presented with a string, **"ALL YOUR BASE ARE BEL"**, this points us to the identity of the string. Invoking the **command "strings bomb"** we were able to display all strings located in the code and were able to identify our answer:



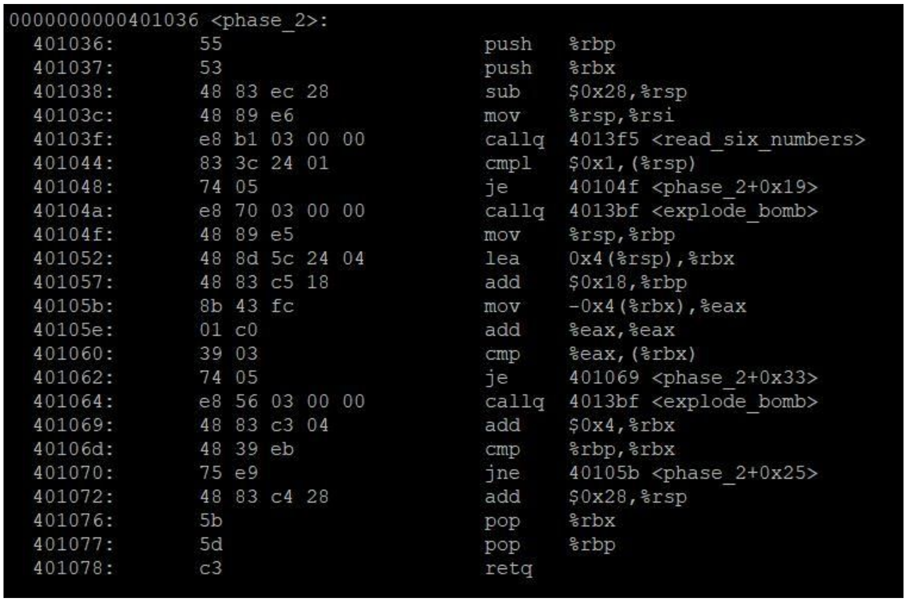
```
%s: Error: Couldn't open %s
Usage: %s [<input_file>]
That's number 2.  Keep going!
Halfway there!
Good work!  On to the next...
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
So you got that one.  Try this one.
Wow! You've defused the secret stage!
All your base are belong to us.
flyers
maduiersnfotvbylInvalid phase%s
exploded
defused
%d:%s:%d:%s
```

<u>ANSWER</u>: "All your base are belong to us."

● PHASE 2:

An overview of phase 2 is that we are supposed to input 6 spaced integers, following a mathematical pattern i.e. powers of two or a Fibonacci sequence. Most commonly used commands for this phase were **“disas”**, **“p/x”**, **“i r” and “break”**.

The process to solve phase 2 required us to initially fire up the gdb and load the bomb assembly file as to it. Upon analyzing the assembly file pictured below:



```
0000000000401036 <phase_2>:
  401036:       55                      push   %rbp
  401037:       53                      push   %rbx
  401038:       48 83 ec 28             sub    $0x28,%rsp
  40103c:       48 89 e6                mov    %rsp,%rsi
  40103f:       e8 b1 03 00 00          callq  4013f5 <read_six_numbers>
  401044:       83 3c 24 01             cmpl   $0x1,(%rsp)
  401048:       74 05                   je     40104f <phase_2+0x19>
  40104a:       e8 70 03 00 00          callq  4013bf <explode_bomb>
  40104f:       48 89 e5                mov    %rsp,%rbp
  401052:       48 8d 5c 24 04          lea    0x4(%rsp),%rbx
  401057:       48 83 c5 18             add    $0x18,%rbp
  40105b:       8b 43 fc                mov    -0x4(%rbx),%eax
  40105e:       01 c0                   add    %eax,%eax
  401060:       39 03                   cmp    %eax,(%rbx)
  401062:       74 05                   je     401069 <phase_2+0x33>
  401064:       e8 56 03 00 00          callq  4013bf <explode_bomb>
  401069:       48 83 c3 04             add    $0x4,%rbx
  40106d:       48 39 eb                cmp    %rbp,%rbx
  401070:       75 e9                   jne    40105b <phase_2+0x25>
  401072:       48 83 c4 28             add    $0x28,%rsp
  401076:       5b                      pop    %rbx
  401077:       5d                      pop    %rbp
  401078:       c3                      retq
```

We see that the assembly code takes in the user input as six integers which is evident from the call to the **“<read_six_numbers>”** function located on “**40103f**” informs us that this phase requires the user to input six integers to diffuse the bomb. We observe that the assembly consists of three different “**compare”** statements. The first one located on **“401044”** compares the first number of the user input to the value **“0x1”** which is 1 in decimal. This tells us for sure that our first value is **“1”.** After this our program shifts to **“401048”** where the address of the next number is stored on **“rbx”** and **“rbp”** gets the address right after the address of the last number read by the code. On **“40105b”** the previous number is copied into **“eax”** then the next instruction duplicates this value on **“eax”** which is then compared with our second number. If they are equal the function will continue execution at **“401069”**.

