CSE 331: Software Design & Engineering Fall 2024

Homework 1

Due: Saturday, October 5th, 11pm

Ensure you've completed the course Software Set-up, then check out the starter code for this assignment:

```
git clone https://gitlab.cs.washington.edu/cse331-24au-materials/hw1-parsing.git
```

Navigate to the `hw1-parsing` directory and run `npm install --no-audit`.

In Tasks 1–3, you are asked to *think through* a few programming problems and type in your solutions. In Task 4, on the next page, we will explain how to test out your solutions to see if they work. For reasons that will be clear then, you must wait until Task 4 to test them out.

Task 1 – A Pain in the Parse [25 pts]

Implement the function `parseBuildings` in `src/routes.js`. This function is passed in a list of lines of text (from the file `data/campus_buildings.csv`), each of which is a string of the form:

```
MOR,Moore Hall,2317.1749,1859.502
```

with commas separating the four provided pieces of information about the building.

Parse each line into a record containing fields called `shortName`, `longName`, `x`, and `y`, corresponding to the four parts above. The two names should be strings, while `x` and `y` should be numbers. Each of these records should be added to the `buildings` array.

See the final page for descriptions of functions that may be helpful. For all tasks, you should not use functions or methods other those listed there.

Task 2 – It's Name Time [25 pts]

Implement the function `findByName` in `routes.js`. This function should find buildings (from Task 1) whose long names contain the text provided in the query parameter called `text`. The "`query`" field of the request object, `req`, is a record containing each of the query parameters as a field.

Your code should send back (via the response object, `res`) a record containing a single field called "`results`" that contains an array of the buildings found. Each building should be a record as above.

Your solution must follow these additional rules:

- Return at most 3 results. If more than 3 buildings contain the provided text in their long names, then return the *first* three buildings from the original list.
- Your text matching should be case-*insensitive*. I.e., searching for "engineering" should find "Paul G. Allen Center for Computer Science & Engineering" even though the case is different.

Task 3 – It Takes All Finds [25 pts]

Implement the function `closest` in `routes.js`. This function should find the **three** closest buildings to the (x, y) position provided in the query parameters `x` and `y`.

Your code should send back the results in the same format as in Task 2.

Task 4 – Every Log Has Its Day [25 pts]

In this task, you will try out your solutions to Tasks 1–3. Finding the bugs and analyzing their causes is the most important part of this assignment. For that reason, we require you to carefully **track** and **document** your time spent debugging. For each bug, you must also provide the following information:

- What (incorrect) behavior you saw that told you there was a bug.
- How many **minutes** it took you to find the bug.
- What kind of experiments you performed to try to locate the bug (checking the network tab, scanning for typos, `Console.logs`, etc.)
- Where an error first occurs in the code that leads to the incorrect behavior.
- What the defect was that caused that bug (if you ever found it).
- Whether a (Java) **type checker** would have spotted the bug.

  Note that the type checker not only finds cases where data of the wrong type is used (e.g., a number where a string is expected) but also when the names of fields or functions are misspelled.

We have provided a debugging log website for you to use to record this information. Some tips and rules for using the site:

- Don't forget to save! Hit the "Save Log" button to avoid losing your entries.
- Keep your log on standby. Having the log open before you run your app and as you test it will help you remember to write everything down. Even a simple typo is a bug, and you should log it!
- Keep your logging brief. We want to understand your process and diagnosis, but we prefer to read short descriptions over paragraphs.

Since debugging is the most important part of this assignment, you can still get full credit for submitting an incorrect solution provided that you have made a reasonable attempt at Tasks 1-3 and fully document **4 hours** (240 minutes) of debugging. You are forbidden from spending significantly more than 4 hours debugging (if you go over by a few minutes, that's okay). Note that the 4 hour time limit applies only to this debugging task, the time to complete Tasks 1-3 should not be counted.

If you think your solution is correct, and you have not yet reached 4 hours of debugging, be careful! Have someone else try out your app and see if they catch any bugs.

