Quiz 1 Review Session

November 15th, 2015

Topics (non-exhaustive)

- stacks
- queues
- linked lists
- hash tables
- trees
- Huffman Coding
- tries
- TCP/IP
- HTTP
- HTML
- CSS
- PHP
- MVC
- HTTP statuses
- DOM
- JavaScript
- jQuery
- Ajax
- security
- AI
- ...

Linked Lists

- benefits of linked lists
  - unlike arrays, size changes dynamically
  - useful for hash tables
- basic operations
  - all Ω(1)
  - insert O(1), delete O(*n*), search O(*n*)
    - assuming not sorted

Stacks

- last-in, first-out (LIFO)
- picture a stack of trays!
- elements are **pushed** on and **popped** off
- if using an array, keep track of both the **size** and **capacity**!

Queues

- first-in, first-out (FIFO)
- picture a line!
- elements are **enqueued** and **dequeued**
- if using an array, keep track of the **size**, **capacity**, and **head**

Hash Table

- implementation of an associative array where the position of each element is decided by a hash function

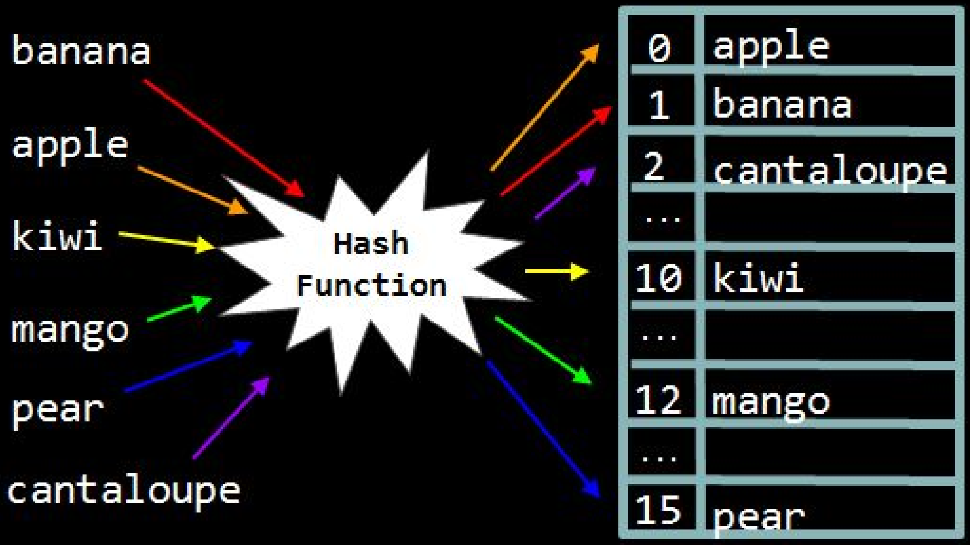


image from study.cs50.net

Hash Function

determines where to insert or lookup a word

```
int hash_function(char* key)
{
    // hash on first letter of string
    int value = toupper(key[0]) - 'A';
    return value % SIZE;
}
```