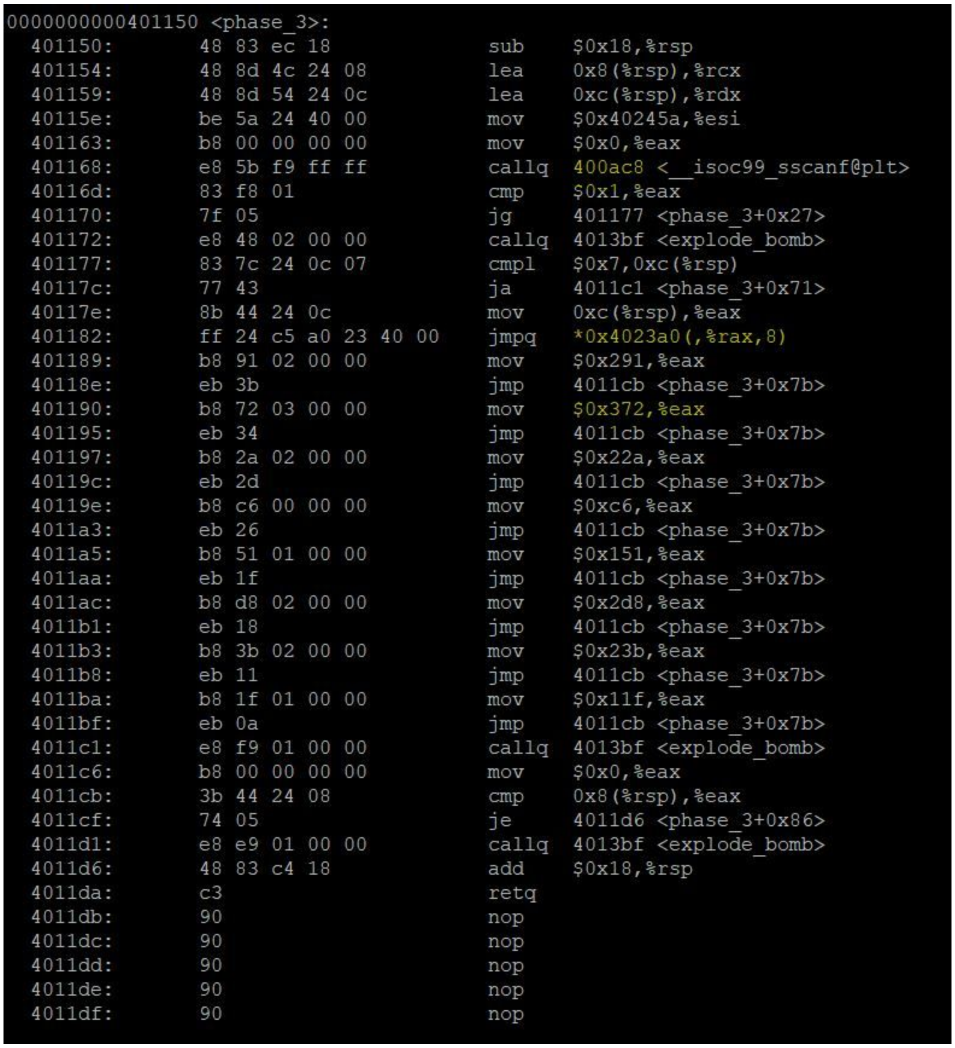
On **“401069”** the pointer goes to the next number. Next it checks if the pointer passed the last number which means all six numbers were checked. If it didn't it goes back to **“40105b”** to check the next number and if all numbers were already checked it will jump back to the **“main”** function. A summarized code for this process would basically be powers of 2. Since we know our first number is 1 for sure next six will be 5 powers of 2.

<u>ANSWER:</u> “1 2 4 8 16 32”

- **PHASE 3**

An overview of phase three is inputting two strings in the input which include two positive integer. The input was confirmed by looking into the memory address located at **"400ac8"**. Commonly used commands for this phase were **"disas"**, **"p/x"**, **"i r" and "break"**.

