Name :-

Purval Madhukar Bhude

Roll No. S20230010193

Subject :- CA

Bomb Lab

Bomb No. 304

Phase 1:

```
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) break phase_1
Breakpoint 1 at 0x15a7
(gdb) run
Starting program: /mnt/c/IIITS ASSIGNMENTS/Sem 2/Computer Arch/Bomb Lab 3/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
hgfd

Breakpoint 1, 0x00005555555555a7 in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
=> 0x00005555555555a7 <+0>:     endbr64
   0x00005555555555ab <+4>:     sub    $0x8,%rsp
   0x00005555555555af <+8>:     lea    0x1b9a(%rip),%rsi        # 0x555555557150
   0x00005555555555b6 <+15>:    call   0x555555555aa1 <strings_not_equal>
   0x00005555555555bb <+20>:    test   %eax,%eax
   0x00005555555555bd <+22>:    jne    0x5555555555c4 <phase_1+29>
   0x00005555555555bf <+24>:    add    $0x8,%rsp
   0x00005555555555c3 <+28>:    ret
   0x00005555555555c4 <+29>:    call   0x555555555bb5 <explode_bomb>
   0x00005555555555c9 <+34>:    jmp    0x5555555555bf <phase_1+24>
End of assembler dump.
(gdb) x/s 0x555555557150
0x555555557150: "Brownie, you are doing a heck of a job."
```

First, set a breakpoint for phase_1 and run the program. Then, disassemble it, proceeding step by step. In the second line, we check whether the input string is the same as 0x555555557150 and then convert it to a string using x/s

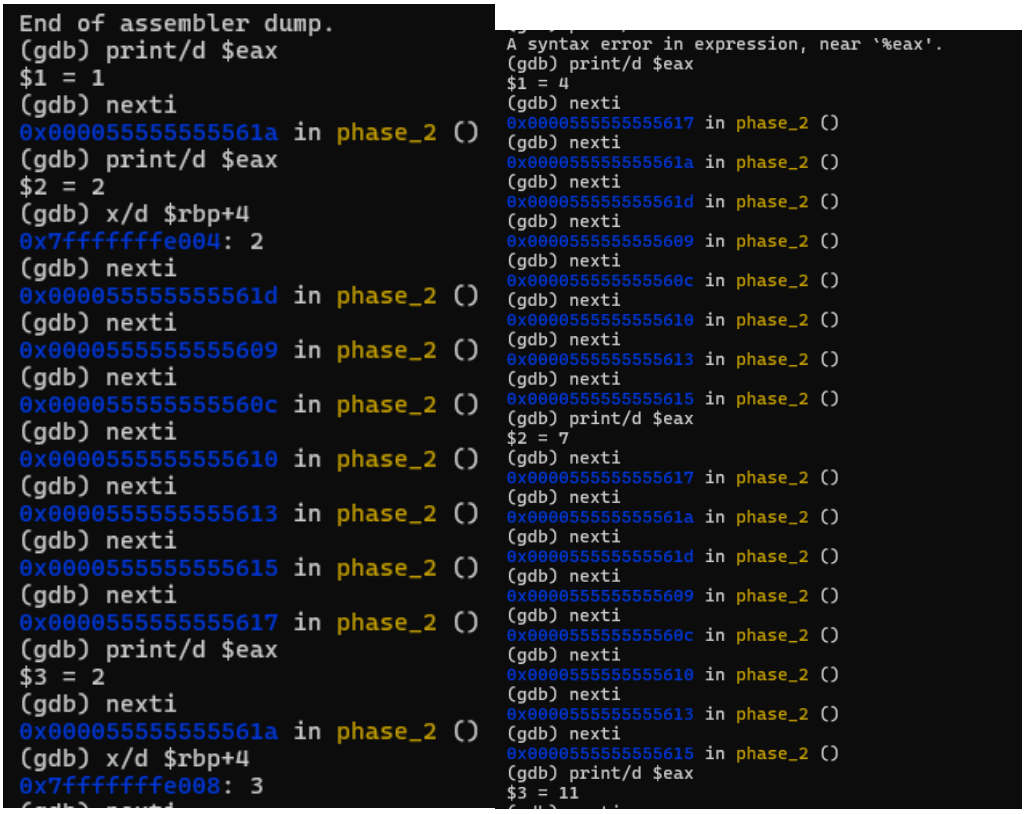
Phase_1 answer : Brownie, you are doing a heck of a job.

