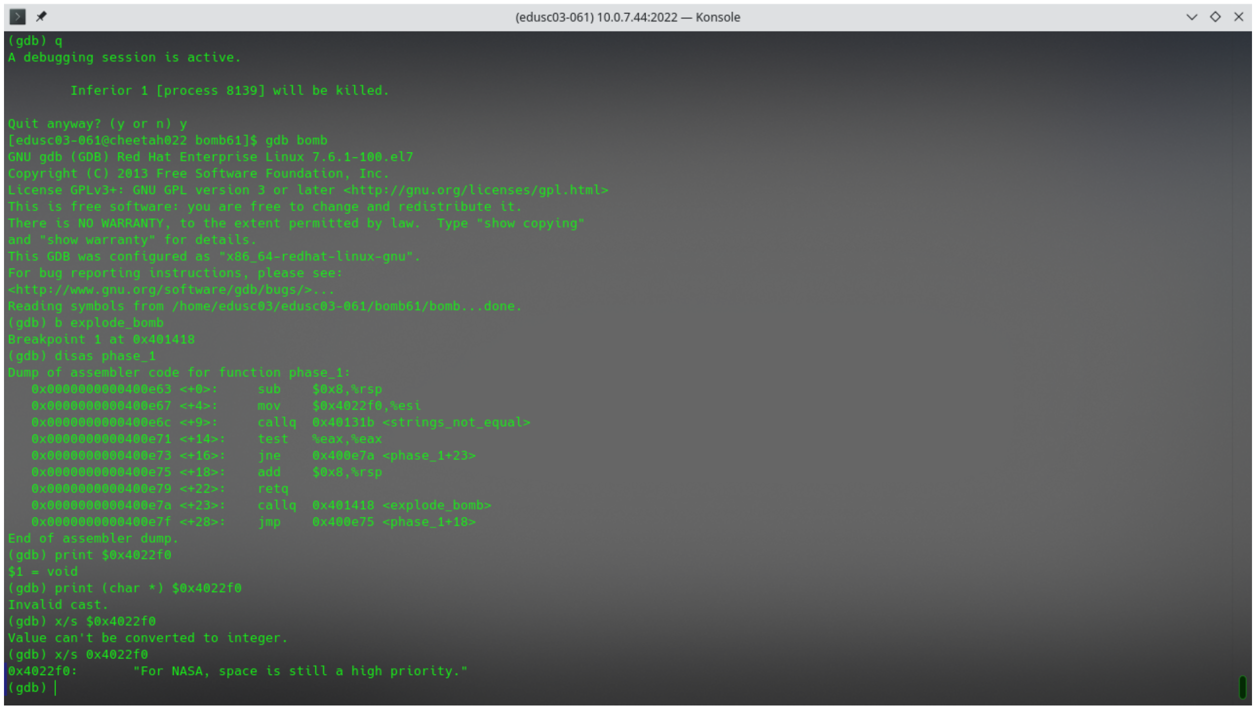
Phase 1



(edusc03-061) 10.0.7.44:2022 — Konsole

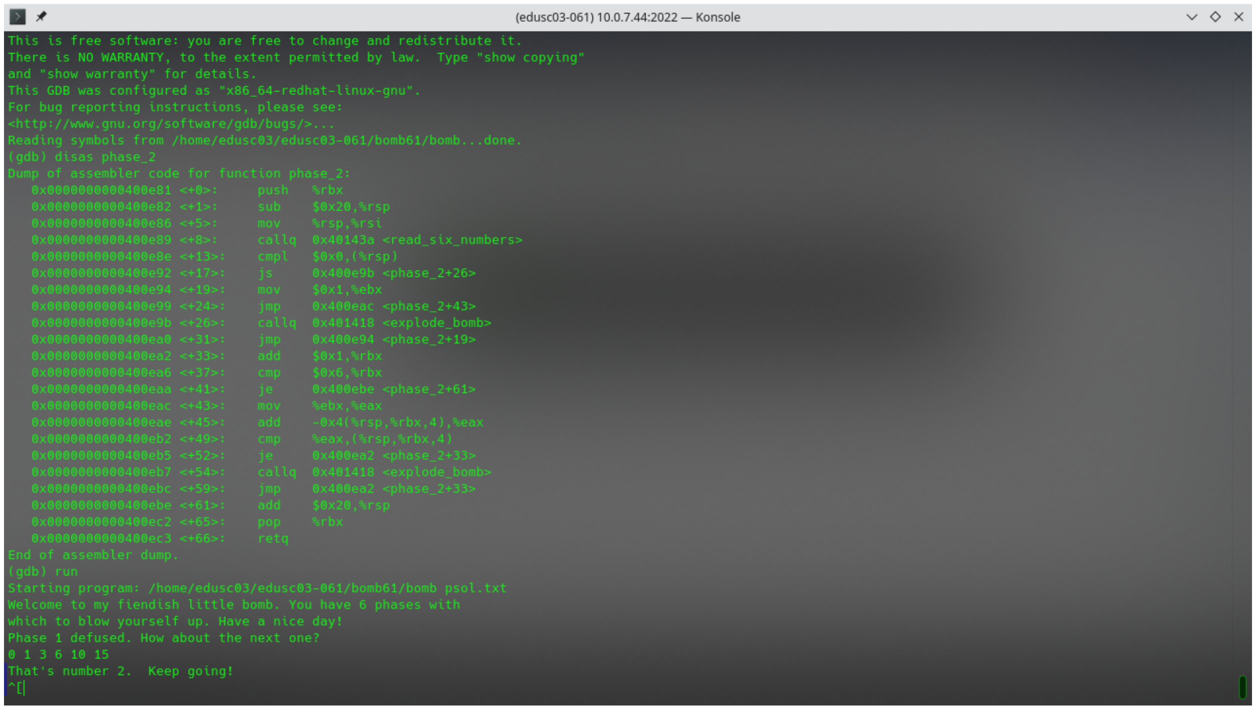
```
(gdb) q
A debugging session is active.

        Inferior 1 [process 8139] will be killed.

Quit anyway? (y or n) y
[edusc03-061@cheetah022 bomb61]$ gdb bomb
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-100.el7
Copyright (C) 2013 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/edusc03/edusc03-061/bomb61/bomb...done.
(gdb) b explode_bomb
Breakpoint 1 at 0x401418
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400e63 <+0>:     sub    $0x8,%rsp
   0x0000000000400e67 <+4>:     mov    $0x4022f0,%esi
   0x0000000000400e6c <+9>:     callq  0x40131b <strings_not_equal>
   0x0000000000400e71 <+14>:    test   %eax,%eax
   0x0000000000400e73 <+16>:    jne    0x400e7a <phase_1+23>
   0x0000000000400e75 <+18>:    add    $0x8,%rsp
   0x0000000000400e79 <+22>:    retq
   0x0000000000400e7a <+23>:    callq  0x401418 <explode_bomb>
   0x0000000000400e7f <+28>:    jmp    0x400e75 <phase_1+18>
End of assembler dump.
(gdb) print $0x4022f0
$1 = void
(gdb) print (char *) $0x4022f0
Invalid cast.
(gdb) x/s $0x4022f0
Value can't be converted to integer.
(gdb) x/s 0x4022f0
0x4022f0:       "For NASA, space is still a high priority."
(gdb) |
```

I used disas to check the code of phase 1. And saw that it just checks our string with some string in the 0x4022f0 address. So using x/s command I found the answer string which is ”`For NASA, space is still a high priority.`”.

