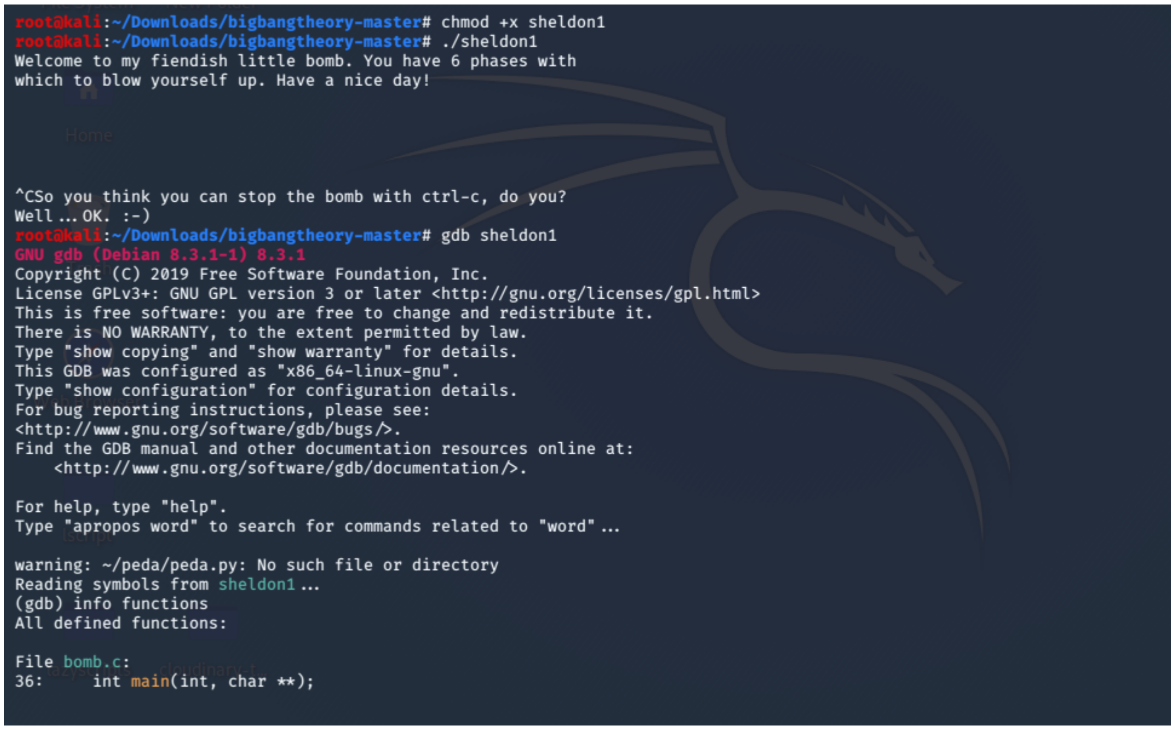
Sheldon1

Phase 1

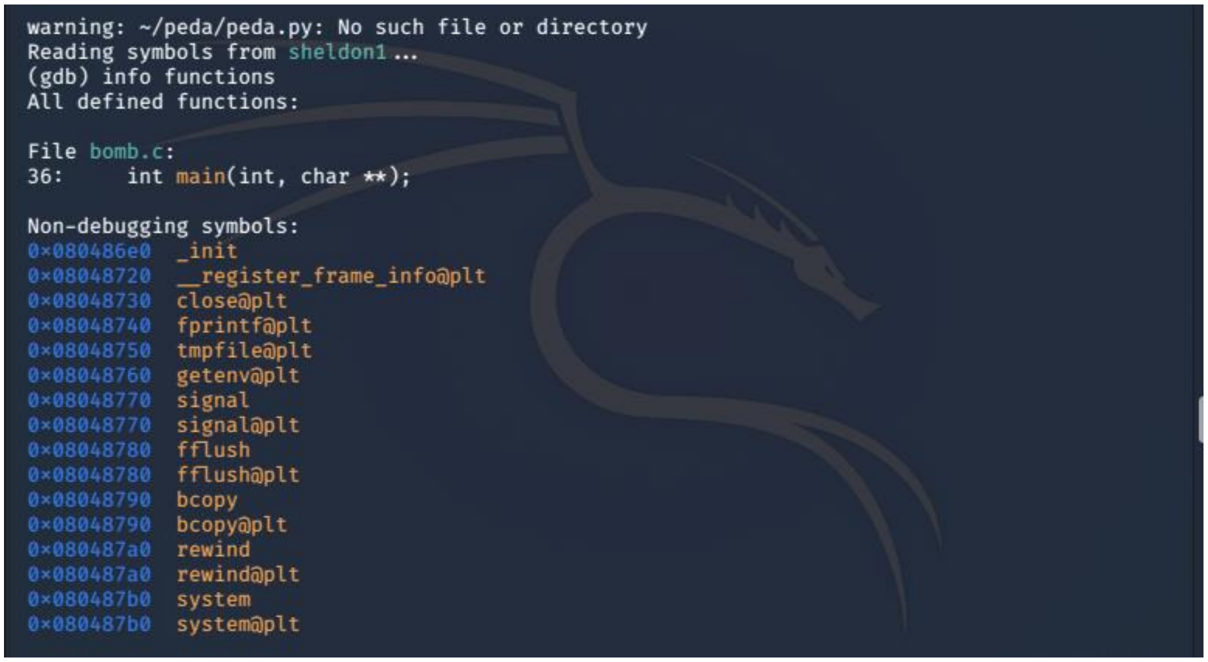
First run the sheldon1 file using " chmod +x sheldon1" command and "./sheldon1" command.

Then open it using gdb.

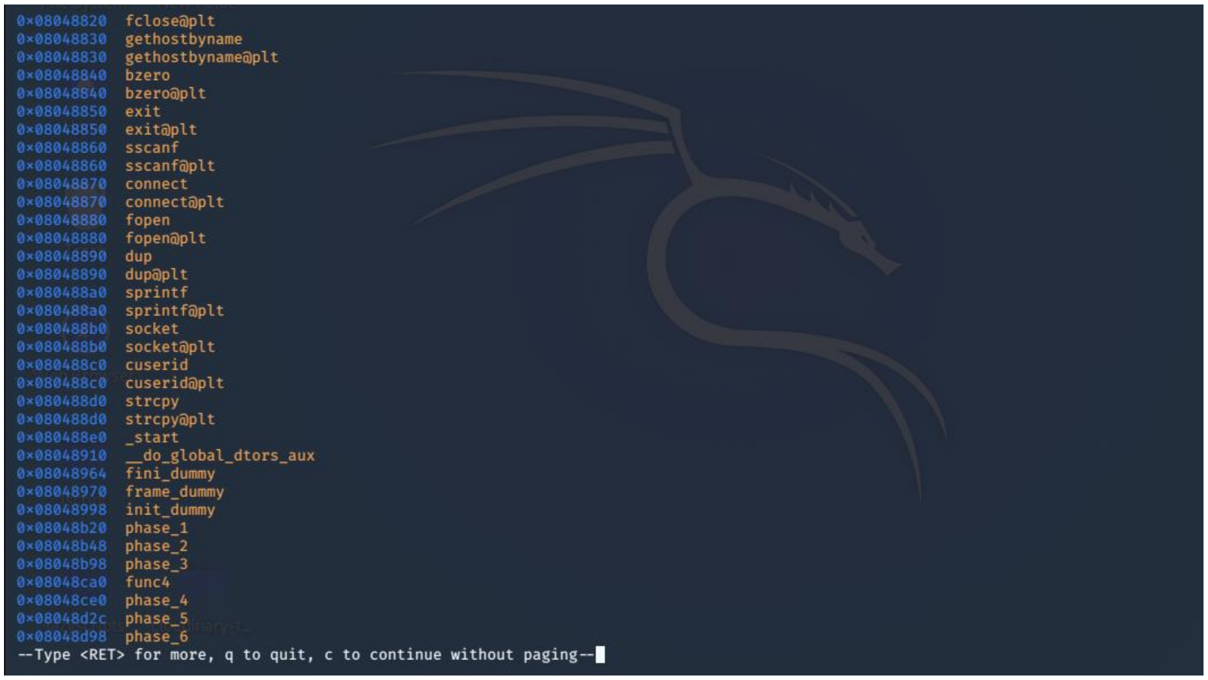


```
root@kali:~/Downloads/bigbangtheory-master# chmod +x sheldon1
root@kali:~/Downloads/bigbangtheory-master# ./sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!



^CSo you think you can stop the bomb with ctrl-c, do you?
Well...OK. :-)
root@kali:~/Downloads/bigbangtheory-master# gdb sheldon1
GNU gdb (Debian 8.3.1-1) 8.3.1
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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"...

warning: ~/peda/peda.py: No such file or directory
Reading symbols from sheldon1...
(gdb) info functions
All defined functions:

File bomb.c:
36:     int main(int, char **);
```

Look for the functions inside sheldon1 using the command "info functions". As we can see there are separate functions for each phase there.

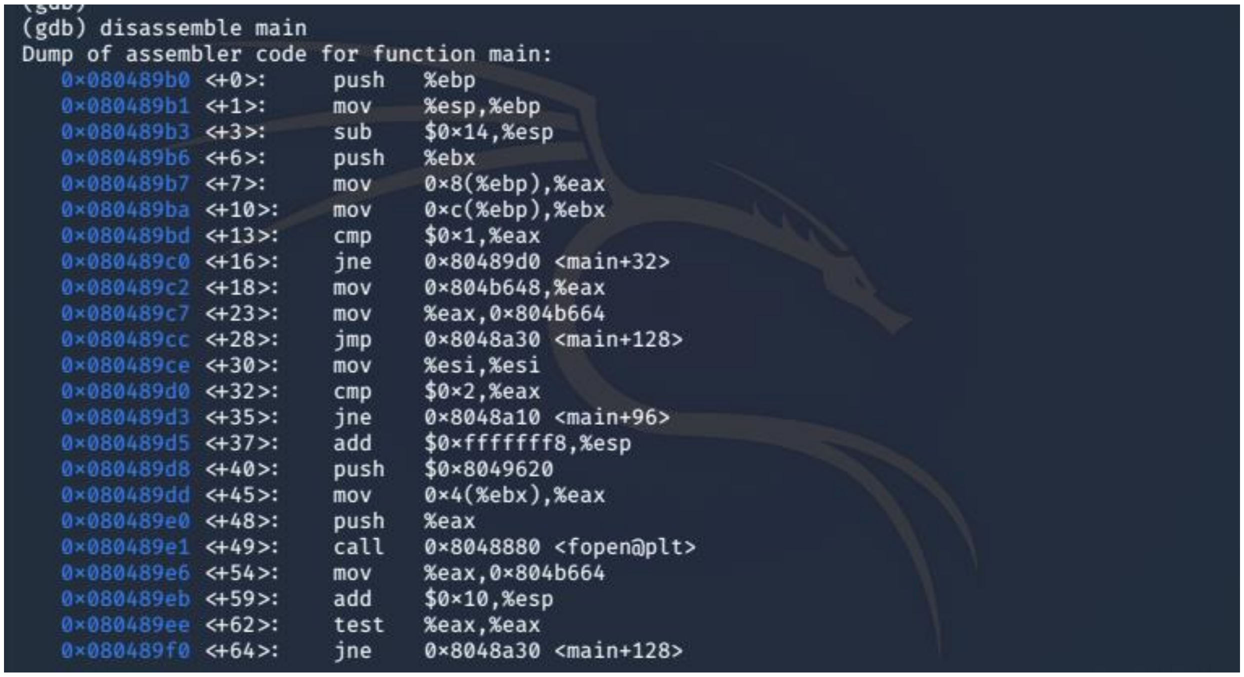


```
warning: ~/peda/peda.py: No such file or directory
Reading symbols from sheldon1...
(gdb) info functions
All defined functions:

File bomb.c:
36:     int main(int, char **);

Non-debugging symbols:
0x080486e0  _init
0x08048720  __register_frame_info@plt
0x08048730  close@plt
0x08048740  fprintf@plt
0x08048750  tmpfile@plt
0x08048760  getenv@plt
0x08048770  signal
0x08048770  signal@plt
0x08048780  fflush
0x08048780  fflush@plt
0x08048790  bcopy
0x08048790  bcopy@plt
0x080487a0  rewind
0x080487a0  rewind@plt
0x080487b0  system
0x080487b0  system@plt
```

