COMP110 - WS5 - Name: _____ PID: _____

1. Given the functions below, answer the following questions about filter, map, and reduce. If the returned value is a list (`Node`), write your answer in arrow list notation. (For example, `listify("Thanksgiving", "dinner")` should be written as `"Thanksgiving" -> "dinner" -> null`).

```
let numbers = listify(2, 7, 5, 6, 3, 9);
let booleans = listify(true, false, false, true, false);
let strings = listify("Thanksgiving", "fall", "chilly", "pie", "dinner");

let f1 = (s: string): boolean => {
    return s[3] === "n";
};

let f2 = (n: number): boolean => {
    return n % 4 === 1;
};

let f3 = (b: boolean): boolean => {
    return !b;
};

let f4 = (s: string): number => {
    return s.length;
};

let f5 = (n: number): number => {
    return n * 2;
};

let f6 = (s: string): string => {
    return s[2];
};

let f7 = (s: string, n: number): string => {
    return s + n;
};

let f8 = (n: number, m: number): number => {
    if (n < m) {
        return n + m;
    } else {
        return m;
    }
};

let f9 = (s: string, t: string): string => {
    return s + t;
};
```

1.1 What *type* of function is `f6` (which functional interface does it implement)?

Transform<string, string>

1.2 Which of these functions implements the `Reducer` functional interface?

f7, f8, f9

Write the returned values of the following calls:

1.3 `filter(booleans, f3);`

false -> false -> false -> null

1.4 `map(filter(numbers, f2), f5);`

10 -> 18-> null

1.5 `reduce(filter(strings, f1), f9, "");`

Thanksgivingdinner

1.6 `map(strings, f4);`

12 -> 4 -> 6 -> 3 -> 6 -> null

1.7 `filter(map(numbers, f5), f2);`

null

1.8 `reduce(map(strings, f6), f9, "");`

alien

2. Answer the following questions by cleary writing **True** or **False** in the blank.

2.1 Functions can be passed as arguments to other functions. **True**

2.2 The `filter` and `reduce` functions both always return a `Node`. **False**

2.3 TypeScript can always infer the parameter types and return type of a function.
False

2.4 Generic functional interfaces do not require a function's parameters to have a specific type.
True

2.5 Functional interfaces assign a name to a function's type. **True**

2.6 Function literals cannot be passed as arguments to higher order functions, but named functions can be.
False

3. Given the code to the left, answer the questions that follow.

```
1  export let main = async () => {
2      let arr = [2, 7, 3, 1, 0];
3      let aa = a(arr);
4      let bb = b(arr);
5      let cc = c(arr);
6  };
7
8  export let a = (x: number[]): number[] => {
9      let y = [];
10     for (let i = 0; i < x.length; i++) {
11         y[i] = x[i];
12         x[i] *= 2;
13     }
14     return y;
15 };
16
17 export let b = (x: number[]): number[] => {
18     x[x.length] = x[0] * 4;
19     return x;
20 };
21
22 export let c = (x: number[]): number[] => {
23     let y = [];
24     for (let i = 0; i < x.length; i++) {
25         y[i] = 3 * i;
26     }
27     x = y;
28     return x;
29 };
30
31 main();
```

3.1 What is the value of `arr` after line 3 runs?
[4, 14, 6, 2, 0]

3.2 What is the value of `aa` after line 3 runs?
[2, 7, 3, 1, 0]

3.3 What is the value of `arr` after line 4 runs?
[4, 14, 6, 2, 0, 16]

3.4 What is the value of `bb` after line 4 runs?
[4, 14, 6, 2, 0, 16]

3.5 What is the value of `arr` after line 5 runs?
[4, 14, 6, 2, 0, 16]

3.6 What is the value of `cc` after line 5 runs?
[0, 3, 6, 9, 12, 15]

4. Answer the questions assuming the program below is paused at line 9.

```
1  export let main = async () => {
2      let x = 9;
3      let one = x;
4      x = x + 3;
5      let z1 = [2, 2];
6      let z2: number[] = [x, 110];
7      let rv = a(z2);
8      let rv2 = a(z1);
9      // Break here
10 };
11 let a = (arr: number[]): number[] => {
12     if (arr[0] % 3 === 0) {
13         b(arr);
14         return arr;
15     }
16     if (arr[0] % 2 === 0) {
17         arr = b(arr);
18         return arr;
19     }
20     return arr;
21 };
22 let b = (arr: number[]): number[] => {
23     arr = [99, 111];
24     return arr;
25 };
26
27 main();
```

4.1 What is the value of `one`? **9**

4.2 What is the value of `x`? **12**

4.3 What is the value of `rv`? **[12, 110]**

4.4 What is the value of `rv2`? **[99, 111]**

5. In the space below, define a generic function named `reverse`. Given an input `Node` named `list`, `reverse` should return a new list that is the input list reversed. In order to accomplish this, define a new function named `reverseRecur` that has two parameters: a generic Node named `list` and a generic Node named `out`. You should recursively build up `out` and return it. After you have defined `reverseRecur`, return a call to it with appropriate arguments within the `reverse` function.
Example usage is as follows: the list returned by `reverse(listify(1, 2, 3))` is equivalent to the list returned by `listify(3, 2, 1)`.

```
let reverse = <T>(list: Node<T>) => {
    return reverseRecur(list, null);
};

let reverseRecur = <T>(list: Node<T>, out: Node<T>): Node<T> => {
    if (list === null) {
        return out;
    }
    return reverseRecur(rest(list), cons(first(list), out));
};
```

6. Given the snippet below, determine what will be printed out at each of the stated lines. If the printed value is a list (`Node`), write your answer in arrow list notation. (For example, `listify(1, 2)` should be written as 1 -> 2 -> null).

```
1  let select = (x: string) => {
2      return x.length === 4 || x.length === 8;
3  };
4
5  let oddEven = (x: number) => x % 2 === 1;
6
7  let questions = (x: string) => x + "??";
8
9  let len = (x: string) => x.length;
10
11 let c1 = (m: string, e: string): string => {
12     return m + "␣" + e;
13 };
14 let c2 = (m: string, e: number): string => {
15     return e + "␣" + m;
16 };
17
18 let mult = (m: number, e: number): number => {
19     return m * e;
20 };
21
22 let main = () => {
23     let coolNewList: Node<string> = listify("What",
           "happened", "to", "our", "water");
24     print(reduce(map(filter(coolNewList, select),
           questions), c1, ""));
25     print(reduce(filter(map(coolNewList, len),
           oddEven), mult, 1));
26     print(reduce(map(coolNewList, len), c2, "!"));
27     print(map(map(filter(coolNewList, select), len)
           , (x) => x + 5));
28 };
29
30 main();
```

6.1 Line 24

What?? happened??

6.2 Line 25

15

6.3 Line 26

5 3 2 8 4 !

6.4 Line 27

9->13->null

7. Due to ink costs, the Yellow Pages is no longer adding information about companies with names that are too long. Completely rewrite **all** of line 2 so that `willBeIncluded` is a list containing only the companies whose names have fewer than 10 characters, replacing `insertFunctionalLiteralHere` with a function literal.

```
1     let companies = listify("Walmart", "Target", "Bed␣Bath␣and␣Beyond", "Costco", "Mediterranean␣Deli", "
          Mellow␣Mushroom");
2     let willBeIncluded = filter(companies, insertFunctionalLiteralHere);
```

```
let willBeIncluded = filter(companies, (s) => s.length < 10)
```