NATIONAL UNIVERSITY OF SINGAPORE

CS1101S — PROGRAMMING METHODOLOGY

(AY2020/2021 SEMESTER 1)

MIDTERM ASSESSMENT

Time Allowed: **1 Hour 30 Minutes**

INSTRUCTIONS

1. This assessment contains **20 Questions** in **5 Sections**.

2. The full score of this assessment is **70 marks**.

3. Answer **all questions**.

4. This is a **Closed-Book** assessment, but you are allowed one double-sided **A4 / foolscap / letter-sized sheet** of handwritten or printed **notes**.

5. You are allowed to use up to **4 sheets** of **blank A4 / foolscap / letter-sized** paper as **scratch paper**.

6. Where programs are required, write them in the **Source §2** language. You are allowed access to the online reference page for **Source §2 pre-declared constants and functions** at `https://source-academy.github.io/source/source_2/global.html`

7. In any question, your answer may use **functions given in, or written by you for,** any **preceding question**. You can assume a correct solution from the preceding question as given, even if your solution from that preceding question was not correct.

8. **Follow the instructions of your invigilator or the module coordinator to submit your answers**.

Section A: List Notation [9 marks]

(1) [3 marks]

What is the result of evaluating the following Source program in *list notation*?

```
const xs = list(6, 7, 8);
pair(xs, tail(xs));
```

A. `list( list(6, 7, 8), 7, 8 )`
B. `list( list(6, 7, 8), list(7, 8) )`
C. `list( list(6, 7, 8), [7, 8] )`
D. `[ [ list(6, 7, 8), 7 ], 8 ]`
E. `[ list(6, 7, 8), list(7, 8) ]`
F. `[ list(6, 7, 8), [7, 8] ]`

(2) [3 marks]

What is the result of evaluating the following Source program in *list notation*?

```
const xs = list(3, 4, 5);
accumulate((x, ys) => append(list(x), list(ys)), null, xs);
```

A. `list(3, 4, 5)`
B. `list(5, 4, 3)`
C. `list(list(list(null, 3), 4), 5)`
D. `list(list(list(null, 5), 4), 3)`
E. `list(3, list(4, list(5, null)))`
F. `list(5, list(4, list(3, null)))`

(3) [3 marks]

What is the result of evaluating the following Source program in *list notation*?

```
function fun(xs) {
    return is_null(xs)
        ? null
        : pair( map(x => head(xs), enum_list(1, head(xs))),
                fun(tail(xs)) );
}
fun( list(2, 4, 3) );
```

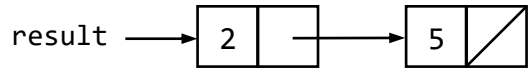
A. `list( list(1, 2), list(1, 2, 3, 4), list(1, 2, 3) )`
B. `list( list(2, 2), list(4, 4, 4, 4), list(3, 3, 3) )`
C. `list( list(1, 2, 3), list(1, 2, 3, 4), list(1, 2) )`
D. `list( list(3, 3, 3), list(4, 4, 4, 4), list(2, 2) )`
E. `[ list(1, 2), [ list(1, 2, 3, 4), list(1, 2, 3) ] ]`
F. `[ list(2, 2), [ list(4, 4, 4, 4), list(3, 3, 3) ] ]`

Section B: Box-and-Pointer Diagrams [8 marks]

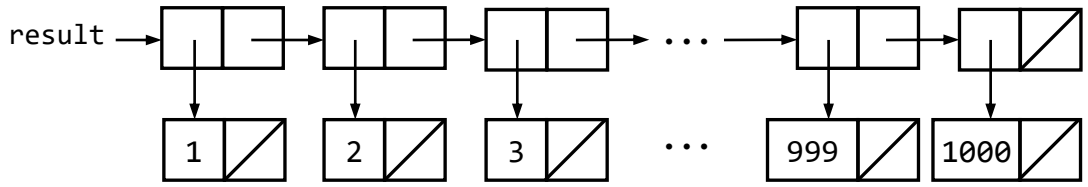
For each of the following box-and-pointer diagrams, write a Source program such that at the end of the evaluation of your program, the name `result` will have the value as shown in the diagram. You must not use any ellipsis (...) in your program.