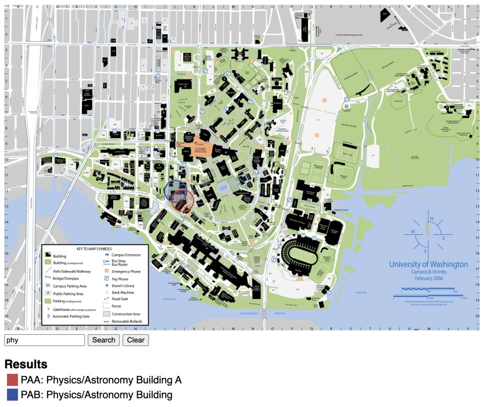
To try your solution, open the file `public/index.html` and add the following line just before `</body>`:

```
<script src="ui.js"></script>
```

Next, in the terminal, run the command `npm run start`. Finally, open `http://localhost:8080` in Chrome. You should see a campus map. Typing in the input box and pressing "Search" should perform a "find by name" search. Clicking anywhere in the map should perform a "closest" search. In both cases, the results you returned should be shown both on the map and in a list below the input box. If they are not, then there is a bug. Good hunting!

The following is a set of examples of app inputs and the expected outputs to try with your app. You should definitely experiment with inputs beyond these.

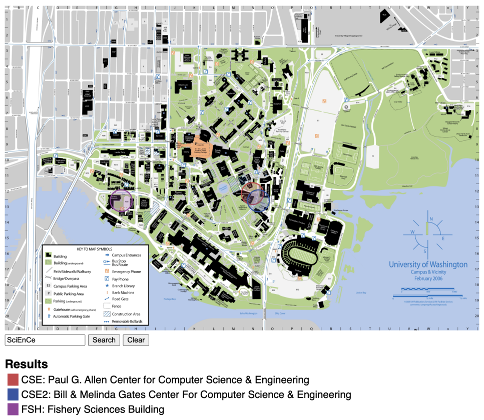
- Input "phy" in the input box and "Search", and the results and map should appear as follows:



Results

🟥 PAA: Physics/Astronomy Building A
🟦 PAB: Physics/Astronomy Building

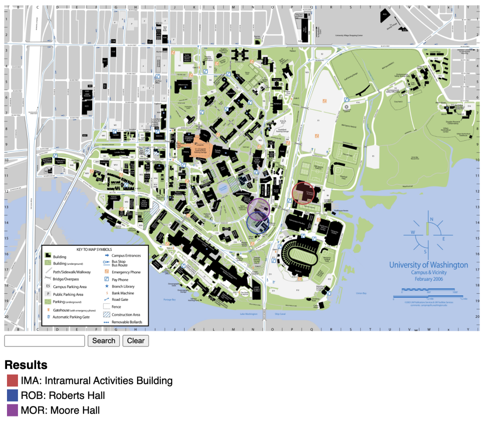
- Input "SciEnCe" in the input box and "Search", and the results and map should appear as follows:



Results

🟥 CSE: Paul G. Allen Center for Computer Science & Engineering
🟦 CSE2: Bill & Melinda Gates Center For Computer Science & Engineering
🟪 FSH: Fishery Sciences Building

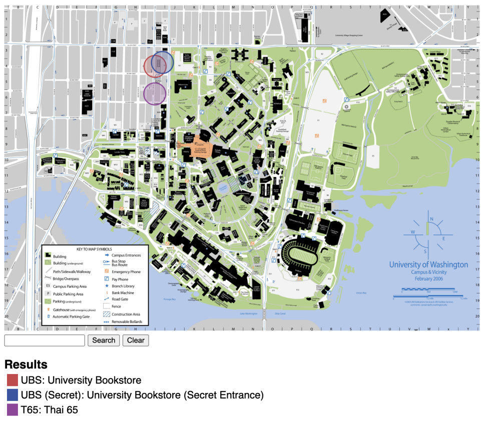
- Click the Women's Softball Field on the map, and the results and map should appear as follows:



Results

🟥 IMA: Intramural Activities Building
🟦 ROB: Roberts Hall
🟪 MOR: Moore Hall

- Click the University Bookstore on the map, and the results and map should appear as follows:



