NATIONAL UNIVERSITY OF SINGAPORE

CS1101S — PROGRAMMING METHODOLOGY

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

(Semester 1 AY2016/2017)

Time Allowed: **2 Hours**

INSTRUCTIONS TO STUDENTS

1. This assessment paper contains **FIVE (5)** questions and comprises **TWENTY-TWO (22)** 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 Week 10** 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 (31 marks)		**Q4 (10 marks)**	
Q2 (13 marks)		**Q5 (12 marks)**	
Q3 (14 marks)		**TOTAL (80 marks)**	

Question 1: Lists, Trees, Streams [31 marks]

A.1. [4 marks]

Complete the function `last_member(x, xs)`, which is similar to the built-in function `member`, but returns the ***last*** postfix sublist whose head is identical (===) to `x`, and returns `null` if the element does not occur in the list `xs`.

```
function last_member(x, xs) {

    function find_last_member(ys, current_last) {
        let next = member(x, ys);

        // WRITE INSIDE THE BOX

    }

    return find_last_member(xs, null);
}
```

A.2. [1 mark]

Let n be the length of the input list `xs`. Describe the runtime of your function `last_member` with respect to n using Θ notation.

A.3. [1 mark]

Let n be the length of the input list `xs`. Describe the space consumption of your function `last_member` with respect to n using Θ notation.

B. [5 marks]

We represent a ***set*** of numbers using a list of **distinct** numbers **sorted** in ascending order. Complete the function, `is_subset(S, T)`, to determine whether set S is a subset of set T, which is true only if every element in S is also an element of T. We assume that $0 \le N_S$ and $0 \le N_T$, where N_S and N_T are the number of elements in S and T respectively. If $N_S = 0$, then S is always a subset of T. **Your function's runtime should have an order of growth of O(N_T).**

```
function is_subset(S, T) {

    if (is_null(S)) {
        [                                        ]
    } else if (is_null(T)) {
        [                                        ]
    } else if (head(S) < head(T)) {
        [                                        ]
    } else if (head(S) === head(T)) {
        [                                        ]
    } else {
        [                                        ]
    }

}
```

C.

You are given the following function, `mystery(xs)`, that takes a list `xs` as argument and returns a pair:

```
function mystery(xs) {
    if (is_null(xs)) {
        return pair(null, null);
    } else {
        let tmp = mystery(tail(xs));
        let ys = pair(head(xs), tail(tmp));
        let zs = head(tmp);
        return pair(ys, zs);
    }
}
```

C.1. [3 marks]

Given the following application of `mystery`, what would be the values of `head(p)` and `tail(p)`?

```
let p = mystery(list(1, 2, 3, 4, 5, 7, 8));
head(p);  // what is the returned value?
tail(p);  // what is the returned value?
```

Value of `head(p)`:   Value of `tail(p)`:

C.2. [2 marks]

Given the following application of `mystery` and the returned values of `head(p)` and `tail(p)`, what should be the value of `input`?

```
let input = ???;
let p = mystery(input);
head(p);  // returns value equal to list("A", "Q", "R").
tail(p);  // returns value equal to list("T", "U", "P").
```

Value of `input`:

D. [4 marks]

Write a function, `mutable_append(xs,  ys)`, which is similar to the built-in function `append`, but in the result of `mutable_append`, **every pair in the result list is an existing pair of the input lists (i.e. no new pair is created)**. The given lists `xs` and `ys` *can* be destroyed in the process.

```
function mutable_append(xs, ys) {












}
```

E. [6 marks]

Recall from the lectures that a tree of numbers is a list whose elements are numbers or trees of numbers. Write a function, `transform_tree(t)`, that takes a tree of numbers `t` and returns a new tree of numbers `s`, such that `display_tree(s)` displays a sequence of numbers that is the reverse of `display_tree(t)`. The `display_tree` function definition is shown below:

```
function display_tree(tree) {
    if (is_null(tree)) {
        ;
    } else if (is_list(head(tree))) {
        display_tree(head(tree));
        display_tree(tail(tree));
    } else {
        display(head(tree));
        display_tree(tail(tree));
    }
}
```