Phase 2:

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) break phase_2
Breakpoint 1 at 0x15cb
(gdb) run
Starting program: /mnt/c/IIITS ASSIGNMENTS/Sem 2/Computer Arch/Bomb Lab 3/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
Phase 1 defused. How about the next one?
1 223 12

Breakpoint 1, 0x000055555555555cb in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
=> 0x000055555555555cb <+0>:     endbr64
   0x000055555555555cf <+4>:     push   %rbp
   0x000055555555555d0 <+5>:     push   %rbx
   0x000055555555555d1 <+6>:     sub    $0x28,%rsp
   0x000055555555555d5 <+10>:    mov    %fs:0x28,%rax
   0x000055555555555de <+19>:    mov    %rax,0x18(%rsp)
   0x000055555555555e3 <+24>:    xor    %eax,%eax
   0x000055555555555e5 <+26>:    mov    %rsp,%rsi
   0x000055555555555e8 <+29>:    call   0x555555555be1 <read_six_numbers>
   0x000055555555555ed <+34>:    cmpl   $0x0,(%rsp)
   0x000055555555555f1 <+38>:    js     0x5555555555fd <phase_2+50>
   0x000055555555555f3 <+40>:    mov    %rsp,%rbp
   0x000055555555555f6 <+43>:    mov    $0x1,%ebx
   0x000055555555555fb <+48>:    jmp    0x555555555615 <phase_2+74>
   0x000055555555555fd <+50>:    call   0x555555555bb5 <explode_bomb>
   0x0000555555555602 <+55>:    jmp    0x5555555555f3 <phase_2+40>
   0x0000555555555604 <+57>:    call   0x555555555bb5 <explode_bomb>
   0x0000555555555609 <+62>:    add    $0x1,%ebx
   0x000055555555560c <+65>:    add    $0x4,%rbp
   0x0000555555555610 <+69>:    cmp    $0x6,%ebx
   0x0000555555555613 <+72>:    je     0x555555555621 <phase_2+86>
   0x0000555555555615 <+74>:    mov    %ebx,%eax
   0x0000555555555617 <+76>:    add    0x0(%rbp),%eax
   0x000055555555561a <+79>:    cmp    %eax,0x4(%rbp)
   0x000055555555561d <+82>:    je     0x555555555609 <phase_2+62>
   0x000055555555561f <+84>:    jmp    0x555555555604 <phase_2+57>