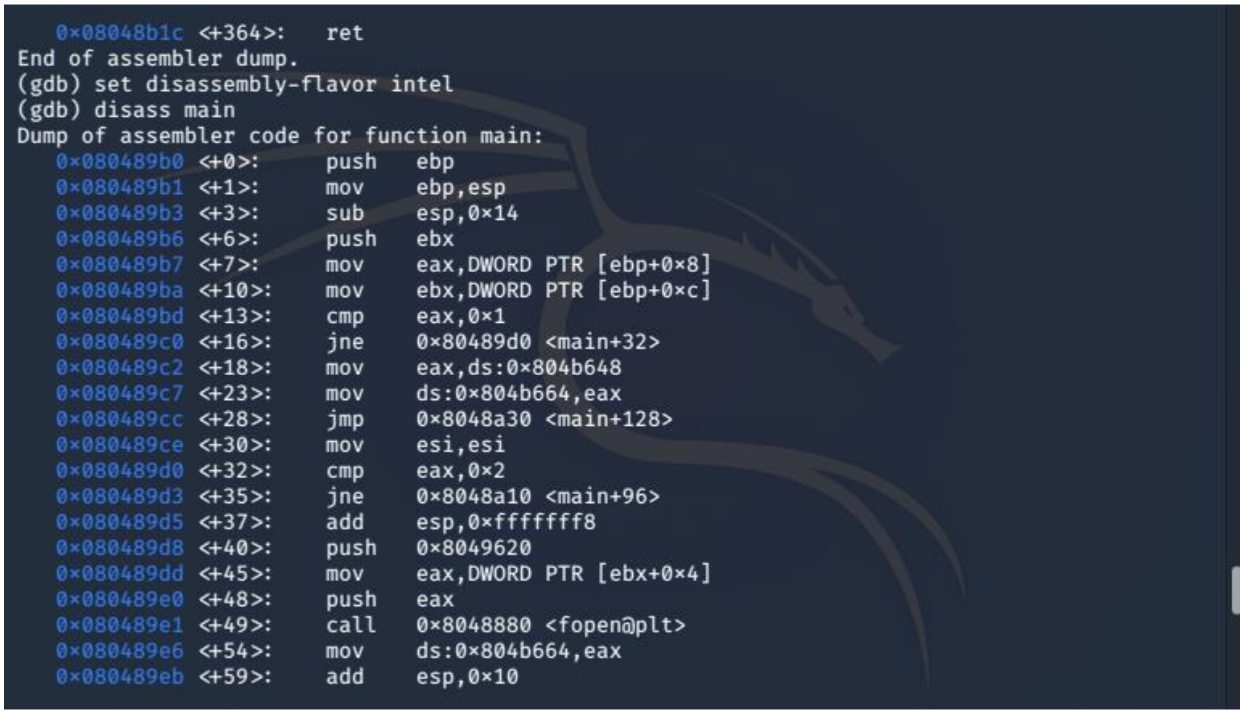


```
0x08048820  fclose@plt
0x08048830  gethostbyname
0x08048830  gethostbyname@plt
0x08048840  bzero
0x08048840  bzero@plt
0x08048850  exit
0x08048850  exit@plt
0x08048860  sscanf
0x08048860  sscanf@plt
0x08048870  connect
0x08048870  connect@plt
0x08048880  fopen
0x08048880  fopen@plt
0x08048890  dup
0x08048890  dup@plt
0x080488a0  sprintf
0x080488a0  sprintf@plt
0x080488b0  socket
0x080488b0  socket@plt
0x080488c0  cuserid
0x080488c0  cuserid@plt
0x080488d0  strcpy
0x080488d0  strcpy@plt
0x080488e0  _start
0x08048910  __do_global_dtors_aux
0x08048964  fini_dummy
0x08048970  frame_dummy
0x08048998  init_dummy
0x08048b20  phase_1
0x08048b48  phase_2
0x08048b98  phase_3
0x08048ca0  func4
0x08048ce0  phase_4
0x08048d2c  phase_5
0x08048d98  phase_6
--Type <RET> for more, q to quit, c to continue without paging--
```

Then view the assembly code of the main function using " disassemble main" command.

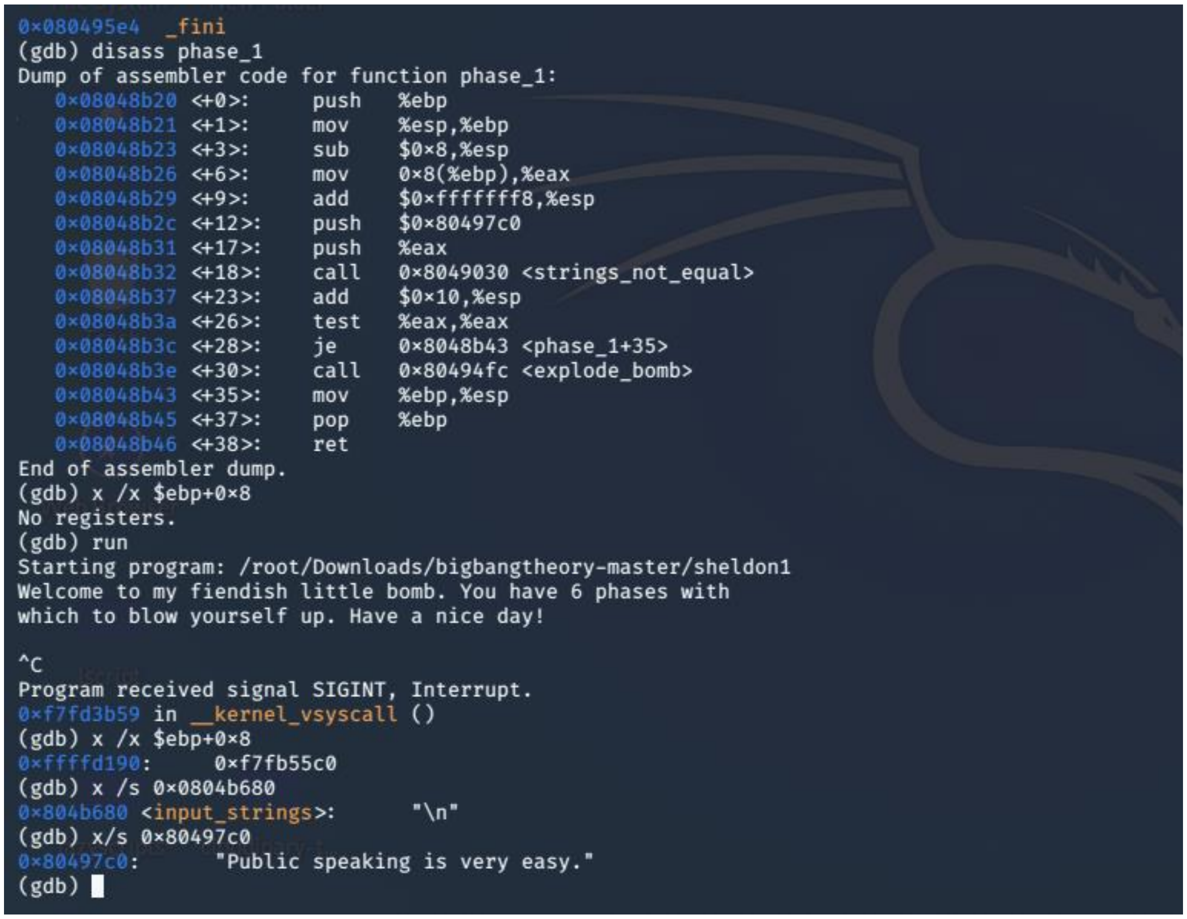


```
(gdb) disassemble main
Dump of assembler code for function main:
   0x080489b0 <+0>:     push   %ebp
   0x080489b1 <+1>:     mov    %esp,%ebp
   0x080489b3 <+3>:     sub    $0x14,%esp
   0x080489b6 <+6>:     push   %ebx
   0x080489b7 <+7>:     mov    0x8(%ebp),%eax
   0x080489ba <+10>:    mov    0xc(%ebp),%ebx
   0x080489bd <+13>:    cmp    $0x1,%eax
   0x080489c0 <+16>:    jne    0x80489d0 <main+32>
   0x080489c2 <+18>:    mov    0x804b648,%eax
   0x080489c7 <+23>:    mov    %eax,0x804b664
   0x080489cc <+28>:    jmp    0x8048a30 <main+128>
   0x080489ce <+30>:    mov    %esi,%esi
   0x080489d0 <+32>:    cmp    $0x2,%eax
   0x080489d3 <+35>:    jne    0x8048a10 <main+96>
   0x080489d5 <+37>:    add    $0xfffffff8,%esp
   0x080489d8 <+40>:    push   $0x8049620
   0x080489dd <+45>:    mov    0x4(%ebx),%eax
   0x080489e0 <+48>:    push   %eax
   0x080489e1 <+49>:    call   0x8048880 <fopen@plt>
   0x080489e6 <+54>:    mov    %eax,0x804b664
   0x080489eb <+59>:    add    $0x10,%esp
   0x080489ee <+62>:    test   %eax,%eax
   0x080489f0 <+64>:    jne    0x8048a30 <main+128>
```



```
   0x08048b1c <+364>:   ret
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disass main
Dump of assembler code for function main:
   0x080489b0 <+0>:     push   ebp
   0x080489b1 <+1>:     mov    ebp,esp
   0x080489b3 <+3>:     sub    esp,0x14
   0x080489b6 <+6>:     push   ebx
   0x080489b7 <+7>:     mov    eax,DWORD PTR [ebp+0x8]
   0x080489ba <+10>:    mov    ebx,DWORD PTR [ebp+0xc]
   0x080489bd <+13>:    cmp    eax,0x1
   0x080489c0 <+16>:    jne    0x80489d0 <main+32>
   0x080489c2 <+18>:    mov    eax,ds:0x804b648
   0x080489c7 <+23>:    mov    ds:0x804b664,eax
   0x080489cc <+28>:    jmp    0x8048a30 <main+128>
   0x080489ce <+30>:    mov    esi,esi
   0x080489d0 <+32>:    cmp    eax,0x2
   0x080489d3 <+35>:    jne    0x8048a10 <main+96>
   0x080489d5 <+37>:    add    esp,0xfffffff8
   0x080489d8 <+40>:    push   0x8049620
   0x080489dd <+45>:    mov    eax,DWORD PTR [ebx+0x4]
   0x080489e0 <+48>:    push   eax
   0x080489e1 <+49>:    call   0x8048880 <fopen@plt>
   0x080489e6 <+54>:    mov    ds:0x804b664,eax
   0x080489eb <+59>:    add    esp,0x10
```

View the assembly code of the phase_1 using “ disassemble phase_1” command.



```
0x080495e4  _fini
(gdb) disass phase_1
Dump of assembler code for function phase_1:
   0x08048b20 <+0>:     push   %ebp
   0x08048b21 <+1>:     mov    %esp,%ebp
   0x08048b23 <+3>:     sub    $0x8,%esp
   0x08048b26 <+6>:     mov    0x8(%ebp),%eax
   0x08048b29 <+9>:     add    $0xfffffff8,%esp
   0x08048b2c <+12>:    push   $0x80497c0
   0x08048b31 <+17>:    push   %eax
   0x08048b32 <+18>:    call   0x8049030 <strings_not_equal>
   0x08048b37 <+23>:    add    $0x10,%esp
   0x08048b3a <+26>:    test   %eax,%eax
   0x08048b3c <+28>:    je     0x8048b43 <phase_1+35>
   0x08048b3e <+30>:    call   0x80494fc <explode_bomb>
   0x08048b43 <+35>:    mov    %ebp,%esp
   0x08048b45 <+37>:    pop    %ebp
   0x08048b46 <+38>:    ret
End of assembler dump.
(gdb) x /x $ebp+0x8
No registers.
(gdb) run
Starting program: /root/Downloads/bigbangtheory-master/sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!

^C
Program received signal SIGINT, Interrupt.
0xf7fd3b59 in __kernel_vsyscall ()
(gdb) x /x $ebp+0x8
0xffffd190:     0xf7fb55c0
(gdb) x /s 0x0804b680
0x804b680 <input_strings>:      "\n"
(gdb) x/s 0x80497c0
0x80497c0:      "Public speaking is very easy."
(gdb) 
```

Here two strings are compared. First string is our input string for the first phase; second string is the password for phase_1.

