COMPSCI 220 Fall 2018

Midterm 2

November 15, 2018

Time for Exam: 60 minutes

Name: **Spire ID:**

DO NOT OPEN EXAM BOOKLET UNTIL INSTRUCTED TO DO SO

Instructions:

Do not begin your exam until instructed. Please read all rules carefully before beginning your exam. You will have one hour to complete all exam problems to the best of your ability.

Rules:

1. All electronics(including but not limited to cellphones, tablets, computers, smart watches, and calculators) must be turned off and placed out of sight in your backpack.
2. In order to receive credit for your midterm you **MUST** write your name and Spire ID on every page.
3. All answers must fit within the boxes allocated for that question. Any work outside of the box will not be taken into account during grading.
4. There will be no talking or leaving the exam room room during testing.
5. You may use any printed or hand written material you wish as long as it is on paper and not a digital document.
6. All work must be your own and compliant with the Universities Academic Honesty Policy. Any exhibition of Academic Dishonesty will be reported to the Academic Honesty Board.

Name: _____

Question 1: What are the outputs of the following programs? If there are any run-time errors, report which lines will result in errors, and what the error will be.

Part a

```
1  function f1(a, b) {
2    return function(x) {
3      return a * x + b;
4    };
5  }
6  let f2 = f1(7, 9);
7  console.log(f2(4));
```

Part b

```
1  function foo(x) {
2    return {
3      get: function() { return x; },
4      set: function(y) { x = y; }
5    };
6  }
7  let o1 = foo(10);
8  let o2 = foo(20);
9  console.log(o1.get());
10 console.log(o2.get());
```

Name: _____

Part c

```
let a = [
  function (x) { return 3 * x; },
  function (y) { return 5 * y; },
  function (a) { return 2 * a; },
];
let f = a.reduce(function(f1, f2) {
  return function(x) {
    return f1(x) + f2(x);
  }
}, function(x) { return 0; });
console.log(f(1));
console.log(f(2));
console.log(f(3));
```

Part d

```
function foo(x) {
  return {
    get: function() { return x; },
    set: function(y) { x = y; }
  };
}
let o1 = foo(10);
let o2 = foo(20);
o1.set(42);
o2.set(21);
console.log(o1.get());
console.log(o2.get());
```

Name: _____

Part e

```
1  function foo(x) {
2    return {
3      x: x,
4      get: function() { return x; },
5      set: function(y) { x = y; }
6    };
7  }
8  let o = foo(10);
9  o.set(20);
10 console.log(o.x - o.get());
```

Question 2:

Consider the following class definition:

```
class Fluent {
  constructor(a) {
    this.a = a;
  }
  f1(y) {
    return new Fluent(this.a.filter(
      function(x) { return x % y === 0; }
    ));
  }
  f2(y) {
    return new Fluent(this.a.filter(
      function(x) { return (x + y) % 10 === 0; }
    ));
  }
}
```

Name: _____

Part a

What is the output of the following program?

```
let x = new Fluent([1, 2, 3, 4, 5, 6, 7, 8, 9]);
let y = x.f1(3).f1(2).f2(4);
console.log(y.a);
```

Output:

Part b

Provide three distinct inputs for **a** such that the following program produces the specified output:

Program:

```
let a = // your answer below
let x = new Fluent(a);
let y = x.f1(2).f1(5).f2(0);
console.log(y.a);
```

Output:

```
[10, 20, 30]
```

Inputs:

let a = [ ];	let a = [ ];	let a = [ ];

Name: _____

Question 3:

Consider the following incomplete class definition that implements the Observer design pattern as a linked list:

```
const listEnd = {};
class LinkedObservable {
  constructor(f) {
    this.next = listEnd;
    this.f = f;
  }
  update(x) {
    this.f(x);
    if (this.next !== listEnd) {
      this.next.update(x);
    }
  }
  addToEnd(f) {
    if (this.next === listEnd) {
      // Do something...

    } else {
      // Do something...

    }
  }
}
```

Part A

The class declaration has the addToEnd() method incompletely defined. Write code to complete the `addToEnd(f)` method, such that the following unit tests pass:

```
test('Observe 1', function() {
  let result = 0;
  let f = function(x) {
    result = x;
  }
  let l = new LinkedObservable(f);
  l.update(42);
  assert(result === 42);
});

test('Observe 2', function() {
  let result = 0;
  let f = function(x) {
    result = result + x;
  }
  let l = new LinkedObservable(f);
  l.addToEnd(f);
  l.update(1);
  assert(result === 2);
});
```

Name: _____

```
class LinkedObservable {
  // ... other existing declarations omitted for brevity
  addToEnd(f) {
    if (this.next === listEnd) {



    } else {



    }
  }
}
```

Name: _____

Part B

The definition of the update(x) method leads to the observers being called in a specific order. Modify the definition so that the order of calling the observers is reversed, and the following tests pass, **without using any loops in the update method**:

```
test('Observe Order', function() {
  const v1 = Math.random();
  const v2 = Math.random();
  let result = [];
  let f1 = function(x) {
    result.push(x + v1);
  }
  let f2 = function(x) {
    result.push(x + v2);
  }
  let l = new LinkedObservable(f1);
  l.addToEnd(f2);
  l.update(42);
  assert(result.length === 2);
  assert(result[0] === 42 + v2);
  assert(result[1] === 42 + v1);
});
```

```
class LinkedObservable {
  // ... other existing declarations omitted for brevity
  update(x) {









  }
}
```

Name: _____

Question 4:

Write a function called rainfall that consumes an array of numbers representing daily rainfall amounts as entered by a user. The list may contain the number -999 indicating the end of the data of interest. Produce the average of the non-negative values in the list up to the first -999 (if it shows up). There may be negative numbers other than -999 in the list.

Your rainfall function should pass the following tests:

```
test('Correct Average Rain', function() {
  let r = [10, 10, 10, -999];
  assert(rainfall(r) === 10);
});

test('Ignore Negative Rain', function() {
  let r = [10, -10, 20, -999];
  assert(rainfall(r) === 15);
});

test('Return Correct Rainfall of Interest', function() {
  let r = [1, -999, 10, -999];
  assert(rainfall(r) === 1);
});

test('Check For Observations', function() {
  let r = [-1, -999];
  assert(rainfall(r) === 0);
});

test('Check For -999', function() {
  let r = [10, 10, 10];
  assert(rainfall(r) === 0);
});
```

Name: _____

Answer:

```
function rainfall(a) {
  let res = {
    // Total amount of rain.
    total:       , // FILL THIS
    // Number of days with valid rainfall.
    n:           , // FILL THIS
    // Whether -999 has been encountered.
    Found:         // FILL THIS
  };
  let f = function (a, b) {
    // FILL THIS









    return a;
  }
  res = a.reduce(f, res);
  if (                      ) {  // FILL THIS



  } else {  // FILL THIS



  }
}
```