Example:

```
let tree1 = ...;        // a tree of numbers
let tree2 = transform_tree(tree1);
display_tree(tree1); // displays a sequence of numbers
display_tree(tree2); // displays a reverse of the above
```

Note that the new tree should not be produced by flattening the input tree.

```
function transform_tree(t) {
```

(*more writing space next page*)

```
}
```

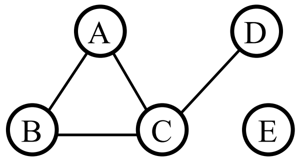
F. [5 marks]

Write a function, `shorten_stream(s, k)`, that takes in a stream `s` and a non-negative integer number `k`, and returns a stream that contains the first `k` elements of `s`. If the length of `s` is less than or equal to `k`, then the result stream will just behave like `s`. Note that `s` may be an infinite stream.

```
function shorten_stream(s, k) {

}
```

Question 2: Networking [13 marks]

A ***network*** consists of ***nodes*** and ***links***, where each node has a name and each link connects one node to another. The diagram shows an example network with 5 nodes and 4 links:



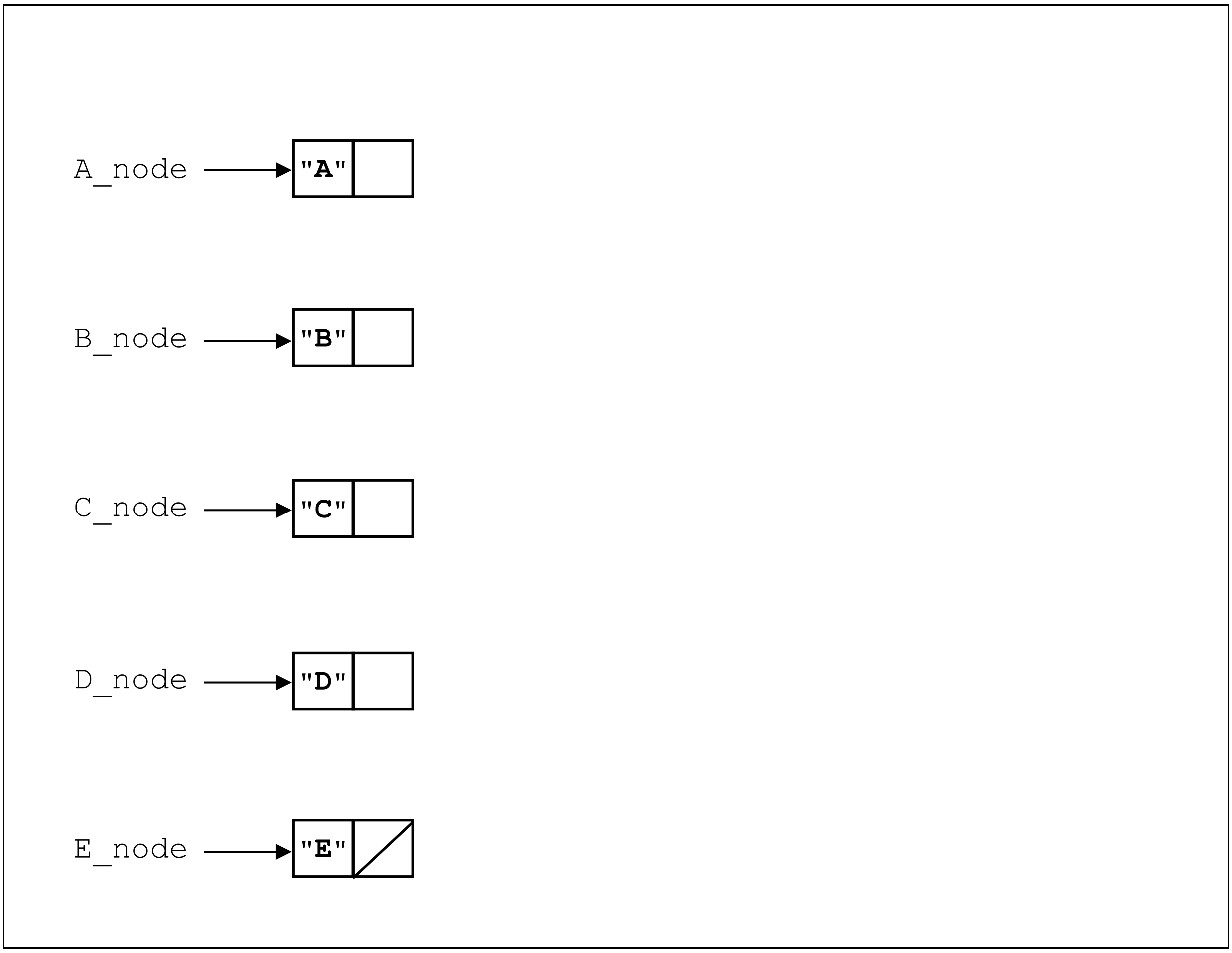
We model a network by representing each node as a list whose first element is the name of the node and the remaining elements are links to other nodes. For example, the following program constructs a representation for the above example network:

```
let A_node = list("A");
let B_node = list("B");
let C_node = list("C");
let D_node = list("D");
let E_node = list("E");
set_tail(A_node, list(B_node, C_node));
set_tail(B_node, list(A_node, C_node));
set_tail(C_node, list(B_node, A_node, D_node));
set_tail(D_node, list(C_node));
```