For example, the following program results in the following diagram on the right:

```
const result = list(2, 5);
```



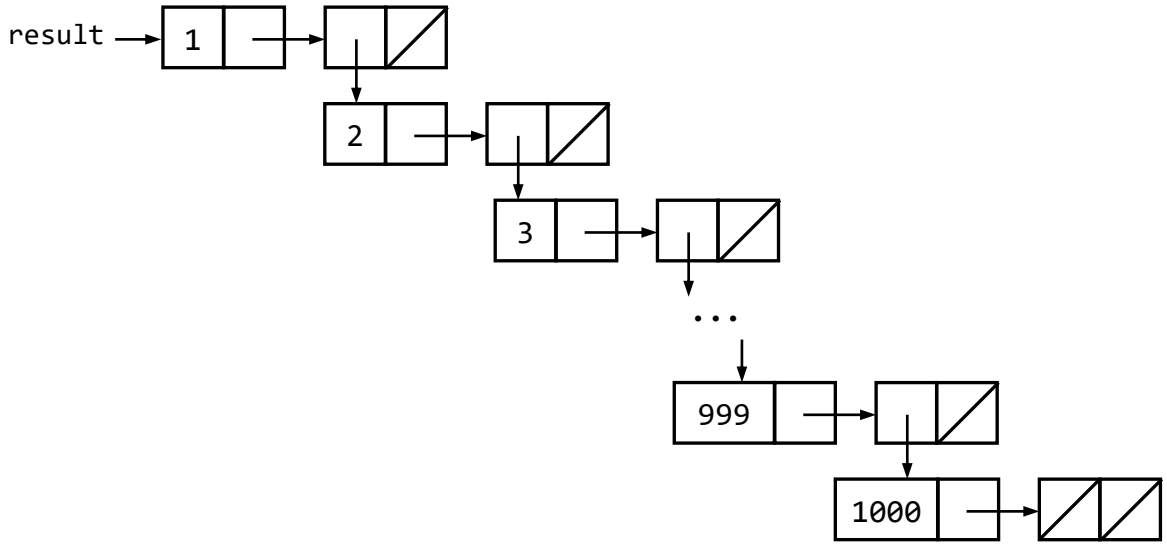
(4) [4 marks]




```
function fun (n) {
    return n === 1000
        ? null
        : list( list(list(n)), fun(n+1) );
}
```

map (x => list(x), enum_list(1, 10));

(5) [4 marks]




```
function fun (n) {
    return n === 1000
        ? null
        : pair( n , pair( fun(n+1) , null ) );
}
```

Section C: Orders of Growth [10 marks]

What is the **order of growth** of the **runtime** of each of the following functions in terms of N using the Θ notation? Note that N is a positive integer value.

(6) [2 marks]

```
function fun(N) {
    return N <= 1 ? 1 : fun(2 * N / 3) * (2 * N / 3);
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

(7) [2 marks]

```
function fun(N) {
    return N <= 1000000 ? N : (N / 1000000) + fun(N - 1000000);
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

(8) [2 marks]

```
function fun(N) {
    return N <= 1 ? N : fun(N / 2) + fun(N / 2) + 1;
}
```

$T(n) = \Theta(1) + 2T(\frac{1}{2}n)$

$= \Theta(n)$

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

B ✗ C.

(9) [2 marks]

```
function fun(N) {
    if (N <= 1) {
        return N;
    } else {
        const h = x => x <= 0 ? 0 : x + h(x - 1);
        return h(N) + fun(N - 1);
    }
}
```

$\Theta(n)$

$\Theta(n)$

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

E ✓

(10) [2 marks]

```
function fun(N) {
    if (N <= 1) {
        return N;
    } else {
        const x = accumulate((x, y) => x + y, 0, enum_list(1, N));
        return x + fun(N / 2) + fun(N / 2);
    }
}
```

A. $\Theta(1)$

B. $\Theta(\log N)$

C. $\Theta(N)$

D. $\Theta(N \log N)$

E. $\Theta(N^2)$

F. $\Theta(N^2 \log N)$

G. $\Theta(N^3)$

H. $\Theta(2^N)$

Section D: Matrices [17 marks]

A $R \times C$ matrix is represented in Source as a list of R lists of numbers and each list of numbers is of length C. Note that R and C are positive integers. For example, the following 3×4 matrix

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

is represented in Source as

```
list( list(1,  2,  3,  4),
      list(5,  6,  7,  8),
      list(9, 10, 11, 12) )
```

(11) [4 marks]

Complete the function `get_elem` that takes as arguments a matrix `M`, a row number `r`, and a column number `c`, and returns the matrix element at Row `r` and Column `c`. Note that the topmost row in the matrix is Row 0 and the leftmost column is Column 0.

```
function get_elem(M, r, c) {
    return /* your solution */
}
```

Example:

```
const A = list( list(1, 2, 3), list(4, 5, 6), list(7, 8, 9) );
get_elem(A, 0, 2);  // returns 3
get_elem(A, 2, 0);  // returns 7
```

```
function get_elem (M, r, c) {
    return list_ref( list_ref (M , r) , c ) ;
}
```