The process to solve phase 3 required us to initially fire up the gdb and load the bomb assembly file as to it. Upon analyzing the assembly file pictured below:



```
0000000000401150 <phase_3>:
  401150:       48 83 ec 18             sub    $0x18,%rsp
  401154:       48 8d 4c 24 08          lea    0x8(%rsp),%rcx
  401159:       48 8d 54 24 0c          lea    0xc(%rsp),%rdx
  40115e:       be 5a 24 40 00          mov    $0x40245a,%esi
  401163:       b8 00 00 00 00          mov    $0x0,%eax
  401168:       e8 5b f9 ff ff          callq  400ac8 <__isoc99_sscanf@plt>
  40116d:       83 f8 01                cmp    $0x1,%eax
  401170:       7f 05                   jg     401177 <phase_3+0x27>
  401172:       e8 48 02 00 00          callq  4013bf <explode_bomb>
  401177:       83 7c 24 0c 07          cmpl   $0x7,0xc(%rsp)
  40117c:       77 43                   ja     4011c1 <phase_3+0x71>
  40117e:       8b 44 24 0c             mov    0xc(%rsp),%eax
  401182:       ff 24 c5 a0 23 40 00    jmpq   *0x4023a0(,%rax,8)
  401189:       b8 91 02 00 00          mov    $0x291,%eax
  40118e:       eb 3b                   jmp    4011cb <phase_3+0x7b>
  401190:       b8 72 03 00 00          mov    $0x372,%eax
  401195:       eb 34                   jmp    4011cb <phase_3+0x7b>
  401197:       b8 2a 02 00 00          mov    $0x22a,%eax
  40119c:       eb 2d                   jmp    4011cb <phase_3+0x7b>
  40119e:       b8 c6 00 00 00          mov    $0xc6,%eax
  4011a3:       eb 26                   jmp    4011cb <phase_3+0x7b>
  4011a5:       b8 51 01 00 00          mov    $0x151,%eax
  4011aa:       eb 1f                   jmp    4011cb <phase_3+0x7b>
  4011ac:       b8 d8 02 00 00          mov    $0x2d8,%eax
  4011b1:       eb 18                   jmp    4011cb <phase_3+0x7b>
  4011b3:       b8 3b 02 00 00          mov    $0x23b,%eax
  4011b8:       eb 11                   jmp    4011cb <phase_3+0x7b>
  4011ba:       b8 1f 01 00 00          mov    $0x11f,%eax
  4011bf:       eb 0a                   jmp    4011cb <phase_3+0x7b>
  4011c1:       e8 f9 01 00 00          callq  4013bf <explode_bomb>
  4011c6:       b8 00 00 00 00          mov    $0x0,%eax
  4011cb:       3b 44 24 08             cmp    0x8(%rsp),%eax
  4011cf:       74 05                   je     4011d6 <phase_3+0x86>
  4011d1:       e8 e9 01 00 00          callq  4013bf <explode_bomb>
  4011d6:       48 83 c4 18             add    $0x18,%rsp
  4011da:       c3                      retq
  4011db:       90                      nop
  4011dc:       90                      nop
  4011dd:       90                      nop
  4011de:       90                      nop
  4011df:       90                      nop
```

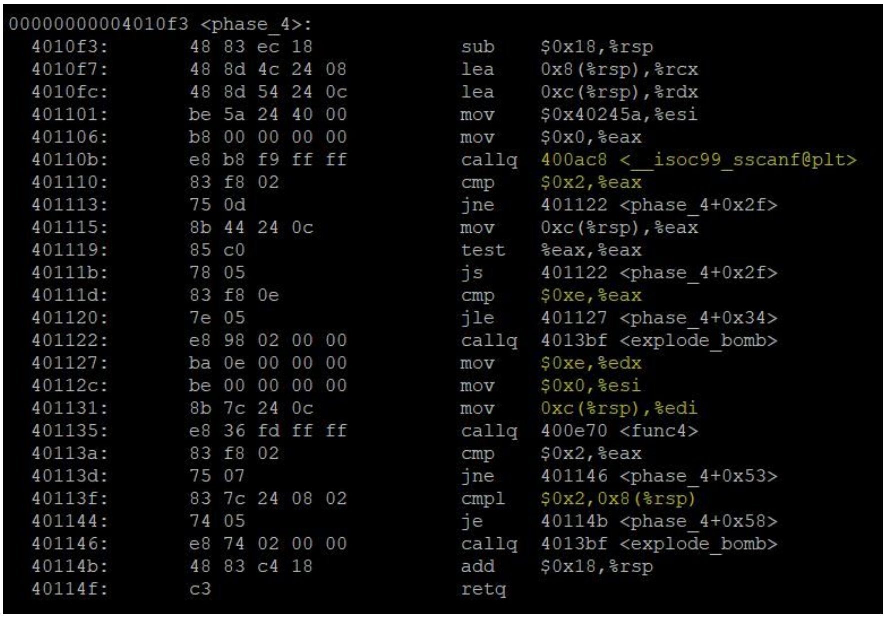
We observe that the code has various **"jmp"** and **"mov"** statements which are indicative of a switch statement algorithm within the assembly code. After the assembly code checks the user input for a valid number of strings it moves onto **"401182"** which is a address computation statement doing the following **"(user_input * $rax + 0x4023a0)",** supposing a user_input of 0 we are returned with value **"0x4023a0"** which we can examine using "**x/d 0x4023a0"** which tells us that we are being pointed to the **"401190"**. At this instruction we see that the number being entered into the register **"$rax0"** is **"0x372"**. This is our second value (first being the user input). Converting this hex value to decimal we get **"882"**, which is the second answer and can be added to the input.