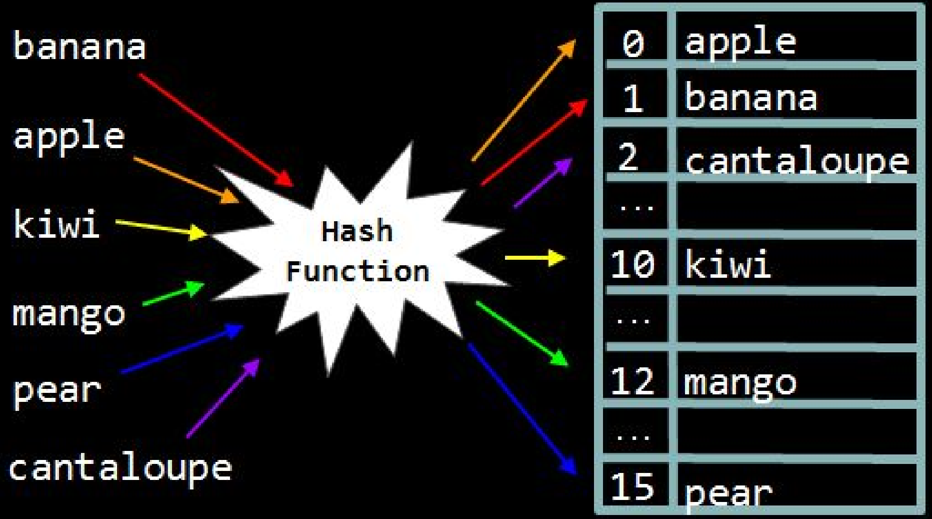


image from study.cs50.net

Collisions

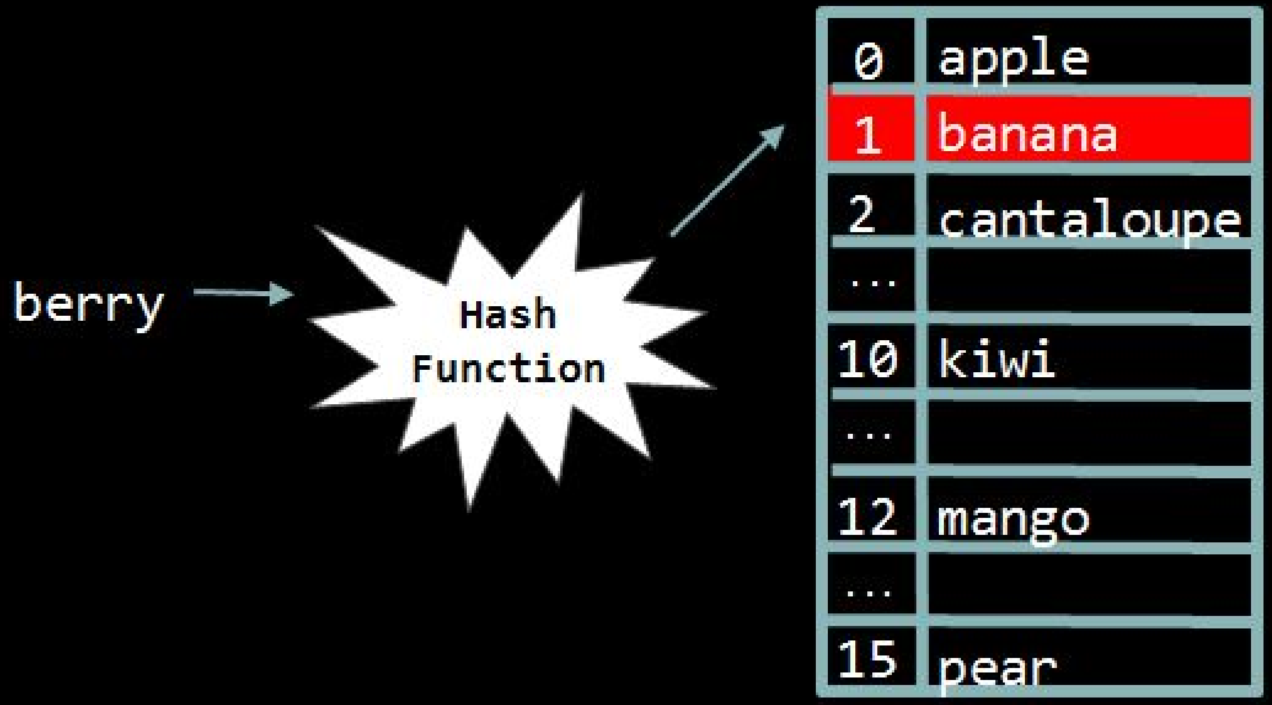


image from study.cs50.net

Linear Probing

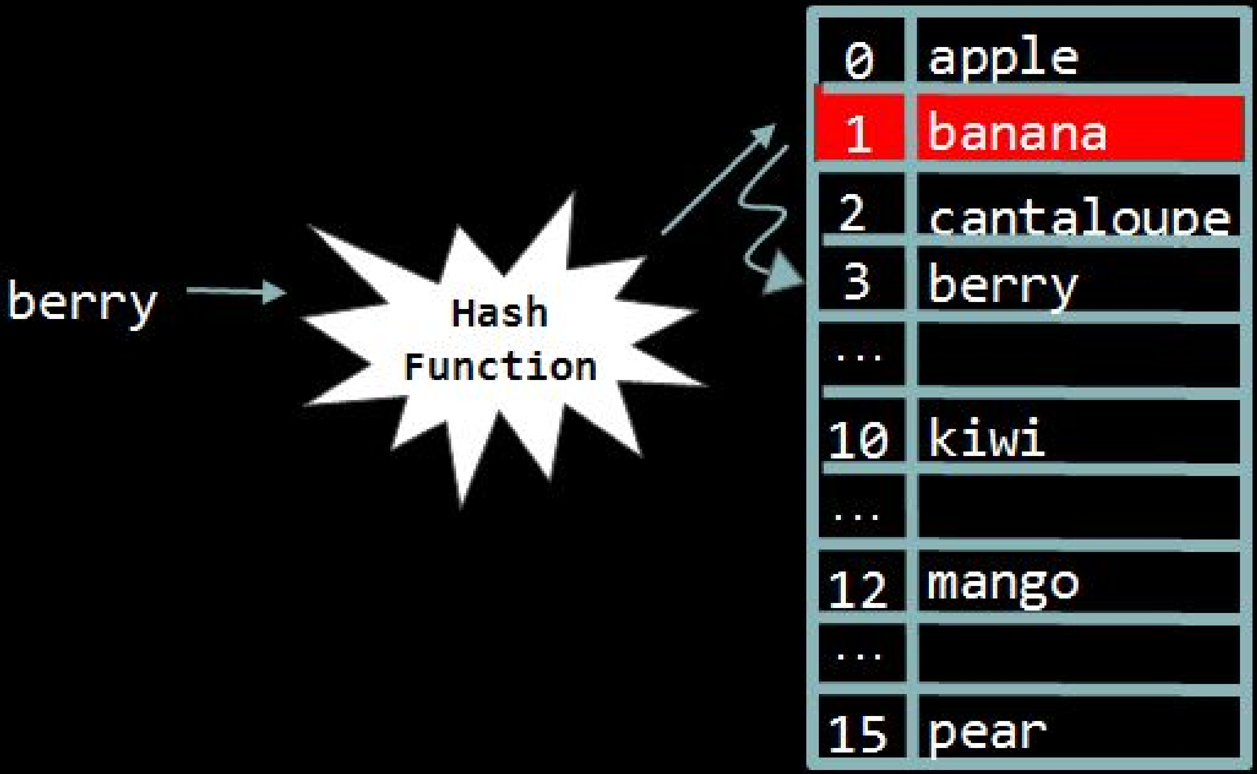


image from study.cs50.net

Separate Chaining

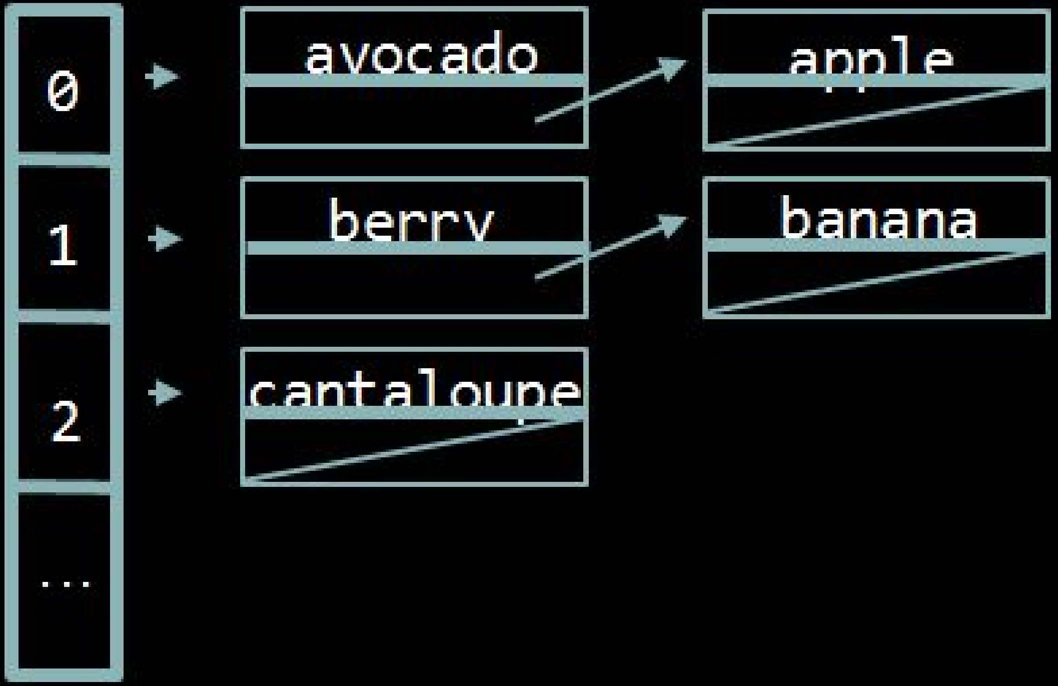


image from study.cs50.net

