UNIVERSIDADE ABERTA
www.uab.pt

Ministry of Education and Science

e-Folio

U.C. 21077

Programming Languages e-Folio A - OCaml Language

-- INSTRUCTIONS –

1) The e-folio has a value of 4.

2) Any attempt at plagiarism will result in a final mark of zero.

3) This e-folio should be solved using the *OCaml language*.

4) A compressed file (ZIP or RAR) with the student's name and number must be submitted:

a) Programme code;

b) Readme.txt file with the information needed to compile and run the programme;

c) Report in *.pdf* format of up to 4 pages describing the solution presented and the tests carried out

E-folio A

In this work, it is proposed that each student implements a **stack structure** using OCaml that simulates a Logistics exercise. A stack is a *last-in*, *first- out* (LIFO) data structure that allows you to stack elements at the top of the stack and unstack elements from the top of the stack.

The application context is as follows: you have just been hired by a logistics company that needs to control the packages being loaded onto a lorry. The packages are loaded onto the lorry in a specific order and unloaded in the reverse order. To keep track of the packages, an employee decided to implement a module in OCaml that used a stack data structure, however this employee went on holiday and left his implementation incomplete, so the task of continuation was assigned to you. Items are loaded onto the lorry whenever an item is loaded the id of the item is recorded. Ids are positive integers.

The code created by your colleague is attached in the *logi.ml* file.

1. Implementing the Stack

In this part of the task, you will be implementing the ***Stack*** module using the module type already created by your colleague who has gone on holiday. With the following functions:

empty : creates an empty stack.

is_empty : returns ***true*** if the stack is empty, ***false*** otherwise.

push : adds an element to the top of the stack.

peek : returns the element from the top of the stack without removing it.

pop : removes the element from the top of the stack and returns it.

size : returns the number of elements in the stack.

His colleague informed him that the implementation was almost finished and only needed a single line.

2. Implementing the Stack Tester

In this part of the task, you will be implementing the ***StackTester*** module. The Stack Tester module provides functions for building and testing stacks. You must provide the following functions:

build : receives a build parameter and returns a stack built from the instructions in the build parameter.

test : receives a stack and a check parameter and returns ***true*** if the stack fulfils the conditions in the check parameter, ***false*** otherwise.

3. Implementing Stack Printing Functionality

In this part of the task, you will be implementing the print function in the ***StackTester*** module.

print : The print function receives a function that converts an element into a string and a stack and prints the elements of the stack.

4. Testing your implementation

In this part of the assignment, you will be testing your implementation of the Stack module (***STACK***) and the Stack ***Tester*** module (***StackTester***). You must provide at least 3 tests of your own and carry out the following 2 tests:

- The lorry arrived at the logistics centre and loaded the IDs in the following order 5,6,8, it made two stops during the day where it unloaded an item, what is the ID of the item that remained on the lorry?
- The lorry arrived at the logistics centre and loaded the IDs in the following order 2,1,5,9,10, at the first stop it unloaded two items and loaded an item with ID 8, at the next stop it unloaded 2 items, at the next stop it loaded an item with ID 3. At the end of the day which items will be left on the lorry?

Your colleague has left an email to direct you on how to take the test:

##############

Dear colleague,

You need to implement your own stack module (e.g. MyStack) to test the *StackTester* module. This is because the *StackTester module* receives a *Stack* module as an argument, and the purpose of the tests is to ensure that the StackTester module works correctly with different stack implementations.

Look at this structure:

module MyStack : STACK = struct

(we implement the module according to the STACK, basically instantiating the variables*) End*

Now we've created a module that uses our stack structure.

module MyStackTester = StackTester(MyStack)

This way we can test an operation, for example putting in IDs 1, 2 and 3 and then removing two of the Ids:

let test = Push (1, Push (2, Push (3, Pop (Pop Empty))))

let result = MyStackTester.build test

let () = MyStackTester.print.....

Well done

Notes:

1. (C3) All choices must be substantiated in the report.
2. (C1) How to implement the various modules proposed.
3. (C2) The way to present a pile is up to each student.
4. (C2) The user-friendliness of the programme is valued (e.g. access menus and other complex structures and terms).