National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2021/2022

R5
List and Tree Processing

Lists

A *list of a certain data type* is `null` or a pair whose head is of that data type and whose tail is a list of that data type.

Consider a list of numbers:

```
const my_list = pair(1, pair(2, pair(3, null))); // shorter: list(1, 2, 3);
```

Trees

A *tree of a certain data type* is a list whose elements are of that data type, or trees of that data type.

Consider a tree of numbers:

```
const tree = list(5, list(1, 2), 6, list(3, 4), 7);
```

Restriction: As “data type”, `null` and pairs are not allowed for trees.

Problems:

1. Consider lists of lists of numbers. Here is an example:

   ```
   const my_matrix = list(list(1, 2, 3), list(4, 5, 6), list(7, 8, 9));
   ```

   Write a function `flatten_list` **using the function `accumulate`**, that “flattens” the given list of lists of numbers: It returns a list of numbers that contains the same numbers in the same order as the given list of list of numbers. Example:

   ```
   const my_list = flatten_list(my_matrix);
   // returns: list(1, 2, 3, 4, 5, 6, 7, 8, 9);
   ```

   ```
   function flatten_list(lst) {
       // your answer here
       return accumulate ((x,y) => append(x,y) null, lst);

   }
   flatten_list(list(list(1, 2, 3), list(4, 5, 6), list(7, 8, 9)));
   // Returns: list(1, 2, 3, 4, 5, 6, 7, 8, 9);
   ```

National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2021/2022

R5
List and Tree Processing

Lists

A *list of a certain data type* is `null` or a pair whose head is of that data type and whose tail is a list of that data type.

Consider a list of numbers:

```
const my_list = pair(1, pair(2, pair(3, null))); // shorter: list(1, 2, 3);
```

Trees

A *tree of a certain data type* is a list whose elements are of that data type, or trees of that data type.

Consider a tree of numbers:

```
const tree = list(5, list(1, 2), 6, list(3, 4), 7);
```

Restriction: As “data type”, `null` and pairs are not allowed for trees.

Problems:

1. Consider lists of lists of numbers. Here is an example:

```
const my_matrix = list(list(1, 2, 3), list(4, 5, 6), list(7, 8, 9));
```

Write a function `flatten_list` **using the function `accumulate`**, that “flattens” the given list of lists of numbers: It returns a list of numbers that contains the same numbers in the same order as the given list of list of numbers. Example:

```
const my_list = flatten_list(my_matrix);
// returns: list(1, 2, 3, 4, 5, 6, 7, 8, 9);

function flatten_list(lst) {
    // your answer here
    return accumulate ((x,y) => append(x,y) null, lst);

}
flatten_list(list(list(1, 2, 3), list(4, 5, 6), list(7, 8, 9)));
// Returns: list(1, 2, 3, 4, 5, 6, 7, 8, 9);
```

Rhythm := null ✓ |

head is num → list<num>

else → | list<Rhythm>

| pair<Rhythm, num>

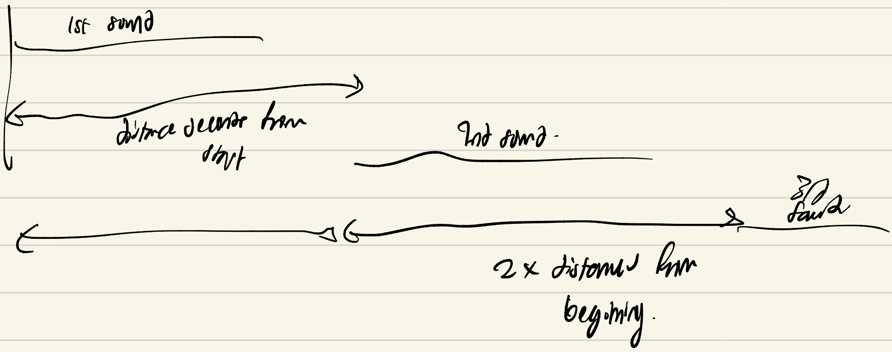
has a tail

is tail a number

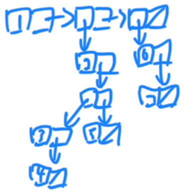
list(list(1,2,3), list(1,2,3))

list(list([1,2], 2,3), list([list(1,2), 2], 2,3))

percussion :



2. Write a function `tree_sum` that takes in a tree of numbers and returns the sum of all the numbers in the tree.



```
function tree_sum(tree) {
    // your answer here
    return is_null(tree)
           ? 0
           : is_number(tree)
             ? head(tree) + tree_sum(tail(tree))
             : tree_sum(head(tree)) + tree_sum(tail(tree));
}

const my_tree = list(1, list(2, list(3, 4), 5), list(6, 7));
tree_sum(my_tree);
// Returns: 28
```

3. Write a function `accumulate_tree` that takes four arguments: a function `f1`, a function `f2`, an `initial` value and a `tree`. The function `f1` takes one argument and is used to get the value for each data item in the tree. The second function takes two arguments and is used to combine the values of two sub-trees.

For example the following programs should compute the `tree_sum` above and the function `count_data_items` from Lecture L5, respectively.

```
function tree_sum(tree) {
    return accumulate_tree( x => x, (x, y) => x + y, 0, tree);
}
```

The following function computes the number of data items in a given tree (`count_data_items` as given in Lecture L5).

```
function count_data_items(tree) {
    return accumulate_tree( x => 1, (x, y) => x + y, 0, tree);
}
```

The following function flattens a given tree into a list.

```
function flatten(tree) {
    return accumulate_tree( x => list(x), append, null, tree);
}
```

Give sufficient conditions for `f2` and `initial` such that the result does not depend on the shape of the tree or the order in which the elements appear?

```
function accumulate_tree(~~tree~~ f1, f2, initial, tree.) {
    // your answer here
    return is_null(tree)
           ? initial
           : is_number(head(tree))
             ? f2(f1(head(tree)), accumulate_tree(f1, f2, initial, tail(tree)))
             : f2(accumulate_tree(f1, f2, initial, head(tree)),
                    accumulate_tree(f1, f2, initial, tail(tree)));
}
```