NATIONAL UNIVERSITY OF SINGAPORE

CS1101S — PROGRAMMING METHODOLOGY

CURATED VERSION OF 14/11/2020 (CORRECTED ON 16/11/2020)

(Semester 1 AY2014/2015)

Time Allowed: **2 Hours**

INSTRUCTIONS TO STUDENTS

1. This assessment paper contains **FOUR (4)** questions and comprises **FIFTEEN (15)** printed pages, including this page.
2. The full score of this paper is **80 marks**.
3. This is a **CLOSED BOOK** assessment, but you are allowed to use **TWO** double-sided A4 sheets of written or printed notes.
4. Answer **ALL** questions **within the space provided** in this booklet.
5. Where programs are required, write them in the **Source §4** language.
6. Write legibly with a **pen or pencil**. **UNTIDINESS will be penalized**.
7. Do not tear off any pages from this booklet.
8. Write your **Student Number** below **USING A PEN**. Do not write your name.

STUDENT NO.: _____

This portion is for examiner's use only

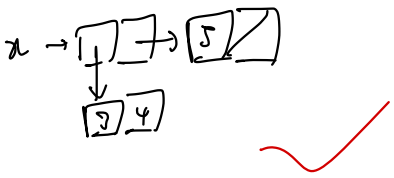
Question	Marks	Question	Marks
Q1 (35 marks)		**Q3 (14 marks)**	
Q2 (20 marks)		**Q4 (11 marks)**	
____	____	**TOTAL (80 marks)**	

Question 1: Miscellaneous [35 marks]

1A. [2 marks]

Draw the box-and-pointer diagram of the value that `x` refers to after the execution of the following statement:

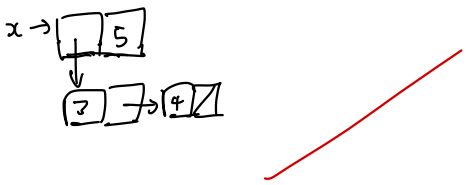
```
let x = list(pair(3, 4), 5);
```



1B. [2 marks]

Draw the box-and-pointer diagram of the value that `x` refers to after the execution of the following statement:

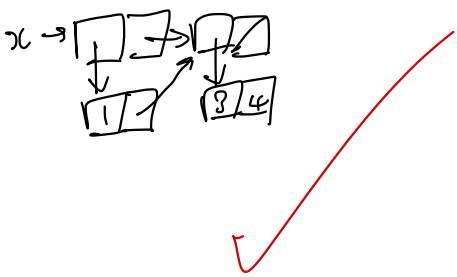
```
let x = pair(list(3, 4), 5);
```



1C. [5 marks]

(i) [3 marks] Draw the box-and-pointer diagram of the value that `x` refers to after the execution of the following statements:

```
let x = list(pair(1, 2), pair(3, 4));
set_tail(head(x), tail(x));
```



(ii) [2 marks] Write a single expression, using `list` and `pair`, that gives `y` a value that is structurally equal to the result of `x` from Part **(i)**.

```
let y =   list( pair(1, list(pair(7,4))), pair(3,4) );


equal(x, y);  // returns true.
```

1D. [6 marks]

