This is **CS50**.
Harvard College Fall 2011

Quiz 0

out of 100 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" × 11") of notes, typed or written, and a pen or pencil, nothing else.

Scrap paper is included at this document's end.
Unless otherwise noted, assume that any problems herein refer to C.
Unless otherwise noted, you may call any functions we've encountered this term in code that you write.

Circle your teaching fellow's name.

Alex Chang	Jordan Jozwiak	Neal Wu
Alex Hugon	Joseph Ong	Patrick Thornycroft
Andrew Wang	Joshua Lee	Paul Bowden
Ashin Shah	Julia Mitelman	Paul Handorff
Bannus Van der Kloot	Julie Zhang	Peter Hung
Bo Han	Karen Xiao	R.J. Aquino
Bob Kinney	Kenny Yu	Rob Bowden
Cheng Huang	Kevin Zhang	Sebastian Pierce-Durance
Cragin Godley	Larry Ehrhardt	Sophie Chang
Dan Bradley	Levi Roth	Steven Tricanowicz
David Palmer	Lexis Ross	Tommy MacWilliam
Doug Lloyd	Matthew Chartier	Tony Ho
Jack Greenberg	Melissa Niu	Travis Downs
Jason Hirschhorn	Michael Chen	Vanessa Tan
Jenny Ye	Michael Tingley	Wellie Chao
Jimmy Sun	Michelle Luo	Yacoub Kureh
John Lee	Naomi Bolotin	Zak Burke

for staff use only

final score out of 100

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.

0. (0 points.) Okay, so how do you spell *caterpillar*?

   a. cat
   b. catepillar
   c. caterpillar
   d. google.com

1. (1 point.) How many times maximally can you tear a 1,024-page phonebook in half (without tearing individual pages like, ahem, Jason)?

   a. 8
   b. 10
   c. 16
   d. 32

2. (1 point.) How many distinct values can you represent with a sequence of 3 bits?

   a. 3
   b. 6
   c. 8
   d. 9

3. (1 point.) How many bits does a single hexadecimal digit (*e.g.*, `F`) ordinarily represent?

   a. 2
   b. 4
   c. 8
   d. 15

4. (1 point.) Assuming a 26-letter alphabet, how many *n*-letter keywords are possible when using Vigenère's cipher?

   a. 26
   b. n
   c. n^{26}
   d. 26^{n}

5. (1 point.) A function that calls itself is said to be

   a. an infinite loop.
   b. buggy.
   c. iterative.
   d. recursive.

for staff use only

–

True or False.

For each of the statements below, circle T if the statement is true or F if the statement is false.

6. T F (0 points.) David's favorite words are *just* and, sadly, *gonna*.
7. T F (1 point.) Any `.c` file must have a `main` function.
8. T F (1 point.) `malloc` allocates memory on the stack.
9. T F (1 point.) `NULL` is a special `char` that demarcates the end of a `string`.

O hai, Scratch.

10. (3 points.) Suppose that a Scratch project has a single sprite (a cat) that's implemented with exactly two scripts, pictured below.

Even though this project is supposed to simulate a cat that doesn't like to be petted, the cat never seems to roar when touched with one's cursor (*i.e.*, mouse-pointer), even after the green flag has been clicked. Instead, the cat seems only to meow infinitely.

Explain why in a sentence and explain how to fix in another. No need to draw puzzle pieces; words suffice.

for staff use only

–

O hai, C.

11. (1 point.) Consider the program below.

```
#include <stdio.h>

int
main(int argc, char *argv[])
{
    printf("%d\n", argc);
    return 0;
}
```

Suppose that this program, once compiled as `a.out`, is executed as follows:

```
./a.out nom nom nom
```

Exactly what gets printed?

***O* no, it's Omega.**

12. (7 points.) Complete the table below by specifying lower (Ω) and upper (*O*) bounds for each algorithm. Assume that the input to each algorithm is an array of size *n*. We've plucked off three cells for you.

	lower (Ω)	**upper (*O*)**
Binary Search		
Bubble Sort	n	
Linear Search		n
Merge Sort		
Selection Sort		n^2

for staff use only
–

Scratch meets C.

13. (6 points.) Recall that Scratch can generate pseudorandom integers between any two values (*e.g.*, 50 and 164), inclusive, via the puzzle piece below:

By "inclusive," we mean that the above may return 50, 164, or any integer in between.

