OCaml 길잡이

2009. 가을. 프로그래밍 원리

이우석, 장수원, 조성근

(Slides Originally Made by 허기홍)

서울대학교 프로그래밍 연구실

ML의 특징

- 값중심
  - 모든 것은 값
  - 한 번 정의되면 바뀌지 않음
- 함수형
  - 함수도 값
  - 인자나 리턴 값으로 사용 가능
- 강한 타입 시스템
  - 안전한 결과물을 보장

ML의 종류

- OCaml
  - INRIA, France
- SML
  - Bell lab. & Princeton, USA
- nML
  - SNU/KAIST, KOREA

기본편

선언문(let)

- 값에 이름을 붙임
- 이름의 유효범위를 결정

```
(* ex1.ml *)
let a = 10

let add x y =
    x + y

let sumofsquare x y =
    let sqx = x * x in
    let sqy = y * y in
        sqx + sqy
```

```
(* ex2.ml *)

let sumofsquare x y =
    let square x = x * x in
        square x + square y
```

조건문(if, match)

- 조건에 따른 분기

```
(* ex3.ml *)
let isEven n =
    if n mod 2 = 0 then true
    else false
```

```
(* ex4.ml *)
let isEven n =
    match n mod 2 with
      0 -> true
    | 1 -> false
```

```
(* ex5.ml *)
let is3mulitple n =
    match n mod 3 with
      0 -> true
    | _ -> false
```

함수

- 경우에 따라 입맛에 맞게
  - 일반 함수

```
let isEven n =
    if n mod 2 = 0 then true
    else false
```

  - 재귀 함수

```
let rec fac n =
    if n = 0 then 1
    else n * fac(n-1)
```

  - 이름 없는 함수 : fun

```
sigma (a, b, (fun x -> x + 1))
```

  - 이름 없는 함수 : function

```
sigma (a, b, (function x -> x + 1))
```

타입

- 명시하지 않아도 자동으로 유추
- 강한 타입 시스템
  - 타입 오류를 방지
- 다양한 타입
  - 기본(int, string 등), list, tuple, record
  - 사용자 정의 타입

강한 타입 시스템

- 타입간 구분이 엄격
- 2 + 2.5 (X)
- (float_of_int 2) +. 2.5 (O)

강한 타입 시스템

- 대등한 구문의 타입은 같아야 함
  - 예) if 의 양쪽 가지, match의 모든 가지

```
(* C *)
int lucky(){
    int r;
    while (1){
        r = rand() % 100;
        if (r == 50)
            return r;
        else
            printf("again\n")
    }
}
```

```
(* OCaml *)
let rec lucky () =
    let r = Random.int 100 in
    if r = 50 then
        r
    else
        (printf("again\n"); lucky ())
```

다양한 타입

- 기본 타입
  - int, float, string, char, unit
- 복합 타입
  - list, tuple, record
- 사용자 정의 타입

리스트

- 같은 타입 값이 나열된 구조
- 머리와 꼬리로 나뉨
  - [1;2;3] = 1::[2;3] = 1::2::[3]
  - [“a”] = “a”::[]

```
let ilist1 = [1; 2; 3]

let ilist2 = 4::ilist1

let ilist3 = ilist1@ilist2
```

```
let getHead l =
    match l with
      h::t -> h
    | [] -> raise Error
```

튜플

- 값의 모임

```
let man = (“age”, 24)
let input = (10,  100, (fun x -> x*x))
```

```
let getfirst l =
    match l with
      (f, _, _) -> f
```

레코드

- 이름표가 붙은 값의 모임
  - C의 구조체와 비슷

```
type subject = {name : string; credit : int}
let subject = {name = "PL"; credit = 3}
```

```
let getName r =
    match r with
      {name= n; credit = c} -> n
```

사용자 정의 타입

- 편의에 따라 다양한 타입 표현 가능
  - 기존 타입을 다시 이름 붙이기
  - 새로운 타입을 생성하기

```
type value = int
type tree = Leaf | Node of value * tree * tree

let t = Node (5, Leaf, Node (4, Leaf, Leaf))
```

```
let rec sum t =
    match t with
      Node (v, t1, t2) -> v + sum t1 + sum t2
    | Leaf -> 0
```

사용자 정의 타입

- 다형 타입(Polymorphic type)

```
type itree = Leaf of int
           | Node of int * itree * itree

type 'a tree = Leaf of 'a
             | Node of 'a * 'a tree * 'a tree

let a = Leaf 5
let b = Node ("b", Leaf "l", Leaf "r")
```

