Common Lisp プログラミングプロジェクト入門

Gnu Common Lisp には Break Package があり，エディタで少しプログラムして，実行し，エラーの発生した環境（変数の束縛環境）で原因を調べるばかりか，プログラミングの訂正や新規のプログラミングまでしてしまえる．直観的には図 1 のようなプロセスを経ることになる．

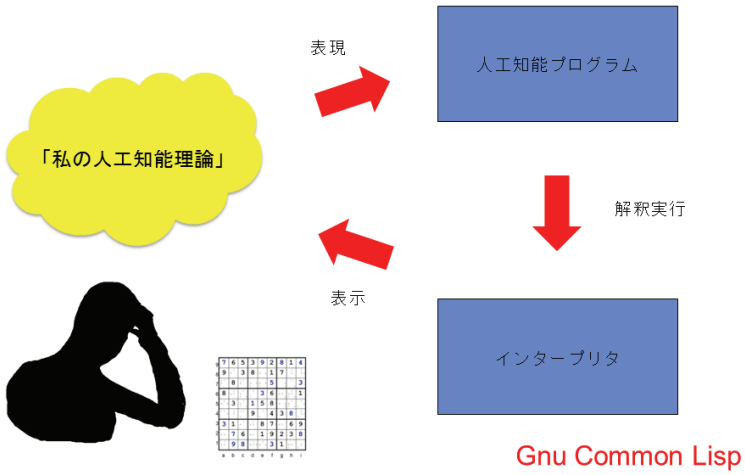

図 1： Lisp プログラミングプロジェクト

実行系を終了すると，折角与えた定義は消えてしまうというのでは困るので，普通のプログラミングでは，定義式をファイルにあらかじめ与えておいて，読み込むという手順を踏む．このとき，Common Lisp 特有の面白い性質は，エラーが生じない限り，定義式を与えたファイルを読ませることはコンソールの入力を（コマンド実行結果に関わらず）与えることと全く同じであることだ．たとえば，My documents/gcl/の下に，sample1.lsp というテキストファイルを作成し，そのなかに先に与えた (defun factorial …)を書いておいて，gcl から，それを読み込めばよい．

上でデフォールトパス名として，“/Users/Nishida/Documents/gcl”を与えて，いちいちこの部分を繰り返さなくてもよいようにするためには変数*default-pathname-defaults*に当該ファイルのあるディレクトリパスをセットしておく．例えば，

```
(setq *default-pathname-defaults* "C:/Users/Nishida/Documents/gcl/")
```

を gcl 立ち上げ時に実行しておくと，

```
(load "sample1")
```

によって， C:/Users/Nishida/Documents/gcl/sample1.lsp にある lisp 式が read-eval-print によって実行される(.lsp は省略可能)．sample1.lsp には lisp インタープリタが実行できる式であれば何を書いても構わない．通常は，(defun factorial …)といった定義式を書いておいて，定義式が実行され，定義が行われるようにする．

さらに発展させると，プログラムを部品から作るのは正反対に，トップレベルから順に作っていく「トンネル掘りプログラミング」ができてしまう．意図的にエラーを引き起こす関数として，(break *s*) がある．

```
(break s)
```

⇒この式が評価された時点で S 式 *s* の評価値が表示され，ブレークパッケージに入る．ブレークパッケージでは，

〈S 式〉 … その環境で〈S 式〉を評価する
`:r` … 処理を続行する
`:q` … トップレベルに帰る
`:b` … どの計算過程でエラーになったか，呼び出し関係を表示する