Input :

(gdb) x /x $ebp+0x8

0xbffff440:   0xf7fb55c0

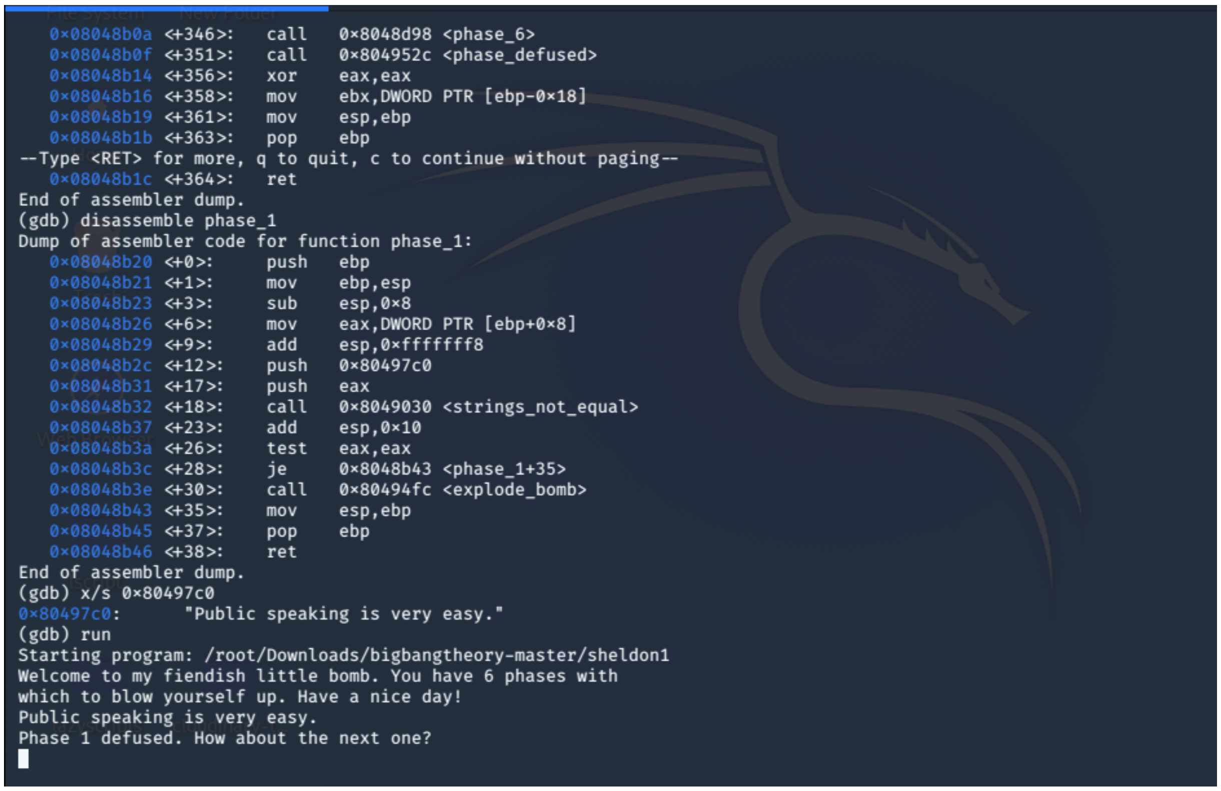
(gdb) x /s 0x0804b680

0x804b680 <input_strings>:   "\n"

Password for phase_1:

(gdb) x/s 0x80497c0

0x80497c0:   "Public speaking is very easy."

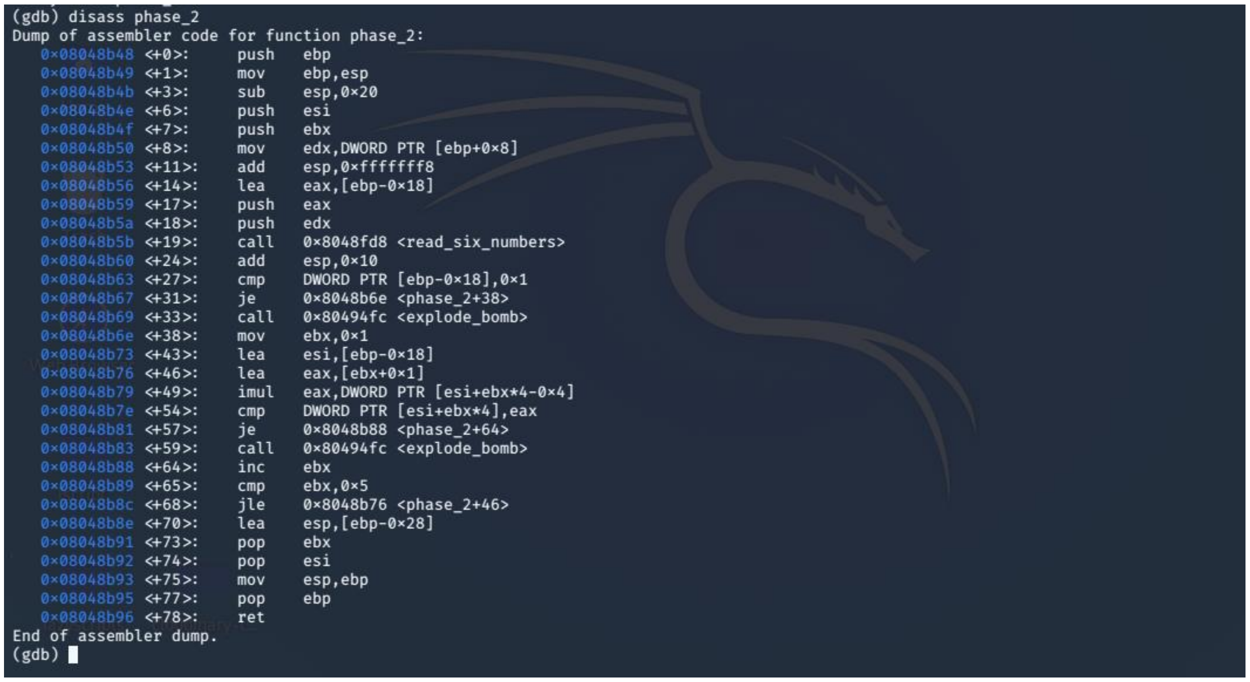


```
   0x08048b0a <+346>:   call   0x8048d98 <phase_6>
   0x08048b0f <+351>:   call   0x804952c <phase_defused>
   0x08048b14 <+356>:   xor    eax,eax
   0x08048b16 <+358>:   mov    ebx,DWORD PTR [ebp-0x18]
   0x08048b19 <+361>:   mov    esp,ebp
   0x08048b1b <+363>:   pop    ebp
--Type <RET> for more, q to quit, c to continue without paging--
   0x08048b1c <+364>:   ret
End of assembler dump.
(gdb) disassemble phase_1
Dump of assembler code for function phase_1:
   0x08048b20 <+0>:     push   ebp
   0x08048b21 <+1>:     mov    ebp,esp
   0x08048b23 <+3>:     sub    esp,0x8
   0x08048b26 <+6>:     mov    eax,DWORD PTR [ebp+0x8]
   0x08048b29 <+9>:     add    esp,0xfffffff8
   0x08048b2c <+12>:    push   0x80497c0
   0x08048b31 <+17>:    push   eax
   0x08048b32 <+18>:    call   0x8049030 <strings_not_equal>
   0x08048b37 <+23>:    add    esp,0x10
   0x08048b3a <+26>:    test   eax,eax
   0x08048b3c <+28>:    je     0x8048b43 <phase_1+35>
   0x08048b3e <+30>:    call   0x80494fc <explode_bomb>
   0x08048b43 <+35>:    mov    esp,ebp
   0x08048b45 <+37>:    pop    ebp
   0x08048b46 <+38>:    ret
End of assembler dump.
(gdb) x/s 0x80497c0
0x80497c0:      "Public speaking is very easy."
(gdb) run
Starting program: /root/Downloads/bigbangtheory-master/sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
```

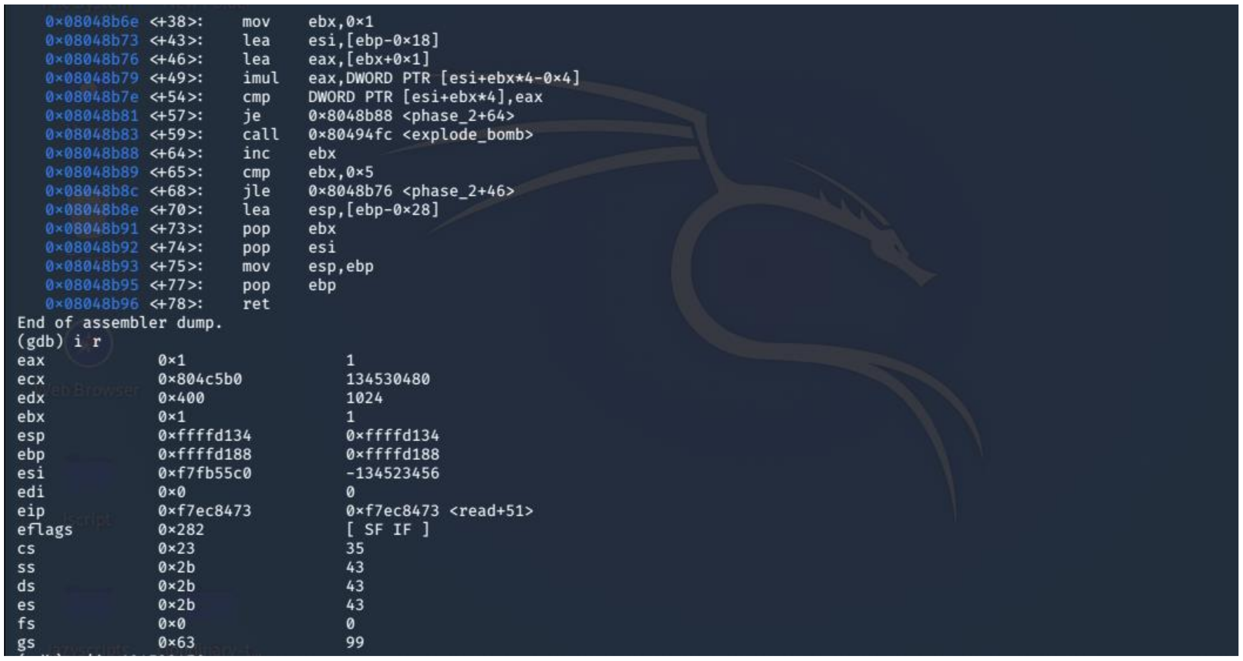
Passphase for phase_1 : Public speaking is very easy.

