Quiz 1

Answer Key

Answers other than the below may be possible.

Stack Attack.

0.
```
bool push(int datum)
{
    if (stack.size == CAPACITY)
    {
        return false;
    }
    stack.data[stack.size] = datum;
    stack.size++;
    return true;
}
```

1.
```
bool pop(int* location)
{
    if (stack.size == 0)
    {
        return false;
    }
    *location = stack.data[stack.size - 1];
    stack.size--;
    return true;
}
```

Hash this out.

2. Because the hash function bases its output only on the first letter of its input, it will tend to output some values more frequently than others, since some letters are more common at the start of English words than others. Some of the hash table's chains will thus grow longer (and slower to traverse) than others.

3.
```
unsigned int size(void)
{
    unsigned int size = 0;
    for (int i = 0; i < 26; i++)
    {
        node* ptr = table[i];
        while (ptr != NULL)
        {
            size++;
            ptr = ptr->next;
        }
    }
    return size;
}
```

Trie this.

4.
```
unsigned int size(node* n)
{
    unsigned int counter = 0;
    if (n == NULL)
    {
        return 0;
    }
    if (n->word)
    {
        counter++;
    }
    for (int i = 0; i < 26; i++)
    {
        counter += size(n->children[i]);
    }
    return counter;
}
```

Grammatically Correct.

5.

start-line	GET /register.php?name=David&dorm=Matthews+Hall HTTP/1.1
request-line	GET /register.php?name=David&dorm=Matthews+Hall HTTP/1.1
method	GET
request-target	/register.php?name=David&dorm=Matthews+Hall
HTTP-version	HTTP/1.1
absolute-path	/register.php
query	name=David&dorm=Matthews+Hall
header-field	Host: localhost
field-name	Host
field-value	localhost

6.

start-line	HTTP/1.1 200 OK
status-line	HTTP/1.1 200 OK
HTTP-version	HTTP/1.1
status-code	200
reason-phrase	OK
header-field	Content-Type: application/json
field-name	Content-Type
field-value	application/json
message-body	{"symbol":"FREE","name":"FreeSeas Inc.","price":0.15}

Better GET permission.

7. `200`, because `index.html` is world-readable (and its ancestors are world-executable).

8. `200`, because `index.html` is world-readable (and its ancestors are world-executable).

9. `404`, because `quote.html` doesn't exist.

10. `403`, because `img` isn't world-executable.

Delete cookies?!

11. A cookie is a key-value pair sent via a `Set-Cookie` HTTP header from a web server to web browser to be stored in RAM or on disk by the latter. On subsequent HTTP requests, the web browser is expected to present the cookie to the web server via a `Cookie` HTTP header so that the web server knows that it's the same user again.

12. `PHPSESSID` is a cookie whose value is a string that uniquely identifies a user's browser. After receipt of that cookie, a browser, by nature of HTTP, will include that cookie's value in all subsequent requests to the website that set it. That value maps, server-side, to a file (or database row) that contains the contents of `$_SESSION`, a PHP superglobal in which a website can store key-value pairs.

13. If the advertiser sets a cookie when responding to a browser's initial request for the image, the browser will send that same cookie with every subsequent request for the image (unless the browser has disabled "third-party" cookies). If the browser also reveals the URL of the page that's been visited (as via a `Referer` HTTP header or via some query string that the advertiser includes in the tag), the advertiser will know on which page the tag is embedded and, thus, which page the user has visited. If a user visits multiple sites that the advertiser has pixeled, the advertiser will receive the same cookie from the user across those sites and thus know which sites the user has visited.

DOM, DOM DOM DOM.

14.
```
typedef struct node
{
    // a pointer to this node's first child, if any; this node's
    // other children are accessible via that first child's nextSibling
    struct node* childNodes;

    // pointer to this node's next sibling, if any; this node is thus
    // effectively the start of a linked list of siblings
    struct node* nextSibling;

    // node's name, if any, is just a string
    char* nodeName;
}
node;
```

Frosh IMs.

15. `GET /register.php?name=David&dorm=Matthews+Hall HTTP/1.1`

16.
```
$(function() {
    $("#registration").submit(function(eventObject) {
        if ($("#name").val() == "" || $("#dorm").val() == "")
        {
            return false;
        }
        return true;
    });
});
```

Having said that...

17. Having said that, JavaScript (and thus client-side validation) can be disabled by users (and users could even send HTTP manually via `curl`, `telnet`, or the like), so implementing server-side validation is also still necessary.

18. Having said that, because PHP is an interpreted (and not compiled) language, PHP programs tend to run more slowly than equivalent C programs.

19. Having said that, because an array (sorted or not) is of fixed size (at least in C), growing or shrinking it is more expensive time-wise than growing or shrinking a linked list.

Why bother?

20. Even though lookups in a hash table are still in $O(n)$ asymptotically, they take wall-clock time on the order of n/k, where k is the number of chains, assuming the chains are of uniform length (i.e., n/k). Lookups in a linked list, meanwhile, take wall-clock time on the order of n (i.e., more time), since a linked list is like one chain of length n.

21. External stylesheets allow CSS properties to be shared across multiple pages. Not only can the stylesheets thus be modified centrally (without having to edit HTML files individually), they can be cached by browsers, which expedites subsequent page loads.

World Wide Wait.

22. The browser first looks up the IP address of `bankofamerica.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, the headers and message body of which are encrypted with HTTPS (i.e., SSL or TLS), to port 443 of that address using TCP, which ensures any dropped data will be re-sent. The network's default router (i.e., gateway) then relays the request to its destination. Upon receipt of the request, the server responds with the HTML that composes the server's home page.