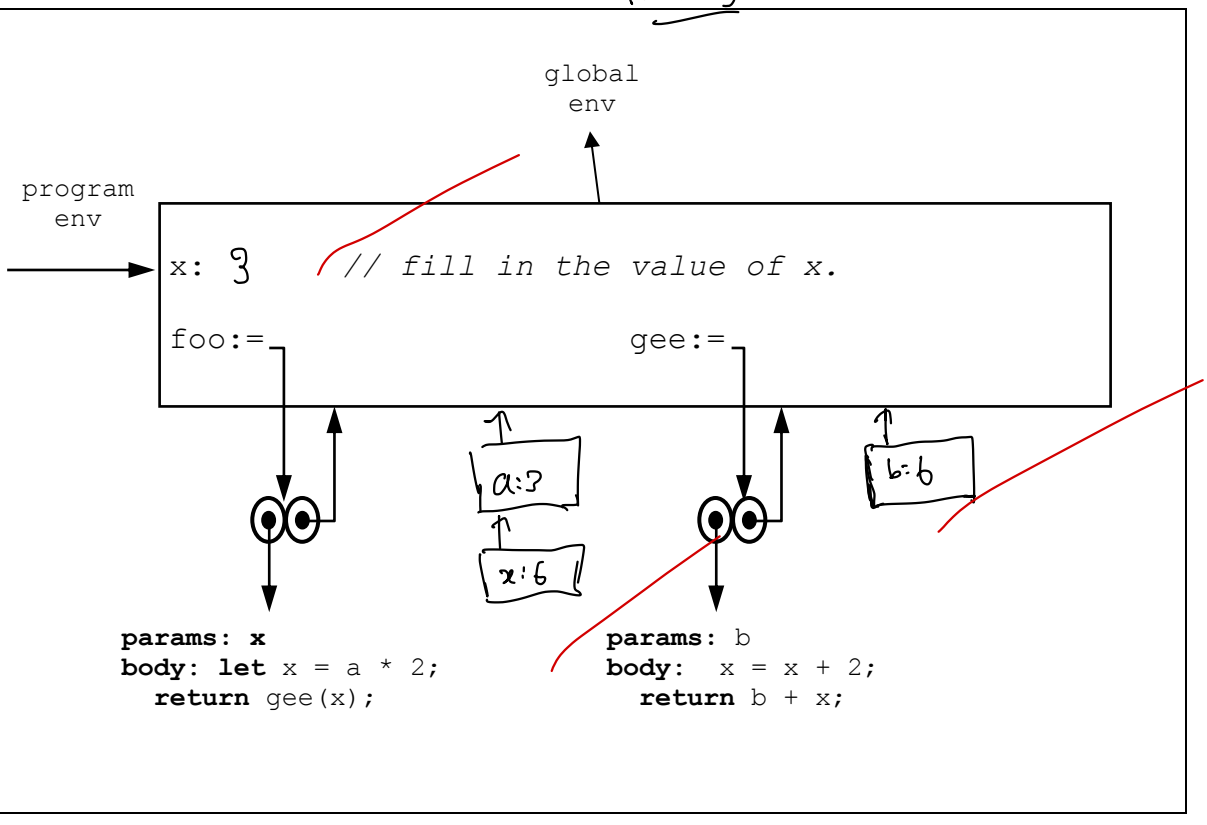
(i) [4 marks] Given the following Source program, complete the following environment model diagram to show all the environments at the point of execution marked ***HERE***.

```
let x = 1;

function foo(a) {
    let x = a * 2;
    return gee(x);
}

function gee(b) {
    x = x + 2;
    // HERE
    return b + x;
}

foo(3);
```



(ii) [2 marks] What is the value of `foo(3)` in the program given in Part **(i)**?

9.

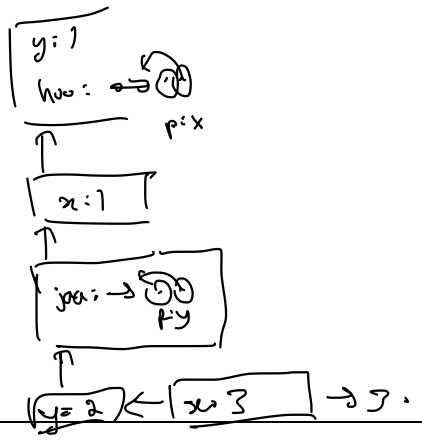
1E. [5 marks]