(12) [4 marks]

Complete the function `horizontal_flip` that takes as argument a matrix `M`, and returns a matrix that is the *horizontal flip* of `M`.

```
function horizontal_flip(M) {
    return /* your solution */
}
```

Example:

```
const A = list( list(1,2,3,4), list(5,6,7,8), list(9,10,11,12) );
horizontal_flip(A);
// returns list( list(4,3,2,1), list(8,7,6,5), list(12,11,10,9) )
```

```
function horizontal_flip(M) {
    // return map( reverse, M );
}
```

(13) [4 marks]

Complete the function `row_sums` that takes as argument a matrix `M`, and returns a list of numbers containing the sum of elements in each row of `M`.

```
function row_sums(M) {
    return map( /* your solution */ , M );
}
```

Example:

```
const A = list( list(1,2,3,4), list(5,6,7,8), list(9,10,11,12) );
row_sums(A);
// returns list(10, 26, 42)
```

```
row => accumulate( (x,y) => x+y , 0 , row)
```

(14) [5 marks]

The *transpose* of a $R \times C$ matrix `M` is a $C \times R$ matrix `T` such that

```
get_elem(M, r, c) === get_elem(T, c, r).
```

Complete the function `transpose` that takes as argument a matrix `M`, and returns a matrix that is the transpose of `M`.

```
function transpose(M) {
    const nR = length(M);        // number of rows
    const nC = length(head(M));  // number of columns

    // You may not need to use nR and/or nC.

    return map( /* your solution */ , enum_list(0, nC - 1) );
}
```

Example:

```
const A = list( list(1,2,3,4), list(5,6,7,8), list(9,10,11,12) );
transpose(A);
// returns list( list(1,5,9), list(2,6,10), list(3,7,11), list(4,8,12) )
```

Section E: Pair-Trees [26 marks]

We introduce the following notion:

A ***pair-tree*** is a number or a pair whose head is a pair-tree and whose tail is a pair-tree.

(15) [2 marks]

Recall that a *tree of numbers* is a list whose elements are numbers or trees of numbers.
Which of the following statements is correct?

A. Every pair-tree is a tree of numbers.
B. No pair-tree is a tree of numbers.
C. Exactly one pair-tree is a tree of numbers.
D. More than one pair-tree, but not all pair-trees, are trees of numbers.
E. A pair-tree has either zero pair or an odd number of pairs.

(16) [2 marks]

Recall that a *tree of numbers* is a list whose elements are numbers or trees of numbers.
Which of the following statements is correct?

A. Every tree of numbers is a pair-tree.
B. No tree of numbers is a pair-tree.
C. Exactly one tree of numbers is a pair-tree.
D. More than one tree of numbers, but not all trees of numbers, are pair-trees.
E. Every tree of numbers is a list of numbers.

(17) [5 marks]

