Computer Science 50: Introduction to Computer Science I
Harvard College
Fall 2008

Quiz 0

Solutions

Answers other than the below may be possible.

Short Answers.

0. `:-)`

1.

```
    10111111
 +  00000001
 -----------
    11000000
```

2. For efficiency, characters printed to `stdout` are buffered until a newline is printed or until a buffer's worth of characters are, at which point the buffer is flushed and those characters appear on screen. Because neither a newline nor (apparently) a buffer's worth of characters are printed by this code, nothing appears to happen for quite some time. Calling `fflush` after each call to `printf` would fix the problem, as would adding a newline to the string being printed.

3.

```
void
foo(int array, int n)
{
    for (i = 0; i <= n; i++)
        n = GetInt();
        array[i] = n;
    return n;
}
```

wrong type; should be: `*array` or `array[]`

missing type; should be: `int i = 0;`

iterates beyond array's bound; should be `<`

not in curly braces and clobbers `n`; should be:

```
array[i] = GetInt();
```

or

```
{
    int input = GetInt();
    array[i] = input;
}
```

returns a value; should be just: `return;`

4.

`printf("%d", a);`	0
`printf("%d", b);`	0
`printf("%d", c);`	0
`printf("%d", *d);`	0
`printf("%d", e);`	0

5. `YES`

strcrazy.

6.

```
char *
strncpy(char *dest, char *src, int n)
{
    // check whether src is shorter than dest
    int length = strlen(src);
    int min = (length < n) ? length : n;

    // copy chars from src to dest
    for (int i = 0; i < min; i++)
        dest[i] = src[i];

    // pad dest with \0's if necessary
    for (int i = min; i < n; i++)
        dest[i] = '\0';

    // return pointer to copy
    return dest;
}
```

Multiple Choice.

7. c[1]
8. d
9. b

[1] assuming a 32-bit x86 architecture

Rapid Fire.

10. Exploiting a buffer (*i.e.*, array) involves filling (or, really, overflowing) that buffer with more bytes than it was intended to contain. If those bytes happen to represent executable code, it's sometimes possible to "trick" a computer into executing those bytes by overwriting a function's return address on the stack.

11. By indexing into an array beyond its bounds.
By calling so many functions (*e.g.*, recursively) that the stack overruns the heap.
By dereferencing a `NULL` (or otherwise invalid) pointer.

12. Memory allocated on the heap (via `malloc`) persists (*i.e.*, belongs to a program) until it is explicitly freed (via `free`), whereas memory allocated on the stack (in the form of functions' local variables) persists only until the function for which the memory was allocated returns.

13. A recursive approach can sometimes be more straightward (or elegant) to implement or read.

14. Vigenère's key space is larger. Whereas Caesar's cipher allows for 26 possible keys, Vigenère's cipher allows for 26^n, where $n \geq 1$, assuming a 26-letter alphabet.

Ugh. Bugs.

15. Because of the (presumably accidental) semicolon on the same line as the condition, the line immediately below (`return true;`) executes unconditionally![2]

16. Because `i` is a signed `int`, the largest positive value that it can store is $2^{31} - 1$, which is 01111111111111111111111111111111 in binary or 2,147,483,647 in decimal. Trying to increment `i` beyond that maximum yields 10000000000000000000000000000000 in binary or -2,147,483,648 in decimal, an instance of overflow; the negative value induces the loop to terminate.

17. Because of the (presumably accidental) use of the assignment (`=`) instead of equality (`==`) operator, the condition always evaluates to `0` (*i.e.*, false), and so only `false` is ever returned.

Role Reversal.

18. In some (or all) cases, the student's function doesn't actually return a value (*e.g.*, the student left out a `return` statement altogether), even though its return type is not `void`.

[2] Quiz 0 went to press with a return type for this function of `void` instead of `bool`, an unintentional bug on our part; we thus accepted answers that focused on that bug as well.

19. The student failed to include

```
#include <cs50.h>
```

atop his or her code.

20. The student failed to compile with `-lcs50`, which tells GCC to link the student's code against (*i.e.*, borrow bits from) CS 50's library.

Argh. Args.

21. `9`
22. `si`
23. `5`

O(mega).

24.

Bound	**Algorithm**
$O(n^2)$	{Bubble Sort, Insertion Sort, Selection Sort}
$\Omega(n^2)$	{Insertion Sort, Bubble Sort, Selection Sort}
$O(n \log n)$	Merge Sort
$\Omega(n \log n)$	Merge Sort
$O(n)$	Linear Search
$\Omega(n)$	{Bubble Sort, Insertion Sort}
$O(\log n)$	Binary Search
$\Omega(1)$	{Binary Search, Linear Search}

25. Given an array of size *n*, selection sort makes $n - 1$ passes over the array. On each pass, it selects the smallest element not previously selected and swaps that element (in constant time) with the leftmost element not previously selected. After *k* passes, the *k* smallest elements thus appear in order in the first *k* locations of the array. So after $n - 1$ passes, the array is sorted. Since each pass can involve as many as $n - 1$ comparisons, this algorithm is in $O(n^2)$.

HAI O.

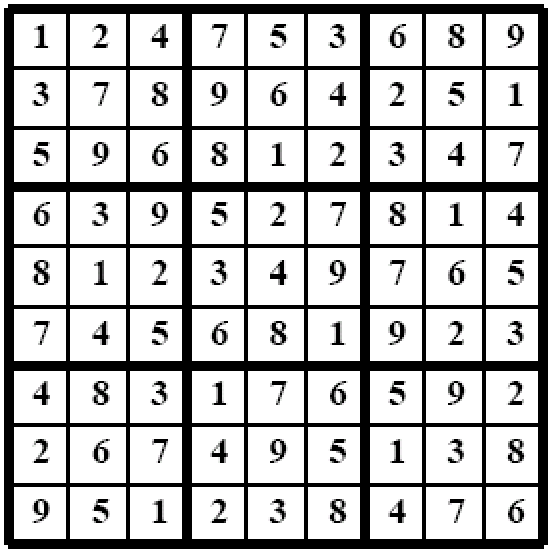
26.
```
bool
rotated(char *s, char *t)
{
    // determine length of s
    int n = strlen(s);

    // ensure lengths are equal
    if (n != strlen(t))
        return false;

    // try all possible rotations of s
    for (int i = 0; i < n; i++)
    {
        int matches = 0;
        for (int j = 0; j < n; j++)
            if (s[(i + j) % n] == t[j])
                matches++;
        if (matches == n)
            return true;
    }

    // not rotated!
    return false;
}
```

Sudoku.



1	2	4	7	5	3	6	8	9
3	7	8	9	6	4	2	5	1
5	9	6	8	1	2	3	4	7
6	3	9	5	2	7	8	1	4
8	1	2	3	4	9	7	6	5
7	4	5	6	8	1	9	2	3
4	8	3	1	7	6	5	9	2
2	6	7	4	9	5	1	3	8
9	5	1	2	3	8	4	7	6