Trees and Tries

- **tree**: a data structure in which data is organized hierarchically

- **trie**: special kind of tree that behaves like a multi-level hash table

Trees

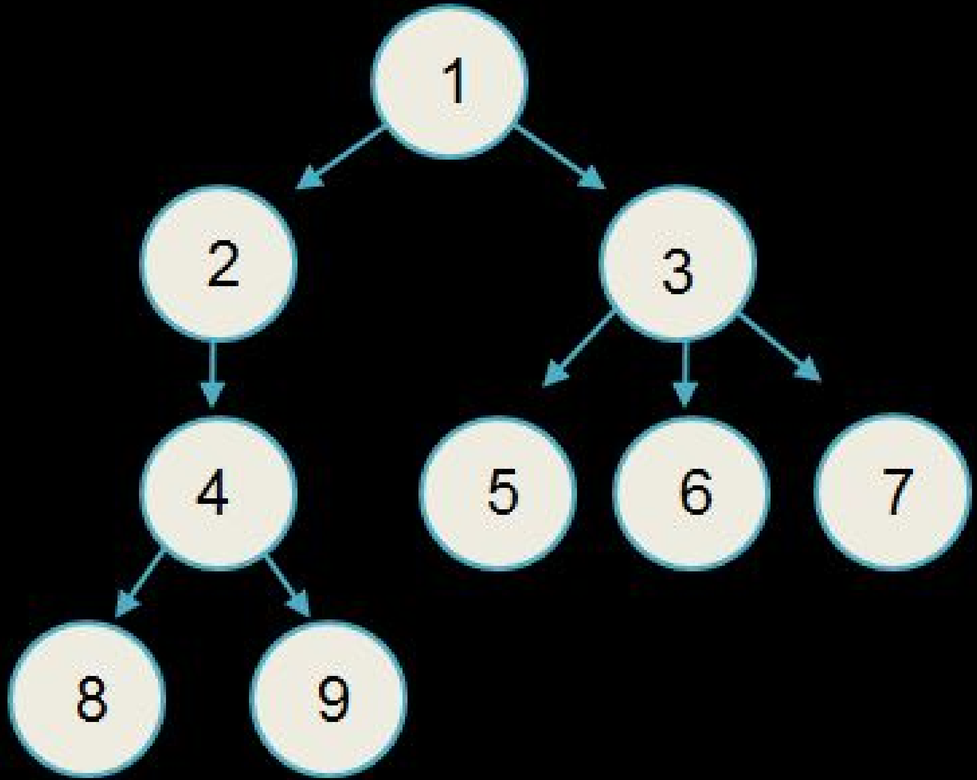


image from study.cs50.net

Binary Trees

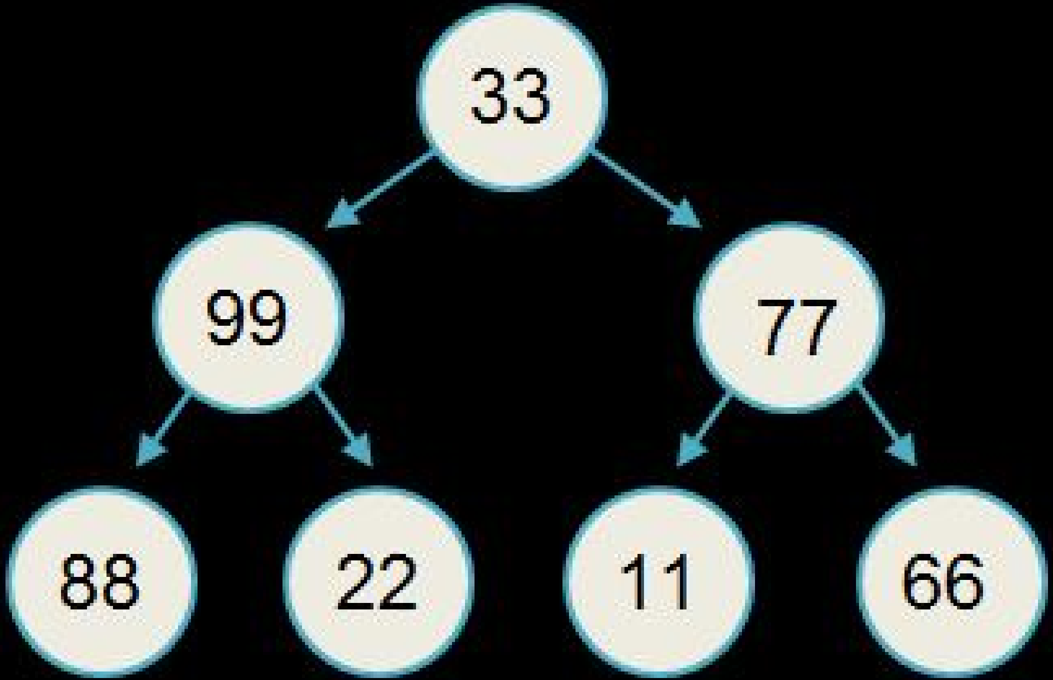


image from study.cs50.net