Phase 2



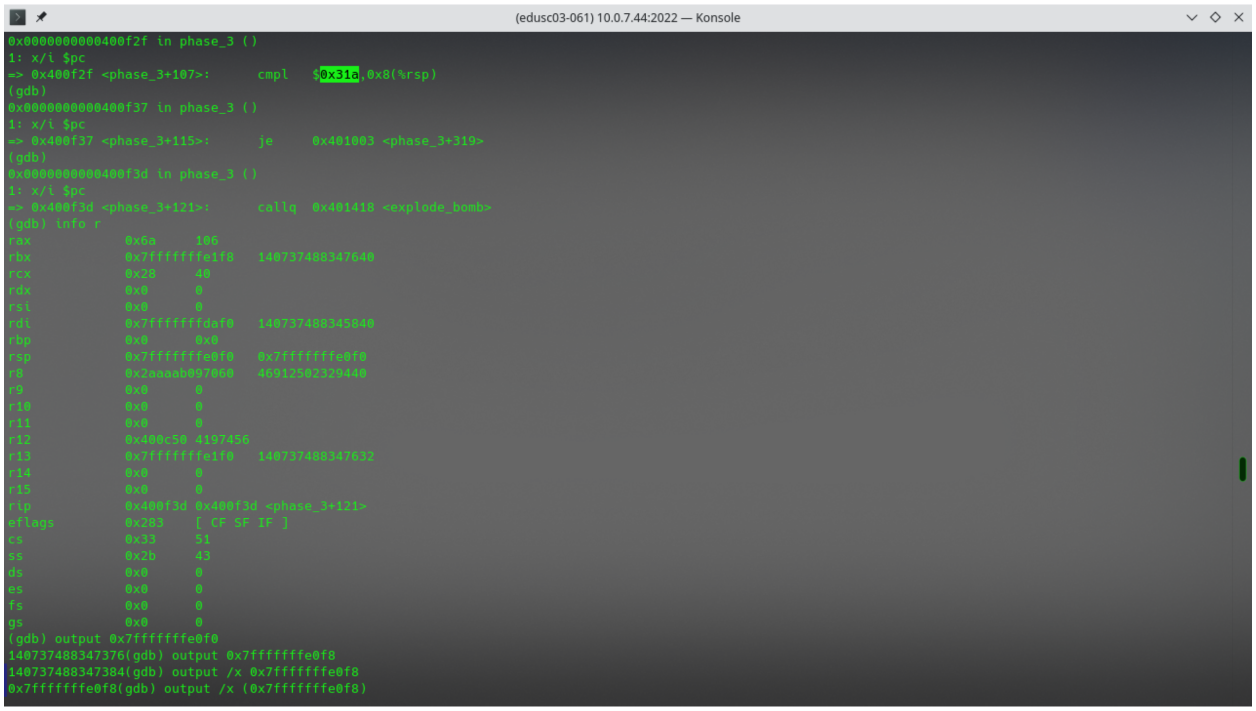
(edusc03-061) 10.0.7.44:2022 — Konsole

```
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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/edusc03/edusc03-061/bomb61/bomb...done.
(gdb) disas phase_2
Dump of assembler code for function phase_2:
   0x0000000000400e81 <+0>:     push   %rbx
   0x0000000000400e82 <+1>:     sub    $0x20,%rsp
   0x0000000000400e86 <+5>:     mov    %rsp,%rsi
   0x0000000000400e89 <+8>:     callq  0x40143a <read_six_numbers>
   0x0000000000400e8e <+13>:    cmpl   $0x0,(%rsp)
   0x0000000000400e92 <+17>:    js     0x400e9b <phase_2+26>
   0x0000000000400e94 <+19>:    mov    $0x1,%ebx
   0x0000000000400e99 <+24>:    jmp    0x400eac <phase_2+43>
   0x0000000000400e9b <+26>:    callq  0x401418 <explode_bomb>
   0x0000000000400ea0 <+31>:    jmp    0x400e94 <phase_2+19>
   0x0000000000400ea2 <+33>:    add    $0x1,%rbx
   0x0000000000400ea6 <+37>:    cmp    $0x6,%rbx
   0x0000000000400eaa <+41>:    je     0x400ebe <phase_2+61>
   0x0000000000400eac <+43>:    mov    %ebx,%eax
   0x0000000000400eae <+45>:    add    -0x4(%rsp,%rbx,4),%eax
   0x0000000000400eb2 <+49>:    cmp    %eax,(%rsp,%rbx,4)
   0x0000000000400eb5 <+52>:    je     0x400ea2 <phase_2+33>
   0x0000000000400eb7 <+54>:    callq  0x401418 <explode_bomb>
   0x0000000000400ebc <+59>:    jmp    0x400ea2 <phase_2+33>
   0x0000000000400ebe <+61>:    add    $0x20,%rsp
   0x0000000000400ec2 <+65>:    pop    %rbx
   0x0000000000400ec3 <+66>:    retq
End of assembler dump.
(gdb) run
Starting program: /home/edusc03/edusc03-061/bomb61/bomb psol.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?
0 1 3 6 10 15
That's number 2.  Keep going!
^[
```

