# 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 4 conditional statements, 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: bigint, x: bigint): bigint => {
  let y = f(n);
  while (x > 0n) {
    y = y + 1n;
    x = x - 1n;
  }
  return y;
}
```

What level of correctness is required now?

## 2. 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):

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

(a) What is the type for the function half? 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) What would the implementation of the body of this function look like in TypeScript?

## 3. The Rings of Pattern

Consider the following TypeScript code:

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

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

## 4. Put Your Mind To Test

(a) Consider the following implementation of a function to calculate $|x|$.

```
export const abs_value = (x: bigint): bigint => {
  if (x > 1n) {
    return x;
  } else {
    return -x;
  }
};
```

Say we included the test cases -1, -2, 2 and 3; 2 cases for each branch. All the test cases passed, but this implementation is actually incorrect. Why were those cases 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: bigint, y: bigint): bigint => {
  if (x >= 0n) {
    if (y >= 0n) {
      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: bigint): bigint => {
  if (n === 0n) {
    return 0n;
  } else if (n === 1n) {
    return 1n;
  } else if (n % 2n === 1n) {  // n is > 1 and odd
    return f(n - 2n) + 1n;
  } else {                     // n is > 1 and even
    return f(n - 2n) + 3n;
  }
}
```

## 5. No Test For the Wicked

We probably won't get to this question in section, so it is **optional** for students making up section materials for being absent. However, this problem may be useful to refer to for the EC problem on HW2.

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

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

(a) What are the values of `f(0n)`, `f(1n)`, `f(2n)`, and `f(3n)` 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?