LISTS

```
function flatten_once(xs){
    return accumulate(append, null, xs);
}

function permutations(ys) {
    return is_null(ys)
        ? list(null)
        : accumulate(append, null,
            map(x => map(p => pair(x, p),
                         permutations(remove(x, ys))),
                ys));
}

function product(xs, ys){
    return is_null(xs) ? null
                       : append(map(x => pair(head(xs),x), ys),
                                product(tail(xs), ys));
}

function combinations(xs, r){
    if ((r!== 0 && is_null(xs)) || r < 0){
        return null;
    } else if (r===0){
        return list(null);
    } else {
        const case1 = combinations(tail(xs), r);
        const case2 = combinations(tail(xs), r-1);
        const case3 = map(x => pair(head(xs), x), case2);
        return append(case1, case3);
    }
}

function list_to_array(lst){
    let out = [];
    let index = 0;
    while (!is_null(lst)){
        out[index] = head(lst);
        index = index + 1;
        lst = tail(lst);
    }
    return out;
}

function remove_duplicates(lst){
    return is_null(lst) ? null
                       : pair(head(lst),
                            remove_duplicates(filter(x => !equal(x, head(lst)),
                                                     tail(lst))));
}

function primes(n){
    let lst = enum_list(2, n);
    function divisible(x){
        const ls = enum_list(1, x);
        if (length(filter(y => x % y === 0, ls)) === 2){
            return false;
        } else {
            return true;
        }
    }
    return filter(x => !divisible(x), lst);
}
```

```
function take(xs, n){
    return n === 0 ? null
                   : pair(head(xs), take(tail(xs), n-1));
}

function drop(xs, n){
    return n === 0 ? xs
                   : drop(tail(xs), n - 1);
}

Insertion_sort
```

STREAMS

```
// Stream that generates “a”, “aa”, “aaa”...
function letter_gen(letter, cur){
    return pair(cur + letter,
                () => letter_gen(letter, cur + letter));
}
const a = letter_gen("a", "");

// Stream that generates 1, f(1), f(2), f(3)...
function f_stream(f) {
    function h(num) {
        return pair(f(num), () => h(num + 1));
    }
    return pair(1, () => h(1));
}

// Stream that generates 1, ff(1), fff(1), ffff(1)...
function twice_stream(f) {
    function h(cur) {
        const x = f(cur);
        return pair(x, () => h(x));
    }
    return pair(1, () => h(1));
}

// Take the every kth element of a stream
function kth_stream(stream, k) {
    function helper(stream, count) {
        if (count === 1) {
            return pair(head(stream), () => helper(stream_tail(stream), k));
        } else {
            return helper(stream_tail(stream), count - 1);
        }
    }
    return helper(stream, k);
}



function stream_to_array(stream, n){
    let result = [];
    function helper(counter, stream){
        if (counter === n || is_null(stream)){
            return result;
        } else {
            result[counter] = head(stream);
            return helper(counter + 1, stream_tail(stream));
        }
    }
    return helper(0, stream);
}
```

```
function add_streams(s1, s2){
    return pair(head(s1) + head(s2),
                () => add_streams(stream_tail(s1), stream_tail(s2)));
}

function array_to_stream(array){
    function helper(array, counter){
        if (counter >= array_length(array)){
            return null;
        } else {
            return pair(array[counter], () => helper(array, counter+1));
        }
    }
    return helper(array, 0);
}
```

ARRAYS

```
function copy_array(M){
    let out = [];
    for (let i = 0; i < array_length(M); i = i + 1){
        out[i] = M[i];
    }
    return out;
}

function array_reverse(M) {
    const out = copy_array(M);
    const len = array_length(out);
    for (let i = 0; i < len/2; i = i + 1) {
        const temp = out[i];
        out[i] = out[len - i - 1];
        out[len - i - 1] = temp;
    }
    return out;
}

function mat_reverse(mat){
    let out = [];
    for (let i = 0; i < array_length(mat); i = i + 1){
        out[i] = array_reverse(mat[i]);
    }
    return out;
}
[[1, 2], [3, 4]] -> [[2, 1], [4, 3]]



function transpose(M){
    let out = [];
    for (let i = 0; i < array_length(M[0]); i = i + 1){
        out[i] = [];
        for (let j = 0; j < array_length(M); j = j + 1){
            out[i][j] = M[j][i];
        }
    }
    return out;
}

// rotates matrix 90 degrees anticlockwise
function rotate_matrix(M){
    return array_reverse(transpose(M));
}
```

```
function array_to_list(arr){
    let output = null;
    for (let i = array_length(arr) -1; i >=0; i = i -1){
        output = pair(arr[i], output);
    }
    return output;
}

function array_filter(pred, arr){
    const output = [];
    for (let i = 0; i < array_length(arr); i = i + 1){
        if (pred(arr[i])){
            output[array_length(output)] = arr[i];
        } else {}
    }
    return output;
}
function sort_ascending(A) {
    const len = array_length(A);
    for (let i = 1; i < len; i = i + 1) {
        const x = A[i];
        let j = i - 1;
        while (j >= 0 && A[j] > x) {
            A[j + 1] = A[j];
            j = j - 1;
        }
        A[j + 1] = x;
    }
}
```

OTHER FUNCTIONS

```
//"mem" is declared in global scope
function read(n, k){
    return (mem[n] === undefined ?
        undefined : mem[n][k];
}

//"mem" is declared in global scope
function write(n, k, value) {
    if (mem[n] === undefined) {
        mem[n] = [];
    } else { }
    mem[n][k] = value;
}

function contains(x, xs){
    return !is_null(member(x, xs));
}
```