CSE216 Programming Abstractions

Modules and Functors

YoungMin Kwon

Contents

- TODO
  - Build Rect module that implements IRect signature
  - Build Polar module that implements for IPolar signature
  - Build CmplxBuilder functor that takes IRect and IPolar modules and returns ICmplx

  - A module is similar to a java class with static fields and static methods

Rect Module

```
type cmplx = R of float * float
           | P of float * float

(*IRect*)
module type IRect = sig
    val re:  cmplx -> float
    val im:  cmplx -> float
    val add: cmplx -> cmplx -> cmplx
    val sub: cmplx -> cmplx -> cmplx
    val to_str: cmplx -> string
end

(*TODO: implement Rect*)
(*Assert error for Polar form cmplx*)
module Rect : IRect =
```

Rect Module

```
(*TODO: implement Rect*)
(*Assert error for Polar form cmplx*)
module Rect : IRect = struct

    let re = function R (r, i) -> r | _ -> assert false
    let im = function R (r, i) -> i | _ -> assert false

    let add a b = (a, b) |> function
                | (R (ra, ia), R (rb, ib)) -> R (ra +. rb, ia +. ib)
                | _ -> assert false
    let sub a b = (a, b) |> function
                | (R (ra, ia) ,R (rb, ib)) -> R (ra -. rb, ia -. ib)
                | _ -> assert false

    let to_str = function
        | R (r, i) ->  Printf.sprintf "%f + %fi" r i
        | _ -> assert false
end
```

Polar Module

```
(*IPolar*)
module type IPolar = sig
    val mag: cmplx -> float
    val ang: cmplx -> float

    val mul: cmplx -> cmplx -> cmplx
    val div: cmplx -> cmplx -> cmplx

    val to_str: cmplx -> string
end

(*TODO: implement Polar*)
(*Assert error for Rect form cmplx*)
module Polar : IPolar =
```

Polar Module

```
(*TODO: implement Polar*)
(*Assert error for Rect form cmplx*)

module Polar : IPolar = struct

    let mag = function P (m, a) -> m | _ -> assert false
    let ang = function P (m, a) -> a | _ -> assert false

    let mul a b = (a, b) |> function
                | (P (ma, aa), P (mb, ab)) -> P (ma *. mb, aa +. ab)
                | _ -> assert false
    let div a b = (a, b) |> function
                | (P (ma, aa), P (mb, ab)) -> P (ma /. mb, aa -. ab)
                | _ -> assert false

    let to_str  = function
                | P (m, a) ->  Printf.sprintf "%f \\_ %f" m a
                | _ -> assert false
end
```

Cmplx Builder Functor

```
(*ICmplx*)
module type ICmplx = sig
    (*constructor*)
    val makeR: float -> float -> cmplx
    val makeP: float -> float -> cmplx

    (*accessor*)
    val re:  cmplx -> float
    val im:  cmplx -> float
    val mag: cmplx -> float
    val ang: cmplx -> float

    (*complex arithmetic*)
    val add: cmplx -> cmplx -> cmplx
    val sub: cmplx -> cmplx -> cmplx
    val mul: cmplx -> cmplx -> cmplx
    val div: cmplx -> cmplx -> cmplx

    (*to string*)
    val to_str: cmplx -> string
end

(*TODO: implement CmplxBuilder functor*)
module CmplxBuilder (Rec:IRect) (Pol:IPolar) : ICmplx =
```

```
(*TODO: mplement CmplxBuilder functor*)
module CmplxBuilder (Rec:IRect) (Pol:IPolar) : ICmplx = struct

    (*constructor*)
    let makeR r i = R (r, i)
    let makeP m a = P (m, a)

    (*helper*)
    let toRect  = function
                | R (r, i) -> R (r, i)
                | P (m, a) -> R (m *. cos a, m *. sin a)
    let toPolar = function
                | P (m, a) -> P (m, a)
                | R (r, i) -> P (sqrt (r*.r +. i*.i), atan2 i r)

    (*accessor*)
    let re  a = toRect  a |> Rec.re
    let im  a = toRect  a |> Rec.im
    let mag a = toPolar a |> Pol.mag
    let ang a = toPolar a |> Pol.ang

…
```

```
    (*complex arithmetic*)
    let add a b = toRect a  |> fun ra ->
                  toRect b  |> fun rb ->
                  Rec.add ra rb
    let sub a b = toRect a  |> fun ra ->
                  toRect b  |> fun rb ->
                  Rec.sub ra rb
    let mul a b = toPolar a |> fun pa ->
                  toPolar b |> fun pb ->
                  Pol.mul pa pb
    let div a b = toPolar a |> fun pa ->
                  toPolar b |> fun pb ->
                  Pol.div pa pb

    (*to string*)
    let to_str  = function
                | R (r, i) -> Rec.to_str (R (r, i))
                | P (m, a) -> Pol.to_str (P (m, a))
end
```

Cmplx Test Builder Functor

```
module CmplxTestBuilder (C: ICmplx) = struct
    let equ a b = max (a -. b) (b -. a) < 1e-10

    let test_add a b =
        let c = C.add a b in
        assert (equ ((C.re a) +. (C.re b)) (C.re c) &&
                equ ((C.im a) +. (C.im b)) (C.im c))

    let test_sub a b =
        let c = C.sub a b in
        assert (equ ((C.re a) -. (C.re b)) (C.re c) &&
                equ ((C.im a) -. (C.im b)) (C.im c))

    let test_mul a b =
        let c = C.mul a b in
        assert (equ ((C.mag a) *. (C.mag b)) (C.mag c) &&
                equ ((C.ang a) +. (C.ang b)) (C.ang c))

    let test_div a b =
        let c = C.div a b in
        assert (equ ((C.mag a) /. (C.mag b)) (C.mag c) &&
                equ ((C.ang a) -. (C.ang b)) (C.ang c))
```

```
    let test () =
        let a = C.makeR 0.1 0.2 in
        let b = C.makeP 0.2 0.3 in
        test_add a b;
        test_sub a b;
        test_mul a b;
        test_div a b
end

module Cmplx  = CmplxBuilder (Rect) (Polar)
module Tester = CmplxTestBuilder (Cmplx)
let _ = Tester.test ()
```