● GNU Common Lisp 用ヒストリ･エディタ･マニュアル

line-edit-pkg.lsp を一言で表現すると、line-edit-pkg.lsp の主関数 line-edit によって用意される編集用バッファを操作するためのツール・ボックスです。

この編集用バッファは 1 行 1 列のウインドウを持ち、行の長さ、行数共に無制限のテキストで、同梱のヒストリ・パッケージと連動させると、これらのテキストが複数個リング状に連結した構造となります。

編集用バッファの中を文字単位で上下左右に移動する基本的関数、単語単位、リスト単位、Ｓ式単位で移動する関数、スタック形式のテキスト・バッファを介してコピー・ペーストを行う関数などが多数定義されています。

定義されている関数を実行するためのキー・シーケンスは関数 global-set-key によって自由に設定できます。emacs-mode.lsp には Emacs 準拠のキー・シーケンスが、vi-mode.lsp には VI 準拠のキー・シーケンスがあらかじめ定義されています。ユーザ自身が他のエディタ互換のコマンドを定義することもできます。

line-edit-pkg.lsp に定義されている関数群は基本的に Emacs 準拠の編集コマンドと 1 対 1 に対応するように実装しています。VI など、別系統のエディタ用のコマンドを実装する場合は用意されている関数を組み合わせることで必要な機能を実装します。vi-mode-function.lsp が実装例です。

このようにして実装した関数群は、

line-edit-pkg.lsp --> vi-mode-function.lsp --> vi-mode.lsp

の順に読み込みます。自分で他のエディタ用のコマンドを定義する際には、添付してある起動時初期設定用ファイル「.gclrc.lsp」などを参考にして下さい。

line-edit-pkg.lsp が定義している編集用関数の詳細はソースコード中のコメントに記述してあります。各コメントの冒頭には emacs-mode.lsp で割り当てているキー・コードを "C-a", "M-d"などの形式で付記してあります。厳密な定義についてはソース・コード自身を読むのが最良ですが、通常はそこまでする必

要はないでしょう。関数名自身も Emacs に準拠しています。また各関数の動作については「GNU Emacs マニュアル」（アスキー、ISBN4-7561-3002-X）に従いました。

● 新しい編集コマンドを追加する

新コマンド用に定義した関数は line-edit-pkg.lsp の評価ループで

(funcall '関数名)

という引数なしの形式で呼び出されます。前置引数で設定される繰り返し回数は、定義する関数内部で

関数 (repeat-count)

を呼び出すことで得ることができます。

定義した関数をキーボードからのコマンドによって呼び出せるようにするには

(global-set-key "コマンド列" '関数名)

とします。たとえば、キーボードから 'd' 'w'とタイプしたときに関数 delete-word を呼び出すようにするには

(global-set-key "dw" 'delete-word)

とします（詳細は後述）。これらの定義は各編集コマンド用の定義ファイルにまとめて記述します。たとえば Emacs 互換編集コマンドの定義は emacs-mode.lsp に、VI 互換編集コマンドの定義は vi-mode.lsp にまとめます。なお、これらのファイルの拡張子を除く部分（emacs-mode や vi-mode）が、編集コマンドのモード名として使用されます。

定義した関数がキーボードからのコマンド入力によって呼び出されたのか、他の関数から (delete-word)のように通常の方法で呼び出されたのかは、

関数 (call-by-keyboard-p)

によって判別できます。

関数 (select-repeat-count n)

は、現在の関数がキーボードからのコマンド入力によって呼び出されている場合は関数(repeat-count)の値を返し、そうでない場合は、自身の引数の値を返します。ただし、関数(repeat-count)の値が nil の場合は、キーボードからのコマンド入力によって呼び出されている場合であっても、自身の引数の値を返します。

したがって、新たなコマンドを定義する関数において、第 1 引数を省略可能な繰り返し回数を受け取る引数として定義し、その第 1 引数に省略時の既定値を与えて関数を定義すれば (select-repeat-count)を利用してキーボードから指定された繰り返し回数を取得できます。

たとえばポイントが存在する行の空白文字でない最初の文字（の直前のポイント）に移動するコマンドは

```
(defun first-char-of-line (&optional (count 1)) ;省略時既定値を持つ引数を定義。
  (setq count (select-repeat-count count))      ;繰り返し回数の取得。
  (beginning-of-line count)                     ;指定回数行分前の行頭へ移動。
  (skip-white-space))                           ;先頭の空白文字をスキップ。
```

と定義できる訳です。

前置引数によって繰り返し回数が与えられている状態でキーボードからのコマンドによって 関数 first-char-of-line が呼び出されれば、キーボードから与えられた繰り返し回数がシンボル count に与えられ、キーボードから繰り返し回数が与えられていなければ、シンボル count の省略時既定値である '1'が与えられます。更に、この関数を

(first-char-of-line)

という形式で他の関数から呼び出した場合にも、定義の際に意図したとおり

```
(first-char-of-line 1)
```

として機能しますし、引数を省略せずに

```
(first-char-of-line 3)
```

と書けば、シンボル count には '3'が与えられることになります。

● キー・シーケンスを定義する

同じ目的の操作でもエディタによって使用するキー･シーケンスは異なります。たとえば１文字分ポインタを進めるという操作には、Emacs では 「C-f」、VI では「l（エル）」です。３文字分ポインタを進めるには Emacs では「C-u 3 C-f」ですし、VI では「3wl」となります。

このように仮にポインタを１文字進める、という機能を備えた関数が用意されていても、この関数を起動するためのキー・シークエンスは自由に、しかもプログラムを変更せずに定義できた方が便利です。