Design Decisions.

23. You should use a `CHAR` when you know in advance that all values in some column will be of (no more than) a fixed length, since the database will be able to search over those values more efficiently if it knows their length. You should use a `VARCHAR` when the values' lengths will vary.

24. You should use `DECIMAL` instead of `FLOAT` when you need a specific number of digits to the right of a decimal point (as is particularly appropriate when storing monetary values), without any risk of floating-point imprecision (as is possible with `FLOAT`).

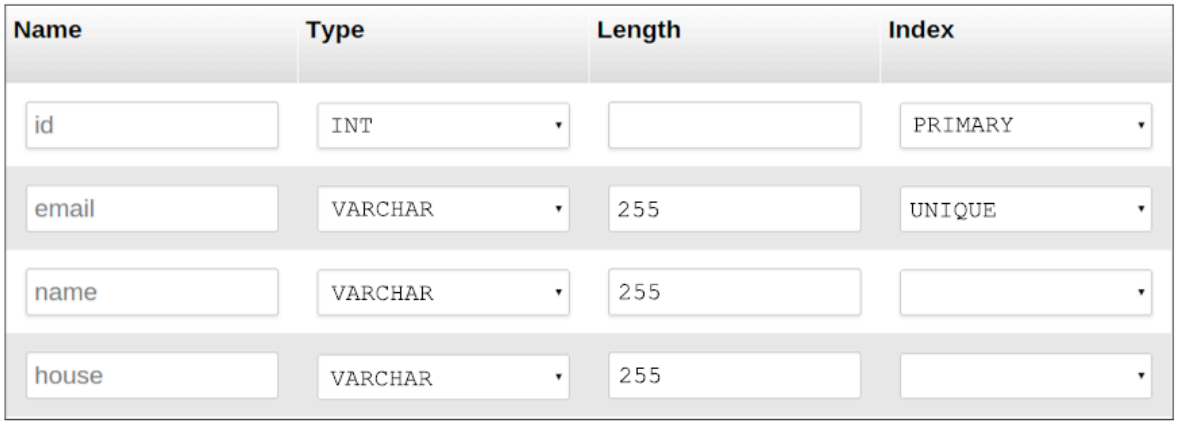
25. You should use `GET` when you want to transmit state (e.g., query strings) via a URL so that the state is bookmarkable or reachable via a clickable link. You should use POST when you do not want state to appear in a URL (and thus a web browser's history or, typically, a web server's logs).

26. You should use `NULL` when you want to assign to a pointer a known sentinel value. You should use `'\0'` to terminate a string. Whereas `NULL` is a pointer (of type `void*`), `'\0'` is a `char`.

27. You should use a CSV file when you want to store small datasets in a (typically read-only) format that can be opened in spreadsheet software. You should use a SQL table when you want to store larger datasets in a format that lends itself to efficient searches (via `SELECT`) and modifications (via `DELETE`, `INSERT`, and `UPDATE`).

Hi, SQL.

28.



29. Suppose that Skroob doesn't yet own any shares of FreeSeas Inc. Because the code's SQL statements are not wrapped in `TRANSACTION`, they will not be executed atomically (i.e., back to back without interruption). If two web browsers, A and B, try talking to the web server simultaneously (and the web server handles those requests with multiple threads), it's possible that A's `SELECT` will execute in one thread, followed immediately by B's `SELECT` in another thread, the result of which is that both threads will see that Skroob doesn't yet own any shares of FreeSeas Inc. If A's thread then proceeds with an `INSERT`, the state of the database will change unbeknownst to B's thread, but B will still try (and fail) to proceed with an `INSERT` (based on the database's previous state) rather than an `UPDATE`. To fix the problem, the code's statements can be wrapped with `BEGIN TRANSACTION` and `COMMIT` or the three statements can be combined into one "upsert" using `ON DUPLICATE KEY`.

Supercookie!

30. Because Verizon is injecting the same header into mobile devices' HTTP requests, that value can be used by web servers receiving that header to track the user across pages, just like a cookie. It simply happens to be a "hand stamp" from Verizon instead of from the servers themselves.

31. Because Verizon's header is injected into an HTTP request after the request leaves a mobile device, it cannot be blocked or deleted by the owner of the mobile device. And so the owner cannot even opt out of being tracked.

32. No. Because HTTPS requests' HTTP headers are encrypted, Verizon cannot inject its header (without knowing the encryption key being used by web browsers and web servers).

Full Circle.

33. The implementation will likely segfault when the queue is of size 1 and its sole `node` is dequeued. Because the code does not check whether `tail` equals `head` (as would be the case when the queue's size is 1), it ultimately frees that sole `node` without setting tail to `NULL`, as is appropriate for a queue of size 0. The addition of a condition, below, fixes the bug.

```
bool dequeue(int* location)
{
    if (location == NULL)
    {
        return false;
    }
    if (tail == NULL)
    {
        return false;
    }
    node* head = tail->next;
    *location = head->datum;
    if (tail != head)
    {
        tail->next = head->next;
    }
    else
    {
        tail = NULL;
    }
    free(head);
    return true;
}
```

34.
```
bool enqueue(int datum)
{
    node* n = malloc(sizeof(node));
    if (n == NULL)
    {
        return false;
    }
    n->datum = datum;
    if (tail == NULL)
    {
       tail = n;
       tail->next = n;
    }
    else
    {
       n->next = tail->next;
       tail->next = n;
       tail = n;
    }
    return true;
}
```