Jacques Garrigue, 2009 年 10 月 14 日

Emacs で OCaml を使う (修正版)

Emacs の起動

情報メディア教育センターでは Emacs のアイコンをクリックするだけでいい．多元数理科学研究科の BSD サーバーでは `\emacs` と入力する．(最初の \ は端末内に表示させるという設定を避けるため)

Emacs の基本操作

Emacs を使ったことがなければ，慣れるのに少しかかるが，プログラミングや TeX の編集に最適なエディタなので，無駄にはならない．

操作には，マウスやカーソルキー以外にコントロールキー (x を同時に押すときは`<C-x>`と書く)，シフトキー (`<S-x>`と書く) とメタキー (`<M-x>`と書く) をベースにしたキー列を利用する．メタキーパソコンにはないので，Mac ではコマンドキー，PC では Alt キーを使う．

最もよく使うコマンドを以下にまとめた．

`<C-x><C-f>`	ファイルを開く (“find”)
`<C-x><C-s>`	ファイルを保存する (“save”)
`<C-x><C-w>`	名前を変えて保存する (”write”)
`<C-x><C-c>`	Emacs を終了させる
`<C-g>`	実行中のコマンドを止める
`<C-S-_>`	直前の操作を取り消す (繰り返し可能)
`<C-a>`	行の先端に移動する
`<C-e>`	行の終わりに移動する
`<C-k>`	行の終わりを削除する (“kill”)
`<C-y>`	削除した文を挿入する (“yank”)

設定ファイルの編集

まず，caml-mode を使えるようにしなければならない．

情報メディア教育センターの計算機では既に使えるようにしてあるが，色が付くようにするために，`<C-x><C-f>.emacs<ret>`でファイルを開き，以下の一行をに追加して下さい．

```
(require 'caml-font)
```

保存した後に一旦 Emacs を終了させて，もう一回起動すると設定が有効になる．

多元数理科学研究科の計算機では`.emacs.personal` に 4 行を加える以下のコマンドで設定が完了する．

```
$ touch .emacs.personal
$ cat /usr/public/lec_examples/caml-mode/dot.emacs >> .emacs.personal
```

上の”>>” に気を付けなければならない．”>” にすると追加ではなく上書きになる．

OCaml を Emacs の中で実行

Emacs の中で `ocaml` を実行するために，以下でキー列をを入力する．

```
<M-x>run-caml<ret><ret>
```

これで新しいバッファの中で以下の内容が表われる．

```
        Objective Caml version 3.09.2

#
```

の後にプログラムを入れると，そのまま実行される．このモードで使える主なキー列は以下のとおりである．

<M-p>	以前の入力文を編集する
<C-c><C-c>	実行を途中で中断させる
<C-c><C-d>	ocaml 自体を終わらせる

ocaml を直接にシェルで起動することもできるが，そうすると編集機能が使えない．

プログラムを編集する

まず，名前が.ml で終わるファイルを作る．

```
<C-x><C-f>test.ml<ret>
```

そのバッファの中でプログラムを書くと，<tab>を押すだけでインデントが自動的に行われる．(構文によって，行を書いてから<tab>を押さないといけない．) また，emacs 21 ではキーワードに色が付く．

編集中のプログラムを一段落ずつ ocaml に実行させることもできる．まず，ocaml を前面に持ってくる．

```
<C-c><C-s>
```

そして，例えば以下の行を書いたら (_はカーソルの位置を表す)

```
let x = 3 * 5;;_
```

今度は次のキー列を入力する (先頭の<C-a>はプログラムの中に戻るため)

```
<C-a><C-c><C-e>
```

そうする実行の結果が ocaml のバッファに表れる．(実行したコードがそちらで表示されないので，先頭の # だけが見える)

```
# val x : int = 15
```

もしもプログラムにエラーがあれば，カーソルがその位置に移る．

ファイルからプログラムを読み込む

Emacs のバッファからの評価は中々便利であるが，ファイルを丸ごと読み込むこともできる．これはトップレベルの機能であり，Emacs を使わなくてもできる．そのとき，Caml は，ファイルの内容があたかも入力ループで入力されたように動作する．

ファイル test.ml の中身は次の通りだとする．

```
let double x = x * 2;;      (* double は引数の 2 倍を計算する *)
let y = 10;;                (* y を適当な値に *)
y + double y;;              (* これで 3 倍だ！ *)
```

test.ml を読み込む．

```
# #use "test.ml";;
val double : int -> int = <fun>
val y : int = 10
- : int = 30
```

このようにファイルからプログラムを読み込む場合は，入力ループで #use "ファイル名";; のように入力すればよい．出力は，読み込んだプログラムによるものである．