

What is the output of this program, and what does it return to the OS?

Asked 16 years, 1 month ago Modified 12 years, 7 months ago Viewed 721 times



It's kind of a C puzzle. You have to tell if the program finish its execution, if so, how much time it takes to run and what it returns to the OS.

2



```
static unsigned char buffer[256];

int main(void)
{
    unsigned char *p, *q;
    q = (p = buffer) + sizeof(buffer);
    while (q - p)
    {
        p = buffer;
        while (!++*p++);
    }
    return p - q;
}
```

[EDIT] I removed the interview-questions tag since that seems to be the primary thing people are objecting to. This is a great little puzzle but as everyone has already pointed out, not a great interview question.

c puzzle

Share

Improve this question

Follow

edited May 4, 2012 at 3:49



Andrew Eisenberg

28.7k ● 9 ● 96 ● 151

asked Nov 11, 2008 at 2:29



lurks

2,606 ● 5 ● 31 ● 39

Horrible interview question, its too low level, and obfuscated besides its not something ANYONE wants to see in production code. – [Robert Gould](#) Nov 11, 2008 at 2:49

While I agree that this is a horrible interview question, it is a great little puzzle for those that like that sort of thing and it has a rewarding simplicity that is hard to appreciate until you work through it. See my answer if you don't want to waste the brain cells to see it ;) – [Robert Gamble](#) Nov 11, 2008 at 4:02

I don't think anyone said it was an interview question, unless it was called that and then edited down later. – [dnord](#) Nov 11, 2008 at 4:09

@dnord, the OP used the interview-questions tag which is what everyone jumped on, I removed that tag. – [Robert Gamble](#) Nov 11, 2008 at 4:14



Despite the fact that this is a horrible interview question, it is actually quite interesting:

13



```
static unsigned char buffer[256];

int main(void)
{
    unsigned char *p, *q;
    q = (p = buffer) + sizeof(buffer);
    /* This statement will set p to point to the beginning of buffer and will
       set q to point to one past the last element of buffer (this is legal) */
    while (q - p)
    /* q - p will start out being 256 and will decrease at an inversely
       exponential rate: */
    {
        p = buffer;
        while (!++*p++);
        /* This is where the interesting part comes in; the prefix increment,
           dereference, and logical negation operators all have the same
           precedence and are evaluated **right-to-left**. The postfix
           operator has a higher precedence. *p starts out at zero, is
           incremented to 1 by the prefix, and is negated by !.
           p is incremented by the postfix operator, the condition
           evaluates to false and the loop terminates with buffer[0] = 1.

           p is then set to point to buffer[0] again and the loop continues
           until buffer[0] = 255. This time, the loop succeeds when *p is
           incremented, becomes 0 and is negated. This causes the loop to
           run again immediately after p is incremented to point to buffer[1],
           which is increased to 1. The value 1 is of course negated,
           p is incremented which doesn't matter because the loop terminates
           and p is reset to point to buffer[0] again.

           This process will continue to increment buffer[0] every time,
           increasing buffer[1] every 256 runs. After 256*255 runs,
           buffer[0] and buffer[1] will both be 255, the loop will succeed
           *twice* and buffer[2] will be incremented once, etc.

           The loop will terminate after about 256^256 runs when all the values
           in the buffer array are 255 allowing p to be incremented to the end
           of the array. This will happen sometime after the universe ends,
           maybe a little sooner on the new Intels ;)
        */
    }
    return p - q;
    /* Returns 0 as p == q now */
}
```

Essentially this is a base-256 (assuming 8-bit bytes) counter with 256 digits, the program will exit when the entire counter "rolls over".

The reason this is interesting is because the code is actually completely legal C (no undefined or implementation defined behavior that you usually find in these types of

questions) and there is actually a legitimate algorithm problem, albeit a little hidden, in the mix. The reason it is a horrible interview question is because I wouldn't expect anyone to remember the precedence and associativity of the operators involved in the while statement. But it does make for a fun and insightful little exercise.