Binary Search Trees

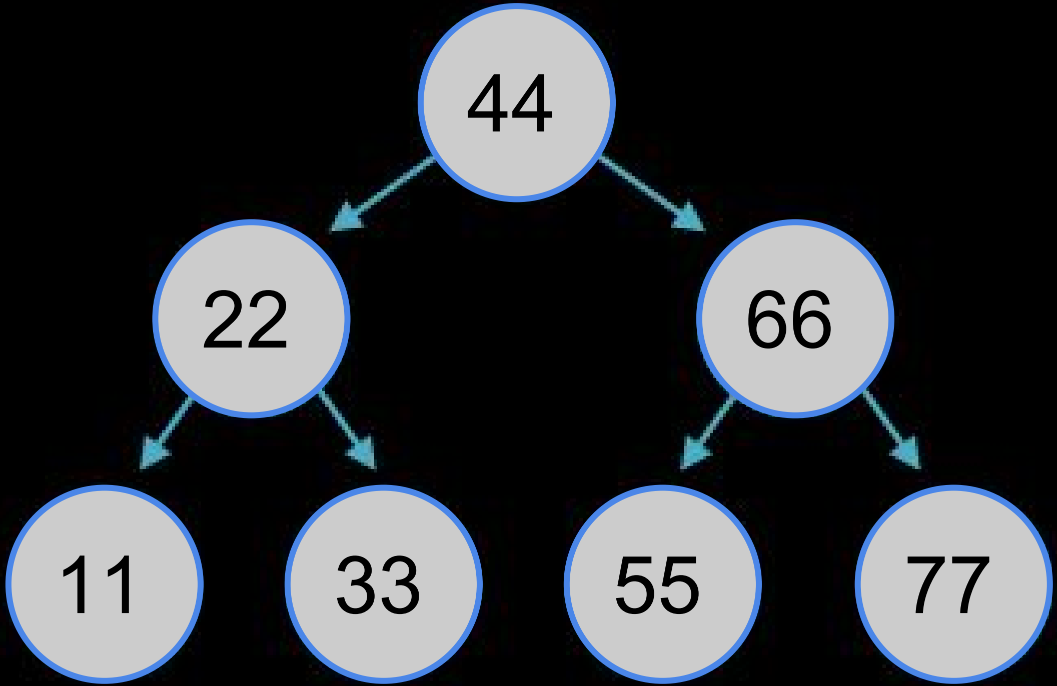


image adapted from study.cs50.net

Tries

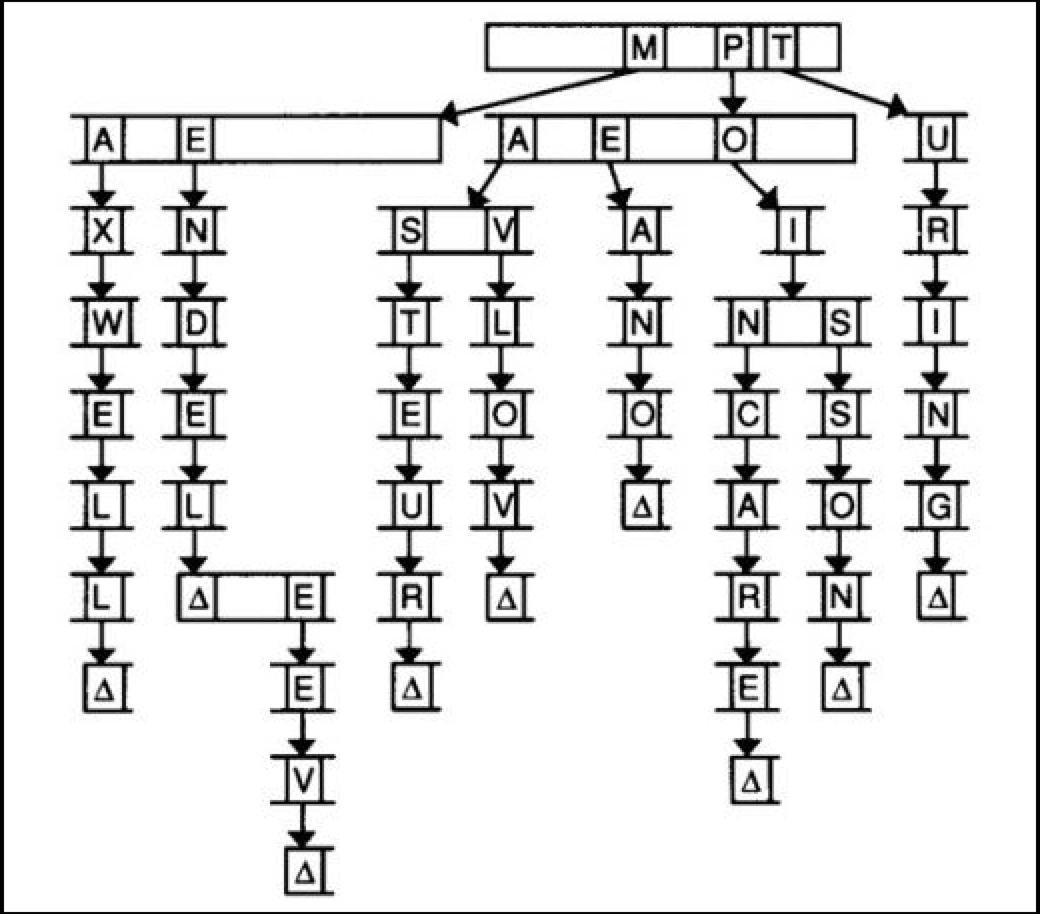


image from study.cs50.net

Tries

```
typedef struct node
{
    // marker for end of word
    bool is_word;

    // array of node*
    struct node* children[27];

}
node;
```

