Cairo University
Faculty of Engineering
Computer Department

Expected Outcomes:

This lab is expected to help you in understanding three concepts of functional programming:

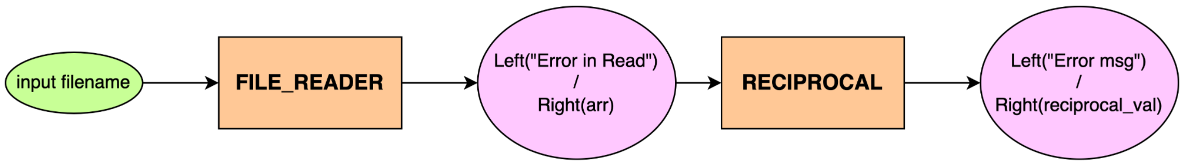
- Functors.
- Pure Error Handling.
- Asynchronous Tasks.

We will deal with these three concepts throughout the lab tasks.

Description:

We need to implement a pipeline function that reads a set of numbers from a file and converts it to an array. Then, it gets the reciprocal of the first entry of the array.

- Pure Error Handling should be applied throughout the pipeline i.e. Either class.


input filename
FILE_READER
Left("Error in Read") / Right(arr)
RECIPROCAL
Left("Error msg") / Right(reciprocal_val)

We can divide our main pipeline into two successive components running sequentially, File Reader and Reciprocal:

1. The File Reader should receive an input filename and output Either object which will be the input for the next stage.
   - In case of success, the Either object should be a Right container having the array as a value.
   - In case of failure, the Either object should be a Left Container having an indicative error message as a value.
2. The Reciprocal should receive an Either object containing the array, or an indicative error message of the last step, And it should also output an Either object as a final result of running the whole pipeline.
   - In case of accepting a previous error from the past step, it should propagate to the output of the pipeline.
   - In case of failure of current step, the Either object should be a Left Container having an indicative error message as a value.
   - In case of success, the Either object should be a Right container having the reciprocal of the first array entry as a value.

Tasks:

- Explore the requirement structure.
- Read utils.js carefully and make sure to use the utility functions whenever you need.
- There is no need to define or implement any functions other than mentioned in tasks.
- Again any function or component that has an error as an input it should propagate that error as its output, hint: use either function in utils.js.

1. File Reader Component(get_file_numbers):
    - It is composed of two sequential steps(functions): read_file, convert_to_array.
    - Implement read_file, it should take a filename as input and return a Task(Error or string in the input file) which is an asynchronous task.
    - Implement convert_to_array, it should take a string and return an array of numbers. Hint: we will need to apply lifting on calling it within the main component.
    - Implement the component function get_file_numbers that takes a string and returns a Task(Error | [Number]).
    - Test each function, and the whole component with different test case scenarios.
2. Reciprocal Component(reciprocal_of_first_array_entry):
    - it is composed of two sequential steps(functions): get_first_array_entry, reciprocal.
    - Implement get_first_array_entry, it should take an array. it should handle errors of non array types and empty arrays, and return the first array entry contained in a Right object in the success case scenario, otherwise an indicative error contained in Left Container.
    - Implement reciprocal, it should take a numeric value. it should handle errors of non numeric types and divide by zero, and return the reciprocal contained in a Right object in the success case scenario, otherwise an indicative error contained in Left Container.
    - Implement the component function reciprocal_of_first_array_entry that takes an Either(Error | [Number]) and returns an Either(Error | Number). An error entering the component or happening throughout the component should be propagated without applying the rest of component functions. But we need to keep component purity and maintain consistent program flow in all cases.
    - Test each function, and the whole component with different test case scenarios.
3. Main Pipeline:
    - The main functionality of the pipeline is that it takes the output of the first component Task(Error | [Number]), runs the asynchronous task, contains the output in Either container and hands it over to the second component.
    - Read the main pipeline.
    - Write down some test cases testing it with different scenarios.