프로그래밍의 원리 2012 가을 - 실습 11
모듈,시그니처, 모듈함수

서울대학교 프로그래밍 연구실
강동옥, 최민아

2012년 11월 29일

이번 실습의 목적은:

- 시그너처를 사용해보고 `with` 구문을 사용하는 법 익히기
- 모듈 함수를 사용해 보기
- 모듈 함수를 사용해서 여러 모듈 조합하기

1. 스택

   스택 모듈을 만들며 모듈이 무엇인지 익혀봅시다. `MyStack` 모듈을 완성해 보세요.

```
module MyStack =
  struct
    type t = int list
    exception Empty
    let empty = ...
    let push x t = ...
    let pop t = ...
    let first t = ...
  end
```

2. 스택 시그니처(Signature) signature를 사용하면 모듈의 정보를 감출 수 있습니다. 위 `MyStack` 모듈에서 `first`를 감추고자 합니다. 시그니처를 만들어 보세요.

```
module type MYSTACK =
  sig
    type ...
    exception ...
    val empty : ...
    ...
  end
```

   인터프리터에서 다음을 입력해보면 `first`함수가 가려지고 있음을 확인할 수 있습니다.

```
# module AbstractStack = (MyStack : MYSTACK);;
module AbstractStack : MYSTACK
# AbstractStack.first [1;2;3] ;;
Unbound value AbstractStack.first
```

3. 스택 모듈함수

   스택 모듈함수를 만들어보면서 모듈함수가 무엇인지 익혀봅시다. 다음 시그니처를 참고하면서 MakeStack 모듈함수를 완성해봅시다.

```
module type STACK =
sig
  type atom
  type 'a stack
  val empty_stack : atom stack
  val push : atom * atom stack -> atom stack
end
```

   MakeStack은 다음과같이 type t를 가지고 있는 모듈을 받습니다. 이 type t를 atom으로 하는 스택을 만들려고 합니다. 완성해보세요.

```
module MakeStack(S: sig type t end) =
struct
  type atom = ...
  type 'a stack = ...
  let empty_stack = ...
  let push x stk = ...
end
```

   이제 다음과같이 여러가지 타입의 스택을 만들 수 있습니다.

```
module IntStk = MakeStack(struct type t = int end)
module StrStk = MakeStack(struct type t = string end)
module PairStk = MakeStack(struct type t = int * string end)
```

   인터프리터에서 다음을 입력해보세요.

```
# let istk = IntStk.empty_stack;;
# let istk = IntStk.push 1 istk;;
# let istk = IntStk.push "ab" istk;; (*error*)
# let strstk = StrStk.push "ab" StrStk.empty_stack ;;
```

4. 숫자 모듈

   숫자의 타입을 정의하고 그 타입을 입력으로 받는 함수들을 모아 모듈을 만들어 봅시다. NUMBER 시그니처는 숫자 모듈이 가져야 할 함수들의 목록을 나타냅니다.

```
module type NUMBER = sig
    type t
    val zero: t
    val add: t -> t -> t
    val mul: t -> t -> t
    val print: t -> unit
    val make: string -> t
end
```

숫자를 나타낼 때 정수(int) 타입을 사용하는 모듈 Integer를 만들어 봅시다. 이 모듈은 NUMBER 시그니처를 따릅니다.

```
module Integer : NUMBER =
struct
    type t = int
    let zero = ???
    let add x y = ???
    let mul x y = ???
    let print x = print_int x
    let make s = int_of_string s
end
```

숫자를 나타낼 때 부동소수점(float) 타입을 사용하는 모듈 FloatingPoint를 만들어 봅시다. 이 모듈도 NUMBER 시그니처를 따릅니다

```
module FloatingPoint : NUMBER =
struct
    type t = float
    ...
end
```

5. 3차 정수 벡터

3차 정수 벡터를 만들고 사용할 때 쓰이는 타입, 함수들을 모아놓은 모듈 IntVector3를 만들어 봅시다. 이 모듈은 VECTOR 시그니처를 따릅니다. 벡터를 나타내는 모듈 타입 VECTOR는 다음과 같습니다.

```
module type VECTOR = sig
    type t
    type elemType

    exception InvalidInput

    val make: elemType list -> t
    val add: t -> t -> t
    val mul: t -> elemType -> t
    val dot: t -> t -> elemType
```

```
    val print: t -> unit
    val to_list: t -> elemType list
end
```

t는 벡터를 나타내는 타입, elemType은 벡터의 원소를 나타내는 타입입니다. 추후 확장을 위하여 IntVector3 의 t를 int list라고 하겠습니다.

```
module IntVector3: VECTOR =
struct
    type t = int list
    type elemType = int

    exception InvalidInput
    ...
end
```

VECTOR 시그니처는 elemType의 구체적인 타입을 밖으로 노출시키지 않기 때문에 elemType을 사용하는 mul함수에 인자를 넣을 수 없습니다. 따라서 다음과 같이 씀으로써 elemType을 밖으로 노출시킬 수 있습니다.

```
module IntVector3: VECTOR with type elemType = int =
struct
    type t = int list
    type elemType = int

    exception InvalidInput
    ...
end
```

6. 3차 숫자 벡터

3차 숫자 벡터를 만들고 사용할 때 쓰이는 타입, 함수들을 모아놓은 모듈 Vector3를 만들어 봅시다. 이 모듈은 펑터로써 모듈을 정의할 때 어떤 숫자를 사용하는 지를 입력으로 받아서 여러 종류의 3차 숫자 벡터를 만들어 냅니다. IntVector3의 정의를 가져다가 Number모듈의 타입으로 치환함으로써 만들 수 있습니다.

```
module Vector3 (Number: NUMBER) : VECTOR =
struct
    ...
end
```

마찬가지로 elemType을 밖으로 노출 시키기 위하여 with를 사용합니다

```
module Vector3 (Number: NUMBER) : VECTOR with type elemType = Number.t =
struct
    ...
end
```

7. N차 숫자 벡터

N차 숫자 벡터를 만들고 사용할 때 쓰이는 타입, 함수들을 모아놓은 펑터 모듈 Vector를 만들어 봅시다. 이 모듈은 NUMBER 시그니처를 따르는 모듈과, 몇차인지를 나타내는 TRAIT시그니처를 따르는 모듈 두개를 인자로 받아 모듈을 인자로 받습니다.

```
module type TRAIT =
sig
    val dim: int
end

module VectorN (Number: NUMBER) (Trait: TRAIT)
    : VECTOR with type elemType = Number.t =
struct
...
end
```

이제 5차 부동 소수점 벡터 모듈을 만들고 사용할 수 있습니다

```
module FloatVector5 = VectorN (FloatingPoint) (struct let dim = 5 end)

let numList = [1.37; 2.90; 3.22; 33.22; 33.33]
let numList2 = [1.; 2.; 3.; 1.; 2.]
let a = FloatVector5.make numList
let b = FloatVector5.make numList2

let c = FloatVector5.print (FloatVector5.add a b)
let _ = print_newline ()
let _ = FloatVector5.print (FloatVector5.mul a 2.)
let _ = print_newline ()
let _ = FloatingPoint.print (FloatVector5.dot a b)
let _ = print_newline ()


2.37;4.9;6.22;34.22;35.33;
2.74;5.8;6.44;66.44;66.66;
116.71
```