   0x0000555555555621 <+86>:    mov    0x18(%rsp),%rax
   0x0000555555555626 <+91>:    xor    %fs:0x28,%rax
   0x000055555555562f <+100>:   jne    0x555555555638 <phase_2+109>
   0x0000555555555631 <+102>:   add    $0x28,%rsp
   0x0000555555555635 <+106>:   pop    %rbx
   0x0000555555555636 <+107>:   pop    %rbp
   0x0000555555555637 <+108>:   ret
   0x0000555555555638 <+109>:   call   0x555555555220 <__stack_chk_fail@plt>
End of assembler dump.
(gdb) stepi
0x000055555555555cf in phase_2 ()
(gdb) nexti
```

Set a breakpoint for phase_2 and then run the program. Upon disassembling it, we receive a hint indicating that there are six numbers expected in the answer, inferred from the function name (read_six_numbers). After stepping through the code and disassembling it seven times, we enter the read_six_numbers function.



And then analysing the function we understand that it is eax = eax + index which can also written as eax+=index then find 6 numbers

eax = 1 + 0 = 1 (index = 0)

eax = 1 + 1 = 2 (index = 1)

eax = 2 + 2 = 4 (index = 2)

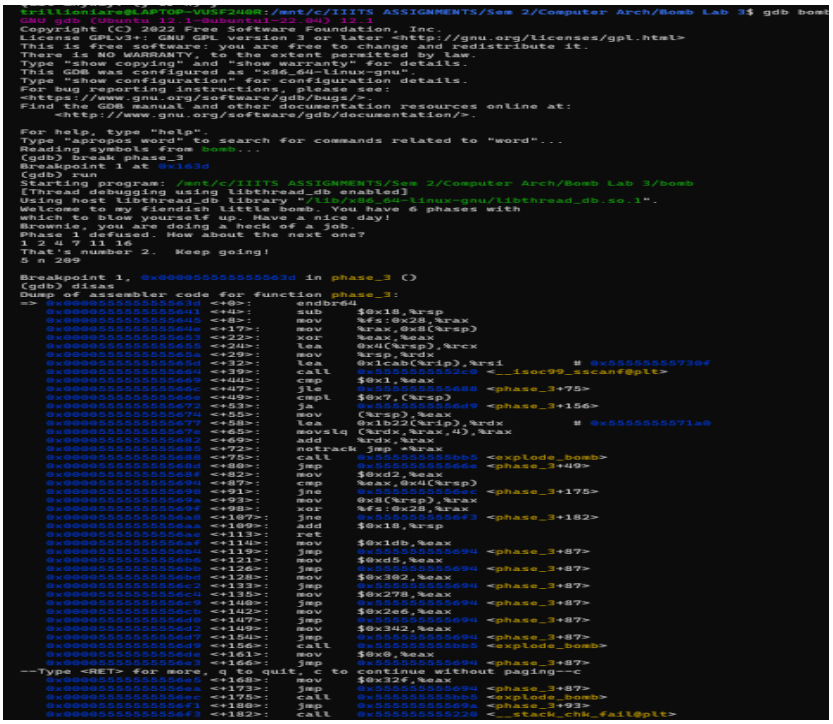
eax = 4 + 3 = 7 (index = 3)

eax = 7 + 4 = 11 (index = 4)

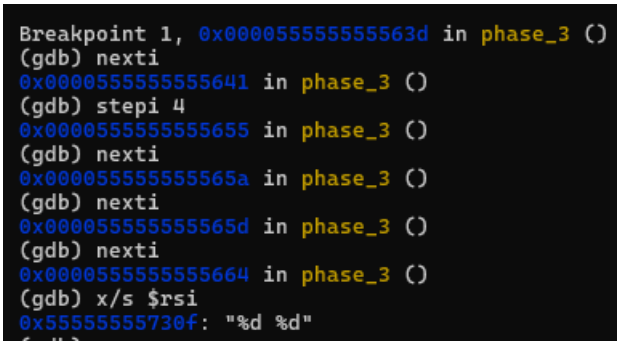
eax = 11 + 5 = 16 (index = 5)

Phase 2 answer: 1 2 4 7 11 16

Phase 3:



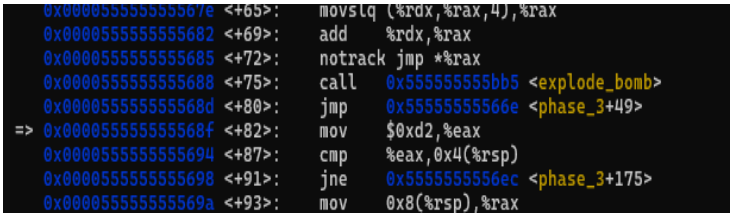
As we done in above 2 phase we first put the breakpoint and then run the program. After running we disas.

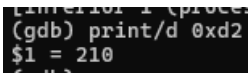


By this we can see that 2 integers are input. So now again run the program and put 2 numbers.

And by (notrack jmp *%rax) we understand that it is switch case

And when I put first number as 1 it jumped to 0x000055555555568f. finding that number using x/wd





Phase_3 answer : 1 210

When we put 2 as first number it jump to 0x00005555555556af