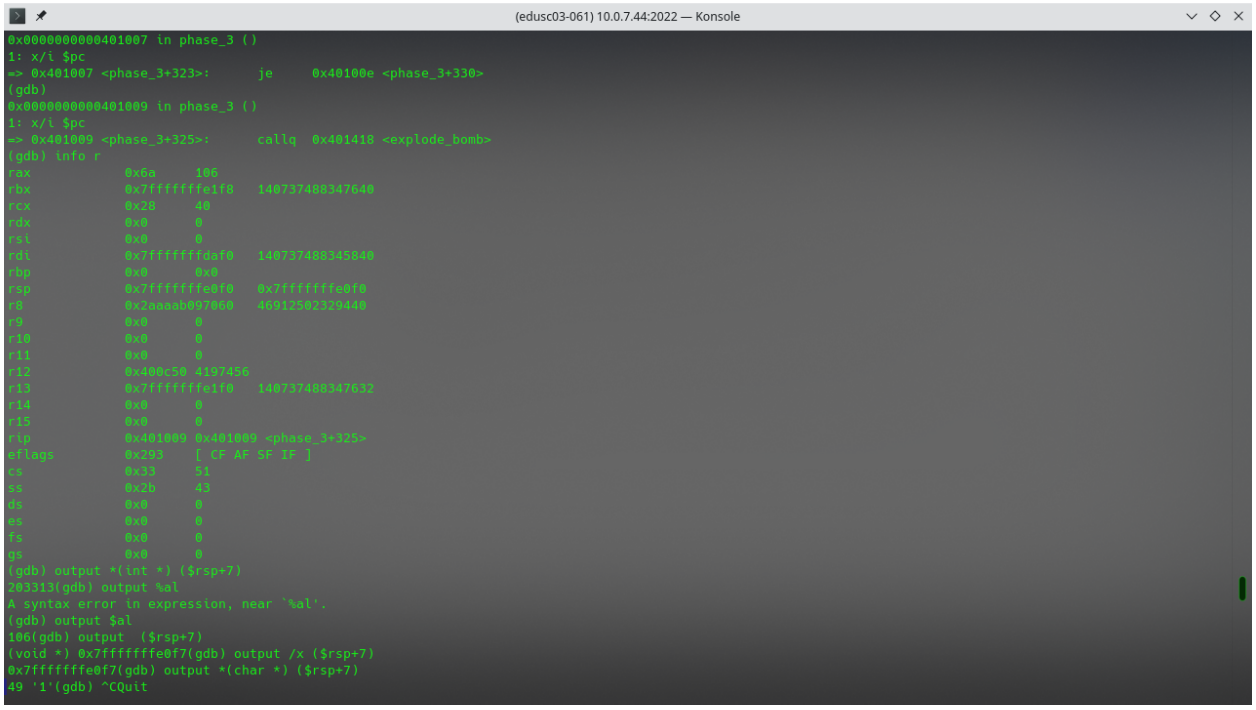
Looking at the code we can see that it takes 6 numbers and checks something with it. Namely, it assigns ebx to 1 and checks the difference between two adjacent numbers whether their difference is ebx and increments ebx and advances to the next two adjacent numbers. After checking all adjacent numbers it will proceed to the next stage.

Phase 3

In order to solve this phase I checked the values of the registers and the instructions of the assembly. I found out that the second last three digits are compared with `0x31a`. After that the first character of the second parameter is checked with j (or ascii 106). I understood that it checks with char because it compares all (8 bits type). The possible type for that in c is only char.