By contrast, C comes with `rand`, which returns a pseudorandom `int` specifically between `0` and `RAND_MAX`, inclusive, where `RAND_MAX` is a large, positive constant. If only there were a way to generate a pseudorandom number in C between any two values! Complete the implementation of `GetRandom` below in such a way that it uses `rand` but returns a pseudorandom `int` between `min` and `max`, inclusive. You may assume that both `min` and `max` will be non-negative and less than or equal to `RAND_MAX`. And you may assume that `max` will be greater than or equal to `min`. You needn't worry about seeding. And no need to `#include` any header files (*e.g.*, `stdlib.h`, in which `rand` is declared).

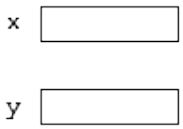
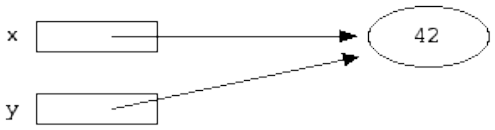
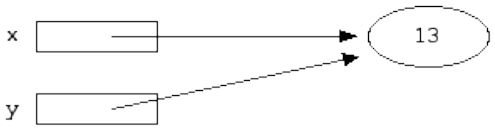
```
int
GetRandom(int min, int max)
{
    int n = rand();
```

for staff use only

–

Pointer Fun without Binky.

14. (6 points.) The column at right in the table below depicts execution of five successive lines of code, but only two of those lines appear in the column at left. Complete the table by supplying, in the column at left, the three missing lines. If more than one answer is possible for some line, you need only provide one.

This statement...	Results in this picture in RAM...
`int *x, *y;`	
`x = malloc(sizeof(int));`	
	
	
	

Don't try this at home.

15. (3 points.) In the space below, write a complete program (however short) that might actually segfault once compiled and run. (You may `#include` any header files that you would like atop `main`.) Explain in a comment why your program might segfault.

for staff use only

–

Binary Time.

16. (2 points.) Perform the following calculation in binary. Show any work (*i.e.*, any 1s carried).

```
  0 0 1 1 0 0 1 0
+ 0 0 1 1 0 0 1 0
```

17. (2 points.) Convert the binary number below to decimal. Show any work (*i.e.*, any arithmetic).

```
1 0 1 0 0 1 1 1 0 0 1
```

Myth Busters.

Refute each of the TF's claims below, explaining why the TF's claim isn't right.

18. (2 points.) Just the other day, Doug claimed: "I have invented an algorithm that can sort an array of n numbers in $O(\log n)$ time!"

19. (2 points.) Just the other day, Melissa claimed: "Just dragging a file to your Recycle Bin or Trash Can doesn't delete it permanently, but emptying your Recycle Bin or Trash Can does."

for staff use only

–

Role Reversal.

Suppose that you're no longer a student but a TF instead. Respond in two or more sentences to each of the following emails in the space below each.

20. (3 points.)

Heyyyy Rob,

So this program is supposed to hand customers their change one penny at a time (by saying "Here's a penny..." for each), but it seems to get stuck in an infinite loop if I input anything other than `0.00`. How come? How can I fix? Below's my code. kthxbai

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

int
main(void)
{
    printf("Change: ");
    float change = GetFloat();
    while (change != 0.00)
    {
        printf("Here's a penny...\n");
        change -= 0.01;
    }
}
```

for staff use only

–

21. (2 points.)

Hi Rob,

So I'm trying to write a program that prints an infinite number of stars (*), one per line. (Don't ask why.) But my program always quits after a while. How come? Here's my code.

```
#include <stdio.h>

int
main(void)
{
    for (int i = 0; i >= 0; i++)
      printf("*\n");
}
```

22. (2 points.)

Hey Matt,

So I used to email Rob with my questions, but I'm worried his answers aren't always ... right. (Please don't tell him I said that!!) Anyhow, this program's supposed to print 50 stars (*), one per line, but it doesn't. What's wrong? How can I fix? Here's my code. (Really, don't tell Rob.)

```
#include <stdio.h>

int
main(void)
{
    for (int i = 0; i < 50; i++)
        printf("*");
        printf("\n");
}
```

for staff use only

–

Design.

Consider the program below.

```
#include <stdio.h>

int
main(int argc, char *argv[])
{
    for (int i = 0; i < strlen(argv[0]); i++)
        printf("%c\n", argv[0][i]);
}
```

23. (1 point.) If this program is compiled as `a.out` and executed with

    ```
    ./a.out
    ```

    exactly what does it print?

24. (1 point.) As short (and uninteresting!) as this program is, it nonetheless manifests bad design because of an inefficiency. Explain in a sentence why this implementation is inefficient.

25. (1 point.) In the space below, re-write the program in such a way that its inefficiency is gone but its output is otherwise identical.

    ```
    #include <stdio.h>

    int
    main(int argc, char *argv[])
    {
    ```

for staff use only

–

Fun with Tables.

26. (4 points.) Consider the function below.