Tries

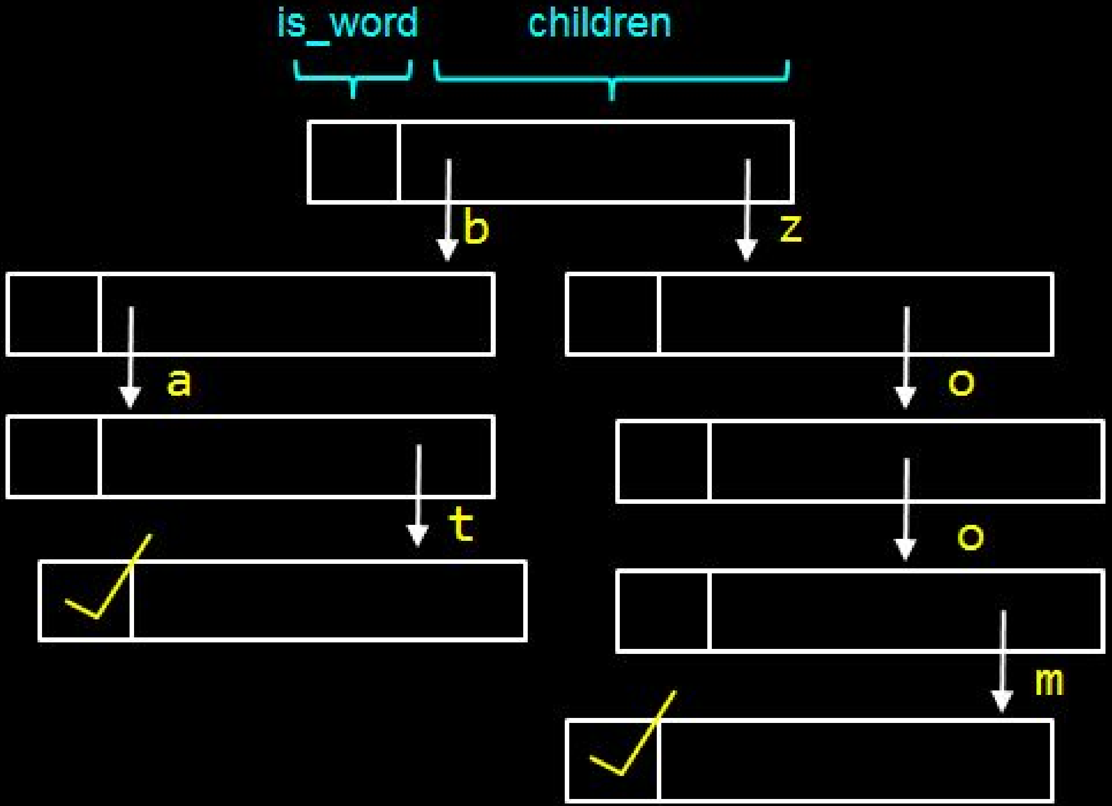


image from study.cs50.net

Tries (vs. Hash Tables)

- **tries** provide constant-time lookup, **but** use large amounts of memory!

Permissions

- chmod ("change mode")
  - Linux command that changes the access permissions of file system objects (i.e., directories, files)
  - to see file permissions: ls -l

Permissions

`d`	`rwx`	`---`	`---`
directory	user	group	world

treat each triad as 3 bits (cumulative value: 0-7)

HTML

- Hypertext Markup Language
- standard markup language used to create web pages

HTML Tags

```html
<!DOCTYPE html>

<html>
    <head>
        <link href="style.css" rel="stylesheet"/>
        <title>CS50</title>
    </head>
    <body>
        <h1 id="title">CS50 Review Session</h1>
        <p class="info">
            Date: Monday, November 15th, 2015
            <br/>
            Time: 2:30 pm - 4:00 pm
        </p>
    </body>
</html>
```

CSS

```css
body
{
    background-color: #000000; /* black */
    color: #ffffff; /* white */
    font-family: "Arial";
}

#title
{
    color: #0000ff; /* blue */
}

.info
{
    color: #ff6666; /* pink */
}
```

CSS

```
tag_name {}

#id {}

.class {}
```

HTML and CSS Best Practices

- close all HTML tags!
- check that your page validates (W3 Validator)
- separate style (CSS) from markup (HTML)

image from xkcd.com

TCP/IP

- Transmission Control Protocol / Internet Protocol
- means of ensuring delivery of data
  - address (e.g., 8.8.8.8)
  - port (e.g., 53)

HTTP

- Hypertext Transfer Protocol
- protocol (i.e., set of conventions) that prescribes how a web browser and web server should communicate

HTTP

request

```
GET / HTTP/1.1
Host: www.google.com
...
```

response

```
HTTP/1.1 200 OK
Content-Type: text/html
...
```

HTTP Statuses

- 200 OK
- 301 Moved Permanently
- 302 Found
- 304 Not Modified
- 400 Bad Request
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error
- 503 Service Unavailable
- ...

PHP

- PHP Hypertext Preprocessor (recursive backronym?!)
- programming language (unlike HTML)

```
<?php
    print("Hello, World!");
?>
```

PHP Basics

- all variable names start with $
    - we don’t specify a variable’s type anymore!
- no main function
- interpreted (as opposed to compiled)
- loosely typed

Arrays

actually an ordered map (associates values to keys)

