



OCaml Pool - d03

Development Environment

42 pedago pedago@staff.42.fr
kashim vbazenne@student.42.fr

Abstract: This is the subject for d03 of the OCaml piscine. The main theme of this day is to introduce the basic development environment of OCaml and the many ways to build a project. It also introduce the notion of tree in OCaml, because trees are good for you.

Contents

I	Ocaml piscine, general rules	2
II	Day-specific rules	4
III	Foreword	5
IV	Exercise 00: Chucalescu - Stupalacci	6
V	Exercise 01: Basic Gardening	7
VI	Exercise 02: All Good Ciphers Go To Heaven	8
VII	Exercise 03: Gardening Research	10
VIII	Exercise 04: Adelson-Velsky-Landis	12

Chapter I

Ocaml piscine, general rules

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names, etc.
- Unless otherwise explicitly stated, the keywords `open`, `for` and `while` are forbidden. Their use will be flagged as cheating, no questions asked.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- Since you are allowed to use the `OCaml` syntaxes you learned about since the beginning of the piscine, you are not allowed to use any additional syntaxes, modules and libraries unless explicitly stated otherwise.
- The exercices must be done in order. The graduation will stop at the first failed exercise. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `ocamlpt`. When you are required to turn in a function, you must also include anything necessary to compile a full executable. That executable should display some tests that prove that you've done the exercise correctly.
- Remember that the special token `";;"` is only used to end an expression in the interpreter. Thus, it must never appear in any file you turn in. Regardless, the interpreter is a powerfull ally, learn to use it at its best as soon as possible!
- The subject can be modified up to 4 hours before the final turn-in time.
- In case you're wondering, no coding style is enforced during the `OCaml` piscine. You can use any style you like, no restrictions. But remember that a code your peer-

evaluator can't read is a code he or she can't grade. As usual, big functions are a weak style.

- You will NOT be graded by a program, unless explicitly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.
- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

Chapter II

Day-specific rules

- For each exercise of the day, you must provide sufficient material for testing during the defence session. **Every functionality that can't be tested won't be graded!**
- During the whole day, we will be using a type tree defined as follows:

```
type 'a tree = Nil | Node of 'a * 'a tree * 'a tree
```

A basic node can be created like this: `Node (42, Nil, Nil)`.

- During the whole day, you are allowed to use any method you prefer to handle the Makefile. (Makefile, OCamlMakefile, ocamlbuild, ...)

Chapter III

Foreword

Level 42 are an english pop-rock and jazz-funk band well known in the 80's and 90's. The band became famous for his bass player and singer Mark King whose percussive slap technique was known as one of the most impressive bass technique of all time. The most successfull single are:

- Lessons in Love
- Love Games
- The Chinese Way
- The Sun Goes Down (Living It Up)


The name of the band is taken from the book The Hitchhiker's Guide to the Galaxy by Douglas Adams. You can view the band in action by clicking [here](#).

Here is a subjectiv list of non-exhaustiv very good bass players :

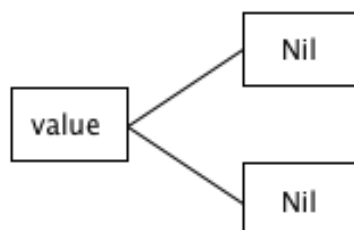
- [Paul McCartney \(The Beatles\)](#)
- [John Entwistle \(The Who\)](#)
- [John Paul Jones \(Led Zeppelin\)](#)
- [Jaco Pastorius \(Weather Report\)](#)
- [John Deacon \(Queen\)](#)
- [Prince](#)
- [Stuart Hamm \(Joe Satriani\)](#)
- No, not Flea from the Red Hot Chili Peppers!
- And many others...

Chapter IV

Exercise 00: Chucalescu - Stupalacci


	Exercise 00
Exercise 00: Chucalescu - Stupalacci	
Turn-in directory : <i>ex00/</i>	
Files to turn in : <code>ft_graphics.ml</code> , <code>Makefile</code>	
Allowed functions : <code>open_graph</code> , <code>lineto</code> , <code>moveto</code> , <code>draw_string</code> functions of the Graphics module and Pervasives module	
Remarks : n/a	

- Write a function `draw_square` that takes three ints `x`, `y`, `size` as parameters and draw a square centered in (`x`, `y`) on a graphical window. You'll have to use the Graphics module, of which the only allowed functions are `lineto`, `moveto` and the `draw_string` functions.
- Write a function `draw_tree_node` that takes a basic tree node `Node(v, Nil, Nil)` or `Nil` as a parameter and draw it according to this example (if you prefer to draw it from up to down it's ok):also, you don't have to handle if the printed value goes out of the square



Chapter V

Exercise 01: Basic Gardening

	Exercise 01
Exercise 01: Basic Gardening	
Turn-in directory : <i>ex01/</i>	
Files to turn in : gardening.ml , Makefile	
Allowed functions : Graphics module and Pervasives module	
Remarks : n/a	

The root of a binary tree is the top node. A leaf of a binary tree is a node with no sub-tree. The size of a binary tree is the number of nodes that are defined. The height of a binary tree is the number of relations on the longest downward path between the root and a leaf.