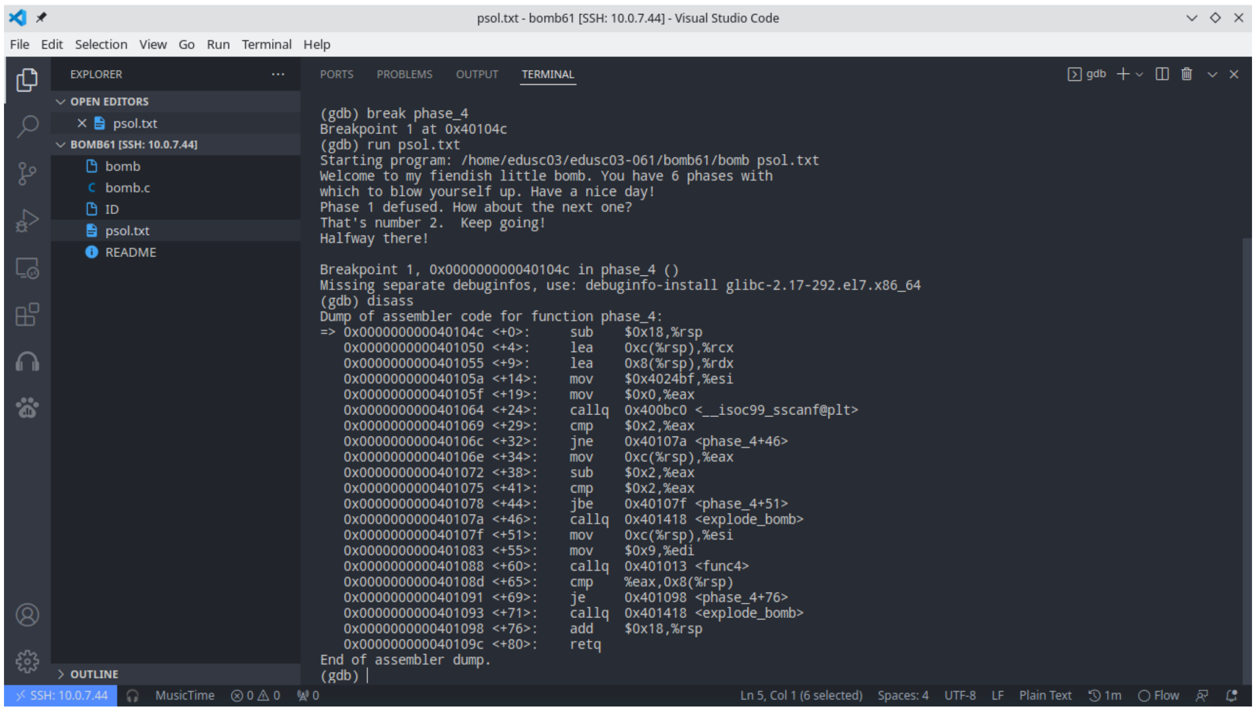
```
0x0000000000400f2f in phase_3 ()
1: x/i $pc
=> 0x400f2f <phase_3+107>:      cmpl   $0x31a,0x8(%rsp)
(gdb)
0x0000000000400f37 in phase_3 ()
1: x/i $pc
=> 0x400f37 <phase_3+115>:      je     0x401003 <phase_3+319>
(gdb)
0x0000000000400f3d in phase_3 ()
1: x/i $pc
=> 0x400f3d <phase_3+121>:      callq  0x401418 <explode_bomb>
(gdb) info r
rax            0x6a     106
rbx            0x7fffffffe1f8   140737488347640
rcx            0x28     40
rdx            0x0      0
rsi            0x0      0
rdi            0x7fffffffdaf0   140737488345840
rbp            0x0      0x0
rsp            0x7fffffffe0f0   0x7fffffffe0f0
r8             0x2aaaab097060   46912502329440
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x400c50 4197456
r13            0x7fffffffe1f0   140737488347632
r14            0x0      0
r15            0x0      0
rip            0x400f3d 0x400f3d <phase_3+121>
eflags         0x283    [ CF SF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb) output 0x7fffffffe0f0
140737488347376(gdb) output 0x7fffffffe0f8
140737488347384(gdb) output /x 0x7fffffffe0f8
0x7fffffffe0f8(gdb) output /x (0x7fffffffe0f8)
