CSE 331: Software Design & Implementation

Section 1

In this section, we will build a very simple web application.

Start by performing the setup steps described on page 3. Then, complete the following parts.

1. Separation of Search and State

In this problem, we will change the application to print out the n-th Fibonacci number, where n is a parameter supplied by the user as a query parameter.

(a) Use the built-in `URLSearchParams` class to parse the query string as follows:

```
const params: URLSearchParams = new URLSearchParams(window.location.search);
```

Then, call the get method on `params` to retrieve the value of the query parameter `"n"` if present. Store the result of the method call in a new variable `nVal`.

Change the code so that, if `"n"` was not provided, it prints a message saying that "The parameter 'n' was not provided.". Otherwise, print out the same message from before.

(b) If `"n"` was provided, parse it into a (base-10) integer value using the built-in parseInt function. Store the result in a new variable `n`.

Change the code so that, if `nVal` was not a valid integer (in base 10) or is not non-negative, then it prints a message saying that "The parameter 'n' was not a non-negative integer." (Hint: the built-in method isNaN may be useful here.)

(c) Change the code so that, if `nVal` did parse into a non-negative integer, then it prints a message saying that "Fibonacci number X is Y.", where X is `n` and Y is `fib(n)`. Use a string literal with substitution rather than concatenation to build the string that is printed.

Note that you will need to import `fib` to be able to call it here.

2. Render Loving Care

In this problem, we will change the messages from printing in the console to showing up in the page as HTML.

(a) Use the built-in `getElementById` method on the `document` object to retrieve the HTML tag (a.k.a. "element") with the ID attribute equal to "main" as follows:

```
const elem: HTMLElement | null = document.getElementById('main');
```

If you look in `index.html`, you will see that there is such an element, so this call should never return `null`. That said, the type checker cannot tell that, so you will see an error message in VS Code.

Change the code to print an error message if this element was not found. If it was found, then it should execute the code from before (looking for a parameter "n" and printing that Fibonacci number).

(b) In order to put HTML into the page, we need to tell React that it will be in charge of placing HTML into this element. You can do so as follows:

```
const root: Root = createRoot(elem);
```

Note that you will need to import `createRoot` and `Root` from `'react-dom/client'` to use them here. You can import multiple names from one file by putting both inside `{..}`, separated by commas.

(c) In order to use HTML inside of JS / TS, add the following special import at the top of the file:

```
import React from 'react';
```

Now, change each of the messages printed in problem 1 to instead be HTML paragraph ("p") tags. You can place HTML into the "main" element we found above using code like the following:

```
root.render(<p>Some message</p>);
```

Remember that HTML supports substitution but uses `{..}` rather than `${..}`.

3. Eat, Link, and Be Merry

In this problem, we will change the HTML to let the user see the next or previous Fibonacci number just by clicking on a link.

(a) Change the HTML placed in the page to contain two paragraph tags instead of one. The first paragraph should be the one from before, with the message about the n-th Fibonacci number. The second should be a paragraph containing link to the page that shows the next largest Fibonacci number. In order to put both paragraphs into the page, you will need to put them together into one HTML tree that is rendered. The easiest way to do so is to render a "div" tag, which has the two "p" tags as children.

Use an anchor ("a") tag to show a link. You will need to calculate the URL for the page showing that Fibonacci number and use substitution to put that URL into the HTML. It should look like this:

```
const nextUrl: string = "/?n=6";
... <p><a href={nextUrl}>Next</a></p> ...
```

but with the number "6" replaced by `n+1` using string substitution.

Try clicking on the link to make sure that it properly changes to the page that displays the next-largest Fibonacci number. Note that each click is loading a **new page**. (Try the back button to confirm.)

(b) Change the code so that, if the `n` is greater than 0, the second paragraph also includes a link to the previous FIbonacci number. If `n` is 0, it should show the HTML we just made in part (a).

Hint: this doesn't require any new JSX knowledge. It should be easy.

(c) Change the code so that, if "n" was not provided, we act as if it was 0. That way, the user will never need to type any number (even 0). They can just use the links to get to the Fibonacci number they want.

Hint: this only requires changing *one line of code* that currently renders an error message. You do not need to change anything else. (You definitely do not need to mutate any variables!)

(d) If you are not using VS Code, run the command `npm run lint` to make sure that your code meets the 331 coding conventions. (Homework solutions that do not pass will not be accepted by Gradescope.)

To get started, check out the starter code using the command
To install the libraries needed by the app, run the command `npm install --no-audit` in the newly-created `sec-fib` directory.

You may see an error about a missing `favicon.ico` file. You can ignore that. The file is missing at development time (now), but if we built for production, it would be there.

Coding Setup

We will assume that you have already installed Git Bash, NPM, and VSCode on your machine. Bash provides an environment where we can type commands to run programs. NPM (node package manager) reads and runs commands from a `package.json` file that describes the organization of our coding project. VSCode is an editor and is optional; you can replace it by any other editor of your choice.

Perform the following steps to get started with the code provided for this section:

1. Navigate to an appropriate directory on your machine in the terminal (using `cd`). Then, run the command

   ```
   git clone
   https://gitlab.cs.washington.edu/cse331-24sp/cse331-24sp-materials/sec-fib.git
   ```

   This will copy the starter code into a new subdirectory of the current directory called `sec-fib`.

2. Change into the `sec-fib` directory (`cd sec-fib`). Then, run the command

   ```
   npm install --no-audit
   ```

   This will nave `npm` download all of the dependencies referenced in the `package.json` file and store them locally (in the `node_modules` subdirectory) so that you can call them from your own code.

   Do not forget to include the `--no-audit` parameter. If you do forget, you will see some scary errors claiming that there are critical vulnerabilities in the code you downloaded that require fixing. This is not true. (If you want to confirm, you can run `npm audit --production` to see that there are no vulnerabilities in the code we are including in the app. `npm audit` is confused and thinks that some of the developer scripts are part of the production code, when they are not.)

3. Start the app by running the command `npm run start`, opening your browser, and navigating to `http://localhost:80`
   So far, the app only prints a message in the Developer Console.

4. Optional: open VSCode, click "Open Folder", and navigate to `sec-fib`.

   You should then be able to click on the `.ts` source files and edit them. You can also run the tests from within VSCode by clicking on the "Terminal" tab and typing `npm run test` there.