関数型言語（OCaml 演習）(8)

いろいろな高階関数

浅井 健一

お茶の水女子大学

代表的なリスト上の高階関数

`List.map` f lst

型：`('a -> 'b) -> 'a list -> 'b list`

lst 中の各要素に f を施したリストを返す。

`List.filter` p lst

型：`('a -> bool) -> 'a list -> 'a list`

lst 中の要素で p を満たすもののみのリストを返す。

`List.fold_right` f lst $init$

型：`('a -> 'b -> 'b) -> 'a list -> 'b -> 'b`

$init$ から始めて lst の要素を右から f 施し込む。

リスト処理の例：成績処理

```
(* 学生の成績データのリストの例 *)
let gakusei_list = [
  {namae="asai";    tensuu=70; seiseki="?"};
  {namae="tanaka"; tensuu=90; seiseki="?"};
  {namae="yamada"; tensuu=60; seiseki="?"};
  ...
]
```

行いたいこと：

- 各 seiseki フィールドに成績を入れる。
- 成績が A の人の数を出す。
- 平均点を出す。
- 成績が A の人の平均点を出す。

各 seiseki フィールドに成績を入れる

```
(* 目的：学生データのリストに成績を入れる *)
(* hyouka_list : gakusei_t list ->
                                gakusei_t list *)
let hyouka_list lst = []
```

リスト中の全要素に対して同じ処理を行っている。

各 seiseki フィールドに成績を入れる

```
(* 目的：学生データのリストに成績を入れる *)
(* hyouka_list : gakusei_t list ->
                                gakusei_t list *)
let hyouka_list lst =
```

リスト中の全要素に対して同じ処理を行っている。

```
    List.map hyouka lst
```

各 seiseki フィールドに成績を入れる

```
(* 目的：学生データのリストに成績を入れる *)
(* hyouka_list : gakusei_t list ->
                                  gakusei_t list *)
let hyouka_list lst =
  (* 目的：学生データ（ひとり）に成績を入れる *)
  (* hyouka : gakusei_t -> gakusei_t *)
  let hyouka gakusei = gakusei (* 未完成 *)

  in List.map hyouka lst
```

各 seiseki フィールドに成績を入れる

```
(* 目的：学生データのリストに成績を入れる *)
(* hyouka_list : gakusei_t list ->
                                  gakusei_t list *)
let hyouka_list lst =
  (* 目的：学生データ（ひとり）に成績を入れる *)
  (* hyouka : gakusei_t -> gakusei_t *)
  let hyouka gakusei = match gakusei with
    {namae=n; tensuu=t; seiseki=s} ->
      gakusei (* 未完成 *)



  in List.map hyouka lst
```

各 seiseki フィールドに成績を入れる

```
(* 目的：学生データのリストに成績を入れる *)
(* hyouka_list : gakusei_t list ->
                                  gakusei_t list *)
let hyouka_list lst =
  (* 目的：学生データ（ひとり）に成績を入れる *)
  (* hyouka : gakusei_t -> gakusei_t *)
  let hyouka gakusei = match gakusei with
    {namae=n; tensuu=t; seiseki=s} ->
      {namae=n; tensuu=t;
       seiseki = if t >= 80 then "A" else
                 if t >= 70 then "B" else
                 if t >= 60 then "C" else "D"}
  in List.map hyouka lst
```

成績が A の人の数を出す

```
(* 目的：成績が A の人の数を返す *)
(* count_A : gakusei_t list -> int *)
let count_A lst = 0
```

- まず、成績が A の人を抽出し、
- その学生たちのリストの長さを返せば良い。

成績が A の人の数を出す

```
(* 目的：成績が A の人の数を返す *)
(* count_A : gakusei_t list -> int *)
let count_A lst =
```

```
  List.length (List.filter seiseki_A lst)
```

