exercise05-solution-p

November 17, 2018

1 Assignment 5.1 (P): Getting Started

Have a quick look at the following OCaml tools.

1.1 REPLs (Read Eval Print Loops)

1.1.1 ocaml

Simply type your OCaml code. If you want the REPL to execute your code, type `;;`. Execute the content of a .ml file with `#use "<filename>"`. To exit the REPL enter `#quit` or type `Ctrl + D`. Note that `#use`, `quit` as well as `;;` are special sequences recognized by the REPL and are not part of OCaml (so do not use them in a .ml file).

1.1.2 utop

utop works in the same way as ocaml, but is more comfortable as it comes with autocompletion, for example, and allows you to change or redo previously typed code much more easily.

1.2 IDEs

1.2.1 Visual Studio Code

Visual Studio Code is a free full-fledged integrated development environment for a lot of programming languages, due to its great extensibility and rich set of existing extensions. The OCaml plugin together with merlin and ocp-indent provides great utilities

- syntax highlighting
- autocompletion
- smart identation
- peek and goto definition
- type information
- immediate error reports
- integrated terminal
- and much more ...

1.3 Compilers

1.3.1 ocamlc

ocamlc is a basic ocaml compiler. When invoked as `ocamlc <file-0> ... <file-n>` it compiles all files into an executable named `a.out`. During this process, `.cmi` and `.cmo` files are generated, representing the compiled interface and object files respectively. Check the documentation (http://caml.inria.fr/pub/docs/manual-ocaml-4.00/manual022.html#toc86) for a list of all command line options and more advanced build steps.

1.3.2 ocamlbuild

ocamlbuild is a more convenient compilation utility, as it resolves and compiles dependencies automatically. In order to build your project with ocamlbuild, simply compile your main file `<file>.ml` with either `ocamlbuild <file>.d.byte` or `ocambuild <file>.native` to produce bytecode (with debug information) or native code, respectively. Check the documentation (https://ocaml.org/learn/tutorials/ocamlbuild/) for more details.

2 Assignment 5.2 (P) Expressions

So far, you learned about the following types of expressions: * Constants * Variables * Unary operators * Binary operators * Tuples * Records * Lists * If-then-else * Pattern matching * Function definition * Function application * Variable binding

Perform the following tasks: 1. For each of these expressions, give the general structure and two concrete examples with different subexpressions.

- Constants: c e.g. `7`, `true`
- Variables: v e.g. `a`, `my_function`
- Unary operators: $op\ e$ e.g. `-x`, `not true`
- Binary operators: $e_0\ op\ e_1$ e.g. `3 + a`, `x = -y`
- Tuples: $e_0, \ldots, e_n$, e.g. `2, x, false, 2 + 4, a, my_function`
- Records (definition): $\{attr_0 = e_0; \ldots attr_n = e_n\}$, e.g. `{ width = 7; height = x - 2 }` (note that a record type with these attributes must be declared)
- Records (access): $e_0.e_1$ e.g. `size.width`, `{ age = 12 }.age`
- Lists: $e_0 :: e_1$ (resp. $[e_0, \ldots, e_n]$, e.g. `2::[]`, `[1,2;3,4;5,6]`
- if-then-else: if e_0 then e_1 else e_2, e.g. `if x > 0 then 2 else -4`, `if a = b then 1.0, 'a' else -0.5, 'b'`
- Pattern matching: match e with $pat_0 \rightarrow e_0 \mid \ldots \mid pat_n \rightarrow e_n$, e.g. `match a with [] -> 1 | x::_ -> x`, `match x,y with 0,0 -> "" | 1,_ -> "x" | _ -> "abc"`
- Function definition: fun $arg_0 \ \ldots \ arg_n \rightarrow e$, e.g. `fun x -> 2`, `fun a b c -> a + b - c`
- Function application: $e\ e_0$ (resp. $e\ e_0\ \ldots\ e_n$), e.g. `f 2`, `(fun x -> 0) (1::[])`
- Variable binding: let $name = e_0$ in e_1, e.g. `let x = 4 in x * x`, `let f = fun a -> a + 2 in f 7`