For any ***proper node*** *x*, if there is a link from *x* to node *y*, then there is a link from node *y* to *x*. Therefore there are altogether 8 links listed in the example.

A. [4 marks]

Complete the following box-and-pointer diagram for the above representation of the example network.



B. [2 marks]
Write a function, `is_linked(x, y)`, that takes in nodes `x` and `y`, and returns true if and only if there is a direct link from node `x` to node `y`.

Examples:
```
is_linked(B_node, C_node);  // returns true
is_linked(B_node, D_node);  // returns false
is_linked(C_node, E_node);  // returns false
```

```
function is_linked(x, y) {

}
```

C. [3 marks]
Recall that for any proper node *x*, if there is a link from *x* to node *y*, then there is a link from node *y* to *x*. Write a function `is_proper(x)` that takes a node `x` as argument and returns true if the node `x` is proper and false otherwise.

```
function is_proper(x) {

}
```

D. [4 marks]

In this question, you can assume that all nodes are proper. Write a function, `is_connected(x, y)`, that takes in nodes `x` and `y`, and returns true if and only if there exists a chain of one or more links that connects node `x` to node `y`. Note that `is_connected(x, x)` is always true.

Examples:

```
is_connected(B_node, C_node);  // returns true
is_connected(A_node, D_node);  // returns true
is_connected(A_node, E_node);  // returns false
is_connected(E_node, E_node);  // returns true
```

(Hint: beware of cycles in the network.)

```
function is_connected(x, y) {
```

Question 3: Continuation Passing Style [14 marks]

In 1975, Gerald Sussman (one of the coauthors of our textbook) and Guy Steele Jr invented a style of programming that they called "*Continuation Passing Style*" (CPS). The CPS style follows a simple rule: Functions should never return a value other than `undefined`. Instead, functions have an additional argument, the *continuation function* that is applied to the result.

This style of programming is useful in many situations, and is used in some web applications.