- まず、成績が A の人を抽出し、
- その学生たちのリストの長さを返せば良い。

List.filter p lst

lst 中の要素で p を満たすもののみのリストを返す。

```
(* filter : ('a -> bool) -> 'a list ->
                                    'a list *)
let rec filter p lst = match lst with
    [] -> []
  | first :: rest ->
      if p first then first :: filter p rest
                 else filter p rest

(* length : 'a list -> int *)
let rec length lst = match lst with
    [] -> 0
  | first :: rest -> 1 + length rest
```

成績が A の人の数を出す

```
(* 目的：成績が A の人の数を返す *)
(* count_A : gakusei_t list -> int *)
let count_A lst =
```

```
  List.length (List.filter seiseki_A lst)
```

- まず、成績が A の人を抽出し、
- その学生たちのリストの長さを返せば良い。

成績が A の人の数を出す

```
(* 目的：成績が A の人の数を返す *)
(* count_A : gakusei_t list -> int *)
let count_A lst =
  (* 目的：学生(ひとり)の成績が A かを調べる *)
  (* seiseki_A : gakusei_t -> bool *)
  let seiseki_A gakusei = false (* 未完成 *)


  in List.length (List.filter seiseki_A lst)
```

- まず、成績が A の人を抽出し、
- その学生たちのリストの長さを返せば良い。

成績が A の人の数を出す

```
(* 目的：成績が A の人の数を返す *)
(* count_A : gakusei_t list -> int *)
let count_A lst =
  (* 目的：学生(ひとり)の成績が A かを調べる *)
  (* seiseki_A : gakusei_t -> bool *)
  let seiseki_A gakusei = match gakusei with
    {namae=n; tensuu=t; seiseki=s} ->
      false (* 未完成 *)
  in List.length (List.filter seiseki_A lst)
```

- まず、成績が A の人を抽出し、
- その学生たちのリストの長さを返せば良い。

成績が A の人の数を出す

```
(* 目的：成績が A の人の数を返す *)
(* count_A : gakusei_t list -> int *)
let count_A lst =
  (* 目的：学生(ひとり)の成績が A かを調べる *)
  (* seiseki_A : gakusei_t -> bool *)
  let seiseki_A gakusei = match gakusei with
    {namae=n; tensuu=t; seiseki=s} ->
      s = "A"
  in List.length (List.filter seiseki_A lst)
```

- まず、成績が A の人を抽出し、
- その学生たちのリストの長さを返せば良い。

平均点を出す

```
(* 目的：学生リストの平均点を出す *)
(* heikin : gakusei_t list -> int *)
let heikin lst = 0
```

方針：

1. 点数の合計　　を出す。
2. それを学生数　　で割る。

平均点を出す

```
(* 目的：学生リストの平均点を出す *)
(* heikin : gakusei_t list -> int *)
let heikin lst =
  add_list lst / List.length lst
```

方針：

1. 点数の合計 (add_list lst) を出す。
2. それを学生数 (List.length lst) で割る。

List.fold_right f lst $init$

$init$ から始めて lst の要素を右から f 施し込む。

```
(* fold_right :
   ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b *)
let rec fold_right f lst init = match lst with
    [] -> init
  | first :: rest ->
      f first (fold_right f rest init)
```

List.fold_right f lst $init$

$init$ から始めて lst の要素を右から f 施し込む。

```
(* fold_right :
   ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b *)
let rec fold_right f lst init = match lst with
    [] -> init
  | first :: rest ->
      f first (fold_right f rest init)

  fold_right f [1; 2; 3] init
```

List.fold_right *f lst init*

init から始めて *lst* の要素を右から *f* 施し込む。

```
(* fold_right :
   ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b *)
let rec fold_right f lst init = match lst with
    [] -> init
  | first :: rest ->
      f first (fold_right f rest init)

  fold_right f [1; 2; 3] init
= f 1 (fold_right f [2; 3] init)
```

List.fold_right *f lst init*

init から始めて *lst* の要素を右から *f* 施し込む。

```
(* fold_right :
   ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b *)
let rec fold_right f lst init = match lst with
    [] -> init
  | first :: rest ->
      f first (fold_right f rest init)

  fold_right f [1; 2; 3] init
= f 1 (fold_right f [2; 3] init)
= f 1 (f 2 (fold_right f [3] init))
= f 1 (f 2 (f 3 (fold_right f [] init)))
```