실전편

패턴 매칭

- ML의 강력한 기능
- 타입과 찰떡궁합
  - h::t, (a, b, _), _
- 복잡한 사용자 정의 타입일수록

```
type exp = Num of int
         | Add of exp * exp
         | Minus of exp * exp
         | Mult of exp * exp
         | Div of exp * exp
```

```
let rec eval exp =
    match  exp with
      Num i -> ...
    | Add (e1, e2) -> ...
    | Minus (e1, e2) -> ...
    | ....
```

다형 함수

- 다형성(Polymorphic)을 함수에 적용
- 여러 타입의 값을 인자로 받는 함수

```
let identity x = x

let trans t =
   match t with
      (a, b) -> (b, a)
```

```
let getHead l =
   match l with
      h::t -> h
   | [] -> raise Error
```

고차 함수

- 함수를 인자로 받거나 결과로 내놓는 함수

```
let add x y = x + y
let incr = add 1
let seven = incr 6

let incrN n =
  (fun x -> x + n)
```

```
let rec map f l =
  match l with
    h::t -> (f h)::(map f t)
  | [] -> []

let incrList = map (fun x -> x + 1)
[1; 2; 3; 4]
```

예외

- 오류 처리, 복잡한 구조 한 번에 탈출
- 발생하면 처리하는 곳까지 돌아옴

exception Error of string

```
let rec fac n =
        if n < 0 then raise (Error “invalid arg”)
        else if n = 0 then 0
        else n + fac (n - 1)

let f n =
        let a = try
                        fac n
                with Error s -> print_endline s; -1
                 | _ -> print_endline “unknown exception”; -1
        in
        print_int a;
```

참조값

- 변하는 값을 만들기 위해 사용

```
let count = ref 0
let acc n =
  count := !count + 1
```

모듈 타입과 모듈

- 값, 타입, 예외 상황, 모듈의 모임
  - 모듈 타입 : 내부 구현 숨김, 모듈의 틀 제공
  - 모듈 : 실제 구현

```
module type StackSig =
sig
   type 'a stack
   exception StackEmpty

   val emptyStack : 'a stack
   val push : 'a stack * 'a -> 'a stack
   val pop : 'a stack -> 'a * 'a stack
end
```

```
module Stack : StackSig =
struct
   type 'a stack = 'a list
   exception StackEmpty
   let emptyStack = []
   let push (stk, itm) = itm::stk
   let pop stk =
      match stk with
        (itm::stack) -> (itm, stack)
      | [] -> raise StackEmpty
end
```

모듈함수(functor)

- 모듈을 받아서 모듈을 결과로 주는 함수

```
(* set.ml *)
module type OrderedType =
   sig
      type t
      val compare : t -> t-> int
   end

module type Make (Ord : OrderedType)
   struct
      type elt = Ord.t
      type t = Empty
             | Node of t * elt * t * int
      ...
   end
```

```
module OrderedString
    struct
        type t = string
        let compare = compare
    end

module StringSet = Set.Make (OrderedString)
module IntSet =
    Set.Make(module
            struct
                type t = int
                let compare = compare
            end)
```

라이브러리 사용

- http://caml.inria.fr/pub/docs/manual-ocaml/libref/
- Ocaml 패키지에 소스 코드도 포함
- 가장 많이 쓸 List를 예로 들어 설명

기본적인 리스트 연산

```
let x = [1;2;3]

List.length x                (* 3 *)

List.hd x                    (* 1 *)

List.tl x                    (* [2;3] *)

List.rev x                   (* [3;2;1] *)
```

리스트 탐색 연산

```
let x = [1;2;3]
let isEven = fun y -> (y mod 2) = 0

List.for_all isEven x                    (* false *)

List.exist isEven x                      (* true *)

List.mem 2 x                             (* true *)

List.filter (fun x -> x < 2) x           (* [1] *)
```

리스트 반복 연산

```
let x = [1;2;3]

List.iter print_int x                          (* unit *)

List.map string_of_int x                       (* ["1";"2";"3"] *)

List.fold_left (fun x y -> x*10+y) 0 x   (* 123 *)

List.fold_right (fun x y -> y*10+x) x    (* 321 *)
```

기타 사항

숙제 관련 팁

- 실행기(ocaml)을 재빠르게 이용
  - 애매한 구문 확인
  - 자잘한 테스트

- 예년 게시판 둘러보기

참고 자료

- OCaml 첫걸음
  - http://www.ocaml-tutorial.org/
- OCaml 표준 라이브러리
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/
  - 라이브러리 소스 코드