Example 1:

Consider the function

```
function plus(x, y) {
    return x + y;
}
```

In CPS, the function works as follows:

```
function plus_cps(x, y, ret) {
    return ret(x + y);
}
```

In order to display the result of the addition of 1 and 2, we can use `plus_cps` as follows:

```
plus_cps(1, 2, display);  // displays the value 3
```

Example 2:

```
function length(xs) {
    if (is_null(xs)) {
        return 0;
    } else {
        return 1 + length(tail(xs));
    }
}
```

Here is the length function in CPS:

```
function length_cps(xs, ret) {
    if (is_null(xs)) {
        return ret(0);
    } else {
        return length_cps(tail(xs),
                     tail_result => ret(1 + tail_result));
    }
}
length_cps(list(10, 20, 30), display); // displays value 3
```

A. [4 marks]

Make use of the functions `plus_cps` in order to compute the sum of three given numbers. Your function `sum_cps` needs to be written in CPS.

```
function sum_cps(x, y, z, ret) {











}
sum_cps(1, 2, 3, display);  // displays the value 6
```

B. [4 marks]

Recall the factorial function:

```
function factorial(n) {
    if (n <= 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Write the factorial function in CPS.

```
function factorial_cps(n, ret) {
















}
factorial_cps(5, display);  // displays the value 120
```

C. [6 marks]

Consider the iterative version of the factorial function:

```
function fact_iter(n, acc) {
    if (n <= 0) {
        return acc;
    } else {
        return fact_iter(n - 1, n * acc);
    }
}
function factorial_iter(n) {
    return fact_iter(n, 1);
}
factorial_iter(5);  // returns 120
```

(i) [4 marks] Write an iterative factorial function in CPS style.

```
function fact_iter_cps(n, acc, ret) {

}
function factorial_iter_cps(n, ret) {
    return fact_iter_cps(n, 1, ret);
}
factorial_iter_cps(5, display);  // displays 120
```

(ii) [2 marks] From your solution, can you characterize iterative functions with respect to CPS? Compare the original continuation function with the continuation function that is passed to the recursive call.

When we turn iterative functions into CPS, the continuation function…

Question 4: Hot Program Reload [10 marks]

A. [6 marks]

The programming environment for Source allows the programmer to re-define functions interactively. For example, consider this session:

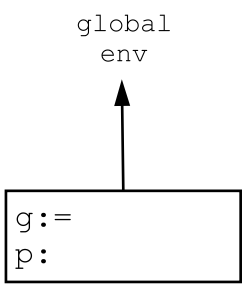
```
< function f(x) { return x + 1; }
< f(3);
> 4

< function f(x) { return x * 2; }
< f(3);
> 6
```

However, such re-definition does not eliminate the old function completely. Consider this program as an example:

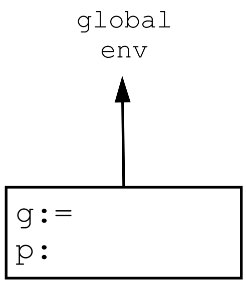
```
< function g(x) { return x + 1; }
< let p = pair(g, null);              // 2nd statement
< (head(p))(3);
> 4

< function g(x) { return x * 2; }     // 4th statement
< (head(p))(3);
> 4
```

Briefly explain why the result in the last line is 4 and not 6. Use the environment model in your explanation and illustrate your written answer using diagrams that you draw.

Show environment after the second statement:



Show environment after the 4th statement:



B. [4 marks]

Programming languages such as Erlang were designed for applications that run continuously for a long time, for example telephone switch systems. Such languages support changing function values *permanently*, while programs run, in order to enhance their functionality and fix bugs. The language designers need to make sure that all references to the function value lead to the new implementation.

Recall that the meta-circular interpreter for the Source language represents compound functions by tagged lists whose `body` element contains the parse tree of the respective function.

```
function make_compound_function(parameters, body, env) {
    return list("compound_function",
                parameters, body, env);
}
function function_body(f) {
    return list_ref(f, 2);
}
```

We would like to add a primitive function `hot_reload` that takes two compound functions as arguments. The function `hot_reload` replaces the body of the first function by the body of the second function, without affecting the parameters and the environment of the function.

Example:

```
function h(x) { return x + 1; }
let p = pair(h, null);
display(head(p))(3);                            // displays 4
hot_reload(h, x => x * 2);
display(head(p))(3);                            // displays 6
```

You can assume that the primitive function `hot_reload` gets as arguments two compound functions (as described above) that have the same number of parameters.