List.fold_right *f lst init*

init から始めて *lst* の要素を右から *f* 施し込む。

```
(* fold_right :
   ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b *)
let rec fold_right f lst init = match lst with
    [] -> init
  | first :: rest ->
      f first (fold_right f rest init)

  fold_right f [1; 2; 3] init
= f 1 (fold_right f [2; 3] init)
= f 1 (f 2 (fold_right f [3] init))
= f 1 (f 2 (f 3 (fold_right f [] init)))
= f 1 (f 2 (f 3 init))
```

List.fold_right *f lst init*

init から始めて *lst* の要素を右から *f* 施し込む。

```
(* fold_right :
   ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b *)
let rec fold_right f lst init = match lst with
    [] -> init
  | first :: rest ->
      f first (fold_right f rest init)
```

List.fold_right は、とても汎用的で強力。
（List.map も List.filter も List.length もその特殊形。）

平均点を出す

```
(* 目的：学生リストの平均点を出す *)
(* heikin : gakusei_t list -> int *)
let heikin lst =
```

```
    add_list lst / List.length lst
```

平均点を出す

```
(* 目的：学生リストの平均点を出す *)
(* heikin : gakusei_t list -> int *)
let heikin lst =
```

```
  (* 目的：学生の点数の合計を出す *)
  (* add_list : gakusei_t -> int *)
  let add_list lst = List.fold_right add lst 0
  in add_list lst / List.length lst
```

平均点を出す

```
(* 目的：学生リストの平均点を出す *)
(* heikin : gakusei_t list -> int *)
let heikin lst =
  (* 目的:学生(ひとり)の点数を goukei に足す *)
  (* add : gakusei_t -> int -> int *)
  let add gakusei goukei = 0 (* 未完成 *)


  in
  (* 目的：学生の点数の合計を出す *)
  (* add_list : gakusei_t -> int *)
  let add_list lst = List.fold_right add lst 0
  in add_list lst / List.length lst
```

平均点を出す

```
(* 目的：学生リストの平均点を出す *)
(* heikin : gakusei_t list -> int *)
let heikin lst =
  (* 目的:学生(ひとり)の点数を goukei に足す *)
  (* add : gakusei_t -> int -> int *)
  let add gakusei goukei = match gakusei with
    {namae=n; tensuu=t; seiseki=s} ->
      t + goukei
  in
  (* 目的：学生の点数の合計を出す *)
  (* add_list : gakusei_t -> int *)
  let add_list lst = List.fold_right add lst 0
  in add_list lst / List.length lst
```

成績が A の人の平均点を出す

1. 成績が A の人を抽出する。
2. その点数の合計を出す。
3. 成績が A の人の数で割る。

```
(* heikin_A : gakusei_t list -> int *)
let heikin_A lst =
  let lst_A = List.filter seiseki_A lst in
  let goukei = List.fold_right add lst_A 0 in
  let ninzuu = List.length lst_A in
  goukei / ninzuu
```

要素を別々に見るのではなく、ひとまとめに扱う

代表的なリスト上の高階関数

`List.map` f lst

型：`('a -> 'b) -> 'a list -> 'b list`

lst 中の各要素に f を施したリストを返す。

`List.filter` p lst

型：`('a -> bool) -> 'a list -> 'a list`

lst 中の要素で p を満たすもののみのリストを返す。

`List.fold_right` f lst $init$

型：`('a -> 'b -> 'b) -> 'a list -> 'b -> 'b`

$init$ から始めて lst の要素を右から f 施し込む。

まとめ

- リストを一括処理するための代表的な命令：

  `List.map` f lst
  `List.filter` p lst
  `List.fold_right` f lst $init$
  `List.length` lst

- これら以外にも多くの命令が定義されている。(マニュアルを参照。)

これらの関数を上手に使えるようになると、リスト全体をまとめて一括処理する関数を簡潔に書けるようになる。