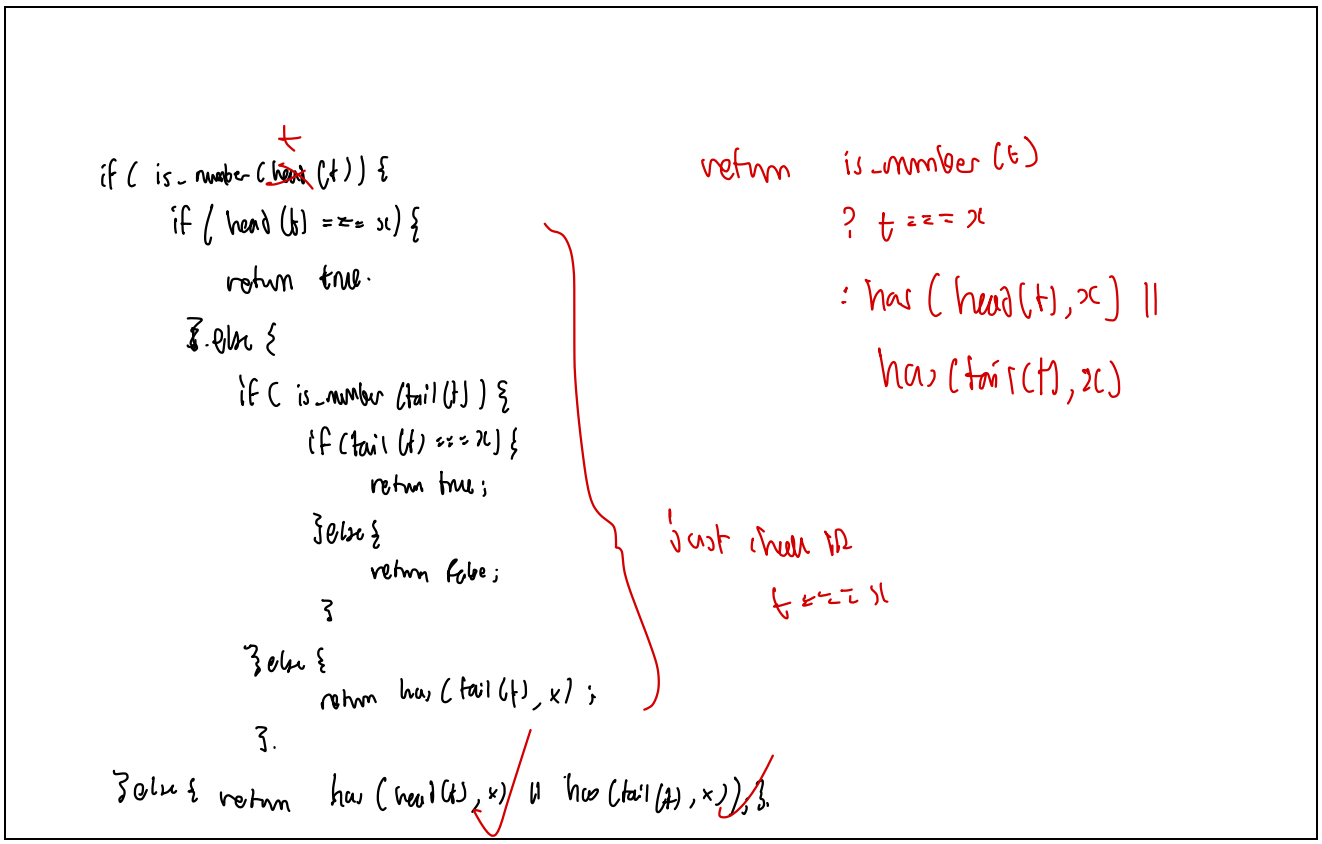
Write the function `has(t, x)` that returns `true` if pair-tree `t` contains the number `x`, otherwise it returns `false`.

```
function has(t, x) {
    /* your solution */
}
```

Examples:

```
const t1 = 8;
has(t1, 4);  // returns false
has(t1, 8);  // returns true

const t2 = pair(pair(1, 2), pair(3, pair(4, 5)));
has(t2, 4);  // returns true
has(t2, 8);  // returns false
```



(18) [5 marks]

A *path* is a list of functions that are successively applied to a pair-tree to reach a subtree. For example, the path `list(head, tail, tail)` applied to `pair(pair(1, pair(2, 5)), 3)` gives you 5. Write the function `apply(p, t)` that takes a path `p` and a pair-tree `t` as arguments and returns the subtree of `t` that `p` refers to. You can assume that the given path `p` is a valid path within pair-tree `t`.

```
function apply(p, t) {
    /* your solution */
}
```

f(1, f(2, f(3, null)))

Examples:

```
const t1 = 8;
apply(null, t1);  // returns 8

const t2 = pair(pair(1, 2), pair(3, pair(4, 5)));
apply( list(tail, tail, head), t2 );  // returns 4
apply( list(head), t2 );  // returns pair(1, 2)
```

```
return accumulate( (x, y) => (list_ref( p, length(p) - x ))(t),
                   null,
                   enum_list( 0, length(p) - 1 ) );
```

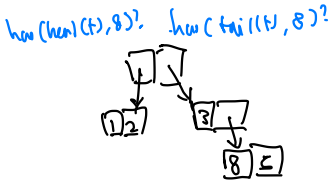
```
return is_null(p)
    ? t  ← return this.
    : apply(tail(p), (head(p))(t));
```

change these 2

(19) [6 marks]

Assume that the number 8 (lucky number) appears *exactly once* in a given pair-tree. Write the function `find_8(t)` that takes the pair-tree as an argument and returns the unique path to 8, i.e. `apply(find_8(t), t)` should return 8 if `t` contains exactly one 8.

```
function find_8(t) {
    /* your solution */
}
```



Examples:

```
const t1 = 8;
find_8(t1);  // returns null

const t2 = pair(pair(1, 2), pair(3, pair(8, 5)));
find_8(t2);  // returns list(tail, tail, head)
```

```
const is_8 = x => equal(x,8);

return is_8(t)
    ? null                                  // base case
    : is_8(head(t))                         // if head is 8
        ? list(head)
        : is_8(tail(t))
        ? list(tail, head)
        : has(head(t),8)
            ? append(list(head), find_8(t))
            : has(tail(t),8)
                ? append(list(tail), find_8(t));
```

```
is_number(t)
? null
: has(head(t),8)
    ? pair(head, find_8(head(t)))
    : pair(tail, find_8(tail(t)));
```

