COMPSCI 220 Spring 2019

Midterm 1

February 28, 2019

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.

Points:

The questions in this exam are worth 100 points. However, you only need to correctly answer 90 points of questions to receive full credit. You will not receive extra credit for answering extra questions. You may attempt all questions if you wish – there is no penalty for doing so.

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?

Part a (5 points)

```
function foo(x) {
  if (x === 5) {
    return;
  }
  console.log(x);
  foo(x + 1);
}
foo(3);
```

Part b (5 points)

```
function app(x, y, f) {
  return f(x, y);
}

app(function() { console.log("A"); },
    function() { console.log("B"); },
    function(m, n) {
     console.log("C");
     return m();
    });
```

Part c (5 points)

```
let a = [{x: 1}];
for (let i = 0; i < 5; ++i) {
  a.push(a[0]);
}
a[0].x = 3;
let s = 0;
for (let i = 0; i < a.length; ++i) {
  s = s + a[i].x;
}
console.log(s);
```

Part d (5 points)

```
function mystery(f, n) {
  if (n === 0) {
    return function(x) { return x; }
  } else {
    return function(x) { return mystery(f, n - 1)(f(x)); }
  }
}

function H(x) {
  return x + 1;
}

console.log(mystery(H, 100)(5));
```

Name: ____

Part e (5 points)

```
function foo(x) {
  console.log(x);
  if (x === 0) {
    return;
  }
  foo(x - 1);
  console.log(x);
}
foo(3);
```

Part f (5 points)

```
function foo() {
  console.log('A');
  function bar() {
    console.log('B');
  }
  console.log('C');
}
foo();
```

Name: ____

Question 2: Consider the following function:

```
function foo(x, y) {
  if (y > 0) {
    console.log('A');
  }
  function bar() {
    if (y < 20) {
      console.log('B');
    }
  }
  if (x % 2 === 0) {
    bar();
  }
}
```

Part a (5 points): Give three values for x and y that will make `foo(x, y)` display:

```
A
B
```

x = y =	x = y =	x = y =

Part b (5 points): Give three values for x and y that will make `foo(x, y)` display:

```
A
```

x = y =	x = y =	x = y =

Part c (5 points): Give three values for x and y that will make `foo(x, y)` display:

```
B
```

x = y =	x = y =	x = y =

Name: ____

Question 3: Consider the following function:

```
function foo(f) {
  f(function(x) {
      if (x > 10) { console.log("A"); }
    },
    function(y) {
      if (y > 0) {
        return function() { console.log("B"); } ;
      } else {
        return function() { console.log("C"); };
      }
    });
}
```

Part a (5 points): Give a value for `f`, such that `foo(f)` will display the following output. **Your answer must not use `console.log`**.

```
A
```

Part b (5 points): Give a value for `f`, such that `foo(f)` will display the following output. **Your answer must not use `console.log`**.

```
A
B
```

Part c (5 points): Give a value for f, such that foo(f) will display the following output. **Your answer must not use `console.log`**.

```
A
C
```

Name: ____

Question 4: In class, we presented several functions that consume and produce immutable linked lists (see the exam notes). Unfortunately, immutability is merely a convention. Nothing prevents a badly-written program from mutating a list that should be immutable. However, we have also seen that closures can be used to hide values. Therefore, we can use closures to implement linked lists in an alternative way that hides the ability to mutate the list.

```
function node(head, tail) {
  return function(op) {
    if (op === 'empty') {
      return false;
    } else if (op === 'head') {
      return head;
    } else if (op === 'tail') {
      return tail;
    } else {
      throw 'invalid operation';
    }
  };
}

function empty() {
  return function(op) {
    if (op === 'empty') {
      return true;
    } else if (op === 'head') {
      throw 'head of empty list';
    } else if (op === 'tail') {
      return 'tail of empty list';
    } else {
      throw 'invalid operation';
    }
  };
}
```

Using these functions, we can write linked lists as follows:

```
let alist1 = node(10, node(20, node(30, empty())));
```

(Problems on the next page.)

Name: ____

Part a (10 points). Write a function to calculate the length of a list.

```
length(node(10, node(20, node(30, empty())))) // produces 3
length(empty()) // produces 0
```

```
// length<T>(list: List<T>): number
function length(list) {

}
```

Part b (10 points). Write the `map` function for linked lists

```
map(function(x) { return x + 1; }, node(10, node(20, node(30, empty()))))
// produces a value equivalent to node(11, node(21, node(31, empty())))
```

```
// map<T>(f: (x: S) => T, list: List<S>): List<T>
function map(f, list) {

}
```

Name: ____

Question 5: The following function, unreduce, is the inverse of reduce:

```
// unreduce<S,T>(pred: (x: S) => boolean, f: (x: S) => [T, S], init: S): T[]
function unreduce(pred, f, init) {
  let arr = [];
  let acc = init;
  while (pred(acc)) {
    let r = f(acc);
    arr.push(r[0]);
    acc = r[1];
  }
  return arr;
}
```

Write the following functions using `unreduce`.

Part a (10 points). Write a function that consumes a number `n`, and produces an array of numbers from `0` up to and excluding `n`. For example:

```
upto(5) // produces [0, 1, 2, 3, 4]
upto(0) // produces []
```

```
// upto(n: number): number[]
function upto(n) {
  let pred = function(x) {






  };
  let f = function(x) {






  };
  let init =                       ;




  return unreduce(pred, f, init);
}
```

Name: ____

Part b (10 points). Write map using unreduce.

```
// map<S,T>(func: (x: S) => T, arr: S[]): T[]
function map(func, arr) {
  let pred = function(x) {





  };
  let f = function(x) {







  };
  let init =                       ;



  return unreduce(pred, f, init);
}
```

End of exam.