Phase 2

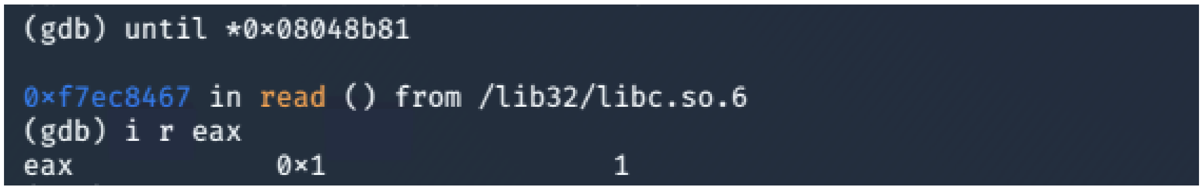
Assembly code for function phase_2:



```
(gdb) disass phase_2
Dump of assembler code for function phase_2:
   0x08048b48 <+0>:     push   ebp
   0x08048b49 <+1>:     mov    ebp,esp
   0x08048b4b <+3>:     sub    esp,0x20
   0x08048b4e <+6>:     push   esi
   0x08048b4f <+7>:     push   ebx
   0x08048b50 <+8>:     mov    edx,DWORD PTR [ebp+0x8]
   0x08048b53 <+11>:    add    esp,0xfffffff8
   0x08048b56 <+14>:    lea    eax,[ebp-0x18]
   0x08048b59 <+17>:    push   eax
   0x08048b5a <+18>:    push   edx
   0x08048b5b <+19>:    call   0x8048fd8 <read_six_numbers>
   0x08048b60 <+24>:    add    esp,0x10
   0x08048b63 <+27>:    cmp    DWORD PTR [ebp-0x18],0x1
   0x08048b67 <+31>:    je     0x8048b6e <phase_2+38>
   0x08048b69 <+33>:    call   0x80494fc <explode_bomb>
   0x08048b6e <+38>:    mov    ebx,0x1
   0x08048b73 <+43>:    lea    esi,[ebp-0x18]
   0x08048b76 <+46>:    lea    eax,[ebx+0x1]
   0x08048b79 <+49>:    imul   eax,DWORD PTR [esi+ebx*4-0x4]
   0x08048b7e <+54>:    cmp    DWORD PTR [esi+ebx*4],eax
   0x08048b81 <+57>:    je     0x8048b88 <phase_2+64>
   0x08048b83 <+59>:    call   0x80494fc <explode_bomb>
   0x08048b88 <+64>:    inc    ebx
   0x08048b89 <+65>:    cmp    ebx,0x5
   0x08048b8c <+68>:    jle    0x8048b76 <phase_2+46>
   0x08048b8e <+70>:    lea    esp,[ebp-0x28]
   0x08048b91 <+73>:    pop    ebx
   0x08048b92 <+74>:    pop    esi
   0x08048b93 <+75>:    mov    esp,ebp
   0x08048b95 <+77>:    pop    ebp
   0x08048b96 <+78>:    ret
End of assembler dump.
(gdb) 
```



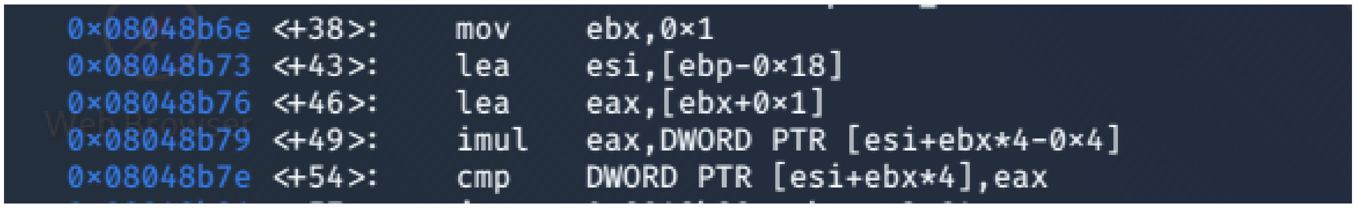
```
   0x08048b6e <+38>:    mov    ebx,0x1
   0x08048b73 <+43>:    lea    esi,[ebp-0x18]
   0x08048b76 <+46>:    lea    eax,[ebx+0x1]
   0x08048b79 <+49>:    imul   eax,DWORD PTR [esi+ebx*4-0x4]
   0x08048b7e <+54>:    cmp    DWORD PTR [esi+ebx*4],eax
   0x08048b81 <+57>:    je     0x8048b88 <phase_2+64>
   0x08048b83 <+59>:    call   0x80494fc <explode_bomb>
   0x08048b88 <+64>:    inc    ebx
   0x08048b89 <+65>:    cmp    ebx,0x5
   0x08048b8c <+68>:    jle    0x8048b76 <phase_2+46>
   0x08048b8e <+70>:    lea    esp,[ebp-0x28]
   0x08048b91 <+73>:    pop    ebx
   0x08048b92 <+74>:    pop    esi
   0x08048b93 <+75>:    mov    esp,ebp
   0x08048b95 <+77>:    pop    ebp
   0x08048b96 <+78>:    ret
End of assembler dump.
(gdb) i r
eax            0x1                 1
ecx            0x804c5b0           134530480
edx            0x400               1024
ebx            0x1                 1
esp            0xffffd134          0xffffd134
ebp            0xffffd188          0xffffd188
esi            0xf7fb55c0          -134523456
edi            0x0                 0
eip            0xf7ec8473          0xf7ec8473 <read+51>
eflags         0x282               [ SF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
```



```
(gdb) until *0×08048b81

0×f7ec8467 in read () from /lib32/libc.so.6
(gdb) i r eax
eax            0×1                 1
```

So the first integer must be 1.

6 numbers are read from our input, and put in a local array variable.



```
   0×08048b6e <+38>:    mov    ebx,0×1
   0×08048b73 <+43>:    lea    esi,[ebp-0×18]
   0×08048b76 <+46>:    lea    eax,[ebx+0×1]
   0×08048b79 <+49>:    imul   eax,DWORD PTR [esi+ebx*4-0×4]
   0×08048b7e <+54>:    cmp    DWORD PTR [esi+ebx*4],eax
```

0x08048b6e <+38>: mov ebx,0x1 //set ebx to 0x1

0x08048b73 <+43>:lea esi,[ebp-0x18] //set esi to the address of the first element of the array

0x08048b76 <+46>:lea eax, [ebx+0x1] //set eax to 0x2

0x08048b79 <+49>:imul eax,DWORD PTR [esi+ebx*4-0x4] // eax = eax * first number = 0x2

0x08048b7e <+54>: cmp DWORD PTR [esi+ebx*4], eax //compare eax (0x2) with the second number

After that, ebx is increased, which acts as an index into the numbers array, and we get back to:

0x8048b76 <phase_2+46>: lea eax, [ebx+0x1] //ebx is 0x2, sets eax to 0x3

0x8048b79 <phase_2+49>: imul eax,DWORD PTR [esi+ebx*4-0x4] //eax = eax * second number = 3*2 = 6

0x8048b7e <phase_2+54>: cmp DWORD PTR [esi+ebx*4], eax //expects the third element to be 6

0x8048b81 <phase_2+57>: je 0x8048b88 <phase_2+64>

0x8048b83 <phase_2+59>: call 0x80494fc <explode_bomb>

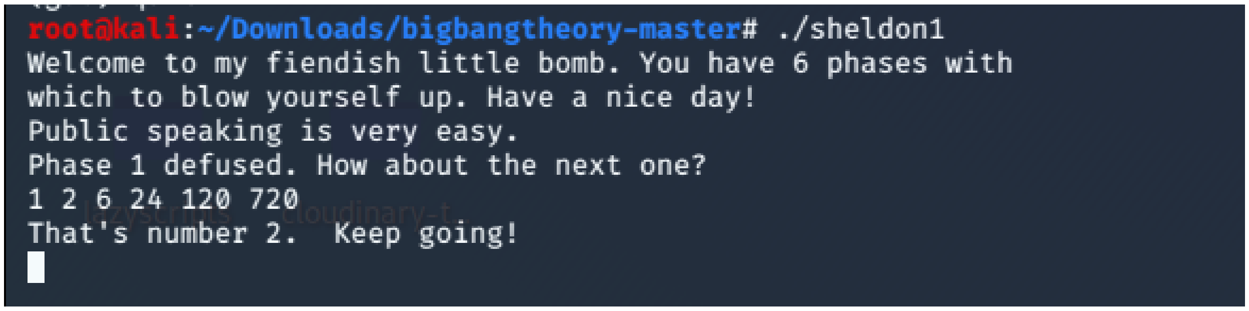
We figure out that the algorithm is as follows:

v[0] = 1

v[i] = (i+i) * v[i-1]

And so we find the solution of phase 2:

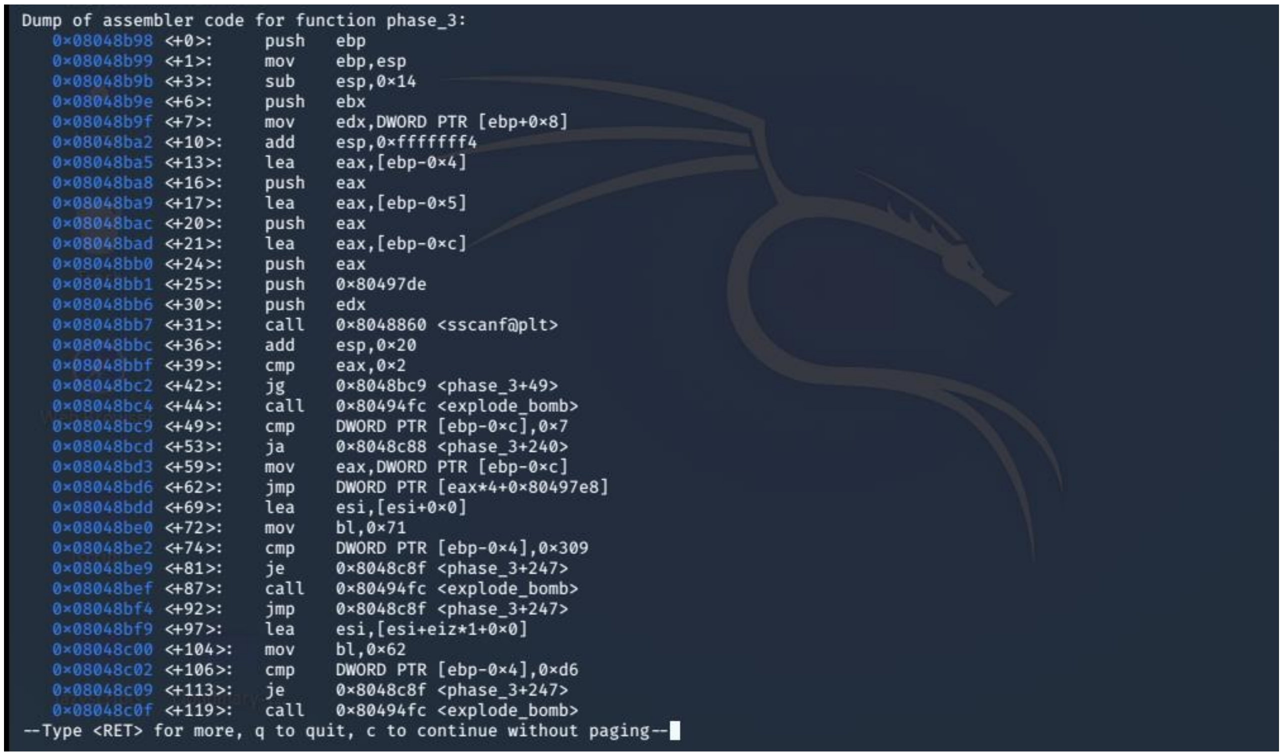
1 2 6 24 120 720



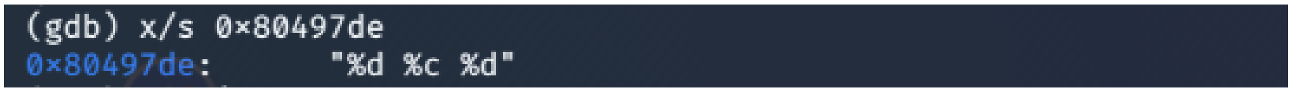
```
root@kali:~/Downloads/bigbangtheory-master# ./sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
1 2 6 24 120 720
That's number 2.  Keep going!
```

Phase 3

Assembly code for the function phase_3:



