From Lecture

Implementation of pair, head and tail:

```javascript
function pair(x, y) {
    return function(m) { return m(x, y); }
}

function head(z) {
    return z(function(p, q) { return p; });
}

function tail(z) {
    return z(function(p, q) { return q; });
}
```

Definitions

The following are two higher-order functions discussed in lecture:

```javascript
function sum(term, a, next, b) {
    if(a > b) {
        return 0;
    } else {
        return term(a) + sum(term, next(a), next, b);
    }
}

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

Note: it is not necessary to memorize these definitions, or even the names of these functions. Definitions of such functions (if they are used) will be given in an Appendix for examinations. What is most important is that students must be able to read the definition for such a function and understand what it does.
Problems:

1. Evaluate the return values of the following sets of statements:

   (a) var x = 12; x;
   Answer: 12

   (b) var x = 12; (function() x = 15; )(); x;
   Answer: 15

   (c) var x = 20; (function() var x = 15; )(); x;
   Answer: 20

2. Write a function my\_sum that computes the following sum, for \( n \geq 1 \):

\[
1 \times 2 + 2 \times 3 + \cdots + n \times (n + 1)
\]

Answer:

```javascript
function my_sum(n) {
  if (n === 1) {
    return 2;
  } else {
    return (n * (n + 1)) + (my_sum(n - 1));
  }
}
```

3. Is the function my\_sum as defined in Question 1 above a recursive process or an iterative process? What is the order of growth in time and in space?

   Answer: Recursive. Time: \( O(n) \), space: \( O(n) \).

4. If your answer in Question 2 is a recursive process, re-write my\_sum as an iterative process. If your answer in Question 2 is an iterative process, re-write my\_sum as a recursive process.

   Answer:

```javascript
function my_sum(n) {
  function my_sum_iter(counter, sum) {
    if (counter === 0) {
      return sum;
    } else {
      return my_sum_iter(counter - 1, sum + (counter * (counter + 1)));
    }
  }
  return my_sum_iter(n, 0);
}
```

5. We can also define my\_sum in terms of the higher-order function \( \text{sum} \). Complete the definition of my\_sum below. You cannot change the definition of \( \text{sum} \); you may only call it with appropriate arguments.

```javascript
function my_sum(n) { return \text{sum}(<T1>, <T2>, <T3>, <T4>); }
```

   Answer:
3. Suppose instead we define \texttt{my\_sum} in terms of the higher-order function \texttt{fold}. Complete the definition of \texttt{my\_sum} below.

\begin{verbatim}
function my_sum(n) {
  return fold(<T1>, <T2>, <T3>); }
\end{verbatim}

\textbf{Answer:}

\begin{verbatim}
T1: function(a, b) { return a + b; }
T2: function(n) {
  if (n === 0) {
    return 0;
  } else {
    return n * (n + 1); }
}
T3: n
\end{verbatim}

7. Write an iterative version of \texttt{sum}.

\textbf{Answer:}

\begin{verbatim}
function sum(term, a, next, b) {
  function sum_iter(a, total) {
    if (a > b) {
      return total;
    } else {
      return sum_iter(next(a), total + term(a));
    }
  }
  return sum_iter(a, 0);
}
\end{verbatim}

8. \textbf{Homework:} Write an iterative version of \texttt{fold}.