ANSWER: **"0 882"**

- ## **PHASE 4:**

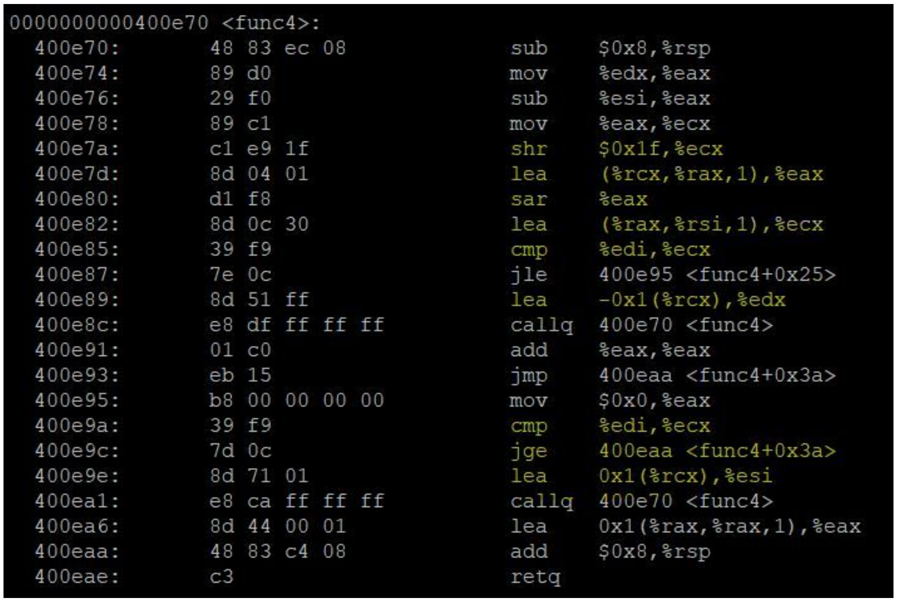
An overview of phase 4 is that it requires two integers as input however it is running a recursive algorithm which makes it particularly more challenging than the previous few phases. The most commonly used commands in this phase were **“disas”**, **“p/x”**, **“i r” and “break”**.

The process to solve phase 4 required us to initially fire up the gdb and load the bomb assembly file as to it. Upon analyzing the assembly file pictured below:



```
00000000004010f3 <phase_4>:
  4010f3:       48 83 ec 18             sub    $0x18,%rsp
  4010f7:       48 8d 4c 24 08          lea    0x8(%rsp),%rcx
  4010fc:       48 8d 54 24 0c          lea    0xc(%rsp),%rdx
  401101:       be 5a 24 40 00          mov    $0x40245a,%esi
  401106:       b8 00 00 00 00          mov    $0x0,%eax
  40110b:       e8 b8 f9 ff ff          callq  400ac8 <__isoc99_sscanf@plt>
  401110:       83 f8 02                cmp    $0x2,%eax
  401113:       75 0d                   jne    401122 <phase_4+0x2f>
  401115:       8b 44 24 0c             mov    0xc(%rsp),%eax
  401119:       85 c0                   test   %eax,%eax
  40111b:       78 05                   js     401122 <phase_4+0x2f>
  40111d:       83 f8 0e                cmp    $0xe,%eax
  401120:       7e 05                   jle    401127 <phase_4+0x34>
  401122:       e8 98 02 00 00          callq  4013bf <explode_bomb>
  401127:       ba 0e 00 00 00          mov    $0xe,%edx
  40112c:       be 00 00 00 00          mov    $0x0,%esi
  401131:       8b 7c 24 0c             mov    0xc(%rsp),%edi
  401135:       e8 36 fd ff ff          callq  400e70 <func4>
  40113a:       83 f8 02                cmp    $0x2,%eax
  40113d:       75 07                   jne    401146 <phase_4+0x53>
  40113f:       83 7c 24 08 02          cmpl   $0x2,0x8(%rsp)
  401144:       74 05                   je     40114b <phase_4+0x58>
  401146:       e8 74 02 00 00          callq  4013bf <explode_bomb>
  40114b:       48 83 c4 18             add    $0x18,%rsp
  40114f:       c3                      retq
```

