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

Quiz 0

out of 61 points

Print your name on the line below.

Do not turn this page over until told by the staff to do so.

This quiz is "closed-book." However, you may utilize during the quiz one two-sided page (8.5" $\times$ 11") of notes, typed or written, and a pen or pencil, but nothing else.

Scrap paper is included at this document's end.
Unless otherwise noted, assume that any code herein is in C.

Please circle your section leader's name.

Aaron Oehlschlaeger

Alex Chang

Andrew Sellergren

Andy Lei
formerly Alex Hugon

Batool Ali

Brett Thomas

Chris Power

Chris Simmons

Daniel Carroll

Daniel Steinbrook

David Ramos

David Wu

Jesse Cohen

Josh Bolduc

Ken Parreno

Kent Rakip

Linfeng Yang

Mike Tucker

Patrick Quinn

Patrick Schmid

Peter Bailis

Peter Lifland

Sanjay Gandhi

Tom Wionzek

Tomo Lazovich

Vivek Sant

Will Phan

Yuhki Yamashita

Zeina Oweis

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

for staff use only

final score out of 61

Short Answers.

0. (0 points.) Just wanted you to start with a smile.[1]

1. (2 points.) Perform the following calculation *in binary*. Be show to show your work (*i.e.*, any 1s carried).

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

[1] It'll be your last.

for staff use only
–

2. (3 points.) Consider the code below.[2]

```
for (int i = 1; i <= 10; i++)
{
    printf("%d, ah ah ah... ", i);
    sleep(1);
}
```

Although this code was meant to simulate counting from 1 to 10, pausing one second between numbers, it instead appears to do nothing for ten seconds, after which it finally prints everything all at once before quitting.

In one or more sentences, explain why. Then explain, in one or more additional sentences (or actual lines of code), how to fix the problem.

3. (4 points.) Suppose that the code below is someone's attempt at writing a function that, given an array of `ints` and the array's size, is supposed to fill that array with `ints` from the user. Point out and correct at least four (4) mistakes that the author has made.

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

for staff use only
–

[2] Think Sesame Street if that makes the problem more fun.

4. (4 points.) Consider the code below.

```
int a = 0;
int b = -1;
int c = b;
c++;
int *d = &b;
int e = *d + 1;
*d += 1;
```

Complete the table below, specifying exactly what would be printed by each call to `printf`, assuming `printf` is called after *all* of the lines above have been executed. We've plucked off the first row in the table for you.

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

5. (2 points.) Consider the code below.

```
char letters[26];
for (int i = 0; i < 26; i++)
    letters[i] = 'A' + i;
printf("%c", letters[24]);
printf("%c", *(letters + 4));
printf("S");
```

What does this code print?

for staff use only

–

strcrazy.

6. (6 points.) Below is an excerpt, somewhat simplified, from the man page for `strncpy` (and `strcpy`).

```
SYNOPSIS
       #include <string.h>

       char *strcpy(char *dest, char *src);

       char *strncpy(char *dest, char *src, int n);

DESCRIPTION
       The strcpy() function copies the string pointed to by src,
       including the terminating null byte ('\0'), to the buffer pointed
       to by dest.  The destination string dest must be large enough
       to receive the copy.

       The strncpy() function is similar, except that at most n bytes of
       src are copied.  Warning: If there is no null byte among the
       first n bytes of src, the string placed in dest will not be NULL-
       terminated.

       If the length of src is less than n, strncpy() pads the remainder
       of dest with NULL bytes.

RETURN VALUE
       The strcpy() and strncpy() functions return a pointer to the
       destination string dest.
```

Suppose that the author of `string.h` only got as far as writing this man page and not `strncpy` itself. Complete the implementation of `strncpy` (not `strcpy`) on the page that follows, without, to be clear, calling the version of `strncpy` that we know exists in reality. Be sure that your implementation adheres to this man page's specs. You may assume that neither `dest` nor `src` will be `NULL`.

```
char *
strncpy(char *dest, char *src, int n)
{
```

for staff use only
–

Multiple Choice.

For each of the following questions or statements, circle the letter (a, b, c, or d) of the one response that best answers the question or completes the statement; you need not explain your answers.

7. (1 point.) The size of a pointer to a `long long` is

   a. 1 byte.
   b. 32 bytes.
   c. 32 bits.
   d. 64 bits.

8. (1 point.) Consider the two segments of code below, one at left, one at right, in which `x` is an `int` initialized to some value.

```
// first segment
while (0 < x)
{
    x -= 1;
}
printf("x = %d\n", x);
```

```
// second segment
do
{
    x -= 1;
}
while (0 < x);
printf("x = %d\n", x);
```

   Under which of the following conditions will the two segments differ in output?

   I. `x` is `0` just before each segment executes
   II. `x` is greater than `0` just before each segment executes
   III. `x` is less than `0` just before each segment executes

   a. I only
   b. III only
   c. I and II only
   d. I and III only

   } circle one of these

9. (1 point.) Suppose that `x` is an `int`. Which statements below print identical output if `x` is (re-)initialized to `0` just before each statement executes?

   I. `printf("%d", 0);`
   II. `printf("%d", x);`
   III. `printf("%d", x++);`
   IV. `printf("%d", ++x);`

   a. I and II only
   b. I and II and III only
   c. III and IV only
   d. I and II and IV only

   } circle one of these

for staff use only

–

Rapid Fire.

