# Quiz 1

**Answer Key**

Answers other than the below may be possible.

**Getting Strings.**

0. Version 1 declares `s` as a pointer but doesn't initialize it to the address of an actual block of memory. The variable thus contains a "garbage value" that is not likely an address to which `scanf` is allowed to write, and so the program might segfault.
1. If a user inputs more than 49 characters, version 2 might segfault because `scanf` will write beyond the boundaries of `s`, which is an array of size 50, which is only large enough for a 49-`char` string plus `'\0'`.
2. If a user inputs more than 49 characters, version 3 might segfault just like version 2. And version 3 might also segfault if `malloc` happens to return `NULL`, since version 3 does not check the function's return value before having `scanf` write to `s`.
3. Whereas version 2 uses 50 bytes on the stack, version 3 uses 50 bytes on the heap.

**Small Bytes.**

4.

| **type** | **bytes** |
|---|---|
| `node` | 8 |
| `node*` | 4 |
| `string` | 4 |
| `struct node*` | 4 |
| `student` | 8 |

**HTTP 1/1.**

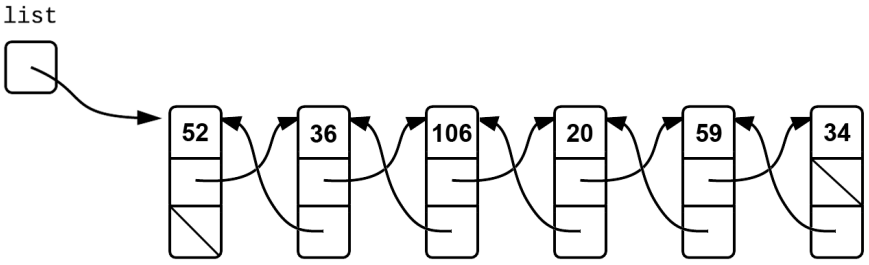
5. If the server can successfully respond to a client's request for some URL, it responds with 200 since all is indeed OK.
6. If a server recognizes a client's request for some URL but would like to redirect the client to some other URL, it responds with 302.
7. If a server is unable to access some file requested by a client (e.g., because it doesn't have permission, as via `chmod`), it responds with 403.
8. If a server is unable to find some file requested by a client (e.g., because it doesn't exist, or its name is misspelled), it responds with 404.
9. `http://tools.ietf.org/html/rfc2324`
10. If a server is improperly configured (e.g., because of a typo in some configuration file) or if there's a syntax error in some file that needs to be interpreted (e.g., a PHP file), it responds with 500.

**Double Trouble.**

11.
```
void insert(int n)
{
    node* ptr = malloc(sizeof(node));
    if (ptr != NULL)
    {
        ptr->n = n;
        ptr->prev = NULL;
        ptr->next = list;
        if (list != NULL)
        {
            list->prev = ptr;
        }
        list = ptr;
    }
}
```

12.
```
void remove(int n)
{
    node* ptr = list;
    while (ptr != NULL)
    {
        if (ptr->n == n)
        {
            if (ptr == list)
            {
                list = ptr->next;
                if (list != NULL)
                {
                    list->prev = NULL;
                }
            }
            else
            {
                ptr->prev->next = ptr->next;
                if (ptr->next != NULL)
                {
                    ptr->next->prev = ptr->prev;
                }
            }
            free(ptr);
            return;
        }
        ptr = ptr->next;
    }
}
```

13.




**World Wide What.**

14. The browser first looks up the IP address of `www.facebook.com` by checking its own cache and then asking the local operating system as needed, which in turn contacts a DNS server as needed. The browser then sends an HTTP request to port 443 of that address via a nearby router, which relays the request to that address. Upon receipt of the request, Facebook responds with the HTML that composes Facebook's home page.

**Quadded.**

15.
```
<ul>
    <?php

        foreach ($quad as $house)
        {
            print("<li>{$house}</li>");
        }

    ?>
</ul>
```

**Tradeoffs.**

16. An upside is that user is informed of errors faster, since server doesn't need to be contacted. A downside is that JavaScript can be disabled, so an adversary could still send invalid data to the server.

17. An upside is that a linked list can grow and shrink dynamically, whereas an array is of fixed size. Downsides include: a linked list requires more memory (to store pointers); and a linked list does not offer random access (as for binary search).