Looking at this assembly code we are able to establish several helpful facts that will allow us to compute our answers. First and foremost the answer requires two integer inputs, this can be verified by looking into the contents of the address **“400ac8”**. Once this is done we see that this phase calls another external function called “**<func4>”** before we dive into function 4 we can establish from lines **“401127”** and **“40112c”** that the second and third argument must be **“0”** and **“e(14)”** looking at function 4 assembly code we see:



```
0000000000400e70 <func4>:
  400e70:       48 83 ec 08             sub    $0x8,%rsp
  400e74:       89 d0                   mov    %edx,%eax
  400e76:       29 f0                   sub    %esi,%eax
  400e78:       89 c1                   mov    %eax,%ecx
  400e7a:       c1 e9 1f                shr    $0x1f,%ecx
  400e7d:       8d 04 01                lea    (%rcx,%rax,1),%eax
  400e80:       d1 f8                   sar    %eax
  400e82:       8d 0c 30                lea    (%rax,%rsi,1),%ecx
  400e85:       39 f9                   cmp    %edi,%ecx
  400e87:       7e 0c                   jle    400e95 <func4+0x25>
  400e89:       8d 51 ff                lea    -0x1(%rcx),%edx
  400e8c:       e8 df ff ff ff          callq  400e70 <func4>
  400e91:       01 c0                   add    %eax,%eax
  400e93:       eb 15                   jmp    400eaa <func4+0x3a>
  400e95:       b8 00 00 00 00          mov    $0x0,%eax
  400e9a:       39 f9                   cmp    %edi,%ecx
  400e9c:       7d 0c                   jge    400eaa <func4+0x3a>
  400e9e:       8d 71 01                lea    0x1(%rcx),%esi
  400ea1:       e8 ca ff ff ff          callq  400e70 <func4>
  400ea6:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  400eaa:       48 83 c4 08             add    $0x8,%rsp
  400eae:       c3                      retq
```

function 4 does some computations which ca be neatly summarised by the C code shown below:

```
int func(int a, int b, int c)

{ int x = c - b;

int y = x >> 31;

 x = x + y;

 x = x >> 1;

y = x + b;

if (y <= a) {

 if (y >= a)

{ return 0; }

 else

{ return 2 * func4(a, y + 1, c) + 1; } }

else { return 2 * func4(a, b, y - 1);
```