(20) [6 marks]

Write the function `find_all_8(t)` that takes a pair-tree `t` as argument and returns the list of all paths that lead to the number 8. The order of the paths in the result list does not matter. Note that the number 8 might appear any number of times in the pair-tree, including not at all.

```
function find_all_8(t) {
    /* your solution */
}
```

Examples:

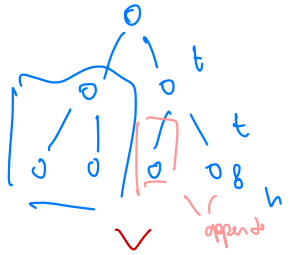
```
const t1 = 8;
find_all_8(t1);  // returns list(null)

const t2 = pair(pair(1, 2), pair(3, pair(4, 5)));
find_all_8(t2);  // returns null

const t3 = pair(8, pair(8, pair(8, 2)));
find_all_8(t3);
// returns list( list(head), list(tail, head), list(tail, tail, head) )
```

```
if ( has( t, 8 ) ) {
        const path = find_8 (t);
        return append (path, find_all_8 ( remove_8 ( path, t ) ) );
} else {
        return null;
}.

function remove_8 ( path, 8, count, sub_t ) {
        const action = list_ref( path, count);
        if ( equal ( action, head ) ){
            return append ( remove_8( sub_part, action(t), count+1, tail(t) ), tail(t) );
        } else if ( equal ( action, tail ) ){
            return append ( head (t), remove_8( sub_part, action(t), count+1, head (t) );
        }
        else {
            return append ( null, tail(t) );
        }
}
```



append.

```
is_number (t)
? t === 8
    ? list(null)
    : null
: append ( map ( x => pair (head, x) ),
           find_all_8 (head (t)) ),
           map ( x => pair (tail, x),
           find_all_8 (tail (t)) );
```

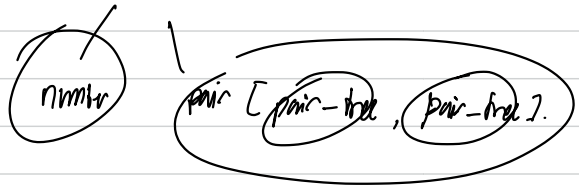
add the list of the head.

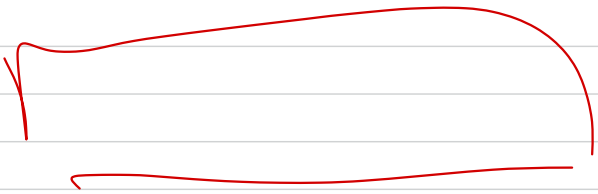
add the list of the tail.

——— END OF PAPER ———

pair tree - number

pair [pair_tree , pair_tree]





$\forall (x_1 \cdots x_n)$

↓

$(f(x_1) \cdots f(x_n))$

acc

$f(x_1, f(x_2, f(x_3 \cdots$

$f(x_n, null/init))$

NATIONAL UNIVERSITY OF SINGAPORE

CS1101S — PROGRAMMING METHODOLOGY

(AY2020/2021 SEMESTER 1)

MIDTERM ASSESSMENT

Time Allowed: **1 Hour 30 Minutes**

SOLUTIONS

INSTRUCTIONS

1. This assessment contains **20 Questions** in **5 Sections**.
2. The full score of this assessment is **70 marks**.
3. Answer **all questions**.
4. This is a **Closed-Book** assessment, but you are allowed one double-sided **A4 / foolscap / letter-sized sheet** of handwritten or printed **notes**.
5. You are allowed to use up to **4 sheets** of **blank A4 / foolscap / letter-sized** paper as **scratch paper**.
6. Where programs are required, write them in the **Source §2** language. You are allowed access to the online reference page for **Source §2 pre-declared constants and functions** at `https://source-academy.github.io/source/source_2/global.html`
7. In any question, your answer may use **functions given in, or written by you for,** any **preceding question**. You can assume a correct solution from the preceding question as given, even if your solution from that preceding question was not correct.
8. **Follow the instructions of your invigilator or the module coordinator to submit your answers**.

