Programming 2
Tutorial 7

User-Defined Types

Nedim Šišić
2023

Exercise 1

Declare a record type *vehicle* with fields *type_* (string), *color* (string), and *manufactureYear* (int). Then, create two different *vehicle* records assigned to variables. Try reading the fields of the variables. Try storing the two variables in a list.

```
type vehicle = { type_ : string; color : string; manufactureYear : int };;
let v1 = { type_ = "car"; color = "blue"; manufactureYear = 2017 };;
let v2 = { manufactureYear = 2005; color = "red"; type_ = "boat"};;
v1.type_;; v2.color;;
[v1; v2];;
```

Exercise 2

Declare a *_2Dpoint* type that has the *x* and *y* float fields. Define a function that swaps the coordinates of a _2Dpoint record (i.e., *x* and *y* are exchanged).

```
type _2Dpoint = { mutable x : float; mutable y : float };;
let swap p1 p2 =
  let temp = p1.x in
    p1.x <- p2.x;
    p2.x <- temp
  ;;
let p1 = { x = 0.5; y = 1.};;
let p2 = { x = 3.; y = -2.};;
swap p1 p2;;
p1;; p2;;
```

Exercise 3

Declare a *_2DpointValue* type that contains integer coordinates *x* and *y* and a value *value* field that can be of any type. Create a *_2DpointValue* record that contains a bool value and one that contains a string value. See if the two *_2DpointValue* records can be stored in a single list.

```
type 'a _2DpointValue = { x : int; y : int; value : 'a };;
let p1 = {x = 5; y = 3; value = true};;
let p2 = {x = 1; y = 3; value = "false"};;
[p1; p2];;
```

Exercise 4

Create a type *floatOrBool* which can take *float* and *bool* values. Declare three *floatOrBool* variables: two having the value *5.0*, and one having a bool value. Try comparing the three variables.

```
type floatOrBool =
| F of float
| B of bool;;

let f1 = F 5.;;
let f2 = F 5.;;
let b = B true;;
f1 = f2;; f1 = b;; f2 = b;;
```

Exercise 5

Create a sum type *color* which can have values *Red*, *Green*, *Blue*, *White*, and an RGB value: one float for each of three channels (red, green, and blue) (hint: integer 3-tuple).

```
type color =
| Red
| Green
| Blue
| White
| RGB of int * int * int;;

let c1 = Red;; let c2 = RGB (200, 50, 0);;
```

Exercise 6

Create a function *rgb* which takes a *color* parameter and returns its RGB components. For values Red, Green, Blue, and White the function should return (255, 0, 0), (0, 255, 0), (0, 0, 255), and (255, 255, 255), respectfully.

```
let rgb c =
match c with
| Red -> (255, 0, 0)
| Green -> (0, 255, 0)
| Blue -> (0, 0, 255)
| White -> (255, 255, 255)
| RGB (r, g, b) -> (r, g, b);;

rgb c1;; rgb c2;;
```

Exercise 8

```
type ('a, 'b) someType = V1 of int | V2 of 'a | V3 of ‘b;;
let a = V2 "b";;
let b = V3 false;;
let c = V2 0;;
```

Guess if can we create the following lists:

a) [a; b]

b) [b; c]

c) [a; c]

Exercise 8

```
type ('a, 'b) someType = V1 of int | V2 of 'a | V3 of ‘b;;
let a = V2 "b";;
let b = V3 false;;
let c = V2 0;;
```

Guess if can we create the following lists:

a) [a; b]

b) [b; c]

c) [a; c]

Exercise 8

Create a (recursive) *sequence* type, which is a parametric list (a list of any type). Create a function *length* that returns the length of a *sequence*.

```
type 'a sequence =
| Empty
| AlphaSeq of 'a * ('a sequence);;

let seq = AlphaSeq (1, AlphaSeq (5, Empty));;
```

Exercise 9

Define a function *length* that returns the length of a *sequence*, and a function append that appends an element to the end of a *sequence*.

```
let rec length s =
  match s with
  | Empty -> 0
  | AlphaSeq (_, t) -> 1 + length t;;
length seq;;

let rec append seq e =
  match seq with
  | Empty -> AlphaSeq (e, Empty)
  | AlphaSeq (h, t) -> AlphaSeq (h, append t e);;
append seq 3;;
```