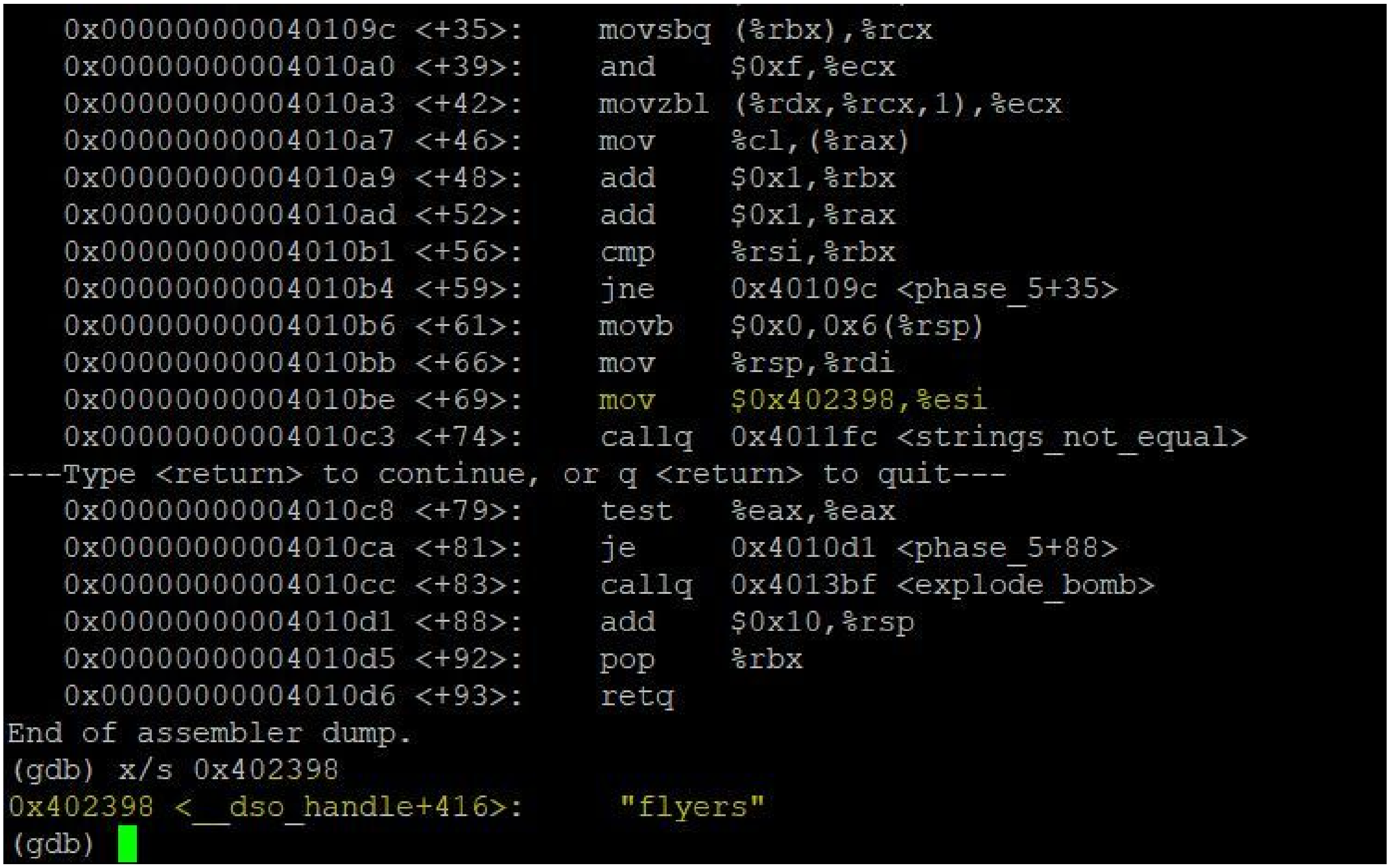
The inputs int a,b,c are user_input_1, 0 and 14 respectively we can use this code and keep on changing the value of **"a"** from 0 to 15 until we get the answer that is the same as the second input **"40113f"** which is **"2"**. Whichever value of a gives us this answer is the first input. We can perform this computation and will arrive at the answer to be at index **"4".**

<u>ANSWER:</u> "4 2"

PHASE 5:

An overview of phase five is that it requires users to find enter a non-spaced string of six alphabets that are basically jumbled up alphabets of a special name which in my case was **“flyers”.**

The process to solve phase 5 required us to initially fire up the gdb and load the bomb assembly file as to it. Upon analyzing the assembly file pictured below:



```
   0x000000000040109c <+35>:    movsbq (%rbx),%rcx
   0x00000000004010a0 <+39>:    and    $0xf,%ecx
   0x00000000004010a3 <+42>:    movzbl (%rdx,%rcx,1),%ecx
   0x00000000004010a7 <+46>:    mov    %cl,(%rax)
   0x00000000004010a9 <+48>:    add    $0x1,%rbx
   0x00000000004010ad <+52>:    add    $0x1,%rax
   0x00000000004010b1 <+56>:    cmp    %rsi,%rbx
   0x00000000004010b4 <+59>:    jne    0x40109c <phase_5+35>
   0x00000000004010b6 <+61>:    movb   $0x0,0x6(%rsp)
   0x00000000004010bb <+66>:    mov    %rsp,%rdi
   0x00000000004010be <+69>:    mov    $0x402398,%esi
   0x00000000004010c3 <+74>:    callq  0x4011fc <strings_not_equal>
---Type <return> to continue, or q <return> to quit---
   0x00000000004010c8 <+79>:    test   %eax,%eax
   0x00000000004010ca <+81>:    je     0x4010d1 <phase_5+88>
   0x00000000004010cc <+83>:    callq  0x4013bf <explode_bomb>
   0x00000000004010d1 <+88>:    add    $0x10,%rsp
   0x00000000004010d5 <+92>:    pop    %rbx
   0x00000000004010d6 <+93>:    retq
End of assembler dump.
(gdb) x/s 0x402398
0x402398 <__dso_handle+416>:     "flyers"
(gdb)
```