Section A: List Notation [9 marks]

(1) [3 marks]

What is the result of evaluating the following Source program in *list notation*?

```
const xs = list(6, 7, 8);
pair(xs, tail(xs));
```

A. `list( list(6, 7, 8), 7, 8 )` **(answer)**
B. `list( list(6, 7, 8), list(7, 8) )`
C. `list( list(6, 7, 8), [7, 8] )`
D. `[ [ list(6, 7, 8), 7 ], 8 ]`
E. `[ list(6, 7, 8), list(7, 8) ]`
F. `[ list(6, 7, 8), [7, 8] ]`

(2) [3 marks]

What is the result of evaluating the following Source program in *list notation*?

```
const xs = list(3, 4, 5);
accumulate((x, ys) => append(list(x), list(ys)), null, xs);
```

A. `list(3, 4, 5)`
B. `list(5, 4, 3)`
C. `list(list(list(null, 3), 4), 5)`
D. `list(list(list(null, 5), 4), 3)`
E. `list(3, list(4, list(5, null)))` **(answer)**
F. `list(5, list(4, list(3, null)))`

(3) [3 marks]

What is the result of evaluating the following Source program in *list notation*?

```
function fun(xs) {
    return is_null(xs)
        ? null
        : pair( map(x => head(xs), enum_list(1, head(xs))),
                fun(tail(xs)) );
}
fun( list(2, 4, 3) );
```

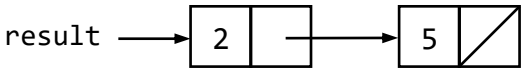
A. `list( list(1, 2), list(1, 2, 3, 4), list(1, 2, 3) )`
B. `list( list(2, 2), list(4, 4, 4, 4), list(3, 3, 3) )` **(answer)**
C. `list( list(1, 2, 3), list(1, 2, 3, 4), list(1, 2) )`
D. `list( list(3, 3, 3), list(4, 4, 4, 4), list(2, 2) )`
E. `[ list(1, 2), [ list(1, 2, 3, 4), list(1, 2, 3) ] ]`
F. `[ list(2, 2), [ list(4, 4, 4, 4), list(3, 3, 3) ] ]`

Section B: Box-and-Pointer Diagrams [8 marks]

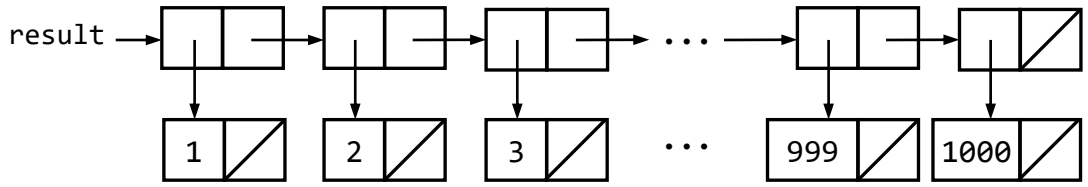
For each of the following box-and-pointer diagrams, write a Source program such that at the end of the evaluation of your program, the name `result` will have the value as shown in the diagram. You must not use any ellipsis (...) in your program.

For example, the following program results in the following diagram on the right:

```
const result = list(2, 5);
```



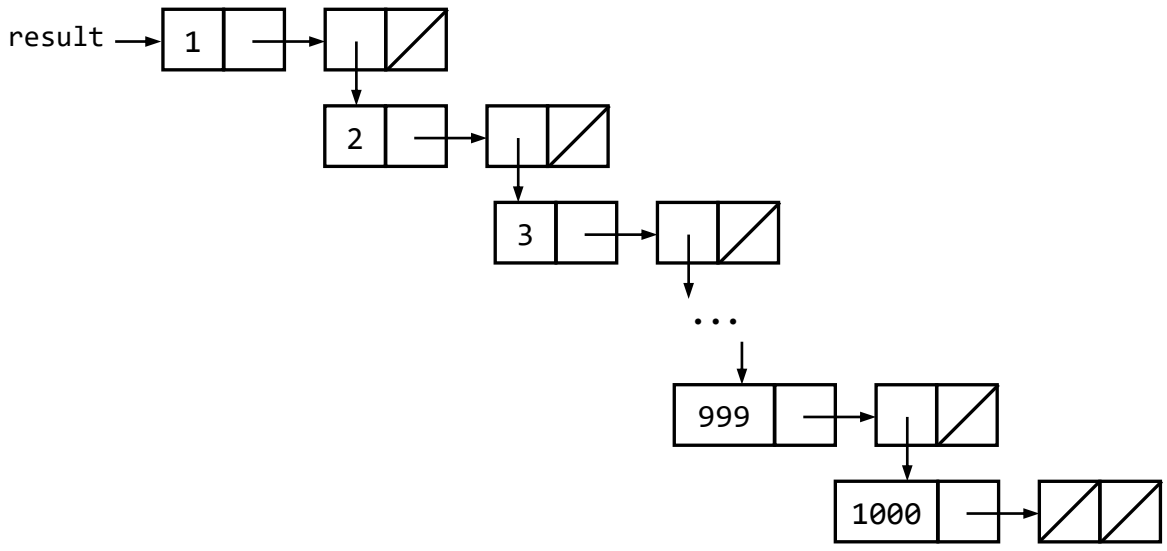
(4) [4 marks]




```
// Solution 1:
const result = build_list(1000, x => list(x + 1));

// Solution 2:
const result = map(x => list(x), enum_list(1, 1000));
```

