# Parallel Programing
# Assignment 3

deadline: 17.3.2024 23:59

## Submission Guidelines

To reduce likelihood of misunderstandings, please follow these guidelines:

1. Work can either be done individually or in pairs.
2. Submission is through the moodle (lamda).
3. Make sure that your solution compiles and runs without any errors and warnings on BIU servers.
4. In the first line of every file you submit, write in a comment your id and full name. For example: “/* 123456789 Israela Israeli */”.
5. Not using openMP in the task will result in an automatic 0, **sequential code is unacceptable.**

## Bonuses

You may add a question suggestion for HW for next semester. The question should be interesting to solve. There is no need to solve the question, but it needs to be reasonably solvable for the homework #3 of the course. Top 3 ideas will get a point in the final grade of course.

## Environment & Submission

For this assignment, you'll need to submit your code in a file named "src.zip". This zip file should contain the files "linkedList.h" and "linkedList.c"

## The Assignment

Our third and final exercise for the semester will be implementing a small library that provides the user an integer linked-list data structure, with operations that can be used safely in parallel.

You are given the file "linkedList.h", and it's your job to implement the functions whose signature is in the file. You may choose your own implementation of the node_s structure, as long as the library functions work the way they are expected.

### node_t *init_list(void);

This function creates a linked list of size 1, where the value inside the single node is INT_MIN.

### void sorted_insert(node_t *head, int val);

Given the address of a linked list created using init_list, this function creates a node with the value "val", and inserts it into the linked list in a sorted manner.

## void remove_val(node_t *head, int val);

Given the address of a linked list created using init_list, this function searches for a node with the value “val”, and removes it from the linked list. If a node with the given value doesn’t exist, the function doesn’t change the state of the list.

## int find_val(node_t *head, int val);

Given the address of a linked list, this function searches for a node with the value “val”. If it exists, the function returns “1”, otherwise - “0”.

## int get_len(node_t *head);

Given the address of a linked list, this function returns its length.

## void free_list(node_t *head);

Given the address of a linked list, this function frees all the memory and destroys all locks used by the list.

**Notes:**

1. You may assume that every function in this library will only be used in parallel with other calls to <u>the same function</u>. I.e: two different functions in the library will not be called in parallel.
2. Init_list and free_list will only be called once each for each list created - when first creating the list, and at the end when freeing the resources it uses.