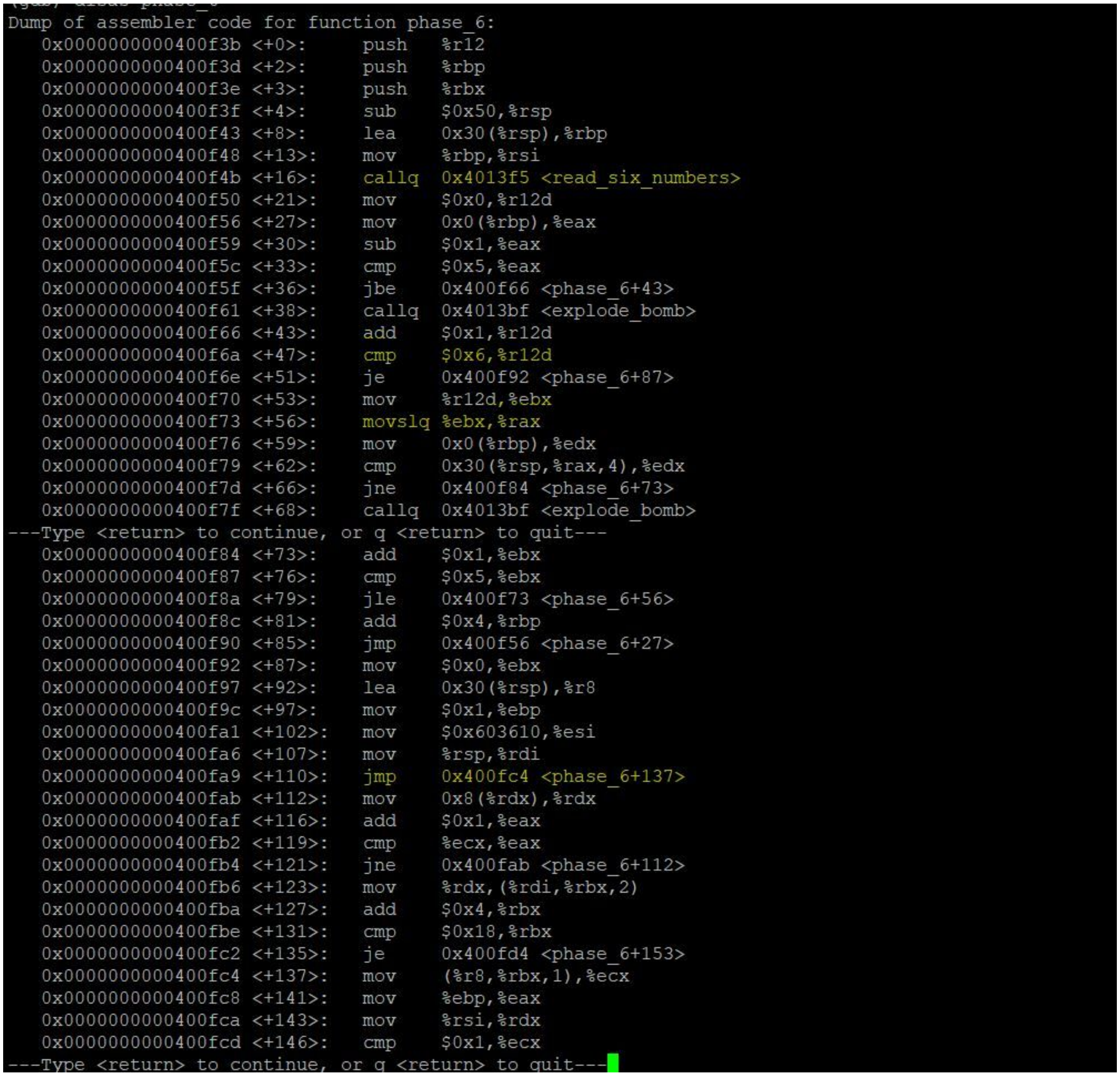
Looking at the assembly code we can look into the various different registers that are being compared and iterated against. Once we examine **“4010be”** we can see the value at the register **“%esi”** is **“flyers”.** Furthermore we can see the position of each character in flyers and then calculate the offset of all these values using the reference string **“maduiersnfotvbyl”** located in register number **“4023e0”.** We can use the position and minus with reference to get the offset and hence get our answer which it maps out.

ANSWER: “)/.%&' ”

- **PHASE 6:**

An overview of this phase is that we need to input 6 integers in a specific order to diffuse this phase.