(5) [4 marks]




```
const result = accumulate((x, y) => list(x, y),
                          null,
                          enum_list(1, 1000));
```

Section C: Orders of Growth [10 marks]

What is the **order of growth** of the **runtime** of each of the following functions in terms of N using the Θ notation? Note that N is a positive integer value.

(6) [2 marks]

```
function fun(N) {
    return N <= 1 ? 1 : fun(2 * N / 3) * (2 * N / 3);
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$ **(answer)**
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

(7) [2 marks]

```
function fun(N) {
    return N <= 1000000 ? N : (N / 1000000) + fun(N - 1000000);
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$ **(answer)**
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

(8) [2 marks]

```
function fun(N) {
    return N <= 1 ? N : fun(N / 2) + fun(N / 2) + 1;
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$ **(answer)**
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

(9) [2 marks]

```
function fun(N) {
    if (N <= 1) {
        return N;
    } else {
        const h = x => x <= 0 ? 0 : x + h(x - 1);
        return h(N) + fun(N - 1);
    }
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$ **(answer)**
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

(10) [2 marks]

```
function fun(N) {
    if (N <= 1) {
        return N;
    } else {
        const x = accumulate((x, y) => x + y, 0, enum_list(1, N));
        return x + fun(N / 2) + fun(N / 2);
    }
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$ **(answer)**
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

Section D: Matrices [17 marks]

A $R \times C$ matrix is represented in Source as a list of R lists of numbers and each list of numbers is of length C. Note that R and C are positive integers. For example, the following 3×4 matrix

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

is represented in Source as

```
list( list(1,  2,  3,  4),
      list(5,  6,  7,  8),
      list(9, 10, 11, 12) )
```

(11) [4 marks]

Complete the function `get_elem` that takes as arguments a matrix `M`, a row number `r`, and a column number `c`, and returns the matrix element at Row `r` and Column `c`. Note that the topmost row in the matrix is Row 0 and the leftmost column is Column 0.

```
function get_elem(M, r, c) {
    return /* your solution */
}
```

Example:

```
const A = list( list(1, 2, 3), list(4, 5, 6), list(7, 8, 9) );
get_elem(A, 0, 2);  // returns 3
get_elem(A, 2, 0);  // returns 7
```

```
function get_elem(M, r, c) {

    return list_ref(list_ref(M, r), c);

}
```

(12) [4 marks]

Complete the function `horizontal_flip` that takes as argument a matrix `M`, and returns a matrix that is the *horizontal flip* of `M`.

```
function horizontal_flip(M) {
    return /* your solution */
}
```

Example:

```
const A = list( list(1,2,3,4), list(5,6,7,8), list(9,10,11,12) );
horizontal_flip(A);
// returns list( list(4,3,2,1), list(8,7,6,5), list(12,11,10,9) )
```

```
function horizontal_flip(M) {

    return map(reverse, M);

}
```

(13) [4 marks]

Complete the function `row_sums` that takes as argument a matrix `M`, and returns a list of numbers containing the sum of elements in each row of `M`.

```
function row_sums(M) {
    return map( /* your solution */ , M );
}
```

Example:

```
const A = list( list(1,2,3,4), list(5,6,7,8), list(9,10,11,12) );
row_sums(A);
// returns list(10, 26, 42)
```

```
function row_sums(M) {

    return map( row => accumulate((x, sum) => x + sum, 0, row) , M );

}
```

(14) [5 marks]

The *transpose* of a $R \times C$ matrix `M` is a $C \times R$ matrix `T` such that

`get_elem(M, r, c) === get_elem(T, c, r)`.

Complete the function `transpose` that takes as argument a matrix `M`, and returns a matrix that is the transpose of `M`.

```
function transpose(M) {
    const nR = length(M);        // number of rows
    const nC = length(head(M));  // number of columns

    // You may not need to use nR and/or nC.

    return map( /* your solution */ , enum_list(0, nC - 1) );

}
```

Example:

```
const A = list( list(1,2,3,4), list(5,6,7,8), list(9,10,11,12) );
transpose(A);
// returns list( list(1,5,9), list(2,6,10), list(3,7,11), list(4,8,12) )
```

```
function transpose(M) {

    const nR = length(M);        // number of rows
    const nC = length(head(M));  // number of columns

    // You may not need to use nR and/or nC.