```



```
0x0000000000401007 in phase_3 ()
1: x/i $pc
=> 0x401007 <phase_3+323>:      je     0x40100e <phase_3+330>
(gdb)
0x0000000000401009 in phase_3 ()
1: x/i $pc
=> 0x401009 <phase_3+325>:      callq  0x401418 <explode_bomb>
(gdb) info r
rax            0x6a     106
rbx            0x7fffffffe1f8   140737488347640
rcx            0x28     40
rdx            0x0      0
rsi            0x0      0
rdi            0x7fffffffdaf0   140737488345840
rbp            0x0      0x0
rsp            0x7fffffffe0f0   0x7fffffffe0f0
r8             0x2aaaab097060   46912502329440
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x400c50 4197456
r13            0x7fffffffe1f0   140737488347632
r14            0x0      0
r15            0x0      0
rip            0x401009 0x401009 <phase_3+325>
eflags         0x293    [ CF AF SF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb) output *(int *) ($rsp+7)
203313(gdb) output %al
A syntax error in expression, near `%al'.
(gdb) output $al
106(gdb) output  ($rsp+7)
(void *) 0x7fffffffe0f7(gdb) output /x ($rsp+7)
0x7fffffffe0f7(gdb) output *(char *) ($rsp+7)
49 '1'(gdb) ^CQuit
```