そこで、関数 global-set-key の第 1 引数に定義すべきキー・シーケンスを文字列として与えることでキー・シーケンスを定義できるようにしています。指定する文字列中の空白は無視するので、たとえば "dw" と "d　w "は同じ意味になります。見やすいように指定して下さい。

空白文字自体をキー・シーケンス文字に含めたい場合は「エスケープ文字」 '¥' を使います。エスケープ文字 '¥'は、直後の文字を評価せずに、その文字そのものとして扱う場合に使用します。したがって空白文字を指定するには '¥ 'とします（バックスラッシュ文字と空白文字)。ただし、Common Lisp も文字列中の '¥'をエスケープ文字として解釈するので評価結果が"¥ "となるようにするために実際には "¥¥ "と記述する必要があります。

コントロール文字とメタ文字を指定する場合は '¥C-' と '¥M-'というエスケ

ープ文字による特別な「文字」を使用します。たとえば Ctrl-A と Meta-x は、それぞれ

"¥¥C-a"
"¥¥M-x"

と記述します。キー・シーケンス文字列中の大文字と小文字は別の文字として扱われるますが、コントロール文字の英字部（たとえば "¥¥C-a"では 'a'の部分）だけは大小文字を区別しません。したがって

"¥¥C-a"　と　"¥¥C-A"

は同じに扱われます（メタ文字の英字部は大小文字を区別する）。

メタ文字を指定する際に使用する Meta キーが存在しないキーボードでは、代わりに Alt キーを使いますがキー・シーケンスは、やはり "¥¥M-"を使います。Esc キーをタイプしてから文字 x をタイプしても M-x と同じ意味になります。したがって "¥¥M-x"は Esc キーの文字コードである "¥¥C-["を使って "¥¥C-[x"とも書けます。

キー・シーケンス文字列には「ラベル」を使うこともできます。キー・シーケンス文字列中では'['と']'で囲まれた文字の列を、単一の文字として扱えるようになっています。

ラベルは関数 define-key によって任意に定義できます。 たとえば、BS キーをタイプしたときに Ctrl-h が発生するならば、

(define-key "[Backspace]"　　"¥¥C-h")

と定義しておけば、line-edit-pkg.lsp は、キーボードから BS（=Ctrl-h）が入力されると "[Backspace]"という「文字」を返すようになります。このとき、

(global-set-key "¥¥M-[Backspace]"　'backward-kill-word)

と定義しておけば、Meta-BS とタイプすると関数 backward-kill-word を実行するようになります。

Deleteキーをタイプすると ESC, '[', '3', '~'という文字コードが発生するのであれば

```
(define-key "[delete]"              "¥¥M-¥¥[¥3¥~")
```

と定義しておき、

```
(global-set-key "[delete]"          'delete-char)
```

とすれば、キーボードの Delete キーをタイプしたときに、関数 delete-char を実行するように設定できます。

'['と']'はラベル指定に使う特別な文字なので、'['と']' 自身をキー・シーケンス文字列中に含めたいときはエスケープ文字 '¥'を使ってエスケープする必要があります。'['は"¥¥["、']'は"¥¥]"と記述します。したがって、たとえば M-[は "¥¥M-¥¥[", M-] は "¥¥M-¥¥["と記します。

line-edit-pkg.lsp のキーボード入力部は、Common Lisp が定義する空白文字（空白やタブなど）とコントロール文字以外の非図形文字が入力されると、自動的に入力された文字コードを 2 桁の 16 進数に変換した「ラベル文字」を返すようになっています。

たとえば Meta キーとスペース・キーの組み合わせをタイプしたときに 16 進数で A0 という文字コードが発生するのであれば、"[0xa0]"というラベル文字を返します（すべて小文字）。したがって、例えば

```
(global-set-key "[0xa0]"            'just-one-space)
```

と定義しておけば Meta キーとスペース・キーの組み合わせをタイプしたときに、関数 just-one-space を実行できますし、もし定義していなければ編集バッファに "[0xa0]"という文字列が入力されます。

Delete キーの例のように、上下左右の矢印キーやファンクション・キーも、それらをタイプしたときに発生する文字コード列が判明しているのであれば、自由に機能を割り当てることができます。

● 未定義の編集コマンドが入力された場合の動作を定義する

様々な形態のエディタ・コマンドを定義できるようにするには、定義していない編集コマンドが入力された場合の動作を定義できるようにしておく必要があります。

line-edit-pkg.lsp は定義されていない編集コマンドが入力された場合は、単に入力された文字を返すようになっています（途中まで一致している場合は最後の文字のみ)。

未定義の編集コマンドが入力された場合の処理を定義するには、関数 global-set-key の第 1 引数に nil を指定して動作を定義します。

emacs-mode.lsp では入力された文字を編集用バッファに入力する処理を行い、vi-mode.lsp では単に無視する処理を行うように定義してあります。

emacs-mode.lsp では

```
(global-set-key nil 'self-insert)
```

vi-mode.lsp では

```
(global-set-key nil 'vi-warning)
```

と定義しています。 ここで指定する関数は line-edit-pkg.lsp から入力された文字(ch とする)を引数として

```
(funcall 'self-insert ch)
```

という形式で呼び出されます。必ず引数をひとつ必要とすることに注意して下さい。ちなみに関数 vi-warning は警告音を鳴らすだけの関数なので引数は不要ですが、前述の条件を満たすために 1 引数の関数として定義し、受け取った引数は単に無視しています。

(daigo@tkf.att.ne.jp)