Quiz 0

Answer Key

Answers other than the below may be possible.

Multiple Choice.

0. c
1. b
2. c
3. b
4. d
5. d

True or False.

6. T
7. F
8. F
9. F

O hai, Scratch.

10. Whereas the lefthand script contains a loop that induces the infinite meowing, the righthand script does not, and so its statements execute only once, the instant the green flag is clicked. By the time the user moves his or her cursor to the cat, the righthand script is already done executing. Wrapping the righthand script's condition with a similar loop would fix.

O hai, C.

11. `4`

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

12.

	Ω	***O***
Binary Search	1	log *n*
Bubble Sort	*n*	n^2
Linear Search	1	*n*
Merge Sort	*n* log *n*	*n* log *n*
Selection Sort	n^2	n^2

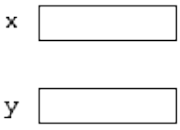
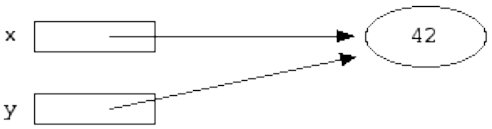
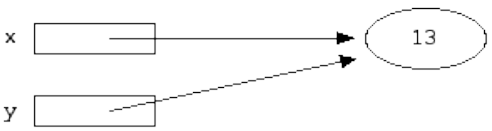
Scratch meets C.

13.

```
int
GetRandom(int min, int max)
{
    int n = rand();
    return min + n % (max - min + 1);
}
```

Pointer Fun without Binky.

14.

This statement...	Results in this picture in RAM...
`int *x, *y;`	
`x = malloc(sizeof(int));`	
`*x = 42;`	
`y = x;`	
`*x = 13;`	

Don't try this at home.

15.
```
#include <stdio.h>

int
main(void)
{
    printf("goodbye, %s\n", NULL);
}
```

Binary Time.

16.

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

17. $1 \cdot 1024 + 1 \cdot 256 + 1 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 1 \cdot 1 = 1337$

Myth Busters.

18. Sorting an array of *n* numbers requires, at least, $\Omega(n)$ time. Even if the input is already sorted, you at least need to examine all *n* numbers to realize as much.

19. When you empty your Recycle Bin or Trash Can, your operating system only "forgets" where the file is; its bits remain untouched. So long as those bits don't get overwritten (as by creating new files), you can usually recover deleted files with special software, like the program I'm about to write for Problem Set 5!

Role Reversal.

20. Floating-point arithmetic is inherently approximate. For example, `0.01` cannot be represented exactly in binary floating-point format. But an equality test requires an exact match. So testing computed floating-point values for equality is rarely useful. In the case of money, it's best to convert your change to pennies (as by multiplying by `100`, rounding, and casting to `int`) so as to perform all subsequent arithmetic with integers.

21. An `int` only has so many bits (32 on a 32-bit machine), and so if you count high enough, you'll end up "wrapping around" back to negative numbers. As soon as `i` becomes negative, your loop's condition will be `false`, and so your loop and, in turn, program will finish.

22. Even though you've indented your second `printf` statement, only the first will execute as part of the loop, so you're printing all of your stars on the same line. To fix, you can wrap both `printf` statements in curly braces, or you can remove the second altogether and have the first print `"*\n"`.

Design.

23.
```
.
/
a
.
o
u
t
```

24. On each iteration of the loop, the program calls `strlen`, even though the length of `argv[0]` isn't changing.

25.
```
int
main(int argc, char *argv[])
{
    for (int i = 0, n = strlen(argv[0]); i < n; i++)
        printf("%c\n", argv[0][i]);
}
```

Fun with Tables.

26.

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

27.

`char`	1
`char *`	4
`int`	4
`int *`	4
`long long`	8
`long long *`	4

Reinventing Some Wheels.

28.

```
int
pow(int x, int y)
{
    if (x < 0 || y < 0)
        return -1;

    int product = 1;
    for (int i = 0; i < y; i++)
        product *= x;
    return product;
}
```

29.

```
bool
isupper(char c)
{
    if (c >= 'A' && c <= 'Z')
        return true;
    else
        return false;
}
```

30.

```
char
tolower(char c)
{
    if (isupper(c))
        return c - 'A' + 'a';
    else
        return c;
}
```

30.

```
int
atoi(char *s)
{
    int sum = 0;
    for (int i = 0, n = strlen(s); i < n; i++)
        sum = 10 * sum + s[i] - '0';
    return sum;
}
```

Rapid Fire.

32. Whereas `\n` moves the cursor down a line, `\r` simply moves the cursor back to the beginning of the current line.

33. In order to store a user's input, `GetString` calls `malloc`, but it never calls `free` to release the memory allocated.

34. Because memory on the stack is used and reused as functions are called and return, a local variable, if not initialized explicitly in code, is said to contain a "garbage value" if its bits just so happen to have been used previously by some other stack frame.

35. In `string.h` is the prototype for `strlen`, which informs the compiler of `strlen`'s existence, along with its return type and parameter. Without the `#include`, `gcc` won't recognize the function and might abort compilation altogether.

36. A breakpoint is a stopping point in a program that's been explicitly defined by a programmer with a debugger. It allows the program to inspect a program's memory at that point in its execution as well as step through subsequent instructions one at a time.

User Input.

37. `scanf` tries to read an `int` from standard input (*e.g.*, a user's keyboard) and store it in `x`.

38. `scanf` needs to be able to change the value of `x` (not a copy thereof), and so it must be passed in via its address, which `&` provides.

Compare and Contrast.

39. Both lines allocate an array of 9 `int`'s (whose size totals, on a 32-bit machine, 36 bytes).

40. Whereas the first line allocates the array on the stack, the second line allocates the array on the heap; the second line additionally allocates 32 bits (on a 32-bit machine) for a pointer called `grades`. (In the first line, `grades` is just a "symbol" that consumes no space.)

Meaning of Life.

41.
```
bool
life(int n)
{
    if (n == 42)
        return true;
    else
        return false;
}
```

42.
```
// declare an integer, life, and assign it a value of 42
int life = 42;

// declare a pointer to an int, ptr, and store the address of life in it
int *ptr = &life;

// dereference ptr and store 50 at that location in RAM (i.e., in life)
*ptr = 50;
```