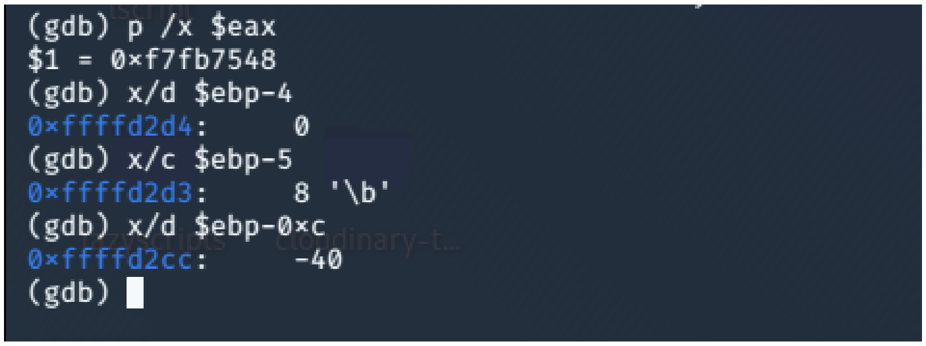
```
Dump of assembler code for function phase_3:
   0x08048b98 <+0>:     push   ebp
   0x08048b99 <+1>:     mov    ebp,esp
   0x08048b9b <+3>:     sub    esp,0x14
   0x08048b9e <+6>:     push   ebx
   0x08048b9f <+7>:     mov    edx,DWORD PTR [ebp+0x8]
   0x08048ba2 <+10>:    add    esp,0xfffffff4
   0x08048ba5 <+13>:    lea    eax,[ebp-0x4]
   0x08048ba8 <+16>:    push   eax
   0x08048ba9 <+17>:    lea    eax,[ebp-0x5]
   0x08048bac <+20>:    push   eax
   0x08048bad <+21>:    lea    eax,[ebp-0xc]
   0x08048bb0 <+24>:    push   eax
   0x08048bb1 <+25>:    push   0x80497de
   0x08048bb6 <+30>:    push   edx
   0x08048bb7 <+31>:    call   0x8048860 <sscanf@plt>
   0x08048bbc <+36>:    add    esp,0x20
   0x08048bbf <+39>:    cmp    eax,0x2
   0x08048bc2 <+42>:    jg     0x8048bc9 <phase_3+49>
   0x08048bc4 <+44>:    call   0x80494fc <explode_bomb>
   0x08048bc9 <+49>:    cmp    DWORD PTR [ebp-0xc],0x7
   0x08048bcd <+53>:    ja     0x8048c88 <phase_3+240>
   0x08048bd3 <+59>:    mov    eax,DWORD PTR [ebp-0xc]
   0x08048bd6 <+62>:    jmp    DWORD PTR [eax*4+0x80497e8]
   0x08048bdd <+69>:    lea    esi,[esi+0x0]
   0x08048be0 <+72>:    mov    bl,0x71
   0x08048be2 <+74>:    cmp    DWORD PTR [ebp-0x4],0x309
   0x08048be9 <+81>:    je     0x8048c8f <phase_3+247>
   0x08048bef <+87>:    call   0x80494fc <explode_bomb>
   0x08048bf4 <+92>:    jmp    0x8048c8f <phase_3+247>
   0x08048bf9 <+97>:    lea    esi,[esi+eiz*1+0x0]
   0x08048c00 <+104>:   mov    bl,0x62
   0x08048c02 <+106>:   cmp    DWORD PTR [ebp-0x4],0xd6
   0x08048c09 <+113>:   je     0x8048c8f <phase_3+247>
   0x08048c0f <+119>:   call   0x80494fc <explode_bomb>
--Type <RET> for more, q to quit, c to continue without paging--
```



```
(gdb) x/s 0x80497de
0x80497de:      "%d %c %d"
```

So the password should look like this : “an integer”- “a character”- “an integer”

I input the passphrase as “1 a 7”. Results:



```
(gdb) p /x $eax
$1 = 0xf7fb7548
(gdb) x/d $ebp-4
0xffffd2d4:     0
(gdb) x/c $ebp-5
0xffffd2d3:     8 '\b'
(gdb) x/d $ebp-0xc
0xffffd2cc:     -40
(gdb) 
```

First condition to not explode the bomb: we have to fill all the 3 variables passed to sscanf.

```
   0x08048bbf <+39>: cmp    eax,0x2
   0x08048bc2 <+42>: jg     0x8048bc9 <phase_3+49>
   0x08048bc4 <+44>: call   0x80494fc <explode_bomb>
```

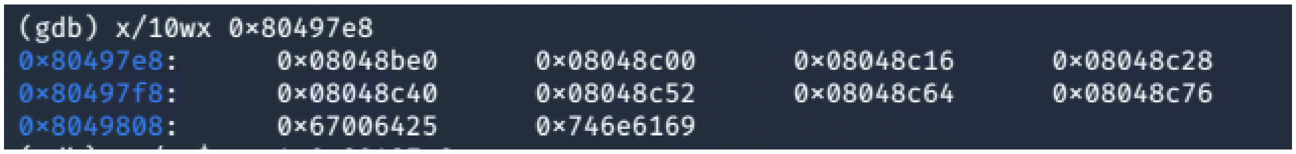
Second condition: first number must be <=7

```
   0x08048bc9 <+49>: cmp    DWORD PTR [ebp-0xc],0x7
   0x08048bcd <+53>: ja     0x8048c88 <phase_3+240>
```

The last part of the function looks like a case structure. We have the following table of addresses:

```
0x08048bd6 <+62>:   jmp    DWORD PTR [eax*4+0x80497e8]
```

In $eax we have the first number, which we chose as 7.



```
(gdb) x/10wx 0x80497e8
0x80497e8:      0x08048be0      0x08048c00      0x08048c16      0x08048c28
0x80497f8:      0x08048c40      0x08048c52      0x08048c64      0x08048c76
0x8049808:      0x67006425      0x746e6169
```

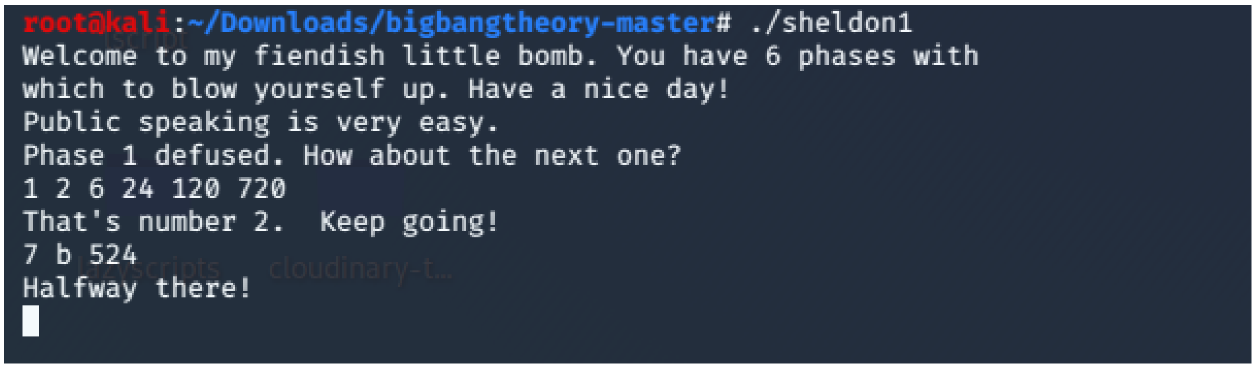
In our case, when the first parameter was 7, we'll jump to 0x08048c76.

```
(gdb) x /x $eax*4+0x80497e8
0x8049804:     0x08048c76
```

```
0x08048c76 <+222>: mov    bl,0x62                                  //ascii letter 'b'
0x08048c78 <+224>: cmp    DWORD PTR [ebp-0x4],0x20c                //524 in decimal
0x08048c7f <+231>: je     0x8048c8f <phase_3+247>
0x08048c81 <+233>: call   0x80494fc <explode_bomb>
0x08048c86 <+238>: jmp    0x8048c8f <phase_3+247>
```

Password for phase_3:

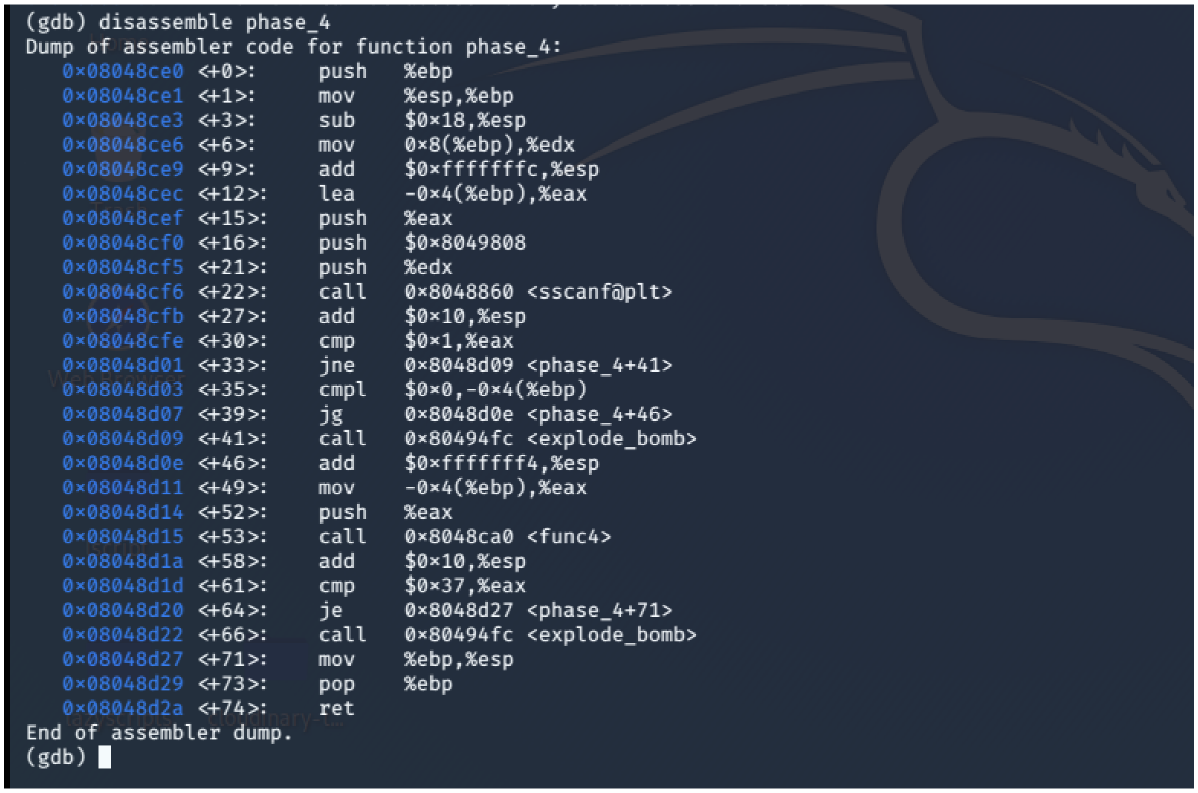
7 b 524



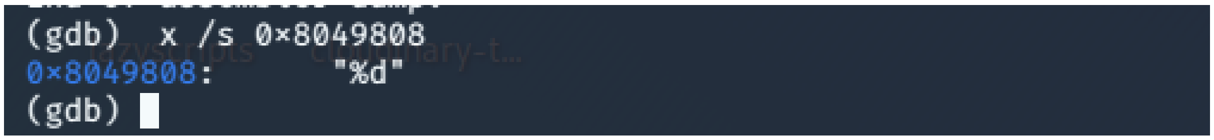
```
root@kali:~/Downloads/bigbangtheory-master# ./sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
1 2 6 24 120 720
That's number 2.  Keep going!
7 b 524
Halfway there!
```

Phase 4

Assembly code for function phase_4:



```
(gdb) disassemble phase_4
Dump of assembler code for function phase_4:
   0x08048ce0 <+0>:     push   %ebp
   0x08048ce1 <+1>:     mov    %esp,%ebp
   0x08048ce3 <+3>:     sub    $0x18,%esp
   0x08048ce6 <+6>:     mov    0x8(%ebp),%edx
   0x08048ce9 <+9>:     add    $0xfffffffc,%esp
   0x08048cec <+12>:    lea    -0x4(%ebp),%eax
   0x08048cef <+15>:    push   %eax
   0x08048cf0 <+16>:    push   $0x8049808
   0x08048cf5 <+21>:    push   %edx
   0x08048cf6 <+22>:    call   0x8048860 <sscanf@plt>
   0x08048cfb <+27>:    add    $0x10,%esp
   0x08048cfe <+30>:    cmp    $0x1,%eax
   0x08048d01 <+33>:    jne    0x8048d09 <phase_4+41>
   0x08048d03 <+35>:    cmpl   $0x0,-0x4(%ebp)
   0x08048d07 <+39>:    jg     0x8048d0e <phase_4+46>
   0x08048d09 <+41>:    call   0x80494fc <explode_bomb>
   0x08048d0e <+46>:    add    $0xfffffff4,%esp
   0x08048d11 <+49>:    mov    -0x4(%ebp),%eax
   0x08048d14 <+52>:    push   %eax
   0x08048d15 <+53>:    call   0x8048ca0 <func4>
   0x08048d1a <+58>:    add    $0x10,%esp
   0x08048d1d <+61>:    cmp    $0x37,%eax
   0x08048d20 <+64>:    je     0x8048d27 <phase_4+71>
   0x08048d22 <+66>:    call   0x80494fc <explode_bomb>
   0x08048d27 <+71>:    mov    %ebp,%esp
   0x08048d29 <+73>:    pop    %ebp
   0x08048d2a <+74>:    ret
End of assembler dump.
(gdb) 
```



```
(gdb)  x /s 0x8049808
0x8049808:      "%d"
(gdb) 
```

This means the answer to the phase_4 should be a single integer.