```
   0x0000555555555682 <+69>:    add    %rdx,%rax
   0x0000555555555685 <+72>:    notrack jmp *%rax
   0x0000555555555688 <+75>:    call   0x555555555bb5 <explode_bomb>
   0x000055555555568d <+80>:    jmp    0x55555555566e <phase_3+49>
   0x000055555555568f <+82>:    mov    $0xd2,%eax
   0x0000555555555694 <+87>:    cmp    %eax,0x4(%rsp)
   0x0000555555555698 <+91>:    jne    0x5555555556ec <phase_3+175>
   0x000055555555569a <+93>:    mov    0x8(%rsp),%rax
   0x000055555555569f <+98>:    xor    %fs:0x28,%rax
   0x00005555555556a8 <+107>:   jne    0x5555555556f3 <phase_3+182>
   0x00005555555556aa <+109>:   add    $0x18,%rsp
   0x00005555555556ae <+113>:   ret
=> 0x00005555555556af <+114>:   mov    $0x1db,%eax
   0x00005555555556b4 <+119>:   jmp    0x555555555694 <phase_3+87>
   0x00005555555556b6 <+121>:   mov    $0xd5,%eax
```

```
(gdb) print/d 0x1db
$1 = 475
```

For first number when it is 2 answer is
Phase 2: 2 475

Phase 4:

```
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
1 2 4 7Phase 1 defused. How about the next one?
 11 16
1That's number 2.  Keep going!
 210
Halfway there!
12 12

Breakpoint 1, 0x000055555555572e in phase_4 ()
(gdb) disas
Dump of assembler code for function phase_4:
=> 0x000055555555572e <+0>:     endbr64
   0x0000555555555732 <+4>:     sub    $0x18,%rsp
   0x0000555555555736 <+8>:     mov    %fs:0x28,%rax
   0x000055555555573f <+17>:    mov    %rax,0x8(%rsp)
   0x0000555555555744 <+22>:    xor    %eax,%eax
   0x0000555555555746 <+24>:    lea    0x4(%rsp),%rcx
   0x000055555555574b <+29>:    mov    %rsp,%rdx
   0x000055555555574e <+32>:    lea    0x1bba(%rip),%rsi        # 0x55555555730f
   0x0000555555555755 <+39>:    call   0x5555555552c0 <__isoc99_sscanf@plt>
   0x000055555555575a <+44>:    cmp    $0x2,%eax
   0x000055555555575d <+47>:    jne    0x555555555765 <phase_4+55>
   0x000055555555575f <+49>:    cmpl   $0xe,(%rsp)
   0x0000555555555763 <+53>:    jbe    0x55555555576a <phase_4+60>
   0x0000555555555765 <+55>:    call   0x555555555bb5 <explode_bomb>
   0x000055555555576a <+60>:    mov    $0xe,%edx
   0x000055555555576f <+65>:    mov    $0x0,%esi
   0x0000555555555774 <+70>:    mov    (%rsp),%edi
   0x0000555555555777 <+73>:    call   0x5555555556f8 <func4>
   0x000055555555577c <+78>:    cmp    $0x12,%eax
   0x000055555555577f <+81>:    jne    0x555555555788 <phase_4+90>
   0x0000555555555781 <+83>:    cmpl   $0x12,0x4(%rsp)
   0x0000555555555786 <+88>:    je     0x55555555578d <phase_4+95>
   0x0000555555555788 <+90>:    call   0x555555555bb5 <explode_bomb>
   0x000055555555578d <+95>:    mov    0x8(%rsp),%rax
   0x0000555555555792 <+100>:   xor    %fs:0x28,%rax
   0x000055555555579b <+109>:   jne    0x5555555557a2 <phase_4+116>
   0x000055555555579d <+111>:   add    $0x18,%rsp
   0x00005555555557a1 <+115>:   ret
   0x00005555555557a2 <+116>:   call   0x555555555220 <__stack_chk_fail@plt>
End of assembler dump.
```

Similarway, putting break point and running then disas. Here we known that