Phase 4

Here we need to enter two numbers separated with space. At first, I entered 1 and 2. Then checked registers before callq explode_bomb and found 176 and 2.

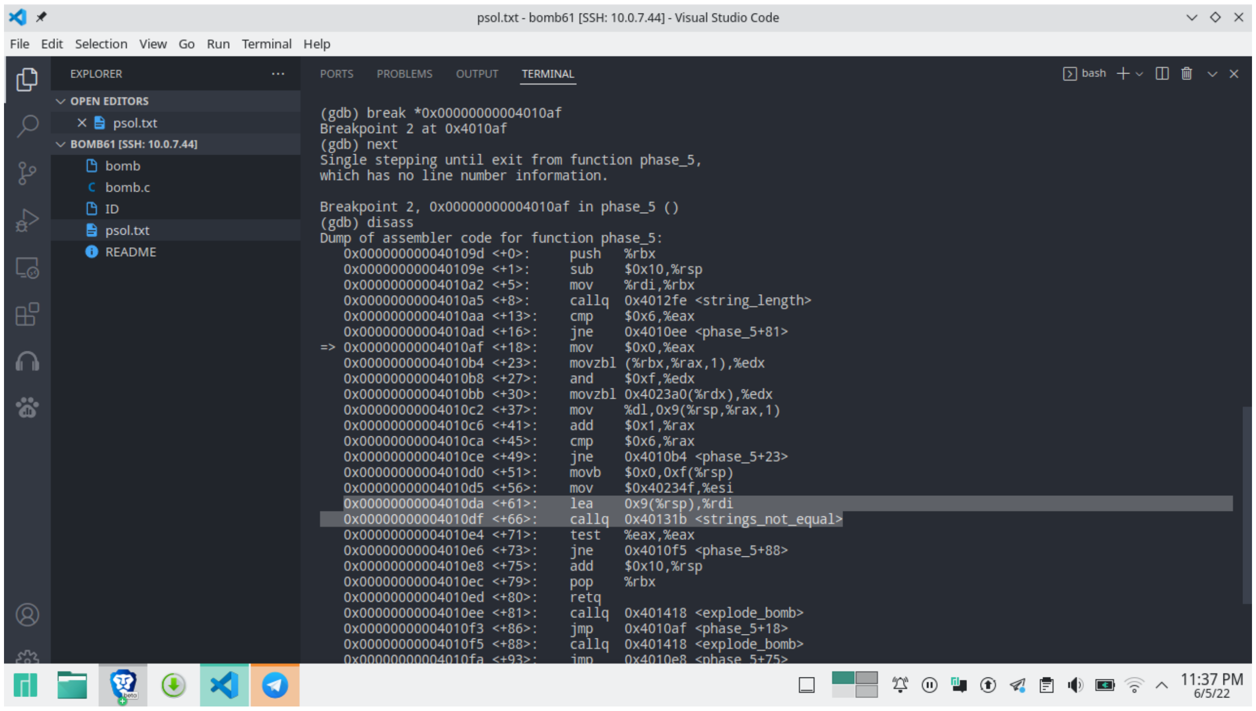


```
(gdb) break phase_4
Breakpoint 1 at 0x40104c
(gdb) run psol.txt
Starting program: /home/edusc03/edusc03-061/bomb61/bomb psol.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!

Breakpoint 1, 0x000000000040104c in phase_4 ()
Missing separate debuginfos, use: debuginfo-install glibc-2.17-292.el7.x86_64
(gdb) disass
Dump of assembler code for function phase_4:
=> 0x000000000040104c <+0>:     sub    $0x18,%rsp
   0x0000000000401050 <+4>:     lea    0xc(%rsp),%rcx
   0x0000000000401055 <+9>:     lea    0x8(%rsp),%rdx
   0x000000000040105a <+14>:    mov    $0x4024bf,%esi
   0x000000000040105f <+19>:    mov    $0x0,%eax
   0x0000000000401064 <+24>:    callq  0x400bc0 <__isoc99_sscanf@plt>
   0x0000000000401069 <+29>:    cmp    $0x2,%eax
   0x000000000040106c <+32>:    jne    0x40107a <phase_4+46>
   0x000000000040106e <+34>:    mov    0xc(%rsp),%eax
   0x0000000000401072 <+38>:    sub    $0x2,%eax
   0x0000000000401075 <+41>:    cmp    $0x2,%eax
   0x0000000000401078 <+44>:    jbe    0x40107f <phase_4+51>
   0x000000000040107a <+46>:    callq  0x401418 <explode_bomb>
   0x000000000040107f <+51>:    mov    0xc(%rsp),%esi
   0x0000000000401083 <+55>:    mov    $0x9,%edi
   0x0000000000401088 <+60>:    callq  0x401013 <func4>
   0x000000000040108d <+65>:    cmp    %eax,0x8(%rsp)
   0x0000000000401091 <+69>:    je     0x401098 <phase_4+76>
   0x0000000000401093 <+71>:    callq  0x401418 <explode_bomb>
   0x0000000000401098 <+76>:    add    $0x18,%rsp
   0x000000000040109c <+80>:    retq
End of assembler dump.
(gdb) |
```

Phase 5

Here at **0x00000000004010da <+61>:   lea   0x9(%rsp),%rdi,** we can see that it's comparing the string we entered to some other string. If we put breakpoint and check the value of %rdi, we can see that this string == bruins, while the string we entered **abcdef is converted to aduier.** It means there's some type of MAPPING going on.

