SML お勉強の巻（１）

海野 秀之

2009-03-13 (fri)

1 はじめに

TigerBook 輪講もかなり進んで来ました。ここらで、あやふやなまま来てしまっていた部分を「復習」しておかないと、わりとヤヴァいことになってしまいかねません。

私に関していえば、いまだに SML に不馴れで、不自由なのが悩みの種です。ここは、恥を忍んで（？）SML のほんとの基本から見直してみた方がいいのかも知れません。

2 ファイル入出力

あんまり、SML 入門編の最初らしくありませんが、ファイルの入出力をやってみます。「お気楽 Standard ML of New Jersey 入門[*1] 」を参考にしてます。

2.1 cat1: ファイルを表示する（１）

つぎのようなコードを、cat1.sml というファイルに書いておきます。

```
1  fun cat1(path) =
2    let
3      val f = TextIO.openIn(path)
4      fun cat_sub(NONE) = ()
5      |   cat_sub(SOME c) =
6            (TextIO.output1(TextIO.stdOut, c); cat_sub(TextIO.input1(f)))
7    in
8      cat_sub(TextIO.input1(f));
9      TextIO.closeIn(f)
10   end
```

それでは、実行してみます。まずは、sml を起動して、プログラムを読み込みます。

```
1  % sml
2  Standard ML of New Jersey v110.65 [built: Thu Jun 14 00:21:56 2007]
```

[*1] http://www.geocities.jp/m_hiroi/func/smlnj08.html#chap19

```
- use "cat1.sml";
[opening cat1.sml]
[autoloading]
[library $SMLNJ-BASIS/basis.cm is stable]
[autoloading done]
val cat1 = fn : string -> unit
val it = () : unit
```

つづいて、実行：

```
- cat1 "cat1.sml";
fun cat1(path) =
  let
    val f = TextIO.openIn(path)
    fun cat_sub(NONE) = ()
    |   cat_sub(SOME c) =
          (TextIO.output1(TextIO.stdOut, c); cat_sub(TextIO.input1(f)))
  in
    cat_sub(TextIO.input1(f));
    TextIO.closeIn(f)
  end
val it = () : unit
-
```

いくつか気が付いた点などを列挙しておきます：

- ファイルに保存してあるプログラム片を読み込むために use を使いました。
- let in end や関数定義におけるパターンマッチも使いました。
- いちいち TextIO. と書くのは面倒くさい。
- NONE とか SOME とか見慣れませんね。

2.2 cat2: ファイルを表示する（2）

```
fun cat2(path) =
  let
    open TextIO
    val f = openIn(path)
    val c = ref (NONE: char option)
  in
    while (c := input1(f); isSome(!c)) do output1(stdOut, valOf(!c));
    closeIn(f)
```

```
9   end
```

さきほどの cat1 は、再帰で書かれていましたが、「繰り返しの構文で書くこともできます」というやつ。

c は、**Option 型**を格納する **ref 変数**です。NONE では型がわかんないので、`char option` に指定してあります。

ここでも、いくつか見慣れないものが出てきました。あまり気にせず、箇条書きにしておきましょう。

- こんどは open 宣言でストラクチャ TextIO をオープンしました。
- while 構文をつかってます。
- Option 型
- ref 型、!c
- isSome, valOf も気になりますね。

3 参照型

SML における変数は、こんな感じでした。値を束縛している。

```
1  % sml
2  Standard ML of New Jersey v110.65 [built: Thu Jun 14 00:21:56 2007]
3  - val a = 10;
4  val a = 10 : int
5  - val a = fn x => x;
6  val a = fn : 'a -> 'a
```

これとは違って、参照型というのもある。

```
1  % sml
2  Standard ML of New Jersey v110.65 [built: Thu Jun 14 00:21:56 2007]
3  - val x = ref 10;
4  val x = ref 10 : int ref
5  - x;
6  val it = ref 10 : int ref
7  - !x;
8  val it = 10 : int
9  - val y = x;
10 val y = ref 10 : int ref
11 - !y;
12 val it = 10 : int
13 - x := 0;
14 val it = () : unit
15 - !y;
16 val it = 0 : int
```

:= で代入できること、値を取り出す際に ! が用いられることがわかります。

なお、参照型の変数は、宣言時に型が固定されるので、型の異なる値を代入することはできません。

```
- val i = ref 0;
val i = ref 0 : int ref
- i := 10;
val it = () : unit
- i := "hello.";
stdIn:10.1-10.14 Error: operator and operand don't agree [tycon mismatch]
  operator domain: int ref * int
  operand:        int ref * string
  in expression:
    i := "hello."
```

4 Option 型

```
datatype 'a option = NONE | SOME of 'a
```

もしかしたら、NONE, SOME は SML の予約語じゃなくて、option 型定義のときに、いっしょに定義されるものだったりするんだろうか？？

```
Standard ML of New Jersey v110.69 [built: Tue Feb 17 17:53:30 2009]
- datatype 'a myopt = MYNONE | MYSOME of 'a;
datatype 'a myopt = MYNONE | MYSOME of 'a
- MYNONE;
val it = MYNONE : 'a myopt
- MYSOME 10;
val it = MYSOME 10 : int myopt
```

なるほど。なんとなくわかってきたぞ。SML は、パターンマッチによって、新たな構文のように見えるものを定義できるんだね。

```
Standard ML of New Jersey v110.69 [built: Tue Feb 17 17:53:30 2009]
- datatype 'a opt2 = nil | some of 'a;
datatype 'a opt2 = nil | some of 'a
- nil;
val it = nil : 'a opt2
- some 1;
val it = some 1 : int opt2
- some "hoge";
val it = some "hoge" : string opt2
```

ふむ、ふむ。

SML に慣れない目には、NONE や SOME が予約語のように見えてしまいましたが、そうじゃなかったんですね。

では、SML/NJ の option 型について、いくつか確認しておきます:

```
Standard ML of New Jersey v110.69 [built: Tue Feb 17 17:53:30 2009]
- SOME 10;
val it = SOME 10 : int option
- SOME "hoge";
val it = SOME "hoge" : string option
- NONE;
val it = NONE : 'a option
- isSome;
val it = fn : 'a option -> bool
- valOf;
val it = fn : 'a option -> 'a
```

5 おしまい（ええ？！）

なんか、お茶を濁しました感ありありで恐縮ですが、ぼちぼちいきましょう……という……ことで。

今回は TigerBook とは関係のないプログラム片をもとに、そこに出てきた構文を確認してみました。次回以降は、TigerBook の PROGRAM, 課題を前から再確認しつつ、それらを理解するのに必要な SML 知識を確認していくこととします。

- Option 型、SOME, NONE は理解できましたか？
- SML のパターンマッチ強力！ っていうか、パターンマッチを避けて SML を理解するのは無理。
- use, open も使ってみたよ。