- Write a function **size** that takes a tree as a parameter and returns its size. The function must be typed as :

```
val size : 'a tree -> int
```

- Write a function **height** that takes a tree as a parameter and returns its height. The function must be typed as :


```
val height : 'a tree -> int
```

- Write a function **draw_tree** that takes a tree as a parameter and draws it using the graphics module. We assume that the size of the window is sufficient to draw it, so you don't have to handle it. We'll consider only trees of type **string tree** for this function, so you can draw the values without conversion problems. The function must be typed as :

```
val draw_tree : string tree -> unit
```


Chapter VI

Exercise 02: All Good Ciphers Go To Heaven

	Exercise 02
Exercise 02: All Good Ciphers Go To Heaven	
Turn-in directory : <i>ex02/</i>	
Files to turn in : <code>cipher.ml</code> , <code>uncipher.ml</code> , <code>Makefile</code>	
Allowed functions : <code>Pervasives</code> module and the <code>String.map</code> function	
Remarks : n/a	

This Exercise is what people usually called fun, a kind of distraction between two trees. The main purpose of this exercise is to introduce to you the concept of very very basic ciphering. Of course, needless to say that the **Cryptokit** module of OCaml is strictly forbidden for the entire exercise.


- Write a function and its opposite `rot42` and `unrot42` that takes a string as parameter and returns the string by 42-rotating all of its char.
- Write a function and its opposite `caesar` and `uncaesar` that takes a string and an int as parameters and returns the string by rotating all of its char to the right according to the int (just like a `rotn` actually).
- Write a function `xor` that takes a string and an int `key` as parameters and returns the string by xor-ing all of its char with the `key`. This function is its own opposite.
- Write a function and its opposite `ft_crypt` and `ft_uncrypt` that takes a string and a list of functions caesar-like or xor-like as parameters and returns the new string after the application of the functions. Of course, wrap everything in a program to prove that your work works.

```
val ft_crypt : string -> (string -> string) list -> string
val ft_uncrypt : string -> (string -> string) list -> string
```

- Every cipher functions must go in the file "`cipher.ml`" and every uncipher functions must go in the file "`uncipher.ml`".

Chapter VII

Exercise 03: Gardening Research

	Exercise 03
Exercise 03: Gardening Research	
Turn-in directory : <i>ex03/</i>	
Files to turn in : btree.ml	
Allowed functions : Pervasives module and List.hd	
Remarks : n/a	

Ok, enough fun, back to the trees. A binary search tree (BST) is a binary tree where the nodes are sorted in a way that for every node, all the nodes of the left subtree of the root are inferior to the current node and all the nodes of the right subtree of the root are superior to the current node. A perfect btree is a tree where all the leaves have 0 or 2 Sons and all the leaves are at equal distance of the root.


- Write a function **is_bst** that takes a tree as a parameter and returns a boolean. True if the tree is an bst, false otherwise.
- Write a function **is_perfect** that takes a tree as a parameter and returns a boolean. True if the tree is perfect BST, false otherwise.
- Write a function **is_balanced** that takes a tree as a parameter and returns a boolean. True if the tree is a height-balanced BST, false otherwise.
- Write a function **search_bst** that takes a value and a BST as a parameter and returns a boolean. True if the the value is in the tree, false otherwise.
- Write a function **add_bst** that takes a value and a BST as a parameter and returns BST with the new value inserted.
- Write a function **delete_bst** that takes a BST as a parameter and a value and returns a BST with the value deleted.



You may encounter a case in which one of your functions cannot return a value because it wouldn't make sense. In that case, instead of returning a value, you must use the following expression : `failwith "your error message here"`, with your custom error message obviously.

Chapter VIII

Exercise 04: Adelson-Velsky-Landis

	Exercise 04
Exercise 04: Adelson-Velsky-Landis	
Turn-in directory : <i>ex04/</i>	
Files to turn in : <i>avl.ml</i>	
Allowed functions : <i>Pervasives</i> module	
Remarks : <i>n/a</i>	

An AVL(**Georgy Adelson-Velsky and Landis**) tree is a Self-Balancing Binary Search Tree (BST). In an AVL tree, the heights of the two child subtrees of any node differ by at most one. If at any time they differ by more than one, rebalancing is done to restore this property.

- Write a function `insert_avl` that takes a value and an AVL tree as parameters. The function insert the value in the tree, check if the tree is still balanced and rebalance it if necessary by performing one or more rotation. The function returns the newly created AVL.
- Write a function `delete_avl` that takes a value and an AVL tree as parameters. The function delete the value in the tree, check if the tree is still balanced and rebalance it if necessary by performing one or more rotation. The function returns the newly created AVL.



You may encounter a case in which one of your functions cannot return a value because it wouldn't make sense. In that case, instead of returning a value, you must use the following expression : `failwith "your error message here"`, with your custom error message obviously.