Ocaml 길잡이

2011.09.07
2011 가을, 프로그래밍 언어

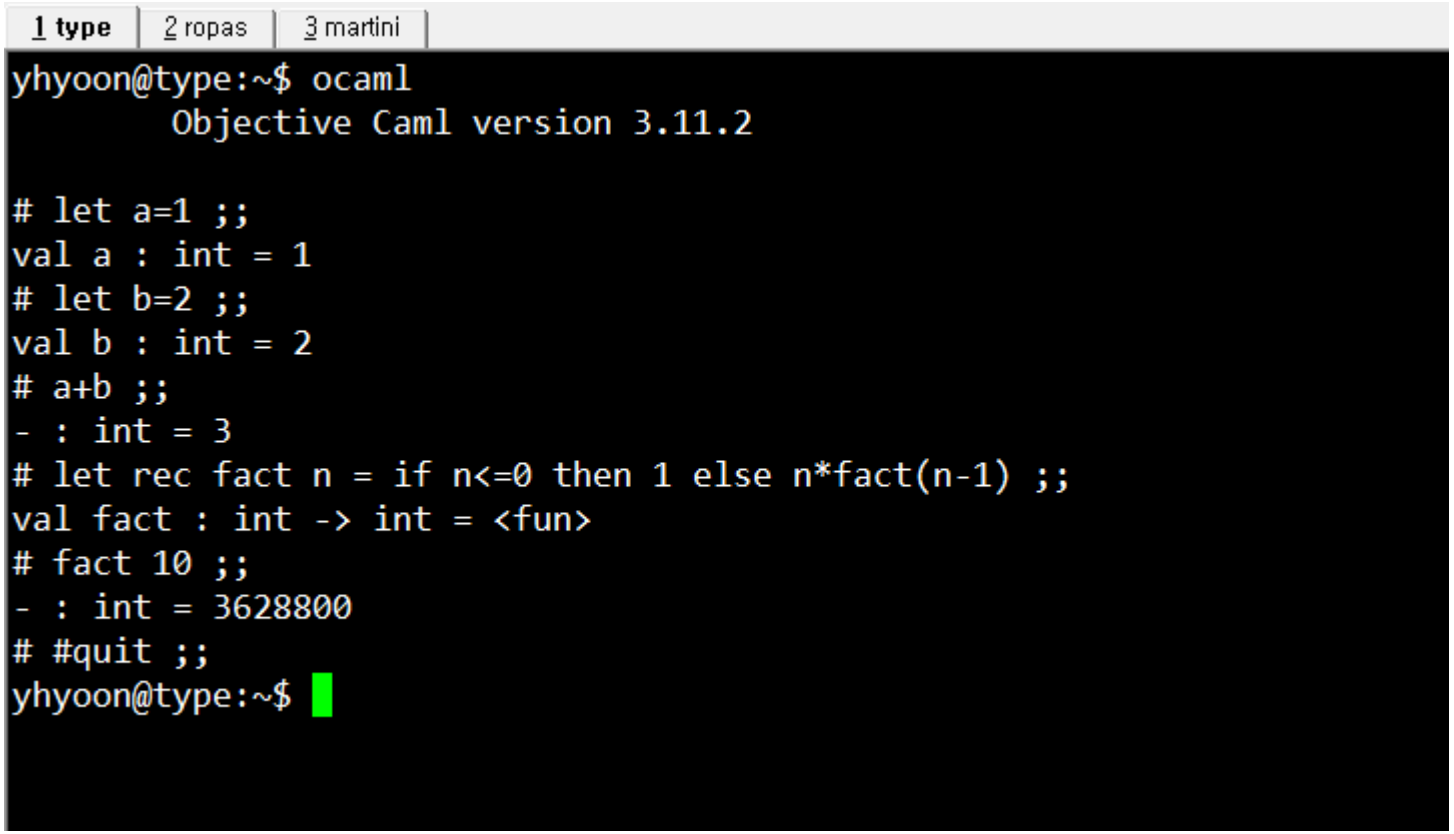
윤용호, 김진영
{yhyoon, jykim}@ropas.snu.ac.kr
서울대학교 프로그래밍연구실

