Consider the following C code, where I have replaced the actual integers in each case statement with "XXX"

```
switch(a){

  case XXX:
    i=5;
    break;

  case XXX:
    i=4;
    break;

  case XXX:
    i=2;
    break;

  case XXX:
    i=7;
    break;

  case XXX:
    i=9;
    break;
```

```
 default:
  i=8;
 }

 fprintf (stderr,"i:%d\n", i);
```

We will give you the output of this code - your job is to figure out what the original value of variable "a" must have been for this output. But first you need to figure out the real values for the XXX's in the code above.

The switch statement (with real integers in place of the XXX) and printf statement compiles to:

```
 40056c:       83 e8 2a                sub    $0x2a,%eax
 40056f:       83 f8 06                cmp    $0x6,%eax
 400572:       77 39                   ja     4005ad <main+0x69>
 400574:       89 c0                   mov    %eax,%eax
 400576:       48 8b 04 c5 e0 06 40    mov    0x4006e0(,%rax,8),%rax
 40057d:       00
 40057e:       ff e0                   jmpq   *%rax
 400580:       c7 45 f0 05 00 00 00    movl   $0x5,-0x10(%rbp)
 400587:       eb 2b                   jmp    4005b4 <main+0x70>
 400589:       c7 45 f0 04 00 00 00    movl   $0x4,-0x10(%rbp)
 400590:       eb 22                   jmp    4005b4 <main+0x70>
 400592:       c7 45 f0 02 00 00 00    movl   $0x2,-0x10(%rbp)
 400599:       eb 19                   jmp    4005b4 <main+0x70>
 40059b:       c7 45 f0 07 00 00 00    movl   $0x7,-0x10(%rbp)
```

```
4005a2:   eb 10                  jmp    4005b4 <main+0x70>
4005a4:   c7 45 f0 09 00 00 00   movl   $0x9,-0x10(%rbp)
4005ab:   eb 07                  jmp    4005b4 <main+0x70>
4005ad:   c7 45 f0 08 00 00 00   movl   $0x8,-0x10(%rbp)
4005b4:   b9 d8 06 40 00         mov    $0x4006d8,%ecx
4005b9:   48 8b 05 f8 03 20 00   mov    0x2003f8(%rip),%rax        # 6009b8 <stderr@@GLIBC_2.2.5>
4005c0:   8b 55 f0               mov    -0x10(%rbp),%edx
4005c3:   48 89 ce               mov    %rcx,%rsi
4005c6:   48 89 c7               mov    %rax,%rdi
4005c9:   b8 00 00 00 00         mov    $0x0,%eax
4005ce:   e8 7d fe ff ff         callq  400450 <fprintf@plt>
```

And here is some gdb interaction that should prove helpful:

```
(gdb) x/56xb 0x4006e0
0x4006e0 <__dso_handle+16>:    0x80    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x4006e8 <__dso_handle+24>:    0xad    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x4006f0 <__dso_handle+32>:    0x89    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x4006f8 <__dso_handle+40>:    0xad    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x400700 <__dso_handle+48>:    0x92    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x400708 <__dso_handle+56>:    0x9b    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x400710 <__dso_handle+64>:    0xa4    0x05    0x40    0x00    0x00    0x00    0x00    0x00
```

The printf output from this code is:

i:2

What is the value of variable a?