Answer the questions below in no more than two sentences each.[3]

10. (2 points.) What does it mean to exploit a buffer overflow?

11. (2 points.) Name two ways in which you can induce a segmentation fault.

12. (2 points.) With respect to memory management, how is the heap different from the stack?

13. (2 points.) Why implement a function recursively if an iterative solution is also possible?

14. (2 points.) Why is Vigenère's cipher more secure than Caesar's?

[3] Don't think we won't count!

for staff use only

–

Ugh. Bugs.

Answer the questions below in no more than two sentences each.[4]

15. (1 point.) Even though the function below is supposed to return `true` only if `n` is positive, it always returns `true`. Why?

```
bool
f(int n)
{
    if (n > 0);
        return true;
    return false;
}
```

16. (1 point.) Although the function below appears to induce an infinite loop, it does, in fact, eventually stop printing asterisks. Why?

```
void
f()
{
    for (int i = 1; i > 0; i++)
        printf("*");
}
```

17. (1 point.) Even though the function below is supposed to return `true` if `n` equals `0`, it always returns `false`. Why?

```
bool
f(int n)
{
    if (n = 0)
        return true;
    else
        return false;
}
```

for staff use only

–

[4] Don't think we won't count!

Role Reversal.

18. (1 point.) Suppose that you're a TF and a student comes up to you at office hours. The student explains that GCC is spitting out the message below.

```
control reaches end of non-void function
```

In one or more sentences, explain to this student (well, us) what this warning means with respect to the student's code.

19. (1 point.) Suppose that you're still a TF and that same student comes up to you again at office hours. The student explains that GCC is now spitting out the message below.

```
implicit declaration of function 'GetString'
```

In one or more sentences, explain to this student (nicely!) what he or she is doing wrong now.

20. (1 point.) Suppose that you're still a TF and your favorite student comes up to you yet again at office hours, this time bringing an apple, which kinda makes it all worthwhile. The student explains that GCC is now spitting out the message below.

```
undefined reference to 'GetString'
```

In one or more sentences, explain to this student what he or she is doing wrong now.

for staff use only

–

Argh. Args.

For the three questions below, suppose that you've executed some program, `a.out`, as follows.

```
username@cs50.net (~/quiz0/): a.out is such a silly name for a program
```

21. (1 point.) If `main` contains only the line below, what gets printed?

```
printf("%d", argc);
```

22. (2 points.) If `main` instead contains only the line below, what gets printed?

```
printf("%c%c", argv[2][0], argv[1][0]);
```

23. (1 point.) If `main` instead contains only the line below, what gets printed?

```
printf("%d", strlen(argv[0]));
```

for staff use only

–

O(mega).

24. (7 points.) Consider the laundry list of algorithms below.

    Binary Search, Bubble Sort, Insertion Sort, Linear Search, Merge Sort, Selection Sort

    Complete the table below, providing next to each bound one (and only one) algorithm known to be constrained by that bound; you may assume any implementation details that you wish. You may draw only from the laundry list above, but you may use each algorithm in that list once, more than once, or not at all; the table below may thus contain duplicates. We've plucked off the first row in the table for you.

Bound	**Algorithm**
$O(n^2)$	Selection Sort
$\Omega(n^2)$	
$O(n \log n)$	
$\Omega(n \log n)$	
$O(n)$	
$\Omega(n)$	
$O(\log n)$	
$\Omega(1)$	

25. (4 points.) In one or more sentences, explain (in English, not code or pseudocode) how Selection Sort works. Then, in one or more additional sentences, explain why Selection Sort is in $O(n^2)$.

for staff use only

–

HAI O.

26. (6 points.) A string *t* is said to be a "cyclic rotation" of a string *s* if it's possible to "rotate" the characters in *s* by some number of positions, *n*, such that *s* becomes *t*. We don't mean "rotate" in a Caesar sense here but in a left-right (or right-left) sense. For instance, *t* = "abanan" is a cyclic rotation of *s* = "banana" because rotating the characters in "banana" to the right by *n* = 1 (*i.e.*, by one position), wrapping accordingly, yields "abanan". It's worth noting that rotating the characters in *s* by *n* = 7 (or 13, 19, 25, *etc*.) would also yield *t*, since the length of "banana" is 6. Similarly is *t* = "nabana" a cyclic rotation of *s* = "banana" because rotating the characters in "banana" by *n* = 2 (or 8, 14, 20, *etc*.) positions yields "nabana". It's even valid to say that *s* = "banana" is a cyclic rotation of *t* = "banana" because rotating *s* by *n* = 0 (or 6, 12, 18, *etc*.) positions indeed yields "banana".

    Complete the implementation of `rotated` on the page that follows in such a way that `rotated` returns `true` if `t` is a cyclic rotation of `s` (*i.e.*, there exists some *n* by which you can rotate *s* in order to get *t*) else `false`. You may assume that neither `s` nor `t` will be `NULL`. For time's sake, comments are not necessary. But if you find yourself running out of time, you may resort to pseudocode for partial credit.

```
bool
rotated(char *s, char *t)
{
```

for staff use only

–

Sudoku.

This is not extra credit. 'Tis just for fun.

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

Scrap Paper.

Nothing on this page will be examined by the staff unless otherwise directed in the space provided for some question.

Scrap Paper.

Nothing on this page will be examined by the staff unless otherwise directed in the space provided for some question.