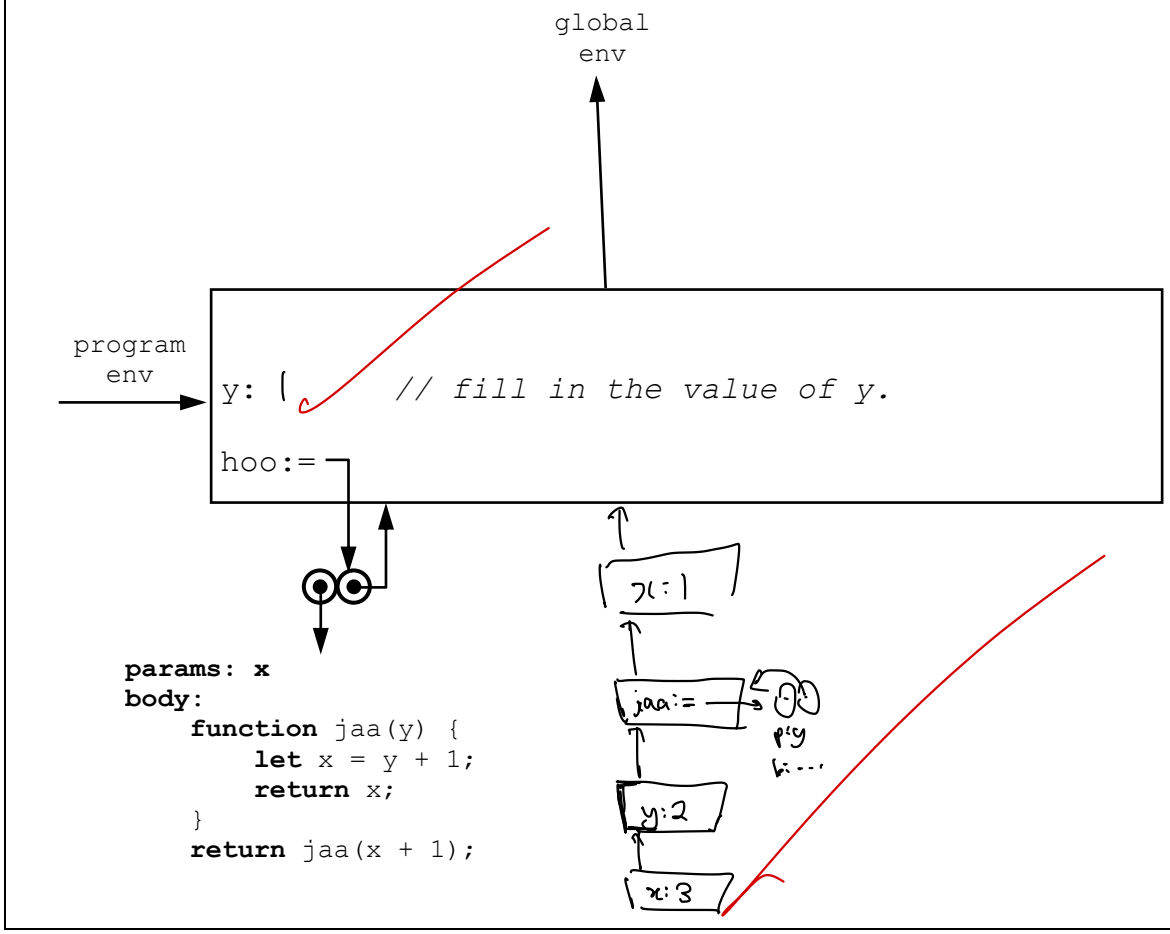
Given the following Source program, complete the following environment model diagram to show all the environments at the point of execution marked ***HERE***.

```
let y = 1;

function hoo(x) {
    function jaa(y) {
        let x = y + 1;
        // HERE
        return x;
    }
    return jaa(x + 1);
}

hoo(1);
```





The following question is not relevant for CS1101S as of 2020/21.

~~1F. [4 marks]~~

~~What is the output of the following program?~~

```
function Test(a) { this.coco = a; }

Test.prototype.func = function() {

    function inner() {
        display("B: " + this.coco);
        this.coco = 678;
        display("C: " + this.coco);
    }

    display("A: " + this.coco);
    inner();
    display("D: " + this.coco);
};

let test = new Test(123);
test.func();
```

1G. [6 marks]

Write a function to return a stream that contains 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, ...

```
function n_of_n_stream() {
    let n = 1
    function helper(counter) {
        if (counter === 0) {
            n = n+1
            return helper(n);
        } else {
            return pair(n, () => helper(counter-1));
        }
    }
    return helper(n);
}
```

1H. [5 marks]

Write a function `table_to_snake_list` that takes a 2D array of numbers as an argument, and visits all the elements in the array in a snake-like manner and returns a list that contains the elements in the visited order. The snake-like visit always starts from the first row, going left-to-right. Then it alternates between right-to-left and left-to-right as it goes from one row to the next.

