CSE 331: Software Design & Implementation

Section 2

1. The Level You Know

Answer each of the following questions in writing.

(a) Consider the following mathematical function 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 one conditional.

```
const h = (n: number, 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?

2. Test Foot Forward

To begin the coding problems, check out the starter code using the command

```
git clone https://gitlab.cs.washington.edu/cse331-23au-materials/sec-levels.git
```

Then, install the libraries needed by the app by running the command `npm install --no-audit` in the newly-created `sec-levels` directory.

Confirm that you can run the tests with the command `npm run test`. At this point, you should see a message saying that tests are failing. You will fix these test failures while completing the assignment.

(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_value` in `src/funcs.ts` so that it passes the tests provided in `funcs_test.ts` but is wrong.

Your implementation must be a single "`if`" statement (i.e., a conditional), with one branch returning "`x`" and the other branch returning "`-x`". You can choose the branch condition.

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. A Barrel of Halfs

Consider the following function, which calculates half when given an **even** number but also accepts other inputs (though it doesn't perform the same behavior in those cases):

$$\begin{aligned}
\textbf{func}\ \ &\text{half}(\text{null}) &&:= 0 \\
&\text{half}(\text{undefined}) &&:= 0 \\
&\text{half}(n : \mathbb{N}) &&:= n/2 &&\text{if } n \text{ is even} \\
&\text{half}(n : \mathbb{N}) &&:= -(n+1)/2 &&\text{if } n \text{ is odd}
\end{aligned}$$

(a) What is the type for the function half? (There are 2 possibilities.) Use the notation half : $A \to B$ to indicate that half takes inputs of type A and produces outputs of type B.

(b) What would the declarations of this function look like in TypeScript based on the type?

(c) Implement the mathematical function half as a Typescript function in `funcs.ts`. Make sure it is exported.

Verify that your implementation is correct by importing half and uncommenting the tests for it in `funcs_test.ts` and running `npm run test` to confirm that they pass.

4. The Rings of Pattern

Consider the following TypeScript code:

```
const maybeDouble = (t: {b: boolean, v: [boolean, number]}): number => {
  if (t.b) {
    if (t.v[0]) {
      return 2 * t.v[1];
    } else {
      return t.v[1];
    }
  } else {
    return 0;
  }
};
```

How would you translate this into our math notation using pattern matching?

5. Put Your Mind To Test

Answer each of the following questions in writing. This question is **optional** as we have not yet covered test heuristics fully in lecture. However, this is a useful set of practice questions for the homework, so we highly recommend you revisit these later on!

(a) We included 4 tests for `abs_value`, two for each branch. Why was that not enough to detect the problem? What heuristic did we forget about?

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

```
const s = (x: number, y: number): number => {
  if (x >= 0) {
    if (y >= 0) {
      return x + y;
    } else {
      return x - y;
    }
  } else {
    return y;
  }
}
```

(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?

```
const 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;
  }
}
```

6. No Test For the Wicked

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.

```
const 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?