Syntax:

```
$arr = [
    key1  => value1,
    key2 => value2,
    ...
];
```

or

```
$arr = [1, 2, 3, 4];
```

foreach

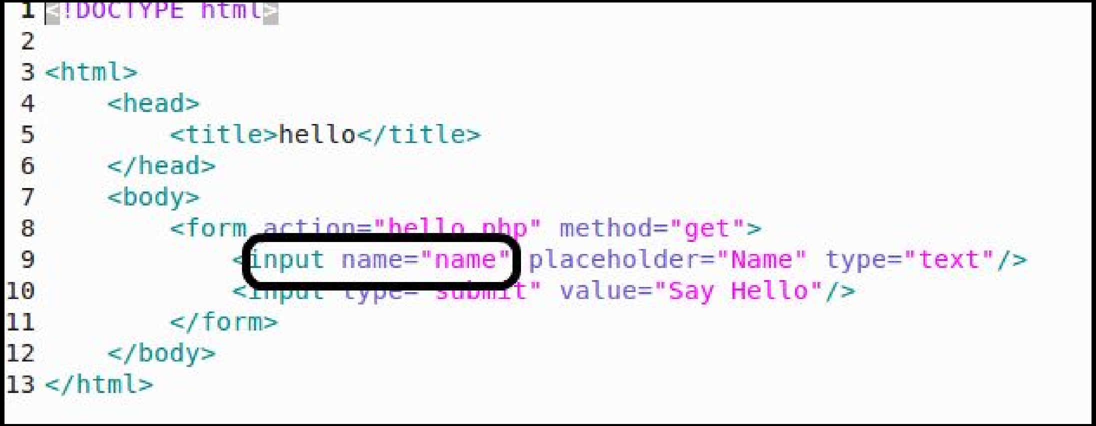
```
Syntax:
foreach ($arr as $value)
{
    // do something with $value
}
```

Example:

```
$arr = ["foo" => "bar", "baz" => "qux"];
foreach ($arr as $key => $value)
{
    // do something with $key and/or $value
}
```

PHP + HTML

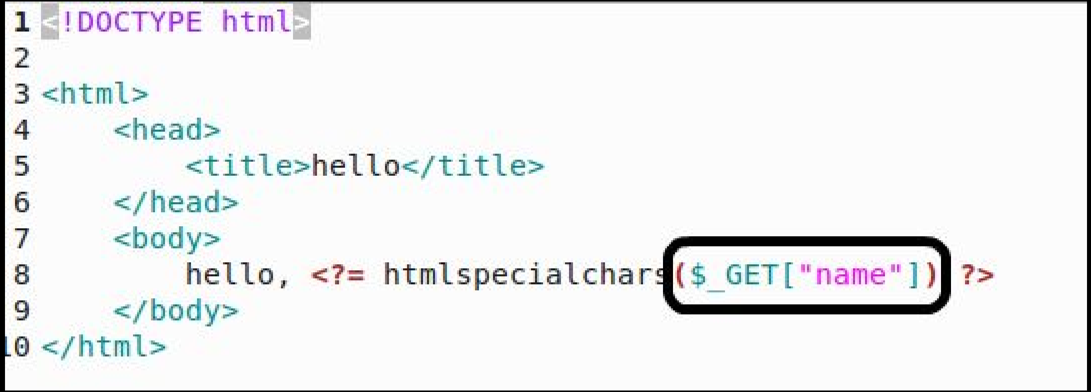
hello.html



```html
<!DOCTYPE html>

<html>
    <head>
        <title>hello</title>
    </head>
    <body>
        <form action="hello.php" method="get">
            <input name="name" placeholder="Name" type="text"/>
            <input type="submit" value="Say Hello"/>
        </form>
    </body>
</html>
```

hello.php



```php
<!DOCTYPE html>

<html>
    <head>
        <title>hello</title>
    </head>
    <body>
        hello, <?= htmlspecialchars($_GET["name"]) ?>
    </body>
</html>
```

GET vs. POST

- two main ways to pass data in an HTTP request
- GET: information is passed via the URL (e.g., YouTube’s URLs)
- POST: passes data in the HTTP message body
  - unlike GET, the data is “hidden” from the user

SQL

- Structured Query Language
- designed for managing data held in a relational database management system
- four common SQL queries:
    - UPDATE
    - INSERT
    - SELECT
    - DELETE

SQL: UPDATE

- update data in a database

```
UPDATE table SET col1 = val1, col2 = val2, …
# update table, changing values in all rows

UPDATE table SET col1 = val1 WHERE house = "Currier"
# update table, changing col1 to val1 at all rows where
the house is "Currier"
```

SQL: INSERT

- insert certain values into a table

```
INSERT INTO table VALUES (val)
# insert into table a new row containing val

INSERT INTO table (col1, col2) VALUES (val1, val2)
# insert a new row into table containing values val1 and
val2 under columns col1 and col2
```

SQL: SELECT

- select data

```
SELECT * FROM table WHERE col = "something"
# select row(s) from table based on col’s value

SELECT * FROM table
# select all columns and all rows from a table
```

SQL: Delete

- delete from table

```
DELETE FROM table WHERE col = "something"
# delete all rows from table where col = "something"
```

MySQL – students example

id	name	year	house
0	Hannah	2015	Cabot House
1	Maria	2018	Cabot House

MySQL – students example

id	name	year	house
0	Hannah	2015	Cabot House
1	Maria	2018	Cabot House
2	Rob	2014	Kirkland House

```
INSERT INTO students (name, year, house)
VALUES ('Rob', 2014, 'Kirkland House');
```

MySQL – students example

id	name	year	house
0	Hannah	2016	Cabot House
1	Maria	2018	Cabot House
2	Rob	2014	Kirkland House

```
SELECT * FROM students;
```

-> returns all fields of all rows

```
SELECT name FROM students WHERE year >= 2016;
```

-> returns Hannah and Maria

```
SELECT id, year FROM students WHERE house = 'Cabot House';
```

-> returns Hannah and Maria

MySQL – students example

id	name	year	house
0	Hannah	2016	Cabot House
1	Maria	2018	Cabot House
~~2~~	~~Rob~~	~~2014~~	~~Kirkland House~~

```
DELETE FROM students WHERE name = 'Rob';
```

MySQL – students example

id	name	year	house
0	~~Hannah~~ Daven	2016	Cabot House
1	~~Maria~~ Daven	2018	Cabot House

```
UPDATE students SET name = 'Daven' WHERE house = 'Cabot House';
```

[A Few] SQL: Data Types

- **CHAR**
  Fixed length string up to 255 characters.
- **VARCHAR**
  Variable length string up to 65,535 characters.
- **INT**
  32-bit integer.
- **FLOAT**
  32-bit floating-point value.

  ....