For example, given that the 2D array, `table`, of height 4 and width 3 is

```
let table = [[ 1,  2,  3],
             [ 4,  5,  6],
             [ 7,  8,  9],
             [10, 11, 12]];
```

`table_to_snake_list(table, 4, 3)` returns a result structurally equal to `list(1, 2, 3, 6, 5, 4, 7, 8, 9, 12, 11, 10)`.

In the following function definition, the parameters `height` and `width` are the height and width of the input 2D array. You can assume they are at least 1.

```
function table_to_snake_list(table, height, width) {
    let xs = null;
    for (let i = 0; i < height; i = i + 1) {
        for (let j = 0; j < width; j = j + 1) {
            if (i % 2 === 0) {
                xs = append(xs, list(table[i][j]));
            } else {
                xs = append(xs, list(table[i][width - j]));
            }
        }
    }
    return xs;
}
```

Question 2: Merging Two Lists [20 marks]

2A. [6 marks]

Given two ordered lists of numbers, each sorted in non-decreasing order, you are to write a function `mergeA` to merge the two lists into one single ordered list. For example, given `list(1, 3, 7, 9)` and `list(2, 3, 5, 6, 11)`, your function should return a list that is structurally equal to `list(1, 2, 3, 3, 5, 6, 7, 9, 11)`. Any of the input lists may be an empty list.

You **must not** use **`for`** loop or **`while`** loop in the function. Your function **must not modify the input lists**, and the **result list must not use any of the existing pairs in the input lists**.

```
function mergeA(xs, ys) {
            function copy(xs) {
                return is_null(xs)
                    ? null
                    : pair(head(xs), copy(tail(xs)));
            }
            if (xs === null) {
                return copy(ys);
            } else if (ys === null) {
                return copy(xs);
            } else {
                let xs_num = head(xs);
                let ys_num = head(ys);
                if (xs_num > ys_num) {
                    return pair(ys_num, mergeA(xs, tail(ys)));
                } else if (ys_num > xs_num) {
                    return pair(xs_num, mergeA(tail(xs), ys));
                } else {
                    return pair(xs_num, pair(ys_num, mergeA(tail(xs), tail(ys))));
                }
            }
}
```

2B. [7 marks]

Similar to Part **A**, you are to write a function `mergeB` to merge two ordered lists into one single ordered list.

You **must not** use **`for`** loop or **`while`** loop in the function. Your function **must not create any new pairs**, and **every pair in the result list must be an existing pair of the input lists**. You should use the function `set_tail` to produce the result list.

```
function mergeB(xs, ys) {
    let N = length(xs) + length(ys);
    if (xs === null) {
        set_tail(xs, ys);
    } else if (ys === null) {
        set_tail(ys, xs);
    } else {
        let xs_num = head(xs);
        let ys_num = head(ys);
        if (xs_num > ys_num) {
            set_tail(ys, mergeA(xs, tail(ys)));
        } else if (ys_num > xs_num) {
            set_tail(xs, mergeA(tail(xs), ys));
        } else {
            set_tail(xs, mergeA(tail(xs), tail(ys)));
            set_tail(ys, mergeA(tail(xs), tail(ys)));
        }
    }
    helper(xs, ys);
    if (length(xs) === N) {
        return xs;
    } else {
        return ys;
    }
}
```

2C. [7 marks]

Similar to Part **A**, you are to write a function `mergeC` to merge two ordered **arrays** of numbers into one single ordered **array**. For example, given array `[1, 3, 7, 9]` and array `[2, 3, 5, 6, 11]`, your function should return an array equivalent to `[1, 2, 3, 3, 5, 6, 7, 9, 11]`. Any of the input arrays may be an empty array.