```
bool
mystery(char *s)
{
    for (int i = 0, j = strlen(s) - 1; i < j; i++, j--)
    {
        if (s[i] != s[j])
            return false;
    }
    return true;
}
```

Suppose that `mystery` is passed each of the arguments in the table below, one at a time. Complete the table by specifying, to the right of each argument, what the return value of `mystery` would be when passed that particular argument.

argument	return value
`"a"`	
`"roar"`	
`"radar"`	
`"monkey"`	

27. (4 points.) Complete the table below by recording to the right of each type its size in bytes (not bits). Assume a 32-bit architecture like the CS50 Appliance. We've plucked off two cells for you.

type	bytes
`char`	
`char *`	
`int`	4
`int *`	4
`long long`	
`long long *`	

for staff use only

–

Reinventing Some Wheels.

28. (4 points.) Complete the implementation of `pow` below in such a way that the function returns x^y (*i.e.*, `x` raised to the power of `y`) unless `x` or `y` (or both) is negative, in which case the function should instead return `-1`. Recall that, mathematically, x^0 is `1` and that x^1 is `x`. You needn't worry about integer overflow. Suffice it to say you may not call the version of `pow` that's declared in `math.h`!

```
int
pow(int x, int y)
{
```

29. (4 points.) Complete the implementation of `isupper` below in such a way that the function returns `false` unless its argument is an uppercase letter, in which case the function should instead return `true`. You may assume that `true` and `false` have been defined for you, as via `cs50.h`; no need to `#include` any header files yourself. Suffice it to say you may not call the version of `isupper` that's declared in `ctype.h`! Nor may you call `isalpha` or `islower`.

```
bool
isupper(char c)
{
```

for staff use only

–

30. (4 points.) Complete the implementation of `tolower` below in such a way that the function returns its argument, lowercased, unless the argument is not alphabetical, in which case the function should return its argument unchanged. Suffice it to say you may not call the version of `tolower` that's declared in `ctype.h`! But you may call your own version of `isupper` if you would like.

```
char
tolower(char c)
{
```

31. (8 points.) Recall that `atoi` is a function that converts a `string` (aka `char *`) that looks like an integer (*e.g.*, `"123"`) to an actual integer (*e.g.*, `123`). Odds are you used that function to convert `argv[1]` to a numeric key for `caesar` in Problem Set 2. Suppose that `atoi` doesn't yet exist and so you have to implement it yourself. Complete the implementation of `atoi` below. You may assume that `s` will be a string of non-zero length composed entirely of numbers (`0` through `9`); it will not be `NULL`. Suffice it to say you may not call the version of `atoi` that's declared in `stdlib.h`. Nor may you call `sscanf`. But you may call other functions (from `math.h`, `stdio.h`, `stdlib.h`, *etc.*) if you would like; no need to `#include` any header files. You needn't worry about integer overflow.

```
int
atoi(char *s)
{
```

for staff use only

–

Rapid Fire. (2 points each.)

32. What's the difference between `\n` and `\r`?

33. In what sense does `GetString`, as defined in the CS50 Library, have a memory leak?

34. What's a garbage value in the context of a local variable on the stack?

35. Why is it necessary to have

```
#include <string.h>
```

before a function that calls `strlen`?

36. What's a breakpoint in the context of a debugger?

for staff use only

–

User Input.

Recall the program below from lecture.

```
#include <stdio.h>

int
main(void)
{
    int x;
    printf("Number please: ");
    scanf("%d", &x);
    printf("Thanks for the %d!\n", x);
}
```

37. (1 point.) What does `scanf` do in this program?

38. (2 points.) Why must `x` be preceded by `&` in order for `scanf` and, in turn, this program to work?

Compare and Contrast.

Consider the two lines of code below.

```
// first line
int grades[9];

// second line
int *grades = malloc(sizeof(int) * 9);
```

39. (1 point.) Compare the two lines of code in a sentence: how are they similar?

40. (1 point.) Contrast the two lines of code in a sentence: how are they different?

for staff use only

–

Meaning of Life.

41. (2 points.) Consider the implementation of `life` below.

```
bool
life(int n)
{
    return (n == 42) ? true : false;
}
```

Re-implement `life` below in such a way that it does not use C's ternary operator but otherwise behaves identically.

```
bool
life(int n)
{
```

42. (3 points.) Atop each line of code below, write a comment that explains, with respect to memory, precisely what the line of code does.

```
//
int life = 42;


//
int *ptr = &life;


//
*ptr = 50;
```

for staff use only

–

Scrap Paper.

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