ML 演習第 3 回
課題

末永, 遠藤, 大山

問題 1 (必須)

“Turtle” を表すデータ型と, その turtle を操作する関数群を定義せよ.

解説　turtle はプログラミング言語 LOGO に出てくる二次元平面上を動き回るカメである. このカメは LOGO で記述した指示通りに画面上を動き回り図形を描画する. カメは現在位置と頭の向きを記憶しており, LOGO で与えられた ” 前に n 歩進め”, “頭を左に k 度回転せよ” などの指示を解釈し動き回る. 今回はこの turtle を表すデータ型を定義して,

- 新しい turtle を (0, 0) の位置に作る.
- turtle を現在の頭の向きに n 歩進ませる.
- turtle を左に k 度回転させる.
- 現在の turtle の位置を返す.

の 4 つの turtle 操作関数を定義してもらう.

仕様　以下に今回実装すべき関数の型を示す.

```
type turtle = (* any specification is O.K. *)
val new_turtle : unit -> turtle (* 座標 (0, 0) に新しい turtle を作る *)
val advance : turtle -> float -> unit (* turtle を現在の頭の向きに進ませる *)
val rotate : turtle -> float -> unit (* turtle を左に回転させる *)
val locate : turtle -> float * float (* turtle の現在位置を返す *)
```

実行例　turtle を作り, (1.0, 0.0), (1.0, 1.0), (0.0, 1.0), (0.0, 0.0) の各点に移動させる様子を以下に示す.

```
# let t1 = new_turtle ();;
val t1 : turtle = ...
# advance t1 1.0; locate t1;;
- : float * float = (1., 0.)
# rotate t1 90.0; advance t1 1.0; locate t1;;
- : float * float = (1., 1.)
# rotate t1 90.0; advance t1 1.0; locate t1;;
- : float * float = (0., 1.)
# rotate t1 90.0; advance t1 1.0; locate t1;;
- : float * float = (0., 0.)
(* 計算誤差で正確に (0., 0.) にならないことがあるが，気にしなくてよい．*)
```

ヒント

- OCaml では sin, cos 等の三角関数があらかじめ使えるようになっている (ちなみに, これらの関数で角度はラジアンで表現される)
- 円周率は atan 1.0 *. 4.0 で表現できる.
- turtle のデータ型は変更可能なフィールドを持つレコードを使うといいかもしれない.

問題 2 (必須)

Stack のデータ構造を表現する多相型を定義して, 次の操作を実装せよ.

```
type 'a stack = { mutable s : 'a list }
val new_stack : unit -> 'a stack (* 新しい stack を作成する．*)
val push : 'a stack -> 'a -> unit (* 要素を push する．*)
val pop : 'a stack -> 'a (* 要素を pop する．stackが空だったら EmptyStack 例外を投げる．*)
```

例 スタックを作り, 1 と 2 を push して, 順に pop する様子を以下に示す.

```
# let s = new_stack ();;
val s : '_a stack = {s = []}
# push s 1;;
- : unit = ()
# push s 2;;
- : unit = ()
# pop s;;
```

```
- : int = 2
# pop s;;
- : int = 1
# pop s;;
Exception: EmptyStack.
```

問題 3 (optional)

Queue のデータ構造を表現する多相型を定義して, 次の操作を実装せよ. 但し, 各操作は queue の長さに関わらず定数時間で終了すること.

1. new_queue: 新しい queue を作成する.
2. add: 新しい要素を queue の末尾に追加.
3. take: queue の先頭の要素を取り出す.

解説 Stack と似た問題だが, stack よりずっと難しいので注意すること. ちなみに, queue のデータ型を単純なリストにしてしまうと, add か take かのいずれかの操作が定数時間では終わらなくなる.

問題 4 (optional)

問題 2 の実装で, `let s = new_stack ()` に対するインタプリタの応答が `val s : '_a stack = {s = []}` となっている. これについて以下の問いに答えよ.

1. `'a stack` と `'_a stack` の違いを説明せよ.
2. 型が `'_a stack` ではなくて `'a stack` となった場合に型チェックは通るが実行時に型エラーが起こる例を考えよ.
3. ```
   # let id x = x;;
   val id : 'a -> 'a = <fun>
   # id id;;
   - : '_a -> '_a = <fun>
   ```
   上の例では `id id` の型は多相型でも問題ないのだが単相型になっている. このとき, `id id` を意味を変えずに多相型にするにはどうすれば良いか.
   ```

問題 5 (optional)

(この問題は少し難しい.) Reference があれば, `let rec` を用いなくとも再帰関数を定義することができる.

- `let rec` を用いずに階乗を計算する再帰関数 fact を定義せよ
- 与えられた整数が奇数であるかどうかを判定する関数 odd と偶数であるかどうかを判定する関数 even を `let rec` を用いずに相互再帰で定義せよ.

注意 `for` 文とか `while` 文とかを使わずに定義すること.

ヒント 1 odd と even は `let rec` を使って定義すると

```
let rec odd n =
  if n = 0 then false else even (n - 1)
and even n =
  if n = 0 then true else odd (n - 1)
```

となる. (odd, even ともに正の数のみが与えられると仮定してよい.)

ヒント 2

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

は

`fact'` = `(fun n -> if n = 0 then 1 else n * fact' (n - 1))`

を満たす fact' を求めて, それを fact として定義しているのと同じことである.

補足 演習の第 5 回でこれと全く同じ発想で解く問題が出てくる.

課題 6 (special)

ICFP Programming Contest 2005 に参加し, レポートを提出せよ.

補足　ICFP Programming Contest は毎年行われている「最強のプログラミング言語」を決めるための天下一武道会である．今年は例年と方式が変わっており、6/24 23:00 (JST) に問題がアナウンスされ, 72 時間かけて参加者がその問題を解く[1]．その後 7/9 23:00 (JST) に問題に対する変更がアナウンスされ, 24 時間かけて参加者がその問題を解く．変更された問題に対する結果が重視されるということなので, 仕様変更に強いプログラムを書いたチームが優勝する仕組みになっているようである．詳細は `http://icfp.plt-scheme.org/` を参照のこと．

使用言語は OCaml でなくとも良い．レポートはチーム全体で一通作成して提出のこと．成果物, 成果物の説明, 考察, チーム内の各人がどのような作業を担当したかを書くこと．

[1] このため, 去年は地下室が殺気立った参加者で埋め尽くされた．