The process to solve phase 6 required us to initially fire up the gdb and load the bomb assembly file as to it. Upon analyzing the assembly file pictured below:



```
Dump of assembler code for function phase_6:
   0x0000000000400f3b <+0>:     push   %r12
   0x0000000000400f3d <+2>:     push   %rbp
   0x0000000000400f3e <+3>:     push   %rbx
   0x0000000000400f3f <+4>:     sub    $0x50,%rsp
   0x0000000000400f43 <+8>:     lea    0x30(%rsp),%rbp
   0x0000000000400f48 <+13>:    mov    %rbp,%rsi
   0x0000000000400f4b <+16>:    callq  0x4013f5 <read_six_numbers>
   0x0000000000400f50 <+21>:    mov    $0x0,%r12d
   0x0000000000400f56 <+27>:    mov    0x0(%rbp),%eax
   0x0000000000400f59 <+30>:    sub    $0x1,%eax
   0x0000000000400f5c <+33>:    cmp    $0x5,%eax
   0x0000000000400f5f <+36>:    jbe    0x400f66 <phase_6+43>
   0x0000000000400f61 <+38>:    callq  0x4013bf <explode_bomb>
   0x0000000000400f66 <+43>:    add    $0x1,%r12d
   0x0000000000400f6a <+47>:    cmp    $0x6,%r12d
   0x0000000000400f6e <+51>:    je     0x400f92 <phase_6+87>
   0x0000000000400f70 <+53>:    mov    %r12d,%ebx
   0x0000000000400f73 <+56>:    movslq %ebx,%rax
   0x0000000000400f76 <+59>:    mov    0x0(%rbp),%edx
   0x0000000000400f79 <+62>:    cmp    0x30(%rsp,%rax,4),%edx
   0x0000000000400f7d <+66>:    jne    0x400f84 <phase_6+73>
   0x0000000000400f7f <+68>:    callq  0x4013bf <explode_bomb>
---Type <return> to continue, or q <return> to quit---
   0x0000000000400f84 <+73>:    add    $0x1,%ebx
   0x0000000000400f87 <+76>:    cmp    $0x5,%ebx
   0x0000000000400f8a <+79>:    jle    0x400f73 <phase_6+56>
   0x0000000000400f8c <+81>:    add    $0x4,%rbp
   0x0000000000400f90 <+85>:    jmp    0x400f56 <phase_6+27>
   0x0000000000400f92 <+87>:    mov    $0x0,%ebx
   0x0000000000400f97 <+92>:    lea    0x30(%rsp),%r8
   0x0000000000400f9c <+97>:    mov    $0x1,%ebp
   0x0000000000400fa1 <+102>:   mov    $0x603610,%esi
   0x0000000000400fa6 <+107>:   mov    %rsp,%rdi
   0x0000000000400fa9 <+110>:   jmp    0x400fc4 <phase_6+137>
   0x0000000000400fab <+112>:   mov    0x8(%rdx),%rdx
   0x0000000000400faf <+116>:   add    $0x1,%eax
   0x0000000000400fb2 <+119>:   cmp    %ecx,%eax
   0x0000000000400fb4 <+121>:   jne    0x400fab <phase_6+112>
   0x0000000000400fb6 <+123>:   mov    %rdx,(%rdi,%rbx,2)
   0x0000000000400fba <+127>:   add    $0x4,%rbx
   0x0000000000400fbe <+131>:   cmp    $0x18,%rbx
   0x0000000000400fc2 <+135>:   je     0x400fd4 <phase_6+153>
   0x0000000000400fc4 <+137>:   mov    (%r8,%rbx,1),%ecx
   0x0000000000400fc8 <+141>:   mov    %ebp,%eax
   0x0000000000400fca <+143>:   mov    %rsi,%rdx
   0x0000000000400fcd <+146>:   cmp    $0x1,%ecx
---Type <return> to continue, or q <return> to quit---
```

Input conditions indicate that 6 input numbers are required as the result is compared to **"0x6".**However each number needs to be different from the other. Knowing our answer is 1-6 we can use trial and error by setting a breakpoint at function **"<explode bomb>"** this way we are able to compute which node points to what node next hence are able to place them in the correct position. Even Though it is not an elegant method of working this out the relatively considerable number of permutations allow us to check each order and place the list in a correct ascending order . We can user the following commands below to achieve this goal:

```
x/3x *(*(*(*($eax+8)+8)+8)+8)
```

```
x/3x *(*(*($eax+8)+8)+8)

x/3x *(*($eax+8)+8)

x/3x *($eax+8)

x/3x $eax
```

These commands allow us to compute the missing node by examining the appropriate hex values, these hex values allow us to order the list and subtract 7 over each iteration from the node value.

<u>ANSWER:</u> "3 6 4 1 2 5"