CSE 331
Software Design & Implementation

Autumn 2023

Section 7 – Imperative Programming II

Administrivia

- HW7 released later today
  - Due Wednesday (11/16) @ 11:00pm

Question 2a

```
let i: number = ____________; // fill this in
```
{{ Inv: there is no index $0 \leq j \leq i$ such that words$[j] = w$ }}
```
while (_________________________) {        // fill this in

                                           // fill this in

    i = i + 1;
}
```
{{ there is no index $0 \leq j \leq$ words.length $- 1$ such that words$[j] = w$ }}
```
return -1;
```

Question 2b

```
let i: number = ____________; // fill this in
```
{{ Inv: there is no index $0 \leq j \leq i$ such that words$[j] = w$ }}
```
while (____________________________) {        // fill this in
    i = i + 1;

                                              // fill this in

}
```
{{ there is no index $0 \leq j \leq$ words.length $- 1$ such that words$[j] = w$ }}
```
return -1;
```

Specifying Functions – Review

- By default, no parameters are mutated
  - **Must *explicitly* say that mutation is possible** (default is not)
  - `@modifies` lists anything that may be changed
    - but doesn't promise to modify it (may not be necessary)
  - `@effects` promises about result after mutation
    - Like `@returns` but for mutated values, not return values.
- **Ex:**

```
/**
* Reorder A so that the numbers are in increasing order
* @param A array of numbers to be sorted
* @modifies A
* @effects A contains the same numbers but now in
*          increasing order
*/
quickSort(A: number[]): void { ... }
```

Arrays – Review

- Allows easy access to both ends: `A[0]` and `A[n-1]`. `n = A.length`
  - Bottom-up loops are now easy

- However, when we write "`A[j]`", we must also check $0 \leq j < n$
  - New possibilities for bugs
  - TypeScript will not help us with this

Arrays – Review

- **Array Concatenation** – define operation "⧺" as array concatenation (makes clear arguments are arrays, not numbers)

- **Following properties hold for any arrays, A, B, C:**
  - A ⧺ [] = A = [] ⧺ A ("identity")
  - A ⧺ (B ⧺ C) = (A ⧺ B) ⧺ C ("associativity")

Mutating Arrays – Review

- **Assigning to array elements changes known state:**

  {{ A[j-1] < A[j] for any 1 ≤ j ≤ 5 }}


  ```
  A[0] = 100;
  ```

  {{ A[0] = 100 and A[j-1] < A[j] for any 2 ≤ j ≤ 5 }}

- **Can add to the end of an array:**

  ```
  A.push(100);
  ```

  {{ A = A_0 ⧺ [100] }}

  A has one more element than before

- **Can remove from the end of an array:**

  ```
  A.pop();
  ```

  {{ A = A_0[0…n-2] }}

  A has one fewer element than before

Loop Invariants with Arrays – Review

- **Heuristic for loop invariants:** weaken the postcondition
  - Inv is just a **weakening** of the Post
  - Inv is simple an assertion that allows the post condition as a special case

- Ex:

{{ **Inv**: $s = sum(A[0 .. j - 1])$ and $j \leq A.length$ }} **sum of array**

{{ **Post**: $s = sum(A[0 .. n - 1])$ }}

{{ **Inv**: $contains(A[0 .. j - 1], x) = F$ }} **search an array**

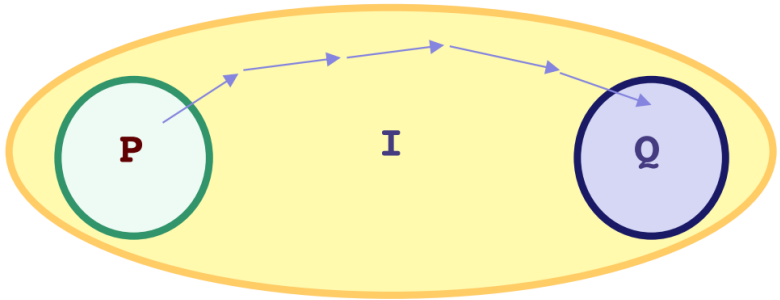
{{ **Post**: $contains(A[0 .. n - 1], x) = F$ }}

we will learn extra facts
when the loop terminates

Loop Invariants with Arrays – Review

- **Algorithm Idea:**
  - How will you get from start (P) to stop state (Q)?
    - e.g. `j=j+1`
  - What does partial progress looks like?
    - e.g. loop invariant

```
{{ P }}
{{ Inv: I }}
while (cond) {
   S
}
{{ Q }}
```



Question 3a

$$
\begin{aligned}
&\textbf{func}\ \text{del-spaces}([]) && := [] \\
&\qquad \text{del-spaces}(L +\!\!+ [\text{“ ”}]) && := \text{del-spaces}(L) \\
&\qquad \text{del-spaces}(L +\!\!+ [c]) && := \text{del-spaces}(L) +\!\!+ [c] \quad \text{if } c \neq \text{“”}
\end{aligned}
$$

Write a specification for the below function signature which mutates the array passed in according to math def del-spaces:

```
/**
*
*
*
*
*/

const delSpaces = (str: string[]): void => {..};
```

Question 3b

Write a loop that overwrites str with del-spaces. Given:

```
let m: number = ____        // Local var at top of loop
```

{{ Post: **str[$0 .. m - 1$] = del-spaces(str$_0$) }}**

How can we weaken the post condition into something easy to make true at the top of the loop (Inv)?

What exit condition will make the loop hold at the end?

Question 3c

```
let m: number = ___;  // From b)

{{ Inv: _______________}}   // From b)
while (_______________) {  // From b)




}
{{ Post: str[0…m-1] = del-spaces(str₀[0…j-1]) }}
```

Question 3

Finish up function by removing any additional elements left over.