2. For these expression, list all contained subexpressions and give their corresponding type. Then evaluate the expression:

```
In [ ]: (* a *) let a = fun x y -> x + 2 in a 3 8 :: []
```

```
In [ ]: (* b *) ((fun x -> x::[]) (9 - 5), true, ('a', 7))
```

(* a *) let a = fun x y -> x + 2 in a 3 8 :: [] : int list fun x y -> x + 2 : int -> 'a -> int x + 2 : int x : int 2 : int a 3 8 :: [] : int list a 3 8 : int a : int -> 'a -> int 3 : int 8 : int [] : int list (* value = [5] *)

(* b *) ((fun x -> x::[]) (9 - 5), true, ('a', 7)) : int list * bool * (char * int) (fun x -> x::[]) (9 - 5) : int list fun x -> x::[] : 'a -> 'a list x::[] : 'a list x : 'a [] : 'a list 9 - 5 : int 9 : int 5 : int true : bool 'a', 7 : char * int 'a' : char 7 : int (* value = ([4], true, ('a', 7)) *)

3 Assignment 5.3 (P): What's the point?

In this assignment, you are supposted to implement some functionalities to compute with 3-dimensional vectors.

1. Define a suitable data type for your point.

```
In [ ]: type vector3 = float * float * float
```

2. Define three points `p1`, `p2` and `p3` with different values.

```
In [ ]: let p1 = -2.5, 5.0, 0.2
        let p2 = 0.0, 0.0, 13.1
        let p3 = 5.4, -8.2, 1.0
```

3. Implement a function `vector3_to_string : vector3 -> string` to convert a vector into a human-readable representation.

```
In [ ]: let vector3_to_string =
            fun (x,y,z) -> "(" ^
                (string_of_float x) ^ "," ^
                (string_of_float y) ^ "," ^
                (string_of_float z) ^ ")"
```

4. Write a function `vector3_add : vector3 -> vector3 -> vector3` to add two vectors.

```
In [ ]: let vector3_add (x1,y1,z1) (x2,y2,z2) = x1+.x2, y1+.y2, z1+.z2
```

5. Write a function `vector3_max : vector3 -> vector3 -> vector3` that returns the larger vector (the vector with the greater magnitude).

```
In [ ]: let vector3_max (x1,y1,z1) (x2,y2,z2) =
            if x1*.x1 +. y1*.y1 +. z1*.z1 > x2*.x2 +. y2*.y2 +. z2*.z2
                then (x1, y1, z1)
                else (x2, y2, z2)
```

6. Compute the result of adding `p1` to the larger of `p2` and `p3` and print the result as a string.

```
In [ ]: vector3_to_string (vector3_add p1 (vector3_max p2 p3))
```

4 Assignment 5.4 (P) Student Database

In this assignment, you have to manage the students of a university.

1. Define a data type for a student. For every student the first name, last name, identification number, current semester as well as the grades received in different courses have to be stored. A course is simply represented by a number.

```
In [ ]: type student = {
            first_name : string;
            last_name : string;
            id : int;
            semester : int;
            grades : (int * float) list
        }
```

The collection of students is now defined as a list of students:

```
In [ ]: type database = student list
```

2. Write a function `insert : student -> database -> database` that inserts a student into the database.

```
In [ ]: let insert s db = s::db
```

3. Write a function `find_by_id : int -> db -> student list` that returns a list with all students with the given id (either a single student or an empty list, if no such student exists).

```
In [ ]: let rec find_by_id id db = match db with [] -> []
            | x::xs -> if x.id = id then [x] else find_by_id id xs
```

4. Implement a function `find_by_last_name : string -> database -> student list` to find all students with a given last name.

```
In [ ]: let rec find_by_last_name name db = match db with [] -> []
            | x::xs -> if x.last_name = name
                then x::find_by_last_name name xs
                else find_by_last_name name xs
```

5 Homework Assignments

The homework assignments can be found in a separate pdf file.