Remember that the quality of the defenses, hence the quality of the of the school on the labor market depends on you. The remote defences during the Covid crisis allows more flexibility so you can progress into your curriculum, but also brings more risks of cheat, injustice, laziness, that will harm everyone's skills development. We do count on your maturity and wisdom during these remote defenses for the benefits of the entire community.

SCALE FOR PROJECT PISCINE OCAML / DAY 06

You should evaluate 1 student in this team

Git repository

Introduction

For the good of this evaluation, we ask you to:

- Stay mannerly, polite, respectful and constructive dunring this evaluation. The trust between you and the 42 community depends on it.

- Bring out to the graded student (or team) any mistake she or he might did.

- Accept that there might be differences of interpretation of the subject or the rules between you and the graded student (or team). Stay open minded and grade as honnestly as possible.

Guidelines

- You must grade only what is present and the graded student's (or team) repository.

- You must stop grading at the first failed exercice, but you are encouraged to continue testing and discussing the following exercices.

Attachments

subject.pdf

Preliminaries

This section is dedicated to setup the evaluation and to test the prerequisits. It doesn't rewards points, but if something is wrong at this

step or at any point of the evaluation, the grade is 0, and an approriate flag might be checked if needed.

Respect of the rules

- The graded student (or team) work is present on her or his repository.

- The graded student (or team) is able to explain her or his work at any time of the evaluation.

- The general rules and the possible day-specific rules are respected at any time of the evaluation.

Yes | No

OCaml piscine D06

- For each exercice, you must compile the exercice using ocamlopt and run the generated executable. If the compilation fails or warns, or an unexpected exception is thrown at runtime, the exercice is failed. - Whether the graded student provided tests or not, you must test her or his work extensivelly and asses if the work is done or not. - Remember to check function names, types, behaviours and outputs.

Ex00, the Set module and the Set.Make functor

This exercice is so straight forward that I won't give any solution here. Check that the graded student actually used the Set.Make functor to create a StringSet module from the String module and that the output is correct.

$> ocamlopt ex00.ml && ./a.out bar baz foo qux quxfoobazbar $>

Yes | No

Ex01, the Hashtbl module and the Hashtbl.Make functor

This exercice is pretty much the same as the previous one, so no answer here neither. The only trick is to write a hash function. Check the following.

- The graded student crated an input module compatible with the signature Hashtbl.HashedType. Note that implementing that signature would require constraint sharing and is out of the scope of this exercice, but count the points anyway.

- The hash function is actually a hash function. Be tolerant, but refuse really dummy hash functions. It was written in red in the subject.

- The output is respected, modulo the order of the lines.

$> ocamlopt ex01.ml && ./a.out k = "Ocaml", v = 5 k = "Hello", v = 5 k = "42", v = 2 k = "H", v = 1 k = "world", v = 5 $>

Yes | No

Ex02, projections

This exercice is the exact opposit of the exemple in the videos.
The FIXME must be replaced by the following code.

module type MAKEPROJECTION = functor (Pair : PAIR) -> VAL

module MakeFst : MAKEPROJECTION = functor (Pair : PAIR) ->
struct let x = fst Pair.pair end

module MakeSnd : MAKEPROJECTION = functor (Pair : PAIR) ->
struct let x = snd Pair.pair end

Check the output.

$> ocamlopt ex02.ml && ./a.out Fst.x = 21, Snd.x = 42 $>

Yes | No

Ex03, fixed point

First real world exemple, the fixed number functor is a functor
I use in my personnal OCaml codes. Given the FIXED signature
from the subject and the test code below, one can deduce the
input signature FRACTIONNAL_BITS.

module type FRACTIONAL_BITS = sig val bits : int end

And the MAKE functor signature.

module type MAKE = functor (Fbits : FRACTIONAL_BITS) -> FIXED

The implementation of the functor Make is pretty straight
forward, the only tricky functions beeing of_float and to_float
so I won't give answers here. Check that the output from the
test code is good.

$> ocamlopt ex03.ml && ./a.out 42.421875 0. 0.0625 0.125 0.1875 0.25 0.3125 0.375 0.4375 0.5 0.5625 0.625 0.6875 0.75 0.8125 0.875 0.9375 1. $>

According to the subject, the graded student must also provide some addionnal test to prove that every functions in the signature FIXED work as intended. check that this work is done.

Yes | No

Ex04, evalexpr is so easy it hurts

This exercice is very representative of OCaml. Writing an evalexpr in C or C++ is at best painful. With OCaml, it's easy and elegant.

Given the signature VAL and the description of the EVALEXPR signature, this signature must be similar to the following.

module type EVALEXPR = sig

type t

type expr = | Add of (expr * expr) | Mul of (expr * expr) | Value of t

val eval : expr -> t

end

The types VAL.t and EVALEXPR.t beeing abstract, the MAKEEVALEXPR functor signature must include a constraint sharing on those types.

module type MAKEEVALEXPR = functor (Val:VAL) -> EVALEXPR with type t = Val.t * OR * module type MAKEEVALEXPR = functor (Val:VAL) -> EVALEXPR with type t := Val.t

I won't give out the implementation of the functor MakeEvalExpr as it is a very good programming exercice. Don't forget to check that this functor actually implements the MAKEEVALEXPR signature, obviously.

The six addtionnal constraint sharing from the last part of the exercice are the following.

module IntVal : (VAL with type t = int) = <...> module FloatVal : (VAL with type t = float) = <...> module StringVal : (VAL with type t = string) = <...> module IntEvalExpr : (EVALEXPR with type t = IntVal.t) = <...> module FloatEvalExpr : (EVALEXPR with type t = FloatVal.t) = <...> module StringEvalExpr : (EVALEXPR with type t = StringVal.t) = <...>

The four destructive substitution are the following.

module type MAKEEVALEXPR = functor (Val:VAL) -> EVALEXPR with type t := Val.t module IntEvalExpr : (EVALEXPR with type t := IntVal.t) = <...> module FloatEvalExpr : (EVALEXPR with type t := FloatVal.t) = <...> module StringEvalExpr : (EVALEXPR with type t := StringVal.t) = <...>

Also, check that the output is good.

$> ocamlopt ex04.ml && ./a.out Res = 42 Res = 42.420000 Res = very very long $>

Yes | No

Ratings

Don't forget to check the flag corresponding to the defense

Ok

Empty work | No author file | Invalid compilation | Norme | Cheat

Crash | Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation