Aide memoire Emacs / Lisaac

1 Emacs

File

`Ctrl + x + f` Open a file.
`Ctrl + x + s` Record a file.
`Ctrl + x + c` Leave Emacs.

Selection

`Ctrl + Space` Begin of a selection.
`Ctrl + w` Cut.
`Ctrl + y` Paste.
`Ctrl + k` Remove a line.

Searching / Replacement

`Ctrl + s` Search forward.
`Ctrl + r` Search backward.
`Echap + Shift + %` Replace (`Space` or `y` : Valid replacement ; `n` : Invalid replacement ; `!` : Valid replacements).

Windowing

`Ctrl + x + 2` Split in 2 windows horizontaly.
`Ctrl + x + 3` Split in 2 windows verticaly.
`Ctrl + x + 0` Remove a window.
`Ctrl + x + o` Change active view.

Others

`Tab` Automatic indent.
`Echap + g` Go to line.
`Echap + u` Upper case the word.
`Echap + l` Lower case the word.
`Alt + Shift + /` Automatic completion.

2 Lisaac

Expression

<expr1> `=` *<expr2>*	Referential equality
<expr1> `==` *<expr2>*	Structural equality
<expr1> `|` *<expr2>*	Complete evaluation `or` operator
<expr1> `|| {` *<expr2>* `}`	Partial evaluation `or` operator
<expr1> `&` *<expr2>*	Complete evaluation `and` operator
<expr1> `&& {` *<expr2>* `}`	Partial evaluation`and` operator

Base types

[U]INTEGER_8	8 bits integer
[U]INTEGER_16	16 bits integer
[U]INTEGER_32	32 bits integer
[U]INTEGER_64	64 bits integer
[U]INTEGER_BIG	n bits integer
REAL_32	32 bits float
REAL_64	64 bits float
REAL_80	80 bits float
[U]REAL_16_16	16.16 bits fixed float
[U]REAL_24_8	24.8 bits fixed float
[U]REAL_26_8	26.8 bits fixed float
CHARACTER	Character
BOOLEAN	Boolean (TRUE or FALSE)
STRING_CONSTANT	String constant
STRING	Extendable string (`create` *<capacity>*)
FAST_ARRAY[*<type>*]	Extendable array resizable by upper (`lower = 0`, `create_with_capacity` *<capacity>*)
FAST_ARRAY2[*<type>*]	Array FAST_ARRAY with 2 dimensions (`create` *<cap1>,<cap2>*)
ARRAY[*<type>*]	Extendable array (`create` *<idx_min>* `to` *<idx_max>*)
ARRAY2[*<type>*]	2 dimensions ARRAY array (`create` *<idx_min1>,<idx_min2>* `to` *<idx_max1>,<idx_max2>*)
LINKED_LIST[*<type>*]	Simple linked list (`lower = 1`, `create`)
LINKED2_LIST[*<type>*]	Double linked list (`lower = 1`, `create`)
SET[*<type>*]	Hashable*element set* (`create`)
DICTIONARY[*<type_value>,<type_key>*]	Associative dictionary (`create`)

Conditionnal

(*<cond>*).if { } [else { }]	"`if then else`" conditional
(*<cond>*).if_false { }	Evalutation, if conditional is false
().if { }.elseif {*<Cond>*} then { } [else { }]	"`elseif then else`" conditinal type
<expr>.when *<value>* then { }	"`switch case`" conditional type
<expr>.when *<val1>* to *<val2>* then { }	"`switch case`" conditional type, with bounds

Loop

{*<cond>*}.while_do { }	"`while`" loop type, test at begining
{*<cond>*}.until_do { }	"`repeat until`" loop type, test at begining
{ }.do_while {*<cond>*}	"`do while`" loop type, test at the end
{ }.do_until {*<cond>*}	"`repeat until`" loop type, test at the end
<low>.to *<up>* [by *<step>*] do { *<decl_counter>* }	Incremental iteration
<up>.downto *<low>* [by *<step>*] do { *<decl_counter>* }	Decremental iteration (oposite of incremental iteration)

Base message

<object>.print	Print object at screen
<collection>.append *<collection>*	Add a collection at end of collection
<collection>.prepend *<collection>*	Add a collection at begin of collection
<collection>.lower	First index of a collection
<collection>.upper	Last item of a collection
<collection>.count	Count of collection elements
<collection>.put *<elt>* to *<index>*	Put an element to index
<collection>.add_last *<elt>*	Put an element at end of collection
<collection>.add_first *<elt>*	Put an element at begin of collection
<collection>.remove *<index>*	Remove element at index
<collection>.clear	Remove all element
<collection>.item *<index>*	Return an element of a collection
<collection>.last	Return last element of collection
<collection>.first	Return fisrt element of collection

Examples

Section Header

```
  + name := HELLO_WORLD;
```

Section Public

```
  - main <-
  (
    1.to 10 do { i:INTEGER;
      i.print;
      " Hello world !".print;
    };
  );
```

Section Header

```
  + name := READ_STRING;
```

Section Public

```
  - main <-
  ( + my_string:STRING;
    my_string := STRING.create 100;
    IO.read_line_in my_string;
  );
```

Section Header

```
  + name := PARAMETER;
```

Section Public

```
  - main <-
  ( + low,up:INTEGER;

    low := COMMAND_LINE.lower;
    up  := COMMAND_LINE.upper;
    low.to up do { j:INTEGER;
      COMMAND_LINE.item j.print;
      ' '.print;
    };
  );
```
