Javascript Lab Seminar

MELL-JAVASCRIPT-02

Day 02

Tree

v1.61

Day 02

Tree

repository name: javascript_lab

branch name: day_02

Your repository must contain the totality of your source files.

You must have one file per Task. my_compute_factorial_it is the task one so you need to have a file named `my_compute_factorial_it.js`.

You are only allow to use **var let const if while** and **for**
Do not use any function of any kind that is not your. If you need to use concact() method then create it.

Please note that none of your files must contain a index.js function, unless specified otherwise. We will use our own main functions to compile and test your code.

You only have 4 tasks today… enjoy :)

Task 01

simple list

The purpose of this exercise is to create a set of functions that will let you manipulate a list. We will consider a list as the following:

```
class linkedListNode {

  constructor(data) {

    this.data = data;

    this.next = null;

  }

}
```

Implement the following functions:

int list_get_size**(LinkedListNode** list**)**

Returns the number of elements in the list.

list_dump**(LinkedListNode** list**)**

Displays every element in the list, separated by new-line

bool list_add_elem_at_front(**LinkedListNode** list , **double** elem)

Adds a new node at the beginning of the list with elem as its value.

bool list_add_elem_at_back(**LinkedListNode** list , **double** elem)

Adds a new node at the end of the list with elem as its value.

bool list_add_elem_at_position(**LinkedListNode** list , **double** elem, **int** position)

Adds a new node at the position position with elem as its value. If the value of position is 0, a call to this function is equivalent to a call to list_add_elem_at_front.

bool list_del_elem_at_front**(LinkedListNode** list**);**

Deletes the first node of the list. Returns FALSE if the list is empty, TRUE otherwise.

bool list_del_elem_at_back**(LinkedListNode** list**);**

Deletes the last node of the list. Returns FALSE if the list is empty, TRUE otherwise.

bool list_del_elem_at_position**(LinkedListNode** list, **int** position**);**

Deletes the node at the position position. If the value of position is 0, a call to this function is equivalent to a call to list_del_elem_at_front. Returns FALSE if the list is empty or if position is out of bounds, TRUE otherwise.

double list_get_elem_at_front**(LinkedListNode** list**);**

Return the value of the first node in the list. Return 0 if the list is empty

double list_get_elem_at_back**(LinkedListNode** list**);**

Return the value of the last node in the list. Return 0 if the list is empty

double list_get_elem_at_position**(LinkedListNode** list, **int** position**);**

Return the value of the node at the position. Return 0 if the list is empty or if the position is out of bounds.

Task 02

simple stack

The purpose of this exercise is to create a stack based on the previously created generic list.

```
class Stack extends LinkedListNode {


}
```

Implement the following functions:

bool stack_push**(Stack** stack**)**

Pushes elem to the top of the stack. Returns FALSE if the new element could not be pushed, TRUE otherwise.

bool stack_pop**(Stack** stack**)**

Pops the top element off the stack. Returns FALSE if the stack is empty, TRUE otherwise.

Task 03

simple tree

The purpose of this exercise is to create a set of functions that will let you manipulate a binary tree. We will consider a binary tree as the following:

```
class  Node {

  constructor(data) {

    this.data = data;

    this.left = null;

    this.right = null;

  }

}
```

Implement the following functions:

bool tree_is_empty**(Node** tree**)**

Return TRUE if the tree is empty. FALSE otherwise

int tree_get_size**(Node** tree**)**

Return the number of nodes in tree

int tree_get_deph**(Node** tree**)**

Return the deph of the tree

bool tree_create_node**(Node** node, **double** value**)**

Creates a new node with `value` *as its value and places it at the right. if a right node already exist, then create it at the left*

Returns `FALSE` *if the node could not be added,* `TRUE` *otherwise.*

double tree_get_max_value**(Node** tree**)**

Return the maximal value in tree. Return 0 if the tree is empty

double tree_get_min_value**(Node** tree**)**

Return the minimal value in tree. Return 0 if the tree is empty

tree_infix**(Node** tree**)**

display all value with an infix expression

tree_postfix**(Node** tree**)**

display all value with an postfix expression

tree_prefix**(Node** tree**)**

display all value with an prefix expression

Binary Tree. Postfix ? Prefix ? Infix ?
https://en.wikipedia.org/wiki/Binary_expression_tree#Infix_traversal

Task 04

Order

The purpose of this exercise is to create a function that will order and display a given String.

order**(string** str**)**

exemple*: order(“1435267”)*

output *1234567*

order_desc**(string** str**)**

exemple*: order(“1435267”)*

output *7654321*

You have to use your binary tree !