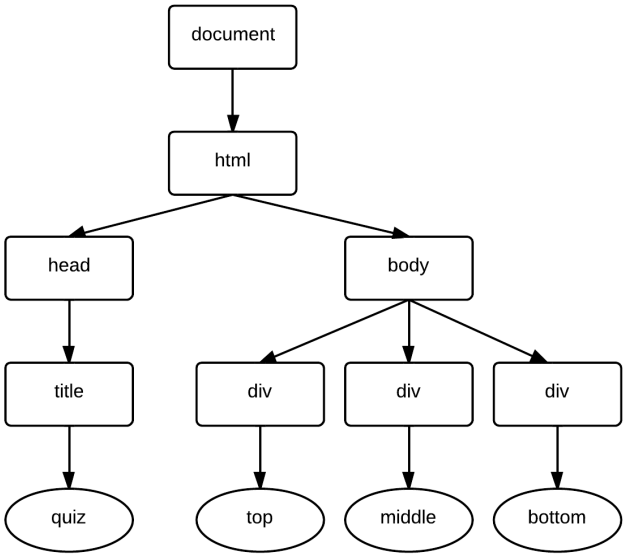
18. Upsides include: PHP has many more functions built into it, which makes it easier to solve problems quickly; and PHP does not need to be compiled, which saves a step. A downside is that programs written in PHP, an interpreted language, are often slower to execute.

19. An upside is that SQL databases can be searched more efficiently than CSVs thanks to indexes. A downside is that a SQL database typically require a server, which involves additional complexity.

20. An upside is that a trie allows for constant-time lookups (or at least $O(k)$, where $k$ is the length of a key), whereas a hash table with separate chaining theoretically only allows for $O(n)$, where $n$ is the number of keys in the structure. A downside is that a trie uses a significant amount of memory, even though much of the space (allocated for pointers) tends to be unused.

**DOM, DOM DOM DOM.**

21.




**Attack!**

22. If an adversary provides more bytes than were anticipated for `bar` via `argv[1]`, some of which collectively represent executable code, and the adversary successfully overwrites some "return address" on the program's stack with the address of that executable code, then a program might be tricked into executing it by "returning" to it from `foo`.

23.
```
void foo(char* bar)
{
    char c[12];
    if (bar != NULL)
    {
        int n = strlen(bar);
        if (n < 12)
        {
            memcpy(c, bar, n);
        }
    }
}
```

**Itsa Mario again?**

24.
```
#!/bin/env php
<?php

    for ($i = 1; $i <= 7; $i++)
    {
        for ($j = 1; $j <= $i; $j++)
        {
            print("#");
        }
        print("\n");
    }
?>
```

**woof am back.**

25.
```
<!DOCTYPE html>

<html>
    <head>
        <script src="http://code.jquery.com/jquery-latest.min.js"></script>
        <script>

            $(document).ready(function() {
                $("#inputs").submit(function() {
                    var value = $("#name").val();
                    if (value != "")
                    {
                        alert("Hello, " + value + "!");
                    }
                    return false;
                });
            });

        </script>
        <title>quiz</title>
    </head>
    <body>
        <form id="inputs">
            <input id="name" type="text"/>
            <input id="button" type="submit" value="Greet"/>
        </form>
    </body>
</html>
```

**MyOhMySQL.**

26.

| Name | Type | Null | Default |
|---|---|---|---|
| **number** | bigint | No | *None* |
| **balance** | decimal | No | 100.00 |

27. `UPDATE accounts SET balance = balance - 20`

28. `SELECT number FROM accounts WHERE balance > 1000`

29. `DELETE FROM accounts WHERE balance = 0`

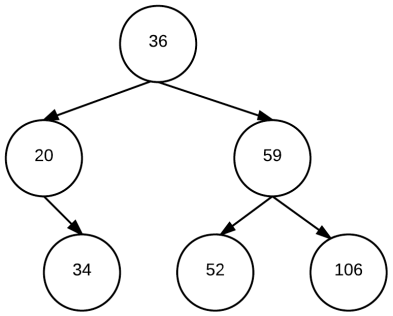
**Queue Guide.**

30.
```
bool enqueue(int n)
{
    if (q.size == CAPACITY || n < 0)
    {
        return false;
    }
    q.numbers[(q.size + q.front) % CAPACITY] = n;
    q.size++;
    return true;
}
```

31.
```
int dequeue(void)
{
    if (q.size == 0)
    {
        return -1;
    }
    q.size--;
    int n = q.numbers[q.front];
    q.front = (q.front + 1) % CAPACITY;
    return n;
}
```

**Pssst, BST.**

32.




33.
```
typedef struct node
{
    int n;
    struct node* left;
    struct node* right;
}
node;
```

34.
```
void traverse(node* root)
{
    if (root == NULL)
    {
        return;
    }
    traverse(root->left);
    printf("%i\n", root->n);
    traverse(root->right);
}
```