CSE216 Programming Abstractions

Modules and Functors

YoungMin Kwon

Interval Module

```
(*icomparable.mli
*)
module type IComparable = sig
    type t
    (* positive if first > second,
       negative if first < second,
       0 otherwise *)
    val compare: t -> t -> int
end
```

```
(*iinterval.mli
*)
module type IInterval = sig
    (*end point type*)
    type p
    (*interval type*)
    type i = Interval of p * p
           | Empty

    val create: p -> p -> i
    val isEmpty: i -> bool
    val contains: i -> p -> bool
    val intersect: i -> i -> i
end
```

Interval Builder

```
(*interval_builder.ml
*)
module IntervalBuilder (EndPoint : IComparable) :
                       (IInterval with type p = EndPoint.t) = struct
    type p = EndPoint.t
    type i = Interval of p * p
           | Empty

    (*TODO: if lo > hi, Empty; otherwise Interval (lo, hi)*)
    let create lo hi =

    (*TODO: if interval is Empty or not*)
    let isEmpty =

    (*TODO: whether p is within interval i*)
    let contains i p =

    (*TODO : intersection of i and j
             define min x y and max x y functions first*)
    let intersect i j =
end
```

App

```
(*app.ml
*)
module IntInterval = struct
    (*TODO: include IInterval with
            EndPoint's type t = int  and
                        comp a b = a - b
    *)
    include

    (*TODO: print [] if interval is Empty
            print [a, b] if interval is Interval (a, b)
    *)
    let print =

end
…
```

IntInterval

```
(* Expected result:
[3, 8]
[4, 10]
[4, 8]
[]
[]
*)

let a = IntInterval.create 3 8
let b = IntInterval.create 4 10
let c = IntInterval.intersect a b
let d = IntInterval.create 3 1
let e = IntInterval.intersect c d
let _ = IntInterval.print a;
        IntInterval.print b;
        IntInterval.print c;
        IntInterval.print d;
        IntInterval.print e
```

Compile

- Run the following commands to compile

```
> ocamlc -c -o icomparable.cmi icomparable.mli

> ocamlc -c -o iinterval.cmi iinterval.mli

> ocamlc -c -o interval_builder.cmo interval_builder.ml

> ocamlc -c -o app.cmo app.ml

> ocamlc -o app.out interval_builder.cmo app.cmo

> ./app.out
```

Makefile

```
#TODO: define macros CMIS and CMOS
CMIS = icomparable.cmi iinterval.cmi
CMOS = interval_builder.cmo app.cmo

RM   = rm   #rm in Linux/Mac, del in windows
TRUE = true #true in Linux/Mac, cd . in windows

#TODO: clear and update .SUFFIXES for .mli, .cmi, .ml, .cmo
.SUFFIXES:
.SUFFIXES: .mli .cmi .ml .cmo

#TODO: add suffix rules for .mli.cmi and .ml.cmo
.mli.cmi:;  ocamlc -c -o $@ $<
.ml.cmo:;   ocamlc -c -o $@ $<

#TODO: add rules for target a.out
a.out: $(CMIS) $(CMOS)
    ocamlc -o a.out $(CMOS)

#TODO: add rules for target clean
clean:
    $(RM) *.cmi | $(TRUE)
    $(RM) *.cmo | $(TRUE)
```