```
(gdb) x/s 0x55555555730f
0x55555555730f: "%d %d"
```

there are 2 integer as a input

```
   0x0000555555555574e <+32>:    lea    0x1bba(%rip),%rsi        # 0x555555557307
   0x0000555555555755 <+39>:    call   0x5555555552c0 <__isoc99_sscanf@plt>
   0x000055555555575a <+44>:    cmp    $0x2,%eax
   0x000055555555575d <+47>:    jne    0x555555555765 <phase_4+55>
   0x000055555555575f <+49>:    cmpl   $0xe,(%rsp)
=> 0x0000555555555763 <+53>:    jbe    0x55555555576a <phase_4+60>
   0x0000555555555765 <+55>:    call   0x5555555555bb5 <explode_bomb>
   0x000055555555576a <+60>:    mov    $0xe,%edx
   0x000055555555576f <+65>:    mov    $0x0,%esi
   0x0000555555555774 <+70>:    mov    (%rsp),%edi
   0x0000555555555777 <+73>:    call   0x5555555556f8 <func4>
```

By this part we understand that first number will be in range of 2 -> 14 (e) and by going in recursive function func4 for 11 as a input we it is giving output as 0x12 for which it is check you can see that below in the image as well so we by hit and trial between 2->14 we can say that first number is 11.

```
   0x000055555555576f <+65>:    mov    $0x0,%esi
   0x0000555555555774 <+70>:    mov    (%rsp),%edi
   0x0000555555555777 <+73>:    call   0x5555555556f8 <func4>
   0x000055555555577c <+78>:    cmp    $0x12,%eax
=> 0x000055555555577f <+81>:    jne    0x555555555788 <phase_4+90>
   0x0000555555555781 <+83>:    cmpl   $0x12,0x4(%rsp)
```

Now, to find 2nd number we can easily see that it is comparing we 0x12 and check whether they are equal or not so second number is 0x12 which is 18.

