exercise06-p

November 25, 2018

0.0.1 Assignment 6.1 (P) Explicit Type Annotation

In OCaml, types are inferred automatically, so there is no need to write them down explicitly. However, types can be annotated by the programmer. Discuss:

1) Annotate all types on the expression `let f = fun x y -> x, [y]`

```
In [ ]: let f = fun x y -> x, [y]
```

2) When can explicitly annotated types helpful?

0.0.2 Assignment 6.2 (P) Local Binding

Local (variable) bindings can be used to give names to intermediate results or helper functions inside other expressions (e.g. functions) such that they can be reused. Furthermore, they allow you to split your computation into smaller steps.

In the following expression, mark all subexpressions and for each use of a variable, decide where this variable is defined and what its value is during the evaluation of this expression.

```
In [ ]: let x =
            let f x y =
                let x = 2 * x in
                let y = y::[x] in
                let y x =
                    let y = x::y in
                    y @ y
                in
                let x = y 3 in
                1::x
            in
            f 1 2
```

0.0.3 Assignment 6.3 (P) Binary Search Trees

In this assignment, a collection to organize integers shall be implemented via binary search trees.

1) Define a suitable data type for binary trees that store integers.

```
In [ ]: type tree = (* TODO *)
```

2) Define a binary tree `t1` containing the values $1, 6, 8, 9, 12, 42$

```
In [ ]: let t1 = (* TODO *)
```

3) Implement a function `to_list : tree -> int list` that returns an ordered list of all values in the tree.

```
In [ ]: let to_list = (* TODO *)
```

4) Implement a function `insert : int -> tree -> tree` which inserts a value into the tree. If the value exists already, the tree is not modified.

```
In [ ]: let insert = (* TODO *)
```

5) Implement a function `remove : int -> tree -> tree` to remove a value (if it exists) from the tree.

```
In [ ]: let remove = (* TODO *)
```

0.0.4 Assignment 6.4 (P) The List Module (part 1)

Check the documentation of the OCaml `List` module and find out what the following functions do. Then implement them yourself. Make sure your implementations have the same type. In cases where the standard functions throw exceptions, you may just `failwith "invalid"`.

```
In [ ]: let hd = (* TODO *)

        let tl = (* TODO *)

        let length = (* TODO *)

        let append = (* TODO *)

        let rev = (* TODO *)

        let nth = (* TODO *)
```