```
function hot_reload(cf1, cf2) {

}
```

The following question is not relevant for CS1101S as of 2019/20.

~~Question 5: Smart Rooms [12 marks]~~

~~Object-oriented programming allows us to naturally capture the behavior of the entities that a computation deals with, including their interactions and relationships. As an example, consider the task of programming a building maintenance system. Among other functions, the system specifies particular actions to be taken when people enter and leave a smart room.~~

```
function Smart_Room(aircon) {
    this.aircon = aircon;
}
```

~~The argument `aircon` of the constructor of the `Smart_Room` class is an `Appliance` object, which has the methods `turn_on` and `turn_off`.~~

~~Smart rooms are equipped with sensors that detect when the first user enters the room and when the last user leaves the room.~~

```
Smart_Room.prototype.first_user_enters = function() {
    this.aircon.turn_on();
};
```

~~Note that the method `turn_on` of `Appliance` objects makes sure the `Appliance` is on, regardless whether it was off or on already.~~

```
Smart_Room.prototype.last_user_leaves = function() {
    this.aircon.turn_off();
};
```

~~Note that the method `turn_off` of `Appliance` objects makes sure the `Appliance` is off, regardless whether it was on or off already.~~

~~A. [5 marks]~~

~~In this question, you can assume a given function `is_daylight_present()` without arguments. This function has access to an outdoor light sensor and returns `true` if the sensor detects enough daylight and `false` otherwise.~~

~~Write a class `Smart_Room_With_Light` whose instances behave like `Smart_Room` objects, with the following exceptions:~~

(1) ~~The constructor has an additional argument `light`, which is an `Appliance`.~~

(2) ~~After the first person enters the room, the room's `light` should be on, if not enough daylight is detected.~~

(3) ~~After the last person leaves the room, the room's `light` should be off.~~

~~In your solution, make best use of the existing class `Smart_Room`.~~

```
function Smart_Room_With_Light(aircon, light) {
    // WRITE HERE








}

Smart_Room_With_Light.Inherits(Smart_Room);


Smart_Room_With_Light.prototype.first_user_enters =
function() {
    // WRITE HERE
















};


Smart_Room_With_Light.prototype.last_user_leaves =
function() {
    // WRITE HERE













};
```

~~B. [7 marks]~~

~~In this question, you should make use of a `Timer` class. Whenever a `Timer` object is created with a given number of seconds `t` and a function `f` as arguments, the function `f` is called with no arguments after `t` seconds.~~

~~Example:~~

```
var my_timer =
        new Timer(10, function() { alert("belated hello"); });
// 10 seconds after this program is evaluated,
// an alert window will pop up with the message "belated hello"
```

~~You need to write a program to describe the behavior of a bathroom that is equipped with ventilation. Instances of your `Smart_Bathroom` class should behave like `Smart_Room_With_Light` objects, except:~~

1. ~~The constructor has an additional argument `ventilation`, which is an `Appliance`.~~
2. ~~After the first person enters the room, the room's `ventilation` should be on.~~
3. ~~Ten seconds after the last person leaves the room, the room's `ventilation` should turn off, if no one is inside the room **at that time**.~~

~~In your solution, make best use of the class `Smart_Room_With_Light` described in Part A of this question.~~

```
function Smart_Bathroom(aircon, light, ventilation) {
    // WRITE HERE





    this.person_inside = false;
}

Smart_Bathroom.Inherits(Smart_Room_With_Light);


Smart_Bathroom.prototype.first_user_enters =
function() {
    // WRITE HERE
```

~~(*more writing space next page*)~~

```
~~};~~

~~Smart_Bathroom.prototype.last_user_leaves =~~
~~function() {~~
~~    // WRITE HERE~~

~~};~~
```

(Scratch paper. Do not tear off.)

(Scratch paper. Do not tear off.)

END OF PAPER