개발 환경

컴파일러 설치

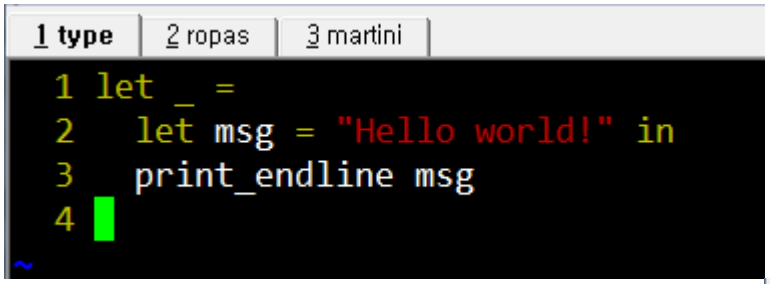
- ocaml.org
  - caml.inria.fr 과 같아요
- Download -> 각자 OS에 맞는 binary 받기
- 혹은 apt-get(Ubuntu), port(MAC) 패키지 매니저
- 혹은 martini 등 snucse 서버 사용
  - martini에는 3.10.2 버전
  - 최신버전은 3.12.1 (금지)
  - 연구실에선 3.11.2 사용

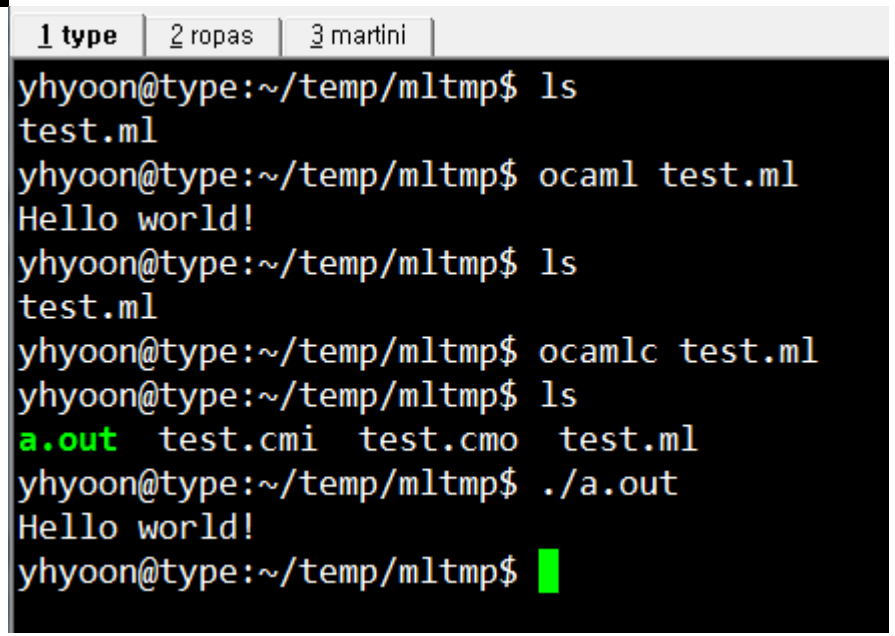
대화형 실행기 환경


1 type
2 ropas
3 martini
yhyoon@type:~$ ocaml
Objective Caml version 3.11.2
let a=1 ;;
val a : int = 1
let b=2 ;;
val b : int = 2
a+b ;;
- : int = 3
let rec fact n = if n<=0 then 1 else n*fact(n-1) ;;
val fact : int -> int = <fun>
fact 10 ;;
- : int = 3628800
#quit ;;
yhyoon@type:~$

컴파일

- 편집 후 컴파일


1 type
2 ropas
3 martini
1 let _ =
2 let msg = "Hello world!" in
3 print_endline msg
4


1 type
2 ropas
3 martini
yhyoon@type:~/temp/mltmp$ ls
test.ml
yhyoon@type:~/temp/mltmp$ ocaml test.ml
Hello world!
yhyoon@type:~/temp/mltmp$ ls
test.ml
yhyoon@type:~/temp/mltmp$ ocamlc test.ml
yhyoon@type:~/temp/mltmp$ ls
a.out test.cmi test.cmo test.ml
yhyoon@type:~/temp/mltmp$ ./a.out
Hello world!
yhyoon@type:~/temp/mltmp$

실행기 vs 컴파일

- 실행기
  - 식을 입력하면 바로 결과를 볼 수 있습니다
  - 식을 입력하고 ;;로 맺습니다
  - 끌 땐 #quit;;

- 컴파일(추천!)
  - ***.ml 파일을 작성하여 컴파일 합니다
  - **;;을 붙이지 않습니다**
  - ml 파일은 정의(let)들의 집합입니다
  - 과제로는 **컴파일이 되는 .ml파일**을 제출합니다

편집기

- vi, emacs
  - 종교
  - 편하신대로
- eclipse 플러그인(OcaIDE)
  - +Cygwin
- 메모장(Notepad)…?
- 기타
  - 문법 강조가 되는 것

기본기

다른 언어, 좀 써보셨어요?