You **must not use recursion** in the function. Your function **must not modify the input arrays**. The parameters `xs` and `ys` are the input arrays, and `xs_len` and `ys_len` are the length of arrays `xs` and `ys` respectively.

```
function mergeC(xs, xs_len, ys, ys_len) {
    let result = [];
    let result_len = xs_len + ys_len;
    let n = 0;
    let m = 0;
    for (let i = 0; i < result_len; i = i + 1) {
        if (xs[n] === undefined) {
            result[i] = ys[m]; m = m + 1;
        } else if (ys[m] === undefined) {
            result[i] = xs[n]; n = n + 1;
        }
        if (xs[n] < ys[m]) {
            result[i] = xs[n];
        } else if (xs[n] >= ys[m]) {
            result[i] = ys[m];
        } else {
            result[i] = xs[n];
        }
    }
    return result;
}
```

Question 3: Sets and Power Sets [14 marks]

3A. [6 marks]

We want to use a **list of numbers** to represent a **set of numbers**. In the list, there is **no duplicate element**. The **empty set** is represented by an empty list `null`. However, a set of numbers can have multiple list representations, where they differ in the order in which the numbers are listed. For example, `list(6, 3, 5, 8)` and `list(8, 3, 5, 6)` represent the same set.

Write a function `are_equal_sets`, that takes as arguments two sets of numbers, represented as lists as described above, and returns `true` if the two sets are the same, otherwise it returns `false`. You get 6 marks if you use at least one of the functions `filter`, `map` and `accumulate` in a correct and meaningful way, and 4 marks for any other correct solution.

```
function are_equal_sets(set1, set2) {
    return length( filter( x => length(filter( y => x === y, set2 )) === 1,
                           set1 ) === length(set2);
}
```

use cheat sheet

3B. [8 marks]

The **power set** of a set ***S*** is a set of all possible subsets of ***S***. For example, if ***S*** = {3, 5, 6}, then its power set is {{3, 5, 6}, {3, 5}, {3, 6}, {5, 6}, {3}, {5}, {6}, {}}.

As in Part **A**, we use a list of numbers to represent a set of numbers, and use a list of lists of numbers to represent the power set. The empty set {} is represented by an empty list `null`.

Write a function `powerset`, that takes as argument a list of numbers that represents a set of numbers, and returns a list of lists of numbers that represents the power set of the input set. Note that the numbers within each subset list can be in any order, and the subset lists within the power set can be in any order too.

```
function powerset(set) {
}
```

L = ps(tail(xs))

append(L, map(x => pair(head(xs), x), L))

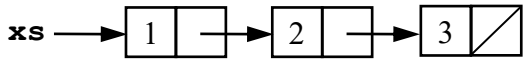
```
function powerset(set) {
    if (set === null) {
        return list(null);
    } else {
        const remain = powerset(tail(set));
        return append(remain, map(x => pair(head(set), x), remain));
    }
}
```

Question 4: Mutable Lists [11 marks]

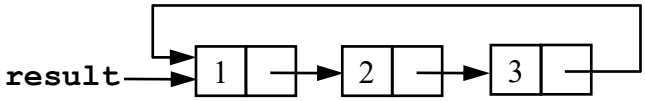
4A. [6 marks]

Write a function `make_circular_copy` that takes a finite list as argument and returns a circular list-like data structure, as depicted in the following example box-and-pointer diagram:

Original:



Copy:



You must make sure that the **original input list is not modified** by your function.

Example:

```
list_ref(make_circular_copy(list(1, 2, 3)), 4);
// should return 2
```

```
function make_circular_copy(xs) {
    function copy(xs) {
        return xs === null
            ? null
            : pair(head(xs), copy(tail(xs)));
    }
    let ys = copy(xs);
    function circular(xs) {
        return tail(xs) === null
            ? set_tail(xs, ys)
            : circular(tail(xs));
    }
    circular(ys);
    return ys;
}
```

4B. [5 marks]

Write a function `make_linear` that takes a circular list-like data structure as argument and changes it to become a list. The function should return `undefined`, but as a side-effect change the data structure as required. Your function **must not create any new pair**.

Example:

```
let ys = make_circular_copy(xs);
make_linear(ys);
equal(xs, ys);  // returns true for any list xs
```

```
function make_linear(xs) {

    function helper(ys){
        return tail(ys) === xs
            ? set_tail(ys, null)
            : helper(tail(ys));
    }

    if(!is_null(xs)){
        helper(xs);
        base case for xs = null?
    }

}
```

4-

(Scratch Paper)

(Scratch Paper)

——— END OF PAPER ———

(Scratch Paper)

——— END OF PAPER ———