First defusing condition:

One parameter must be read:

```
0x08048cfe <+30>: cmp     eax,0x1
0x08048d01 <+33>:jne      0x8048d09 <phase_4+41>
```

Second condition,

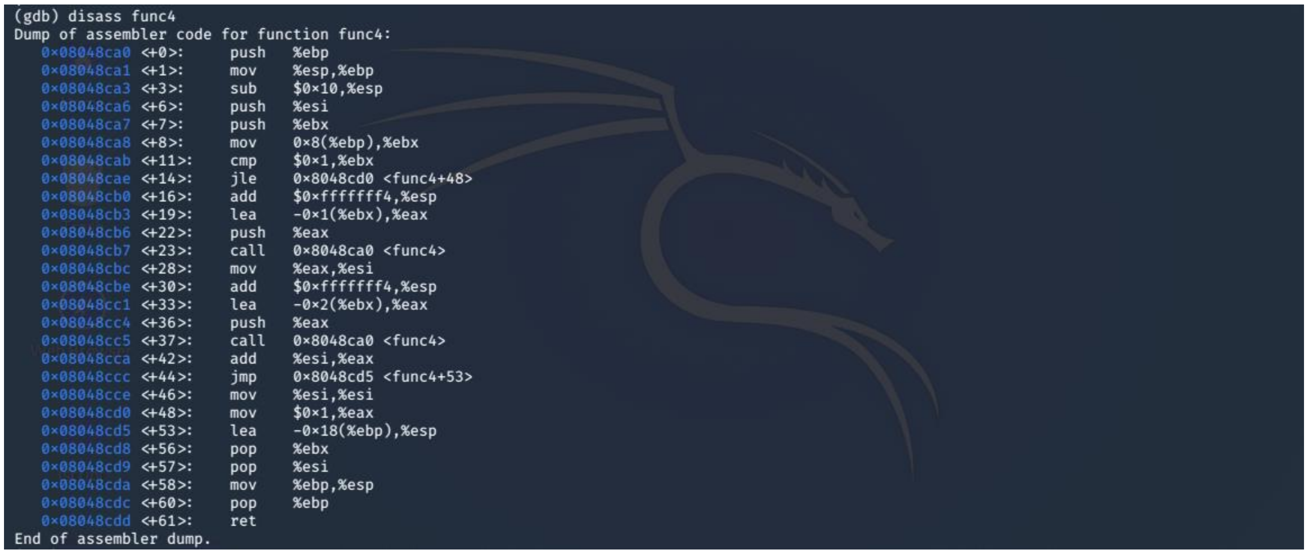
must be greater than 0:

```
   0x08048d03 <+35>:cmp    DWORD PTR [ebp-0x4],0x0
   0x08048d07 <+39>:jg       0x8048d0e <phase_4+46>
```

Then this parameter is passed to func4:

```
   0x08048d11 <+49>:mov     eax, DWORD PTR [ebp-0x4]
   0x08048d14 <+52>:push    eax
   0x08048d15 <+53>:call     0x8048ca0 <func4>
```

Assembly code for fucntion func4:



```
(gdb) disass func4
Dump of assembler code for function func4:
   0×08048ca0 <+0>:     push   %ebp
   0×08048ca1 <+1>:     mov    %esp,%ebp
   0×08048ca3 <+3>:     sub    $0×10,%esp
   0×08048ca6 <+6>:     push   %esi
   0×08048ca7 <+7>:     push   %ebx
   0×08048ca8 <+8>:     mov    0×8(%ebp),%ebx
   0×08048cab <+11>:    cmp    $0×1,%ebx
   0×08048cae <+14>:    jle    0×8048cd0 <func4+48>
   0×08048cb0 <+16>:    add    $0×fffffff4,%esp
   0×08048cb3 <+19>:    lea    -0×1(%ebx),%eax
   0×08048cb6 <+22>:    push   %eax
   0×08048cb7 <+23>:    call   0×8048ca0 <func4>
   0×08048cbc <+28>:    mov    %eax,%esi
   0×08048cbe <+30>:    add    $0×fffffff4,%esp
   0×08048cc1 <+33>:    lea    -0×2(%ebx),%eax
   0×08048cc4 <+36>:    push   %eax
   0×08048cc5 <+37>:    call   0×8048ca0 <func4>
   0×08048cca <+42>:    add    %esi,%eax
   0×08048ccc <+44>:    jmp    0×8048cd5 <func4+53>
   0×08048cce <+46>:    mov    %esi,%esi
   0×08048cd0 <+48>:    mov    $0×1,%eax
   0×08048cd5 <+53>:    lea    -0×18(%ebp),%esp
   0×08048cd8 <+56>:    pop    %ebx
   0×08048cd9 <+57>:    pop    %esi
   0×08048cda <+58>:    mov    %ebp,%esp
   0×08048cdc <+60>:    pop    %ebp
   0×08048cdd <+61>:    ret
End of assembler dump.
```

The next condition is that the func4 should return 0x37

```
   0x08048d1d <+61>:cmp    eax,0x37
   0x08048d20 <+64>:je     0x8048d27 <phase_4+71>
```

In func4, if argument <=1, the return value is 0x1:

```
   0x08048cab <+11>: cmp    ebx,0x1
   0x08048cae <+14>: jle    0x8048cd0 <func4+48>
...
   0x08048cd0 <+48>: mov    eax,0x1
...
```

Func4 :

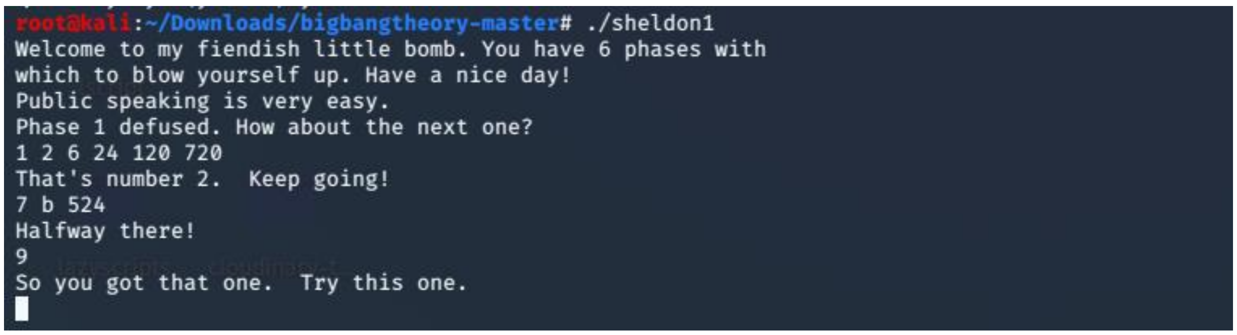
```
func4(x):
    if x <= 1 :
        return 1
    else :
        y = func4(x-1)
        z = func4(x-2)
        return y + z
```

It's the Fibonacci function, implemented recursively. We quickly convert it to python code:

```
#!/usr/bin/python

def func4(x):
    if x <= 1 :
        return 1
    else :
        y = func4(x-1)
        z = func4(x-2)
        return y + z

if __name__ == "__main__":
    print func4(9)
```

We're expecting 0x37, which is 55 decimal, which is Fibonacci(9).

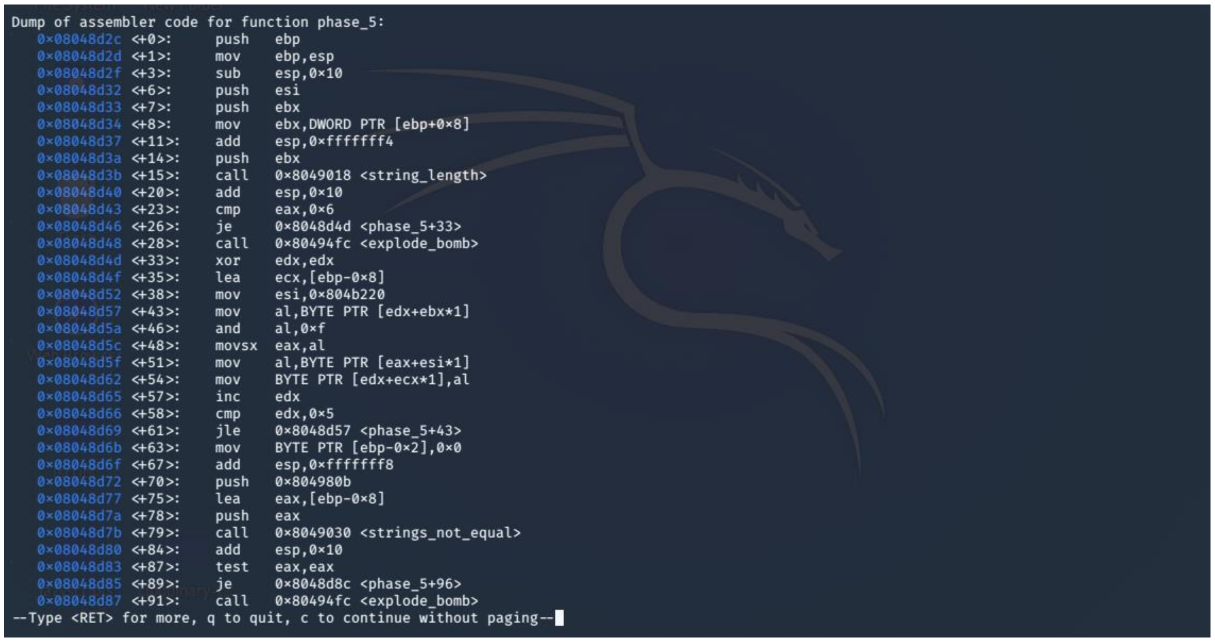
Answer to phase_4 : 9



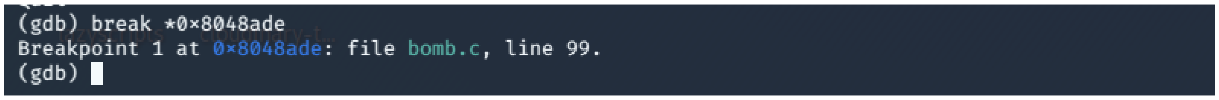
```
root@kali:~/Downloads/bigbangtheory-master# ./sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
1 2 6 24 120 720
That's number 2.  Keep going!
7 b 524
Halfway there!
9
So you got that one.  Try this one.
```