- C/C++, Java, Python
- Scheme, Haskell, ML
- 없다?

Ocaml

- 함수형 언어
  - 값 중심의 언어
  - http://ropas.snu.ac.kr/~kwang/functional.html
  - http://ropas.snu.ac.kr/~kwang/paper/maso/1.pdf

- 어렵지 않아요

let – 값에 이름 붙이기

- `let a=1 in …`
  - 앞으로 1을 a라고도 부르자
  - 변수(variable)가 아님!

- `let incr = fun x -> x+1`
  - x를 받아서 1 큰 수를 뱉어주는 함수를 incr이라고 부르자
  - `incr 10 = 11`

- `let incr x = x+1`

이름 있는 함수

- 함수 만들기

```
1 let incr n = n+1
2
3 let rec fact n =
4     if n<0 then raise (Invalid_argument "factorial")
5     else if n=0 then 1
6     else n * fact(n-1)
```

- 재귀 함수는 이름이 필수

이름 없는 함수

- 꼭 이래야만 하나?
  - `let incr n = n+1 in incr 1`
  - 결과는 2
- 이러면 안 되나?
  - "정수를 받아 1 증가시키는 어떤 함수"에 1을 넣자
  - 결과는 역시 2
- 됩니다
  - `(fun x->x+1) 1`
  - 결과는 2
- 이름은 소중하니까요
  - `f(x), g(x), h(x), F(x), f'(x), …`

이름 없는 함수에 이름 붙이기

- `let incr = (fun n -> n+1)`
- `let incr n = n+1`

- 둘은 같은 의미
  - 취향입니다 존중해주세요
- 상황에 따라 더 편한 것으로

값

- 정수
  - `let i = 1`
- 문자열
  - `let s = “hello world!”`
- 리스트
  - `let l = [1;2;3;4;5]`
  - `let l2 = 1::2::3::4::5::[]`
- 함수
  - `let incr = fun x -> x+1`
  - `let cons a b = a::b`
- 기타 등등

타입

- 정수
  - `let i = 1` `(* int *)`
- 문자열
  - `let s = “hello world!”` `(* string *)`
- 리스트
  - `let l = [1;2;3;4;5]` `(* int list *)`
  - `let l2 = 1::2::3::4::5::[]` `(* int list *)`
- 함수
  - `let incr = fun x -> x+1` `(* int -> int *)`
  - `let cons a b = a::b` `(* 'a -> 'a list -> 'a list *)`
- 기타 등등

코드에 타입 명시하기

- 정수
  - `let i : int = 1 (* int *)`
- 문자열
  - `let s : string = “hello world!” (* string *)`
- 리스트
  - `let l : int list = [1;2;3;4;5] (* int list *)`
  - `let l2 : int list = 1::2::3::4::5::[] (* int list *)`
- 함수
  - `let incr : int -> int = fun x -> x+1 (* int -> int *)`
  - `let cons a b = a::b (* 'a -> 'a list -> 'a list *)`
- 기타 등등

코드에 타입 명시하기 = 뱀발

- 코드에 타입을 쓸 필요가 없습니다
  - C/C++ : `int a=1; string s="abc";`
  - Ocaml : `let a=1  let s="abc"`
- 왜? Ocaml의 타입 유추 시스템
  - 자동으로
  - 정확하게
  - 강한 타입 시스템
- 써야만 하는 경우도 있어요
  - 모듈 타입

타입들

- 대화형 실행기를 띄워놓고 이것저것 입력해보세요!
- int : 정수
- float : 실수
- string : 문자열
- 'a list : 무언가의 리스트
  - int list, string list, float list, int list list, …
- 'a * 'b : 무언가 두 개의 순서쌍
  - (1, 2.0) : int * float
  - ("yhyoon", 20889) : string * int
- 'a -> 'b : 'a를 받아 'b를 만드는 함수
  - incr : int -> int
  - fst : ('a * 'b) -> 'a snd : ('a * 'b) -> 'b
  - List.length : 'a list -> int