```
(gdb) print /d 0x12
$1 = 18
```

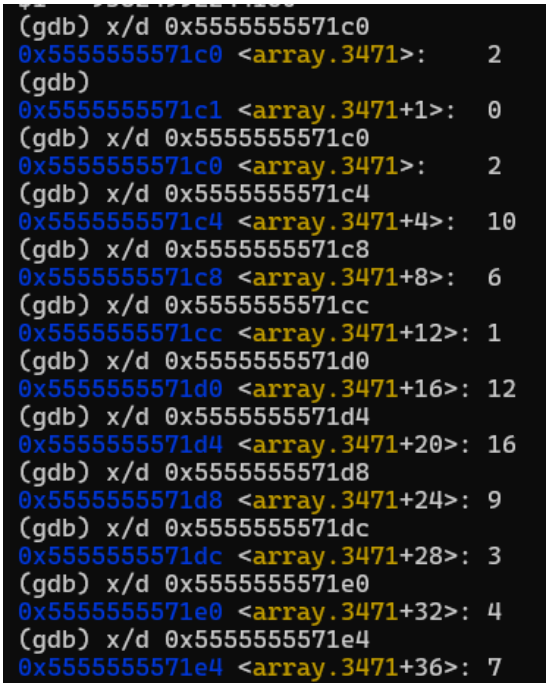
```
   0x0000555555555777 <+73>:    call   0x5555555556f8 <func4>
   0x000055555555577c <+78>:    cmp    $0x12,%eax
   0x000055555555577f <+81>:    jne    0x555555555788 <phase_4+90>
   0x0000555555555781 <+83>:    cmpl   $0x12,0x4(%rsp)
   0x0000555555555786 <+88>:    je     0x55555555578d <phase_4+95>
   0x0000555555555788 <+90>:    call   0x5555555555bb5 <explode_bomb>
   0x000055555555578d <+95>:    mov    0x8(%rsp),%rax
   0x0000555555555792 <+100>:   xor    %fs:0x28,%rax
```

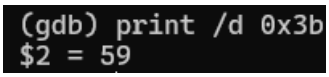
Phase 5:

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) break phase_5
Breakpoint 1 at 0x17a7
(gdb) run
Starting program: /mnt/c/IIITS ASSIGNMENTS/Sem 2/Computer Arch/Bomb Lab 3/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
1 2 4 7Phase 1 defused. How about the next one?
 11 16
1That's number 2.  Keep going!
 210
11 Halfway there!
18
So you got that one.  Try this one.
123 123

Breakpoint 1, 0x00005555555557a7 in phase_5 ()
(gdb) disas
Dump of assembler code for function phase_5:
=> 0x00005555555557a7 <+0>:     endbr64
   0x00005555555557ab <+4>:     push   %rbx
   0x00005555555557ac <+5>:     mov    %rdi,%rbx
   0x00005555555557af <+8>:     call   0x555555555a80 <string_length>
   0x00005555555557b4 <+13>:    cmp    $0x6,%eax
   0x00005555555557b7 <+16>:    jne    0x5555555557e5 <phase_5+62>
   0x00005555555557b9 <+18>:    mov    %rbx,%rax
   0x00005555555557bc <+21>:    lea    0x6(%rbx),%rdi
   0x00005555555557c0 <+25>:    mov    $0x0,%ecx
   0x00005555555557c5 <+30>:    lea    0x19f4(%rip),%rsi        # 0x5555555571c0 <array.3471>
   0x00005555555557cc <+37>:    movzbl (%rax),%edx
   0x00005555555557cf <+40>:    and    $0xf,%edx
   0x00005555555557d2 <+43>:    add    (%rsi,%rdx,4),%ecx
   0x00005555555557d5 <+46>:    add    $0x1,%rax
   0x00005555555557d9 <+50>:    cmp    %rdi,%rax
   0x00005555555557dc <+53>:    jne    0x5555555557cc <phase_5+37>
   0x00005555555557de <+55>:    cmp    $0x3b,%ecx
   0x00005555555557e1 <+58>:    jne    0x5555555557ec <phase_5+69>
   0x00005555555557e3 <+60>:    pop    %rbx
   0x00005555555557e4 <+61>:    ret
   0x00005555555557e5 <+62>:    call   0x555555555bb5 <explode_bomb>
   0x00005555555557ea <+67>:    jmp    0x5555555557b9 <phase_5+18>
   0x00005555555557ec <+69>:    call   0x555555555bb5 <explode_bomb>
   0x00005555555557f1 <+74>:    jmp    0x5555555557e3 <phase_5+60>
End of assembler dump.
```

We put breakpoint, run and then disas. Here we understand that it is string or array and in next line it is comparing with 0x6 which meaning that array or string must be of length 6.





By array's first pointer we can find all the array tell 10th position giving above image. By seeing it we understand that sum of all input of array should be equal to 0x3b. string store index. So we make one of it as 111116 as