// Solution 1:
    return map( c => map(row => list_ref(row, c), M) ,
                enum_list(0, nC - 1) );

// Solution 2:
    return map( c => map(r => get_elem(M, r, c), enum_list(0, nR - 1)) ,
                enum_list(0, nC - 1) );

}
```

Section E: Pair-Trees [26 marks]

We introduce the following notion:

A ***pair-tree*** is a number or a pair whose head is a pair-tree and whose tail is a pair-tree.

(15) [2 marks]

Recall that a *tree of numbers* is a list whose elements are numbers or trees of numbers.
Which of the following statements is correct?

A. Every pair-tree is a tree of numbers.
B. No pair-tree is a tree of numbers. **(answer)**
C. Exactly one pair-tree is a tree of numbers.
D. More than one pair-tree, but not all pair-trees, are trees of numbers.
E. A pair-tree has either zero pair or an odd number of pairs.

(16) [2 marks]

Recall that a *tree of numbers* is a list whose elements are numbers or trees of numbers.
Which of the following statements is correct?

A. Every tree of numbers is a pair-tree.
B. No tree of numbers is a pair-tree. **(answer)**
C. Exactly one tree of numbers is a pair-tree.
D. More than one tree of numbers, but not all trees of numbers, are pair-trees.
E. Every tree of numbers is a list of numbers.

(17) [5 marks]

Write the function `has(t, x)` that returns `true` if pair-tree `t` contains the number `x`, otherwise it returns `false`.

```
function has(t, x) {
    /* your solution */
}
```

Examples:

```
const t1 = 8;
has(t1, 4);  // returns false
has(t1, 8);  // returns true

const t2 = pair(pair(1, 2), pair(3, pair(4, 5)));
has(t2, 4);  // returns true
has(t2, 8);  // returns false
```

```
function has(t, x) {

    return is_number(t)
        ? t === x
        : has(head(t), x) || has(tail(t), x);

}
```

(18) [5 marks]

A *path* is a list of functions that are successively applied to a pair-tree to reach a subtree. For example, the path `list(head, tail, tail)` applied to `pair(pair(1, pair(2, 5)), 3)` gives you 5. Write the function `apply(p, t)` that takes a path `p` and a pair-tree `t` as arguments and returns the subtree of `t` that `p` refers to. You can assume that the given path `p` is a valid path within pair-tree `t`.

```
function apply(p, t) {
    /* your solution */
}
```

Examples:

```
const t1 = 8;
apply(null, t1);  // returns 8

const t2 = pair(pair(1, 2), pair(3, pair(4, 5)));
apply( list(tail, tail, head), t2 );  // returns 4
apply( list(head), t2 );  // returns pair(1, 2)
```

```
function apply(p, t) {

    return is_null(p)
        ? t
        : apply( tail(p), (head(p))(t) );

}
```

(19) [6 marks]

Assume that the number 8 (lucky number) appears *exactly once* in a given pair-tree. Write the function `find_8(t)` that takes the pair-tree as an argument and returns the unique path to 8, i.e. `apply(find_8(t), t)` should return 8 if `t` contains exactly one 8.

```
function find_8(t) {
    /* your solution */
}
```

Examples:

```
const t1 = 8;
find_8(t1);  // returns null

const t2 = pair(pair(1, 2), pair(3, pair(8, 5)));
find_8(t2);  // returns list(tail, tail, head)
```

```
function find_8(t) {

    return is_number(t)
        ? null
        : has(head(t), 8)
            ? pair(head, find_8(head(t)))
            : pair(tail, find_8(tail(t)));

}
```

(20) [6 marks]

Write the function `find_all_8(t)` that takes a pair-tree `t` as argument and returns the list of all paths that lead to the number 8. The order of the paths in the result list does not matter. Note that the number 8 might appear any number of times in the pair-tree, including not at all.

```
function find_all_8(t) {
    /* your solution */
}
```

Examples:

```
const t1 = 8;
find_all_8(t1);  // returns list(null)

const t2 = pair(pair(1, 2), pair(3, pair(4, 5)));
find_all_8(t2);  // returns null

const t3 = pair(8, pair(8, pair(8, 2)));
find_all_8(t3);
// returns list( list(head), list(tail, head), list(tail, tail, head) )
```

```
function find_all_8(t) {

    return is_number(t)
        ? (t === 8 ? list(null) : null)
        : append( map(x => pair(head, x), find_all_8(head(t))),
                  map(x => pair(tail, x), find_all_8(tail(t))) );

}
```

——— END OF PAPER ———