Consider the following C declaration:

```
char * mysticarray[20];
```

and the following gdb interaction:

```
(gdb) print &mysticarray
$1 = (char *(*)[20]) 0x7fffffffe1d0

(gdb) x/256bx 0x7fffffffe1d0
0x7fffffffe1d0: 0xd6    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe1d8: 0xe7    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe1e0: 0x1c    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe1e8: 0x12    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe1f0: 0x08    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe1f8: 0x12    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe200: 0xe7    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe208: 0x04    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe210: 0xf6    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe218: 0xcc    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe220: 0xd0    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe228: 0xec    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe230: 0xfb    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe238: 0x27    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe240: 0xd6    0x06    0x40    0x00    0x00    0x00    0x00    0x00
```

```
0x7fffffffe248: 0xe7    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe250: 0xc8    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe258: 0xe7    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe260: 0x04    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe268: 0x16    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe270: 0x00    0x00    0x00    0x00    0x14    0x00    0x00    0x00
0x7fffffffe278: 0x20    0x04    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe280: 0x70    0xe3    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffe288: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe290: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe298: 0x20    0xed    0x21    0xf4    0x37    0x00    0x00    0x00
0x7fffffffe2a0: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe2a8: 0x78    0xe3    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffe2b0: 0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe2b8: 0x04    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe2c0: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe2c8: 0xc1    0x04    0x0f    0xaa    0xa2    0xb7    0xd6    0x33

(gdb) x/128bx 0x4006C0
0x4006c0 <__dso_handle>:        0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x4006c8 <__dso_handle+8>:      0x61    0x62    0x65    0x00    0x62    0x6f    0x62    0x00
0x4006d0 <__dso_handle+16>:     0x63    0x61    0x72    0x6f    0x6c    0x00    0x64    0x61
0x4006d8 <__dso_handle+24>:     0x6e    0x61    0x00    0x65    0x64    0x64    0x69    0x65
0x4006e0 <__dso_handle+32>:     0x00    0x66    0x72    0x61    0x6e    0x6b    0x00    0x67
0x4006e8 <__dso_handle+40>:     0x65    0x72    0x74    0x00    0x68    0x6f    0x64    0x00
```

```
0x4006f0 <__dso_handle+48>:    0x69    0x7a    0x7a    0x69    0x65    0x00    0x6a    0x6f
0x4006f8 <__dso_handle+56>:    0x65    0x79    0x00    0x6b    0x61    0x72    0x6c    0x00
0x400700 <__dso_handle+64>:    0x6c    0x65    0x6f    0x00    0x6d    0x6f    0x65    0x00
0x400708 <__dso_handle+72>:    0x6e    0x6f    0x72    0x6d    0x00    0x6f    0x74    0x74
0x400710 <__dso_handle+80>:    0x6f    0x00    0x70    0x6f    0x65    0x00    0x71    0x75
0x400718 <__dso_handle+88>:    0x69    0x6e    0x6e    0x00    0x72    0x6f    0x67    0x65
0x400720 <__dso_handle+96>:    0x72    0x00    0x73    0x61    0x75    0x6c    0x00    0x74
0x400728 <__dso_handle+104>:   0x6f    0x6e    0x79    0x00    0x25    0x64    0x20    0x25
0x400730 <__dso_handle+112>:   0x73    0x0a    0x00    0x25    0x64    0x20    0x25    0x64
0x400738 <__dso_handle+120>:   0x20    0x25    0x64    0x20    0x25    0x64    0x0a    0x00
```

What would be printed by the following C statement:

```
printf("%s", mysticarray[12]);
```

Evaluate this C expression:

~(((~(0<<2)+0)^~5)|9)

Your answer should be an integer.

Consider the following C code:

```
#include <stdlib.h>
#include <stdio.h>

union turtles {
  char fogofwar[64];
  struct {
    signed char leo;
    short raph;
    int mikey;
    long don;
  } ninjas;
} x;

int main( int argc, const char* argv[] )
{
  int i,j;
  int shelled;

  srand(atoi(argv[1])); // Set the seed for random number generation
  for (j=0; j<64; j++)
    x.fogofwar[j]=rand();
 /* more data processing here in foo() that will modify the union data */
```

```
  foo();
  shelled=0;
  if (x.ninjas.leo < 0)
    shelled++;
  if (x.ninjas.mikey < 0)
    shelled++;
  if (x.ninjas.don < 0)
    shelled++;
  if (x.ninjas.raph < 0)
    shelled++;
  printf("%d", shelled);
}
```

Your job is to figure out what integer the printf will produce. Here's a dump of variable x at the time of the printf:

```
(gdb) print &x
$2 = (<data variable, no debug info> *) 0x601080 <x>
(gdb) x/64xb 0x601080
0x601080 <x>:   0xd7    0xb2    0x3e    0x55    0xd0    0xc6    0xff    0x4e
0x601088 <x+8>: 0x97    0xa0    0x36    0x29    0x00    0x00    0x00    0x00
0x601090 <x+16>:        0x77    0x07    0xe4    0xfe    0xb0    0x5f    0x6a    0x0a
0x601098 <x+24>:        0x1a    0x82    0x1c    0x6b    0x8d    0x9c    0xcf    0xfa
0x6010a0 <x+32>:        0x4f    0x1c    0x47    0x49    0x67    0x9d    0xff    0x2b
0x6010a8 <x+40>:        0xcb    0x71    0x42    0xfc    0x9e    0x84    0x23    0x16
0x6010b0 <x+48>:        0x8b    0x07    0x14    0x3c    0x66    0x7f    0x46    0x80
0x6010b8 <x+56>:        0x01    0x62    0xec    0x8e    0xff    0xbb    0x89    0x4e
```

Assume:

x and y are randomly generated signed integers

ux and uy are randomly generated unsigned integers

For the following C puzzle, is the statement below true or false?

(x & -4) == Tmin $\rightarrow$ (x & 1) == 0