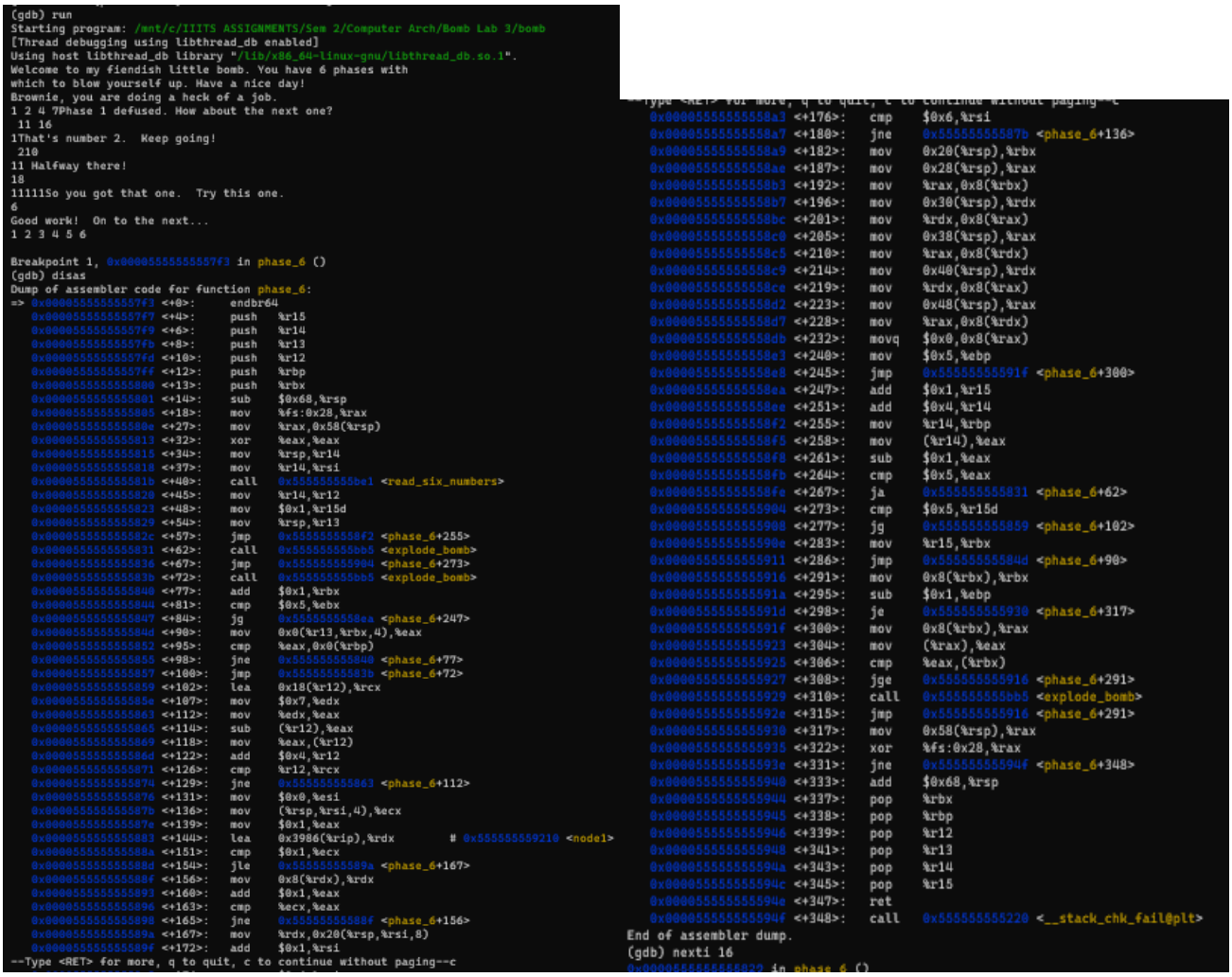
sum = arr[1]+ arr[1]+ arr[1]+ arr[1]+ arr[1]+ arr[6]

= 10 + 10 + 10 + 10 + 10 + 9 = 59

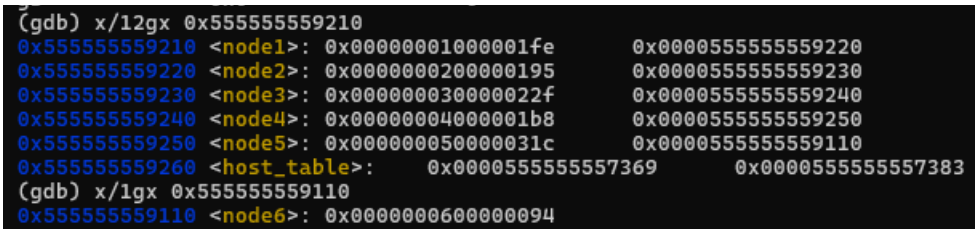
We can make any combination but sum should be 59.

Phase 5 answer: 111116

Phase 6:

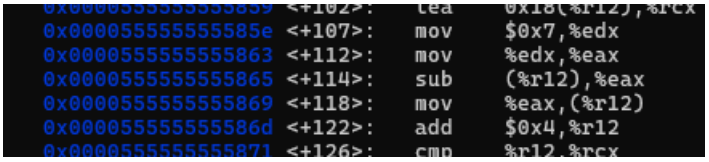


Same as previous we have to do breakpoint, run and then disas by analysing the above it is using linked list and if we take out nodes which is shown in below image.