Phase_5

Assembly code for function phase_5 :



```
Dump of assembler code for function phase_5:
   0x08048d2c <+0>:     push   ebp
   0x08048d2d <+1>:     mov    ebp,esp
   0x08048d2f <+3>:     sub    esp,0x10
   0x08048d32 <+6>:     push   esi
   0x08048d33 <+7>:     push   ebx
   0x08048d34 <+8>:     mov    ebx,DWORD PTR [ebp+0x8]
   0x08048d37 <+11>:    add    esp,0xfffffff4
   0x08048d3a <+14>:    push   ebx
   0x08048d3b <+15>:    call   0x8049018 <string_length>
   0x08048d40 <+20>:    add    esp,0x10
   0x08048d43 <+23>:    cmp    eax,0x6
   0x08048d46 <+26>:    je     0x8048d4d <phase_5+33>
   0x08048d48 <+28>:    call   0x80494fc <explode_bomb>
   0x08048d4d <+33>:    xor    edx,edx
   0x08048d4f <+35>:    lea    ecx,[ebp-0x8]
   0x08048d52 <+38>:    mov    esi,0x804b220
   0x08048d57 <+43>:    mov    al,BYTE PTR [edx+ebx*1]
   0x08048d5a <+46>:    and    al,0xf
   0x08048d5c <+48>:    movsx  eax,al
   0x08048d5f <+51>:    mov    al,BYTE PTR [eax+esi*1]
   0x08048d62 <+54>:    mov    BYTE PTR [edx+ecx*1],al
   0x08048d65 <+57>:    inc    edx
   0x08048d66 <+58>:    cmp    edx,0x5
   0x08048d69 <+61>:    jle    0x8048d57 <phase_5+43>
   0x08048d6b <+63>:    mov    BYTE PTR [ebp-0x2],0x0
   0x08048d6f <+67>:    add    esp,0xfffffff8
   0x08048d72 <+70>:    push   0x804980b
   0x08048d77 <+75>:    lea    eax,[ebp-0x8]
   0x08048d7a <+78>:    push   eax
   0x08048d7b <+79>:    call   0x8049030 <strings_not_equal>
   0x08048d80 <+84>:    add    esp,0x10
   0x08048d83 <+87>:    test   eax,eax
   0x08048d85 <+89>:    je     0x8048d8c <phase_5+96>
   0x08048d87 <+91>:    call   0x80494fc <explode_bomb>
--Type <RET> for more, q to quit, c to continue without paging--
```



```
(gdb) break *0x8048ade
Breakpoint 1 at 0x8048ade: file bomb.c, line 99.
(gdb)
```

The first condition to diffuse phase_5;

the length of the password must be 6:

   0x08048d43 <+23>:cmp   eax,0x6

   0x08048d46 <+26>:je   0x8048d4d <phase_5+33>

We have to form the password 'giants' from the source string "isrveawhobpnutfg".

phase_5 function:

```
phase_5(s) {
        src = "isrveawhobpnutfg"
        dest = "12345"

        for (i=0; i<=5; ++i) {
                idx = s[i] && 0xf
                dest[i] = src[idx]
        }
}
```

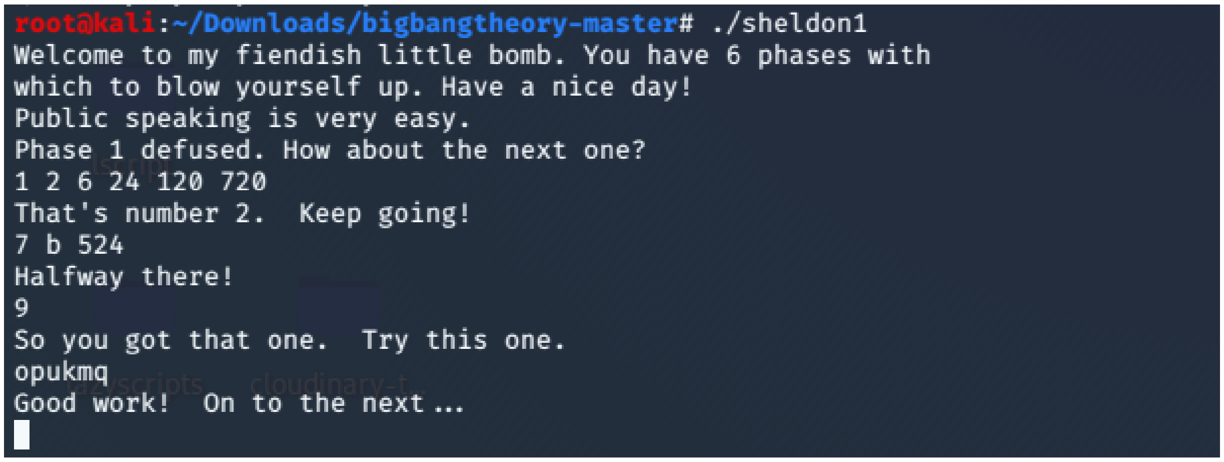
The first hex digit of the password represents the index.

So we need the following indexes:

15 (0xf), 0, 5, 11 (0xb), 13 (0xd) and 1. A possible password is be:

o (0x6f) p (0x70) u (0x75) k (0x6b) m (0x6d) q (0x71)

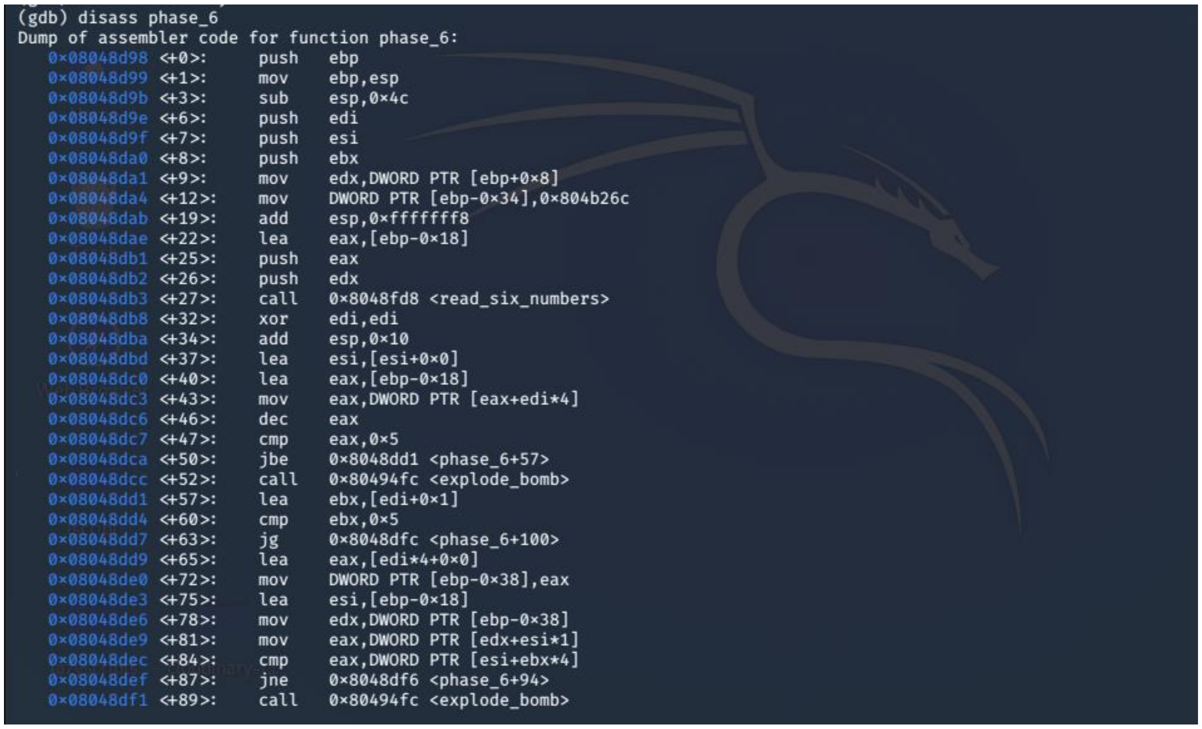
Passphrase: opukmq



```
root@kali:~/Downloads/bigbangtheory-master# ./sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
1 2 6 24 120 720
That's number 2.  Keep going!
7 b 524
Halfway there!
9
So you got that one.  Try this one.
opukmq
Good work!  On to the next...
```

Phase 6

Assembly code for function phase_6:



```
(gdb) disass phase_6
Dump of assembler code for function phase_6:
   0x08048d98 <+0>:     push   ebp
   0x08048d99 <+1>:     mov    ebp,esp
   0x08048d9b <+3>:     sub    esp,0x4c
   0x08048d9e <+6>:     push   edi
   0x08048d9f <+7>:     push   esi
   0x08048da0 <+8>:     push   ebx
   0x08048da1 <+9>:     mov    edx,DWORD PTR [ebp+0x8]
   0x08048da4 <+12>:    mov    DWORD PTR [ebp-0x34],0x804b26c
   0x08048dab <+19>:    add    esp,0xfffffff8
   0x08048dae <+22>:    lea    eax,[ebp-0x18]
   0x08048db1 <+25>:    push   eax
   0x08048db2 <+26>:    push   edx
   0x08048db3 <+27>:    call   0x8048fd8 <read_six_numbers>
   0x08048db8 <+32>:    xor    edi,edi
   0x08048dba <+34>:    add    esp,0x10
   0x08048dbd <+37>:    lea    esi,[esi+0x0]
   0x08048dc0 <+40>:    lea    eax,[ebp-0x18]
   0x08048dc3 <+43>:    mov    eax,DWORD PTR [eax+edi*4]
   0x08048dc6 <+46>:    dec    eax
   0x08048dc7 <+47>:    cmp    eax,0x5
   0x08048dca <+50>:    jbe    0x8048dd1 <phase_6+57>
   0x08048dcc <+52>:    call   0x80494fc <explode_bomb>
   0x08048dd1 <+57>:    lea    ebx,[edi+0x1]
   0x08048dd4 <+60>:    cmp    ebx,0x5
   0x08048dd7 <+63>:    jg     0x8048dfc <phase_6+100>
   0x08048dd9 <+65>:    lea    eax,[edi*4+0x0]
   0x08048de0 <+72>:    mov    DWORD PTR [ebp-0x38],eax
   0x08048de3 <+75>:    lea    esi,[ebp-0x18]
   0x08048de6 <+78>:    mov    edx,DWORD PTR [ebp-0x38]
   0x08048de9 <+81>:    mov    eax,DWORD PTR [edx+esi*1]
   0x08048dec <+84>:    cmp    eax,DWORD PTR [esi+ebx*4]
   0x08048def <+87>:    jne    0x8048df6 <phase_6+94>
   0x08048df1 <+89>:    call   0x80494fc <explode_bomb>
```

Again there’s a <read_six_numbers> function. So the passphrase should be six integers.

