JavaScript

1. pair(a, b): makes a pair from a and b
2. head(c): extracts the value of the first part of the pair c
3. tail(c): extracts the value of the second part of the pair c
4. list(a, b, c, ...): builds a list of the arguments to the function
5. length(list): returns the number of elements in list
6. list_ref(lst, n): returns the nth element of lst
7. append(list1, list2): returns a new list consisting of the elements of the first list followed by the elements of the second list. The new list is made from new pairs for the first argument; the second argument (which need not actually be a list) is merely placed at the end of the new structure.
8. reverse(lst): returns new list containing the elements of lst in reverse order

Problems:

1. Draw the box-and-pointer diagram for the values of the following expressions. Also give the representation that the JediScript Console uses.

   (a) pair(1, 2) => [1, 2]

   ![Diagram for pair(1, 2)]

   1 2

(b) pair(1, pair(3, pair(5, []))) => [1, [3, [5, []]]]

   ![Diagram for pair(1, pair(3, pair(5, [])))]

   1 3 5
2. Write JediScript Week 5 expressions that do not use the array syntax [...], whose values will print out like the following.

\[
\begin{align*}
[1, [2, [3, [[]]]]] \Rightarrow & \text{list(1,2,3)} \\
[1, [2, 3]] \Rightarrow & \text{pair(1,pair(2,3))} \\
[[[1, [2, [[]]]]], [[3, [4, [[]]]]], [[5, [6, [[]]]], [[]]]] \Rightarrow & \text{list(list(1,2),list(3,4),list(5,6))}
\end{align*}
\]

3. Write expressions using head and tail that will return 4 when the lst is bound to the following values:

(a) \text{list(7,6,5,4,3,2,1)}

\textbf{Answer:}

\text{head(tail(tail(tail(lst))))}

(b) \text{list(list(7),list(6,5,4),list(3,2),1)}

\textbf{Answer:}

\text{head(tail(tail(head(tail(lst)))))}
(c) \text{list}(7,\
\text{list}(6,\
\text{list}(5,\
\text{list}(4,\
\text{list}(3,\
\text{list}(2,\
\text{list}(1)))))))

\textbf{Answer:}

\text{head(head(tail(head(tail(head(tail(lst))))))})
(d) list(7, 
    list(list(list(6,5, 
        list(list(4)), 
      3), 
    2), 
  ), 
1) 

Answer:

head(head(head(tail(tail(head(head(tail(lst))))))))

Note: The key idea in this question is that you have to understand how to translate an expression into a box and pointer diagram and to systematically traverse the box and pointer structure.

4. You found a holiday assignment at the Registrar's Office. Your job is to write a program to help students with their scheduling of classes. You are provided with an implementation of the records for each class as follows:

```javascript
function make_class(number, units) {
    return list(number, units);
}

var get_class_number = head;
function get_class_units(cl) {
    return head(tail(cl));
}

function make_units(lecture, tutorial, lab, homework, prep) {
    return list(lecture, tutorial, lab, homework, prep);
}

var get_units_lecture = head;
function get_units_tutorial(units) {
    return head(tail(units));
}

function get_units_lab(units) {
    return head(tail(tail(units)));
}

function get_units_homework(units) {
    return head(tail(tail(tail(units))));
}

function get_units_prep(units) {
    return head(tail(tail(tail(tail(units)))));
}

function get_class_total_units(cl) {
    var units = get_class_units(cl);
    return get_units_lecture(units) +
         get_units_tutorial(units) +
         get_units_lab(units) +
         get_units_homework(units) +
         get_units_prep(units);
}

function is_same_class(c1, c2) {
    return get_class_number(c1) ===
         get_class_number(c2);
}
```
Each class has a course code and an associated number of credit units, e.g., for CS1101S, that’s 3-2-1-3-3. Your job is now to write a schedule object to represent the sets of classes taken by a student.

(a) Write a constructor that returns an empty schedule.

Answer:

```javascript
function empty_schedule() { return []; }
```

It does not make sense to talk about an order of “growth” for this function because there are no arguments that may grow in size.

(b) Write a function that when given a class and a schedule, returns a new schedule including the new class.

Answer:

```javascript
function add_class(class, schedule) {
    return pair(class, schedule);
}
```

Order of growth in time: $O(1)$, space: $O(1)$.

(c) Write a function that computes the total number of units in a schedule.

Answer:

```javascript
function total_scheduled_units(sched) {
    if(is_empty_list(sched)) {
        return 0;
    } else {
        return get_class_total_units(head(sched)) +
               total_scheduled_units(tail(sched));
    }
}
```

Order of growth in time: $O(n)$, space: $O(n)$.

(d) Write a function that drops a particular class from a schedule.

Answer:

```javascript
function drop_class(sched, class) {
    if(is_empty_list(sched)) {
        return sched;
    } else {
        if (get_class_number(head(sched)) !==
            get_class_number(class)) {
            return drop_class(tail(sched), class);
        } else {
            return pair(head(sched),
                    drop_class(tail(sched),
                              class));
        }
    }
}
```

Order of growth in time: $O(n)$, space: $O(n)$.

(e) Implement a credit limit by taking in a schedule, and removing classes until the total number of units is less than max_credits.

Answer:
function credit_limit(sched, max_credits) {
    if (total_scheduled_units(sched) > max_credits) {
        return credit_limit(tail(sched), max_credits);
    } else {
        return sched;
    }
}

Order of growth in time: $O(n^2)$, space: $O(n)$.

The following is an alternate solution:

function credit_limit(sched, max_credits) {
    function helper(s, e) {
        if (e <= 0) {
            return s;
        } else {
            return helper(tail(s),
            e - get_class_total_units(head(s)));
        }
    }
    var total = total_scheduled_units(sched);
    var extra = total - max_credits;
    return helper(sched, extra);
}

Order of growth in time: $O(n)$, space: $O(n)$.

(f) **Homework 1:** Implement `total_scheduled_units` using higher-order functions `accumulate` and `map`.

(g) **Homework 2:** Implement an improved version of `credit_limit` that will return a schedule with a total number of units is less than `max_credits`, but with the maximal number of classes. What is the order of growth of your solution? Is that the best you can do?