ML 코드엔 main이 없다

- 코드의 맨 위부터 순서대로 실행
- `let let let let let` 들의 집합
- 일반적으로...

```
1 let a=1
2
3 let b=2
4
5 let add x y = x+y
6
7 let sum = add a b
8
9 let _ =
10     print_endline ("Hello world!");
11     print_int sum;
12     print_newline()
13
```

ML 코드엔 main이 없다

- 코드의 맨 위부터 순서대로 실행
- `let let let let let` 들의 집합
- 과제는...

```
1 let a=1
2
3 let b=2
4
5 let add x y = x+y
6
7 let sum = add a b
8
```

과제의 예

- Problem 1. 두 정수를 받아 최대공약수를 구하는 함수 `gcd : int -> int -> int` 를 작성하시오

과제의 예

- Problem 1. 두 정수를 받아 최대공약수를 구하는 함수 `gcd : int -> int -> int` 를 작성하시오
  - 는 꿈
  - 이런 파일을 제출하시면 됩니다

```
1 let rec gcd a b =
2     if a=1 || b=1 then 1
3     else if a=b then a
4     else if a<b then gcd b a
5     else gcd (a-b) b
```

한 걸음 더

하지만 꼭 알아야 하는 것들

Currying (1/3)

- 잠깐! `gcd a b` 라고요?
- C, Java 과제 할때는 `gcd(a,b)` 였는데?

Currying (1/3)

- 잠깐! `gcd a b` 라고요?
- C, Java 과제 할때는 `gcd(a,b)` 였는데?

- `gcd(a,b)` 와 `gcd a b`는 다릅니다
  - `gcd(a,b) : (int * int) -> int`
    - 정수 두 개의 쌍을 받아 정수를 만든다
  - `gcd a b : int -> int -> int`
    - 정수를 하나 받고 정수 하나를 더 받아 정수를 만든다

Currying (2/3)

- 어떻게 다를까요?
- `gcd(a,b)` 는 정수 두 개를 한 번에
- `gcd a b` 는 정수를 하나만 받을 수도
  - `gcd 10` 의 결과는?
    - b를 받아 10과의 최대공약수를 구하는 "함수"
- 따라서 이런 것도 가능합니다
  - `let add a b = a + b`
  - `let incr = add 1`

Currying (3/3)

- **과제에서 타입이 틀리면 0점입니다**
  - `int -> int -> int`
    - `let rec gcd a b = …`
  - `(int * int) -> int`
    - `let rec gcd (a,b) = …`
  - 주의하세요
  - 제발
  - 이번 튜토리얼의 핵심!

match – with

- switch – case와 비슷하지만 훨씬 편리

- ```
  match x with
    A -> a
  | B -> b
  | C -> c
  | _ -> default
  ```
  ```

match - with 예시

- gcd를 match with로

```
1 let rec gcd a b =
2     if a=1 || b=1 then 1
3     else if a=b then a
4     else if a<b then gcd b a
5     else gcd (a-b) b
```

```
1 let rec gcd a b =
2   match (a,b) with
3   | (1, _) | (_, 1) -> 1
4   | _ ->
5     if a=b then a
6     else if a<b then gcd b a
7     else gcd (a-b) b
```

- list 다루기

```
1 let rec length l =
2   match l with
3   | [] -> 0
4   | _::t -> 1 + length t
```

```
1 let rec sum_of_list l =
2   match l with
3   | [] -> 0
4   | h::t -> h + sum_of_list t
```

match - with 주의할 점

- 중첩시킬 경우 괄호를 잘 써줘야 합니다

```
1 let rec merge l1 l2 =
2   match l1 with
3   | [] -> l2
4   | h1::t1 ->
5     (match l2 with
6     | [] -> l1
7     | h2::t2 ->
8       if h1<h2 then h1::merge t1 l2
9       else h2::merge l1 t2)
```

- 이게 더 보기 좋아요

```
1 let rec merge l1 l2 =
2   match (l1,l2) with
3   | ([], _) -> l2
4   | (_, []) -> l1
5   | (h1::t1, h2::t2) ->
6     if h1<h2 then h1::merge t1 l2
7     else h2::merge l1 t2
```

try – with, raise

- `raise` : 예외 상황 발생시키기
  - 0으로 나누기, 빈 리스트의 머리 꺼내기, 파일이 존재하지 않습니다…
- `try – with` : 예외 잡기
  - Java의 `try – catch`와 비슷
  - `try … with Exception1 -> …`
  - 문법은 `match – with`와 비슷
    - 괄호 잘 써주세요

try – with 예시

- 0으로 나누기