Share

edited Nov 11, 2008 at 5:41

answered Nov 11, 2008 at 3:50

Improve this answer



Robert Gamble

109k ● 25 ● 147 ● 138

Follow

+1 for taking your time, I still think this is a horrible question though :) – [Robert Gould](#) Nov 11, 2008 at 3:58

*p is accessing uninitialized memory, no matter how you slice it – [Steven A. Lowe](#) Nov 11, 2008 at 4:00

@Steven, look again, the array is static which means all elements are initialized to zero. – [Robert Gamble](#) Nov 11, 2008 at 4:03

I stand (well, sit) corrected. But note that this is true only if the C compiler is ANSI standard (see K&R second edition, circa 1988); I learned C in 1985 (K&R first edition) so I did not know this bit of trivia, thanks! – [Steven A. Lowe](#) Nov 11, 2008 at 4:13

Actually, according to my 1978 copy of K&R 1, this has always been the case, see page 37: "External and static variables are initialized to zero by default, but it is good style to state the initialization anyway." This is mentioned again on pages 82 and 198. – [Robert Gamble](#) Nov 11, 2008 at 4:21

▲ this code is garbage, see comments

12

```
static unsigned char buffer[256];
int main(void)
{
    unsigned char *p, *q;
    q = (p = buffer) + sizeof(buffer);    //p=buffer, q=buffer+256
    while (q - p)    //q-p = 256 on first iteration
    {
        p = buffer;    //p=buffer again
        while (!++*p++);    //increment the value pointed at by p+1 and check for
!0
    }
    return p - q;    //will return zero if loop ever terminates
}
```

it might terminate, it might not; the while loop is essentially scanning an uninitialized buffer so it might throw an access violation instead; i don't remember the binding precedence of ++*p++, nor do i care enough to look it up

if this is really an interview question, my answer is "if this is the kind of code you expect me to work with, i don't want the job"

EDIT: thanks to Robert Gamble for reminding me that static arrays are automatically initialized to zero, so the code is not complete garbage - but I still would not want to maintain it or work with the nutjob that wrote it ;-)

Share

edited Nov 11, 2008 at 4:57

answered Nov 11, 2008 at 2:41

Improve this answer



[Steven A. Lowe](#)

61.1k ● 19 ● 135 ● 204

Follow

i've looked up the precedence of `++*p++` : `++(*p++)` . so it actually increments the value pointed at by p, not at p + 1. then after that it increments p. no worries, i've neither memorized that precedence rules :) – [Johannes Schaub - litb](#) Nov 11, 2008 at 5:47



The right answer to this question is:

4

This code is unmaintainable, untestable, serves no purpose and should be removed or rewritten.



Anything else means that the interviewee is not thinking as a software engineer.



Than again, you might be not be interviewing for engineering position.

Share Improve this answer Follow

answered Nov 11, 2008 at 3:50



[Franci Penov](#)

75.9k ● 18 ● 135 ● 171



```
unsigned char *p, *q;
```

-6

Isn't this wrong on many levels? First of all, is there such a thing as an unsigned char? Second, and I may be wrong here, so don't quote me, but doesn't `char *p, q` produce funky results? It's either that, or it makes it easy to do `char p, q`, which would be bad form.



The following is much better:

```
char* p;  
char* q;
```

Share Improve this answer Follow

answered Nov 11, 2008 at 4:44



[Charles Graham](#)

24.8k ● 14 ● 47 ● 56

Your formatting is bad (escape your *s with `\`), unsigned char *p, *q; is perfectly fine, and char * would not work in this case since char can be signed which would cause undefined behavior on overflow. char should only be used for characters, signed/unsigned char for numbers.

– [Robert Gamble](#) Nov 11, 2008 at 5:34

- 1 You may as well earn your Peer Pressure badge and delete this. – [Jonathan Leffler](#) Nov 16, 2008 at 14:10
-