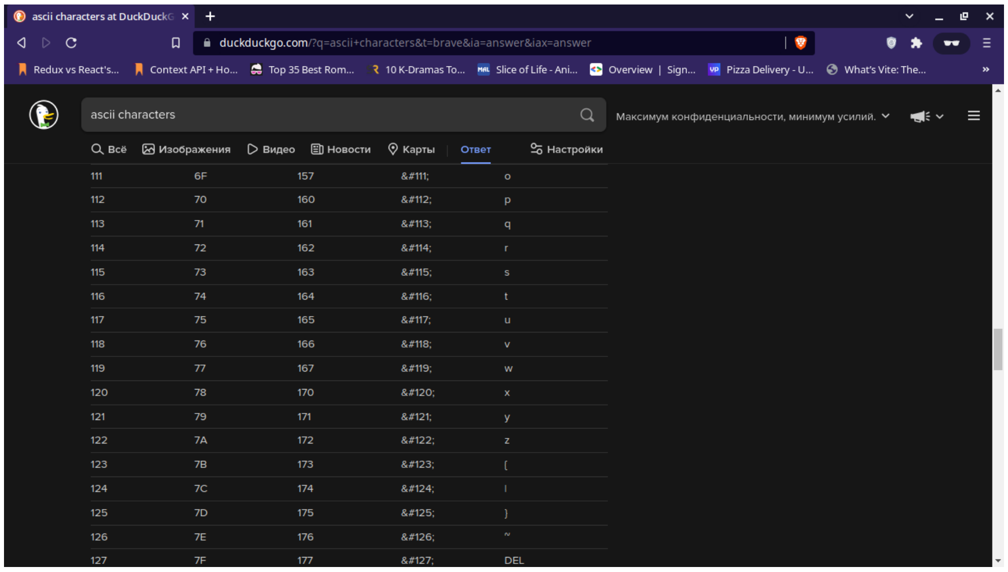


```
(gdb) break *0x00000000004010af
Breakpoint 2 at 0x4010af
(gdb) next
Single stepping until exit from function phase_5,
which has no line number information.

Breakpoint 2, 0x00000000004010af in phase_5 ()
(gdb) disass
Dump of assembler code for function phase_5:
   0x000000000040109d <+0>:     push   %rbx
   0x000000000040109e <+1>:     sub    $0x10,%rsp
   0x00000000004010a2 <+5>:     mov    %rdi,%rbx
   0x00000000004010a5 <+8>:     callq  0x4012fe <string_length>
   0x00000000004010aa <+13>:    cmp    $0x6,%eax
   0x00000000004010ad <+16>:    jne    0x4010ee <phase_5+81>
=> 0x00000000004010af <+18>:    mov    $0x0,%eax
   0x00000000004010b4 <+23>:    movzbl (%rbx,%rax,1),%edx
   0x00000000004010b8 <+27>:    and    $0xf,%edx
   0x00000000004010bb <+30>:    movzbl 0x4023a0(%rdx),%edx
   0x00000000004010c2 <+37>:    mov    %dl,0x9(%rsp,%rax,1)
   0x00000000004010c6 <+41>:    add    $0x1,%rax
   0x00000000004010ca <+45>:    cmp    $0x6,%rax
   0x00000000004010ce <+49>:    jne    0x4010b4 <phase_5+23>
   0x00000000004010d0 <+51>:    movb   $0x0,0xf(%rsp)
   0x00000000004010d5 <+56>:    mov    $0x40234f,%esi
   0x00000000004010da <+61>:    lea    0x9(%rsp),%rdi
   0x00000000004010df <+66>:    callq  0x40131b <strings_not_equal>
   0x00000000004010e4 <+71>:    test   %eax,%eax
   0x00000000004010e6 <+73>:    jne    0x4010f5 <phase_5+88>
   0x00000000004010e8 <+75>:    add    $0x10,%rsp
   0x00000000004010ec <+79>:    pop    %rbx
   0x00000000004010ed <+80>:    retq
   0x00000000004010ee <+81>:    callq  0x401418 <explode_bomb>
   0x00000000004010f3 <+86>:    jmp    0x4010af <phase_5+18>
   0x00000000004010f5 <+88>:    callq  0x401418 <explode_bomb>
   0x00000000004010fa <+93>:    jmp    0x4010e8 <phase_5+75>
```

psol.txt - bomb61 [SSH: 10.0.7.44] - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
psol.txt
BOMB61 [SSH: 10.0.7.44]
bomb
bomb.c
ID
psol.txt
README
PORTS PROBLEMS OUTPUT TERMINAL
bash
11:37 PM
6/5/22

Since there are only 128 ASCII characters, I tried every of them and found out
mf3txw ==> bruins



111	6F	157	&#111;	o
112	70	160	&#112;	p
113	71	161	&#113;	q
114	72	162	&#114;	r
115	73	163	&#115;	s
116	74	164	&#116;	t
117	75	165	&#117;	u
118	76	166	&#118;	v
119	77	167	&#119;	w
120	78	170	&#120;	x
121	79	171	&#121;	y
122	7A	172	&#122;	z
123	7B	173	&#123;	{
124	7C	174	|	|
125	7D	175	&#125;	}
126	7E	176	&#126;	~
127	7F	177	&#127;	DEL