```
1 let s =
2   try
3     string_of_int (
4       let a=read_int() in
5       let b=read_int() in
6       a/b)
7   with Division_by_zero -> "error"
```

- 파일 열기 실패

```
1 let _ =
2   try
3     let input = open_in "input.txt" in
4     let line = input_line input in
5     print_endline "the first line of the file is";
6     print_endline line
7   with Sys_error -> print_endline "fail to open file input.txt"
```

List 다루기 (1/3)

- 정말 정말 정말 정말 많이 쓰입니다
  - ML에서 가장 만만한 자료구조

- 빈 리스트 : `[]`
- 긴 리스트 : `[1;2;3;4;5]`
  - 혹은 `1::2::3::4::5::[]`
- 리스트는 항상
  - 빈 리스트 []이거나
  - 첫 값::나머지 입니다

List 다루기 (2/3)

- 흔한 list의 match 방식

```
1 match l with
2 | [] -> ...
3 | h::t -> ...
```

- 앞에서 본 예제 우려먹기

```
1 let rec length l =
2   match l with
3   | [] -> 0
4   | _::t -> 1 + length t
```

```
1 let rec sum_of_list l =
2   match l with
3   | [] -> 0
4   | h::t -> h + sum_of_list t
```

List 다루기 (3/3)

- 기본 제공 List 라이브러리들
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/List.html
  - 과제할 때 항상 켜둡니다

- List.length : 길이
  - `List.length [1;2;3] = 3`

- List.nth : n번째 값 꺼내기
  - `List.nth [1;2;3] 2 = 2`

- List.rev : 리스트 뒤집기
  - `List.rev [1;2;3] = [3;2;1]`

- List.mem : 값이 있는지 확인하기
  - `List.mem 1 [1;2;3] = true`

타입 만들기

- 정수 리스트 타입을 직접 만들어봅시다

```
 1 type intlist = Nil | List of int * intlist
 2
 3 let rec length l =
 4   match l with
 5   | Nil -> 0
 6   | List(_, t) -> 1 + length t
 7
 8 let rec sum_of_list l =
 9   match l with
10   | Nil -> 0
11   | List(h, t) -> h + sum_of_list t
```

다형 타입 만들기

- 기본 제공 list와 똑같은 일을 할 수 있는 타입을 만들어봅시다

```
type 'a mylist = Nil | List of 'a * 'a mylist
type intlist = int mylist

let rec length l =
  match l with
  | Nil -> 0
  | List(_, t) -> 1 + length t

let rec sum_of_list l =
  match l with
  | Nil -> 0
  | List(h, t) -> h + sum_of_list t
```

- 'a : 여기에 아무 타입이나 와도 상관 없음

생략한 내용

- 모듈과 펑터
- 사용자 정의 예외
- 레퍼런스(변수)
  - `let id = ref 1`

- 상호 재귀 함수
- 사용자 정의 이항 연산자

여러분이 참고하실 자료 (1/2)

- 표준 라이브러리 문서
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/index.html
  - 과제할 땐 항상 띄워둡니다
  - 이러저러 하는 함수가 필요한데...?
    - 반쯤은 이미 만들어져 있습니다.
    - **fold, map, iter**를 특히 적극 활용
- 메뉴얼
  - http://caml.inria.fr/pub/docs/manual-ocaml/index.html
  - 영어공부 합시다

여러분이 참고하실 자료 (2/2)

- 조교 페이지에 있는 예제 코드
  - http://ropas.snu.ac.kr/~ta/4190.310/11f/
  - 피가 되고 살이 됩니다
- Introduction to Objective Caml
  - http://files.metaprl.org/doc/ocaml-book.pdf
- 지난 학기 게시판들
  - 지금 물으려는 그 질문, 누가 이미 했을지도!
  - https://ropas.snu.ac.kr/phpbb/index.php
  - 선배들의 삽질을 반복하지 않길
- 조교에게 질문
  - 최후의 수단으로…ㅠㅠ

제가 참고한 문헌

- 2011년 봄학기 튜토리얼 (by 이원찬)
  - http://ropas.snu.ac.kr/~ta/4190.310/11s/oca ml_tutorial.pdf