This phase also reads 6 numbers into a local array. With the input as "1 2 3 4 5 6",

after the call to read_six_numbers function, we have the numbers in $ebp-0x18:

```
(gdb) x /6w $ebp-0x18
0xbffff3e0:     0x00000001    0x00000002    0x00000003    0x00000004
0xbffff3f0:     0x00000005    0x00000006
```

From the annotated disassemby below, it seems that this phase has more stages, and has a very important

input, a linked list:

- stage1: check that all 6 numbers are between [1,..,6] and all different
- stage2: builds and arranges a second array with pointers to list elements

- stage3: fixes the links between elements from the input list to match the array constructed in stage2

- stage4: check that the elements of the linked list are in reverse sorted order.

```
#edi = i, ebx = j;
        i = 0
  esi = 0
  0x08048db8 <+32>:xor    edi,edi
  0x08048dba <+34>:add    esp,0x10
  0x08048dbd <+37>:lea    esi,[esi+0x0]

  while (i <= 5) :
  0x08048dc0 <+40>:lea    eax,[ebp-0x18] --> ebp-0x18 is the first number read
  0x08048dc3 <+43>:mov    eax,DWORD PTR [eax+edi*4]
  0x08048dc6 <+46>:dec    eax
  eax = &v1[0]
  eax = v1[edi]
  eax --
  0x08048dc7 <+47>:cmp    eax,0x5          --> all numbers should be > 0 and <= 6
  0x08048dca <+50>:jbe    0x8048dd1 <phase_6+57>
  0x08048dcc <+52>:call   0x80494fc <explode_bomb>


  j = i + 1
  while j <= 5 :
          edx = tmp
          if (v[i] == v[j]):
                  explode()
          j += 1
```

(The first stage seems to be that all elements are distinct. Update breakpoint to check 2 0x08048e02)

```
0x08048dd1 <+57>:lea    ebx,[edi+0x1]
0x08048dd4 <+60>:cmp    ebx,0x5
0x08048dd7 <+63>:jg     0x8048dfc <phase_6+100>
0x08048dd9 <+65>:lea    eax,[edi*4+0x0]
0x08048de0 <+72>:mov    DWORD PTR [ebp-0x38],eax
0x08048de3 <+75>:lea    esi,[ebp-0x18]

0x08048de6 <+78>:mov    edx,DWORD PTR [ebp-0x38]
0x08048de9 <+81>:mov    eax,DWORD PTR [edx+esi*1]
0x08048dec <+84>: cmp    eax,DWORD PTR [esi+ebx*4]
0x08048def <+87>: jne    0x8048df6 <phase_6+94>
0x08048df1 <+89>: call   0x80494fc <explode_bomb>
0x08048df6 <+94>: inc    ebx
0x08048df7 <+95>: cmp    ebx,0x5
0x08048dfa <+98>: jle    0x8048de6 <phase_6+78>

i += 1
0x08048dfc <+100>:          inc    edi
0x08048dfd <+101>:          cmp    edi,0x5
0x08048e00 <+104>:          jle    0x8048dc0 <phase_6+40>



// 2nd stage
        i = 0
        ecx = v[0]
```

```
eax = v2
y = v2
while i<=5 :
    elem = list_head
    elem = head
    j = 1
    edx = i
    if (j < v[i] ):
        do {
            elem = elem.next
            j ++
        }while (j < v[i])
    v2[i] = elem
    i++
```

In this stage we have to arrange the values of the list elements,

so that we can pass stage 4 (should be in reverse order).

Current order:

```
(gdb) printf "%08x %08x %08x %08x %08x %08x \n", *0x0804b26c, *0x0804b260,
*0x0804b254, *0x0804b248, *0x0804b23c, *0x0804b230
000000fd 000002d5 0000012d 000003e5 000000d4 000001b0
```

1 4 2 5 0 3

This stage builds a list of pointers to elements, which is used in stage 3 and 4.

Using the previously deduced agorithm, the input numbers (0<n<=1),

which mean how much we move an element, to have them in reverse order, should be:

pos 1: 3 (head->next->next->next which is the biggest num)

pos 2: 1 (head->next, which is the second biggest) pos 3: 5

pos 4: 2

pos 5: 0

pos 6: 4

Because of the advancing algorithm, we add 1 to the previous, and get: 4 2 6 3 1 5

```
   0x08048e02 <+106>:    xor    edi,edi
   0x08048e04 <+108>:    lea    ecx,[ebp-0x18]
   0x08048e07 <+111>:    lea    eax,[ebp-0x30]
   0x08048e0a <+114>:    mov    DWORD PTR [ebp-0x3c],eax
   0x08048e0d <+117>:    lea    esi,[esi+0x0]
   0x08048e10 <+120>:    mov    esi,DWORD PTR [ebp-0x34]
   0x08048e13 <+123>:    mov    ebx,0x1
   0x08048e18 <+128>:    lea    eax,[edi*4+0x0]
   0x08048e1f <+135>:    mov    edx,eax
   0x08048e21 <+137>:    cmp    ebx,DWORD PTR [eax+ecx*1]
   0x08048e24 <+140>:    jge    0x8048e38 <phase_6+160>
   0x08048e26 <+142>:    mov    eax,DWORD PTR [edx+ecx*1]
   0x08048e29 <+145>:    lea    esi,[esi+eiz*1+0x0]
   0x08048e30 <+152>:    mov    esi,DWORD PTR [esi+0x8]
   0x08048e33 <+155>:    inc    ebx
   0x08048e34 <+156>:    cmp    ebx,eax
   0x08048e36 <+158>:    jl     0x8048e30 <phase_6+152>
   0x08048e38 <+160>:    mov    edx,DWORD PTR [ebp-0x3c]
   0x08048e3b <+163>:    mov    DWORD PTR [edx+edi*4],esi
   0x08048e3e <+166>:    inc    edi
   0x08048e3f <+167>:    cmp    edi,0x5
   0x08048e42 <+170>:          jle    0x8048e10 <phase_6+120>

   // 3rd stage
   0x08048e44 <+172>:          mov    esi,DWORD PTR [ebp-0x30]
   0x08048e47 <+175>:          mov    DWORD PTR [ebp-0x34],esi

(gdb) x /x $ebp-0x30
0xbffff3c8:     0x0804b26c
(gdb) x /x $ebp-0x34
0xbffff3c4:     0x0804b26c
(gdb) p /x $esi
$11 = 0x804b26c
```

```
   i = 1
   esi = curr_elem = list_head
   edx = *($ebp-0x30)     // array with list elements addresses
   while i<= 5:
         // the second array contains the addresses of list elements
         (gdb) x/6x $ebp-0x30
               0xbffff3c8:     0x0804b26c    0x0804b260    0x0804b254    0x0804b248
               0xbffff3d8:     0x0804b23c    0x0804b230

               eax = &list[i]
               curr_elem.next = eax
               curr_elem = eax
               i++
0x08048e4a <+178>:      mov    edi,0x1
0x08048e4f <+183>:      lea    edx,[ebp-0x30]
0x08048e52 <+186>:      mov    eax,DWORD PTR [edx+edi*4]
0x08048e55 <+189>:      mov    DWORD PTR [esi+0x8],eax
0x08048e58 <+192>:      mov    esi,eax
0x08048e5a <+194>:      inc    edi
0x08048e5b <+195>:      cmp    edi,0x5
0x08048e5e <+198>:      jle    0x8048e52 <phase_6+186>

// 4th stage
0x08048e60 <+200>:      mov    DWORD PTR [esi+0x8],0x0
0x08048e67 <+207>:      mov    esi,DWORD PTR [ebp-0x34]
*(esi? + 8) = 0
```

Looks like we have a linked list, with the head at 0x804b26c. The last pointer is NULL. The list has 6 elements.

List element is like:

```
list_el {
        int value;          // 4 bytes
        int filler;         // 4 bytes
        next *list_el;
}
```

This is confirmed by gdb:

```
(gdb) x/x 0x0804b26c + 8
0x804b274 <node1+8>:    0x0804b260
```

```
(gdb) x/x 0x0804b260 + 8
0x804b268 <node2+8>:        0x0804b254
(gdb) x/x 0x0804b254 + 8
0x804b25c <node3+8>:        0x0804b248
(gdb) x/x 0x0804b248 + 8
0x804b250 <node4+8>:        0x0804b23c
(gdb) x/x 0x0804b23c + 8
0x804b244 <node5+8>:        0x0804b230
(gdb) x/x 0x0804b230 + 8
0x804b238 <node6+8>:        0x00000000
(gdb)
```

(To get the values that are compared:

```
printf "%08x %08x %08x %08x %08x %08x \n", *0x0804b26c, *0x0804b260, *0x0804b254,
*0x0804b248, *0x0804b23c, *0x0804b230
000000fd 000002d5 0000012d 000003e5 000000d4 000001b0)
```

```
 elem = 0x804b26c   // list head
 i = 0
 while i<=4:
       edx = *(esi+8)
       eax = *(esi)
       if elem < elem->next:
               explode_bomb()
       elem = elem->next
       i++
```

Applying the corect ordering in stage 2, we obtain the desired list:

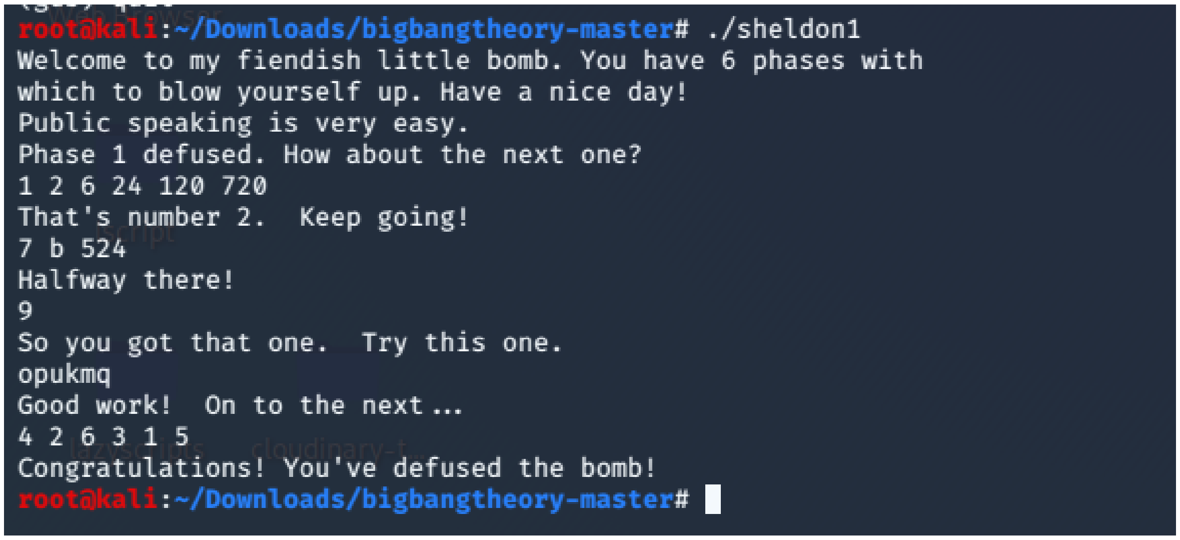
```
1: /x *(int*)($ebp-0x3c) = 0xbffff3c8
```

```
(gdb) p /x $esi
$1 = 0x804b248
(gdb) x /4x $esi
0x804b248 <node4>:  0x000003e5    0x00000004    0x0804b260    0x0000012d
(gdb) x /4x 0x0804b260
0x804b260 <node2>:  0x000002d5    0x00000002    0x0804b230    0x000000fd
(gdb) x /4x 0x0804b230
0x804b230 <node6>:  0x000001b0    0x00000006    0x0804b254    0x000000d4
(gdb) x /4x 0x0804b254
0x804b254 <node3>:  0x0000012d    0x00000003    0x0804b26c    0x000002d5
(gdb) x /4x 0x0804b26c
0x804b26c <node1>:  0x000000fd    0x00000001    0x0804b23c    0x000003e9
(gdb) x /4x 0x0804b23c
0x804b23c <node5>:  0x000000d4    0x00000005    0x00000000    0x000003e5
(gdb)
```

```
   0x08048e6a <+210>:    xor    edi,edi
   0x08048e6c <+212>:    lea    esi,[esi+eiz*1+0x0]
   0x08048e70 <+216>:    mov    edx,DWORD PTR [esi+0x8]
   0x08048e73 <+219>:    mov    eax,DWORD PTR [esi]
   0x08048e75 <+221>:    cmp    eax,DWORD PTR [edx]
   0x08048e77 <+223>:    jge    0x8048e7e <phase_6+230>
   0x08048e79 <+225>:    call   0x80494fc <explode_bomb>
   0x08048e7e <+230>:    mov    esi,DWORD PTR [esi+0x8]
   0x08048e81 <+233>:    inc    edi
   0x08048e82 <+234>:    cmp    edi,0x4
   0x08048e85 <+237>:    jle    0x8048e70 <phase_6+216>
```

```
0x08048e87 <+239>:    lea    esp,[ebp-0x58]
0x08048e8a <+242>:    pop    ebx
0x08048e8b <+243>:    pop    esi
0x08048e8c <+244>:    pop    edi
0x08048e8d <+245>:    mov    esp,ebp
0x08048e8f <+247>:    pop    ebp
0x08048e90 <+248>:    ret
```

Password for phase_6: 4 2 6 3 1 5



```
root@kali:~/Downloads/bigbangtheory-master# ./sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
1 2 6 24 120 720
That's number 2.  Keep going!
7 b 524
Halfway there!
9
So you got that one.  Try this one.
opukmq
Good work!  On to the next...
4 2 6 3 1 5
Congratulations! You've defused the bomb!
root@kali:~/Downloads/bigbangtheory-master# 
```