By this image 

we can say that order should be decreasing order which we found using above image and that numbers are 5 3 1 4 2 6.

And then we also have 

so we have to do 7-number that will be the answer.

arr[0] = 7-5 = 2

arr[1] = 7-3 = 4

arr[2] = 7-1 = 6

arr[3] = 7-4 = 3

arr[4] = 7-2 = 5

arr[5] = 7-6 = 1

Phase 6 answer: 2 4 6 3 5 1

Scream Phase:

Setting the break point in phase_4 and then in scream_phase

```
Reading symbols from bomb...
(gdb) break phase_4
Breakpoint 1 at 0x172e
(gdb) breal secret_phase
Undefined command: "breal".  Try "help".
(gdb) break secret_phase
Breakpoint 2 at 0x1995
```

And then running the program and disas the scream phase

```
Breakpoint 2, 0x0000555555555995 in secret_phase ()
(gdb) disas
Dump of assembler code for function secret_phase:
=> 0x0000555555555995 <+0>:     endbr64
   0x0000555555555999 <+4>:     push   %rbx
   0x000055555555599a <+5>:     call   0x555555555c26 <read_line>
   0x000055555555599f <+10>:    mov    %rax,%rdi
   0x00005555555559a2 <+13>:    mov    $0xa,%edx
   0x00005555555559a7 <+18>:    mov    $0x0,%esi
   0x00005555555559ac <+23>:    call   0x5555555552a0 <strtol@plt>
   0x00005555555559b1 <+28>:    mov    %rax,%rbx
   0x00005555555559b4 <+31>:    lea    -0x1(%rax),%eax
   0x00005555555559b7 <+34>:    cmp    $0x3e8,%eax
   0x00005555555559bc <+39>:    ja     0x5555555559e3 <secret_phase+78>
   0x00005555555559be <+41>:    mov    %ebx,%esi
   0x00005555555559c0 <+43>:    lea    0x3769(%rip),%rdi        # 0x555555559130 <n1>
   0x00005555555559c7 <+50>:    call   0x555555555954 <fun7>
   0x00005555555559cc <+55>:    test   %eax,%eax
   0x00005555555559ce <+57>:    jne    0x5555555559ea <secret_phase+85>
   0x00005555555559d0 <+59>:    lea    0x17a1(%rip),%rdi        # 0x555555557178
   0x00005555555559d7 <+66>:    call   0x555555555200 <puts@plt>
   0x00005555555559dc <+71>:    call   0x555555555d6e <phase_defused>
   0x00005555555559e1 <+76>:    pop    %rbx
   0x00005555555559e2 <+77>:    ret
   0x00005555555559e3 <+78>:    call   0x555555555bb5 <explode_bomb>
   0x00005555555559e8 <+83>:    jmp    0x5555555559be <secret_phase+41>
   0x00005555555559ea <+85>:    call   0x555555555bb5 <explode_bomb>
   0x00005555555559ef <+90>:    jmp    0x5555555559d0 <secret_phase+59>
End of assembler dump.
```

Once checking for <n1> and that's the answer

```
(gdb) x/d 0x555555559130
0x555555559130 <n1>:    36
```

Secret_phase answer: 36

All Answers txt file and giving it input to bomb

```
trillioniare@LAPTOP-VUSF240R:/mnt/c/IIITS ASSIGNMENTS/Sem 2/Computer Arch/Bomb Lab 3$ cat Bomb304_S20230010193_solution.txt
Brownie, you are doing a heck of a job.
1 2 4 7 11 16
1 210
11 18 DrEvil
111116
2 4 6 3 5 1
36
trillioniare@LAPTOP-VUSF240R:/mnt/c/IIITS ASSIGNMENTS/Sem 2/Computer Arch/Bomb Lab 3$ ./bomb Bomb304_S20230010193_solution.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...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```