- All “ ” elements were replaced by the next non-space element, so now there are some left over after others shift down

$\{\{$ Inv: str$[0 \,..\, m-1] =$ del-spaces(str$_0$) and $m \leq$ str.length $\}\}$

```
while (m !== str.length) {
    str.pop();
}
```

$\{\{$ str $=$ del-spaces(str$_0$) $\}\}$

- When the loop terminates we know `m == str.length, so`:

del-spaces(str$_0$) $=$ str$[0 \,..\, m-1] =$ str$[0 \,..\,$ str.length $- 1] =$ str

which matches the post condition!

Client-Side vs Server-Side – Review

- **Client-Side JavaScript**
  - Code so far has run inside the browser
    - webpack-dev-server handles HTTP requests
    - Sends back our code to the browser
  - In the browser, executes code of index.tsx

- **Server-Side JavaScript**
  - Can run code in the server as well
    - Returns different data for each request (HTML, JSON, etc.)
  - Can have code in *both* browser and server

Client-Side vs Server-Side – Review

Client-Side

HTTP GET

index.html
index.tsx etc.

webpack-dev-server

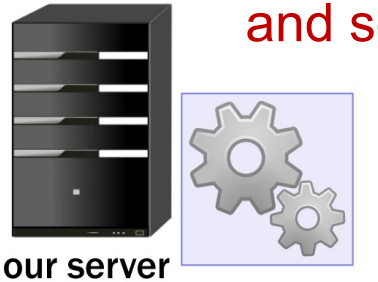
Code only on browser

VS

Server-Side

HTTP GET

response data

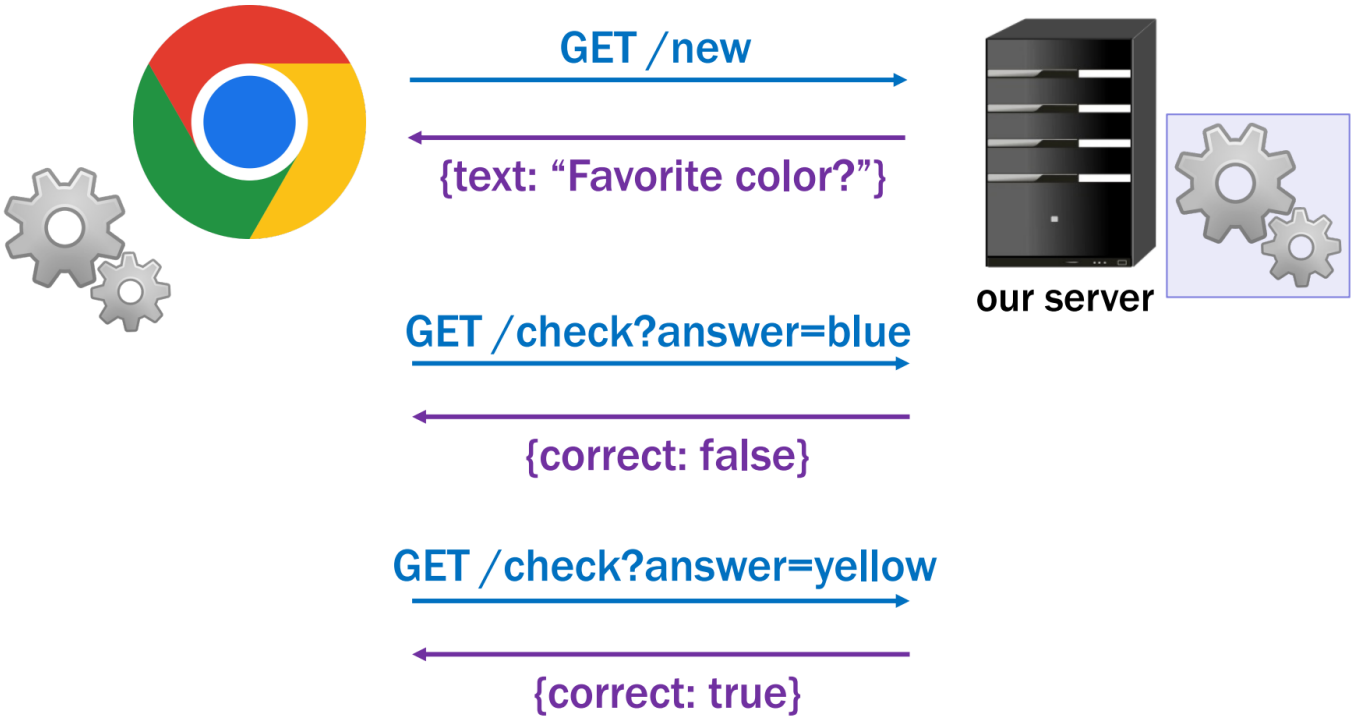


our server

Code on browser and server

Custom Server

- In a custom server, we can define useful routes
- Interacting with app will result in a series of requests and responses



Custom Server

```
function F(req: Request, res: Response): void {
    const name: string | undefined = req.query.name;
    // Error case: send error message, set error status
    if (name === undefined) {
        res.status(400).send(“Missing ‘name’”);
        return;
    }
    // Send proper response, default status (200) used
    // Sent as parseable JSON object, could be text/HTML
    res.send({message: `Hi, ${name}`});
}
// localhost:8080/foo will call F()
const app = express();
app.get(“/foo”, F);
app.listen(8080);
```

Question 4

- Writing some code that runs in the **server** rather than in the browser (the ``client'').
- Inputs are passed in **query parameters** as usual
- The response will be a JavaScript object serialized into text format, **JSON**, and sent back to the browser.
  - Allows for convenient parsing

- Given an uninteresting trivia app, going to make it more interesting!