Results

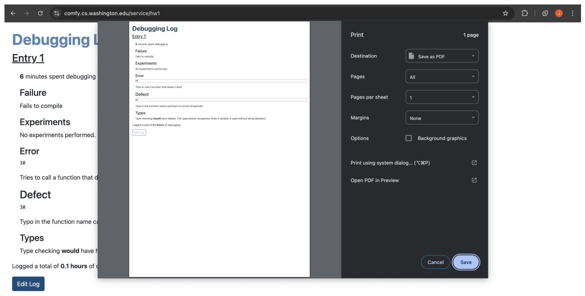
🟥 UBS: University Bookstore
🟦 UBS (Secret): University Bookstore (Secret Entrance)
🟪 T65: Thai 65

Note that the placement of the mouse for a click may cause the *order* of "closest" search results to differ, but provided the click is within the vicinity described in these examples, the *contents* should be the same. For "find by name" search results, both the order and contents should be identical.

Submission

After completing Task 4, download your debugging log:

1. Select "View Log" to open all of your log entries
2. Select File > Print or ctrl+P/Command+P to open the print dialog
3. Set the print destination as "Save as PDF" and "Save"
4. Rename the downloaded PDF as "**DebuggingLog.pdf**"



Debugging Log
Entry 1

6 minutes spent debugging

Failure
Fails to compile

Experiments
No experiments performed.

Error
30
Tries to call a function that doesn't exist

Defect
30
Typo in the function name causing it to not be recognized

Types
Type checking **would** have helped. The typechecker recognizes when a variable is used without being declared.

Logged a total of **0.1 hours** of debugging.

Print — 1 page
Destination: Save as PDF
Pages: All
Pages per sheet: 1
Margins: None
Options: ☐ Background graphics
Print using system dialog... (⌥⌘P)
Open PDF in Preview

Then, submit the downloaded log and your code to the "HW1" assignment on Gradescope. The assignment expects exactly the following files:

DebuggingLog.pdf `routes.js`

After you submit your work, an autograder will run to verify you have submitted the correct files; wait for it to finish and check that the submission looks correct. Don't forget to make sure that the submitted files are up-to-date with all implementation and debugging you completed!

Useful Library Functions & Methods

This page contains library functions (and methods) you can use in your solution. You will also need to use the functions shown in lecture and section for sending server responses.

To keep grading simpler, please restrict yourself to only those listed here. Specifically note that the functions from the `Math` library are **not** allowed, but mathematical operations (+ * - / **) are.

You can read more about these allowed functions (as well as many other library functions) by searching in the Mozilla Developer Network documentation on the web.

Global Functions

`parseFloat` Takes a string as an argument and returns the floating point value it describes. For example, a call to `parseFloat("3.14")` would return the floating point value 3.14. This returns `NaN` (not a number) if the string does not represent any valid number.

`isNaN` Returns true if the value passed is `NaN`, otherwise returns false.

Fields and Methods of `Array`

`length` The length field stores the length of the array, e.g., `[1, 2, 3].length` is 3.

`push` Adds the value passed in as an argument to the end of the array. E.g., if `A = [1, 2]`, then after calling `A.push(3)`, the value of `A` would be `[1, 2, 3]`.

`pop` Removes the last element from the end of the array and returns it. E.g., if `A = [1, 2]`, then after calling `A.pop()`, 2 would be returned, and the value of `A` would be `[1]`.

`slice` Called with no arguments, this returns a (shallow) copy of the array. Returns a copy of a portion of the array when called with arguments selecting the `start` (and optionally `end`) index of the portion.

`sort` Reorders the elements of the array into ascending order. For primitive types, this orders them according to the built-in "<=" operator for that type. For other types, you must pass in an argument, which is a function taking two arguments `(a, b)` and returning a negative value if $a < b$, a positive value if $a > b$, and zero if $a = b$. (For numbers, just `(a, b) => a - b` works!)

Methods of `String`

`indexOf` Takes a character (i.e., a length-1 string) as an argument and returns the first index where that character appears in the string or -1 if the character is nowhere in the string.

`includes` Returns true if the string contains the argument (another string) somewhere within it. E.g., `"a quick brown fox".includes("bro")` returns true.

`split` Takes a character (i.e., a length-1 string) as an argument and returns an array containing each of the contiguous pieces of the string without that character. For example, `"the quick brown fox".split(" ")` returns `["the", "quick", "brown", "fox"]`.

`toLowerCase` Returns the same string but with all upper-case characters replaced by their lower-case versions (all other characters are unchanged).