PHP + SQL

```
$rows = CS50::query("SELECT * FROM history WHERE
user_id = ?", $_SESSION["id"]);
```

CS50’s `query` function protects against SQL injection.

MVC

- design paradigm
- way of organizing and thinking about code

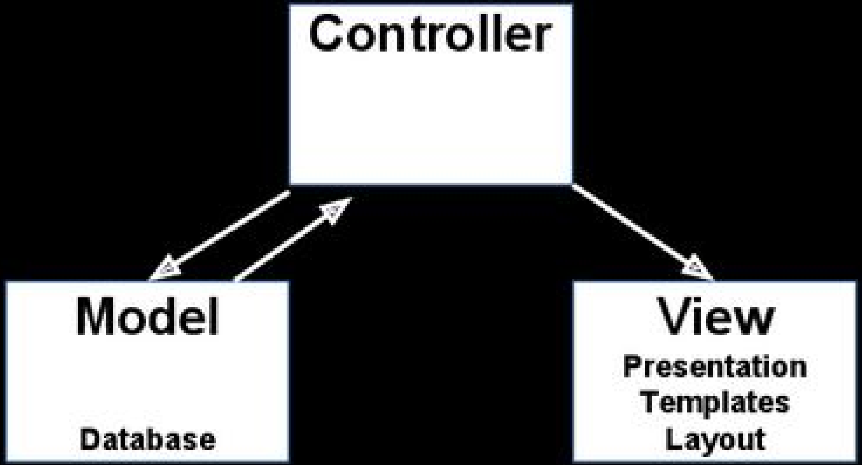


image from study.cs50.net

MVC

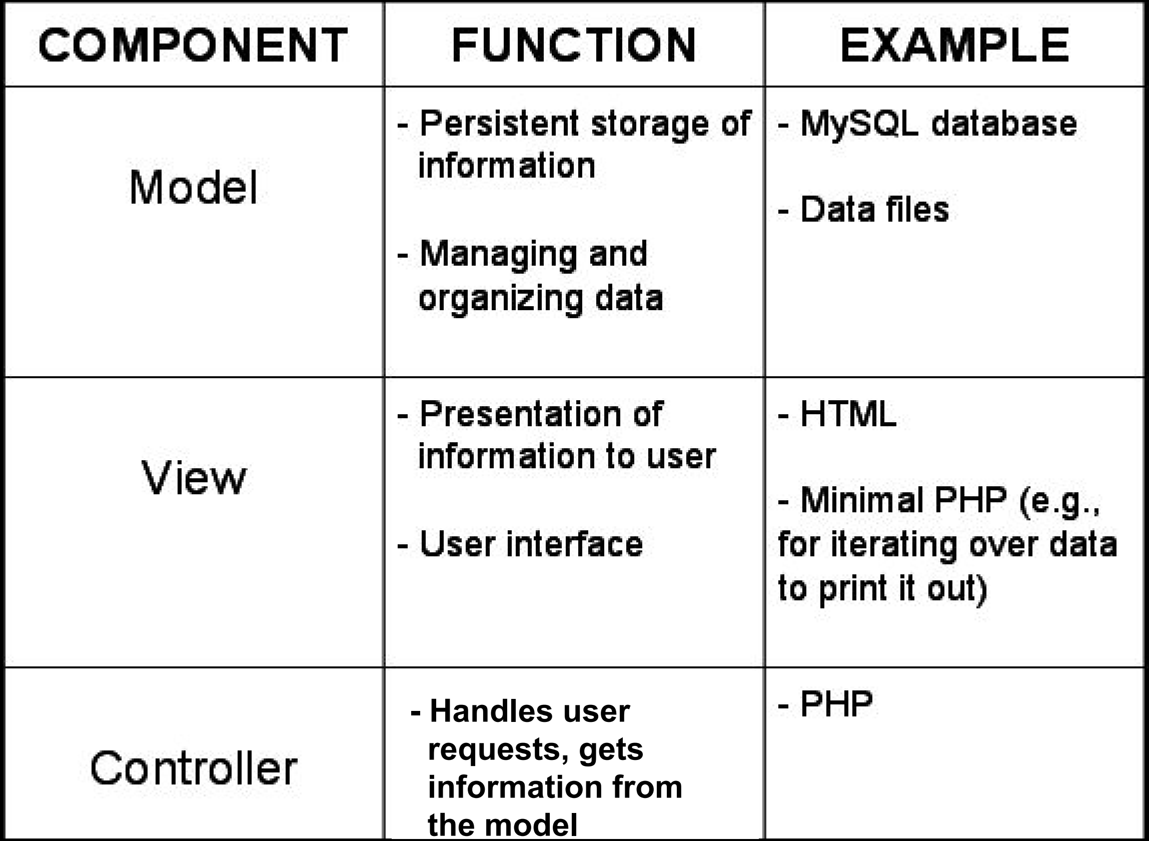
HTTP request is sent to a web server→

controller interprets the user’s request and validates user input→

(optional) controller communicates with a **model** (which might be a database or other functionality) →

controller passes information on to the view

MVC



COMPONENT	FUNCTION	EXAMPLE
Model	- Persistent storage of information - Managing and organizing data	- MySQL database - Data files
View	- Presentation of information to user - User interface	- HTML - Minimal PHP (e.g., for iterating over data to print it out)
Controller	- Handles user requests, gets information from the model	- PHP

image from study.cs50.net

DOM

- HTML documents are organized into a hierarchical tree structure
- DOM: Document-Object Model
  - if we have access to an object representation of the document, then we can manipulate the document like we manipulate objects

DOM

```
<!DOCTYPE html>

<html>
   <head>
      <title>hello, world</title>
   </head>
   <body>
      hello, world
   </body>
</html>
```

