CSE 331: Software Design & Implementation

Section 1

1. The Level You Know

Answer each of the following questions in writing.

(a) Consider the following mathematical function[1] defined on the integers 1, 2, 3, and 4:

$$\begin{aligned} \textbf{func } f(1) &:= 2 \\ f(2) &:= 3 \\ f(3) &:= 4 \\ f(4) &:= 1 \end{aligned}$$

If we implement this directly in TypeScript using a `switch` statement, what level of correctness is required?

(b) Consider the following mathematical function defined on the inputs n and b, where n is 1, 2, 3, or 4 and b is true or false. It is defined in terms of the function f defined in part (a).

$$\begin{aligned} \textbf{func } g(n, \mathsf{T}) &:= f(n) \\ g(n, \mathsf{F}) &:= f(n) \end{aligned}$$

If we implement this in TypeScript using an `if` statement (on b), what level of correctness is required?

(c) Consider the following mathematical function defined on the inputs n and x, where n is 1, 2, 3, or 4 and x is any integer. It is defined in terms of the function f defined in part (a).

$$\textbf{func } h(n, x) := f(n) + x$$

If we implement this in TypeScript using a single `return` statement, what level of correctness is required?

(d) Suppose that we implement the function h with the following TypeScript code. (It calls f, which we will assume is implemented in TypeScript with a simple `switch` statement.)

```
function h(n: 1|2|3|4, x: number): number {
  let y = f(n);
  while (x > 0) {
    y = y + 1;
    x = x - 1;
  }
  return y;
}
```

What level of correctness is required now?

[1] "func" defines a mathematical function, whereas "`function`" is used to define a JavaScript function

2. Test Foot Forward

Start by performing the setup steps described on page 4. Then, complete the following parts:

(a) Fill in the code for the function `quadratic1` and `quadratic2` in `src/funcs.ts` so that it passes the tests provided in `src/funcs_test.ts` but is wrong (not correct on all inputs).

Make your implementation as simple as possible: just return a constant value.

Note that `src/funcs_test.ts` checks that you give the wrong answer for one specific input (see the file for which one), so when `npm run test` succeeds, it means that your code is "wrong" but successfully passes the required cases.

(b) Fill in the code for the function `abs_value1` and `abs_value2` in `src/funcs.ts` so that it passes the tests provided in `funcs_test.ts` but is wrong.

Your implementation must be simple, straight-line code. In `abs_value1`, you should invoke the method `cond` that is provided in `src/funcs.ts`. In `abs_value2`, you should instead use the conditional operator, which is written in the form "`a ? b : c`" in TypeScript.

The previous examples demonstrate that it is *not safe* to skip any of the test cases required by our heuristics. It is possible to write incorrect code that passes all required tests but one, and if that happens to be the one you skipped, then the incorrect code would pass our tests.

(c) If our code does pass all the tests required by our heuristics, does that *guarantee* that it is correct?

3. Put Your Mind To Test

Answer each of the following questions in writing.

(a) In how many test cases did our incorrect code for `abs_value1` and `abs_value2` give correct answers in the previous problem? How many test cases should we have written for this code? Why that many?

(b) How many tests should we write for the following function? Why?

```
function quadrant(x: number, y: number): 1|2|3|4 {
  if (x >= 0) {
    return (y >= 0) ? 1 : 2;
  } else {
    return (y <= 0) ? 3 : 4;
  }
}
```

(c) How many tests should we write for the following function, defined only on the *non-negative* integers? Why? What are the tests that we should use?

```
function f(n: number): number {
  if (n === 0) {
    return 0;
  } else if (n === 1) {
    return 1;
  } else if (n % 2 === 1) {  // n is > 1 and odd
    return f(n - 2) + 1;
  } else {                   // n is > 1 and even
    return f(n - 2) + 3;
  }
}
```

4. No Test For the Wicked

Complete the following written question only if you have enough time. (It is optional but recommended.)

Consider the following recursive function defined on the *non-negative* integers. It references two constants, `A` and `B`, which are numbers defined elsewhere in the code.

```
function f(n: number): number {
  if (n == 0) {
    return 0;
  } else {
    return A * f(n - 1) + B;
  }
}
```

(a) What are the values of $f(0)$, $f(1)$, $f(2)$, and $f(3)$ in terms of A and B?

(b) Suppose that we typed in the wrong value for `A` in the code. If we test the output of f for each input starting from 0, how far do we have to go before we could notice that `A` was wrong?

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 git@gitlab.cs.washington.edu:cse331-23sp-materials/sec-levels.git
   ```

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

2. Change into the `sec-levels` directory (`cd sec-levels`). 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. Confirm that you can run the tests with the command `npm test`.

   At this point, you should see a message saying that tests are failing. You will fix these test failures while completing the assignment.

4. Optional: open VSCode, click “Open Folder”, and navigate to `sec-levels`.

   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.