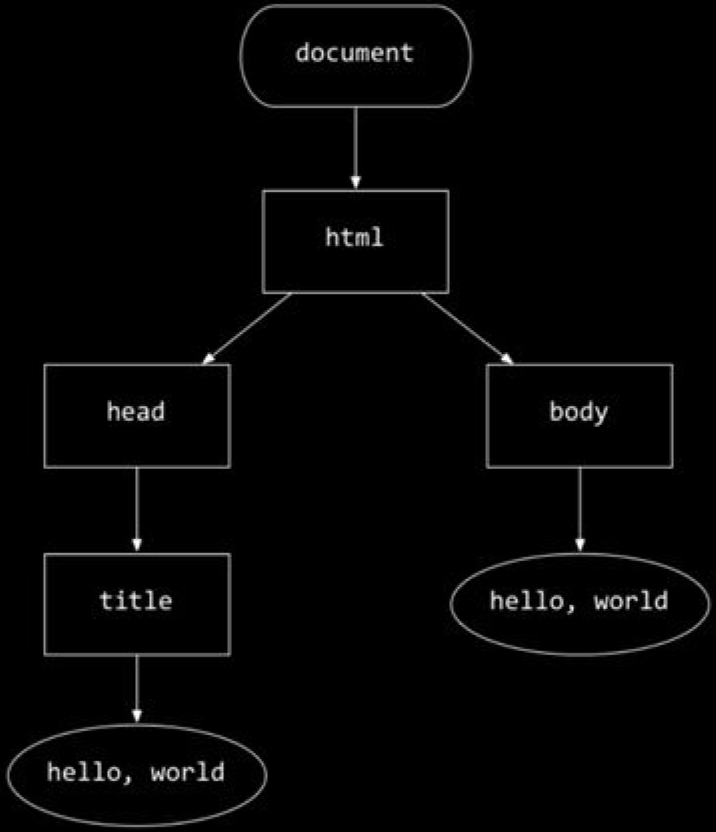


image from study.cs50.net

JavaScript

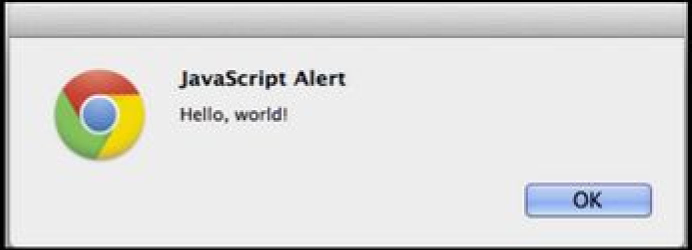
- loosely typed
- interpreted language (no need to compile)
- used to manipulate the content, appearance, and behavior of a web page
- allows users to communicate asynchronously with the browser (via Ajax)
- client-side
  - no need to interact with a server → faster

Hello World

index.html

```
<!DOCTYPE html>


<html>
    <head>
        <script src="hello.js"></script>
        <title>Hello, world!</title>
    </head>
    <body>
        Body HTML here
    </body>
</html>
```



hello.js

```
alert("Hello, world!");
```

Variable Declarations

- take the form `var` ***`name`*** `=` ***`value`***`;`
- no type is specified

```
C
int i = 50;
```

```
PHP
$i = 50;
```

```
JavaScript
var i = 50;
```

Loops

```
for(/* init */; /* condition*/; /* update */)
{}

while(/* condition */)
{}

do
{}
while(/* condition */);
```

Function Declarations

```
function sum(x, y)
{
    return x + y;
}

/* or */

var sum = function(x, y)
{
    return x + y;
}
```

anonymous function: functions without names

functions are treated like values

Arrays

```
var arr = ["Arrays", "in", "JavaScript"];
```

- grow dynamically
- access elements with square brackets
- `arr.length` gives the length of array `arr`

Objects

- conceptually similar to structs in C and associative arrays in PHP
- JSON: JavaScript Object Notation

Objects (JSON)

```
var CS50 = {
  "course": "CS50",
  "instructor": "David J. Malan '99",
  "tfs": ["Maria", "Hannah", "Daven"],
  "psets": 9,
  "recorded": true
};
```

Objects

- can access / set fields in two ways:
  - `objectName.fieldName`
  - `objectName["fieldName"]`

```html
<!DOCTYPE html>

<html>
    <head>
        <title>Search Button Demo</title>
    </head>
    <body>
        <button id="search_button">Push me!</button>
    </body>
</html>
```

Events

```
window.onload = function() {
   var searchButton =
      document.getElementById("search_button");

   searchButton.onclick = function() {
      alert("You clicked the search button");
   };
}
```

jQuery

- “fast, small, and feature-rich JavaScript library”
- easier
    - HTML document traversal and manipulation
    - event handling
    - animation
    - Ajax

quote from https://jquery.com/

jQuery

- basic syntax
    - `$(selector).action()`

jQuery Example

```
$(window).load(function() {
   $("#search_button").click(function() {
      alert("You clicked the search button");
   });
});
```

Compare to “Events” slide - jQuery is much more concise!

Some Useful jQuery

- `$(document).ready(someFunction)`
  - call `someFunction` when DOM has loaded
- `$("#someID")`
  - select the DOM element with ID `someID`
- `.submit(someFunction)`
  - on `<form>` submission, call `someFunction`
- `.val()`
  - get value submitted through a form
- `.html()`
  - access HTML

Ajax

- stands for “Asynchronous JavaScript and XML”
  - (JSON is usually used in place of XML)
- goal: load data in the background and display it when it’s ready (without reloading the whole page)
  - allows us to send additional GET or POST requests

Ajax + jQuery

```
$.getJSON(URL, parameters)
  .done(function(data, textStatus, jqXHR) {
      // if successful, do something
  })
  .fail(function(jqXHR, textStatus, errorThrown) {
      // else handle error
  });
```

Security

Something bad that should look familiar:

```
#include <string.h>

void foo(char* bar)
{
    char c[12];
    memcpy(c, bar, strlen(bar));
}

int main(int argc, char* argv[])
{
    foo(argv[1]);
}
```

The Fix

Always check bounds of arrays!

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

Web Security

True or False

- Using a single password is a good idea

- Padlock icons ensure security

Web Security

True or False

- Using a single password is a good idea
- Padlock icons ensure security

FALSE

Other Types of Attacks

- Session hijacking
- SQL Injection Attack
- Manipulating header data

Questions?