Aide mémoire Emacs / Lisaac

1 Emacs

Fichier

`Ctrl + x + f` Ouverture d'un fichier.
`Ctrl + x + s` Enregistrer un fichier.
`Ctrl + x + c` Quitter Emacs.

Sélection

`Ctrl + Space` Début d'une sélection.
`Ctrl + w` Couper.
`Ctrl + y` Coller.
`Ctrl + k` Supprime une ligne.

Recherche / Remplacement

`Ctrl + s` Rechercher en avant.
`Ctrl + r` Rechercher en arrière.
`Echap + Shift + %` Remplacer (`Space` ou `y` : Valide le remplacement ; `n` : Invalide le remplacement ; `!` : Valide tous les remplacements).

Fenêtrage

`Ctrl + x + 2` Découpe en 2 fenêtres horizontales.
`Ctrl + x + 3` Découpe en 2 fenêtres verticales.
`Ctrl + x + 0` Supprime une fenêtre.
`Ctrl + x + o` Change la fenêtre active.

Autres

`Tab` Indentation automatique.
`Echap + g` Aller à une ligne.
`Echap + u` Mettre le mot en majuscule.
`Echap + l` Mettre le mot en minuscule.
`Alt + Shift + /` Complémentation automatique.

2 Lisaac

Expression

<expr1> `=` *<expr2>*	Égalité référentielle
<expr1> `==` *<expr2>*	Égalité structurelle
<expr1> `|` *<expr2>*	Opérateur `or` à évaluation totale
<expr1> `|| {` *<expr2>* `}`	Opérateur `or` à évaluation partielle
<expr1> `&` *<expr2>*	Opérateur `and` à évaluation totale
<expr1> `&& {` *<expr2>* `}`	Opérateur `and` à évaluation partielle

Type de base

[U]INTEGER_8	Entier 8 bits
[U]INTEGER_16	Entier 16 bits
[U]INTEGER_32	Entier 32 bits
[U]INTEGER_64	Entier 64 bits
UINTEGER_BIG	Entier *n* bits
REAL_32	Réel 32 bits
REAL_64	Réel 64 bits
REAL_80	Réel 80 bits
[U]REAL_16_16	Réel à virgule fixe 16.16 bits
[U]REAL_24_8	Réel à virgule fixe 24.8 bits
[U]REAL_26_8	Réel à virgule fixe 26.8 bits
CHARACTER	Caractère
BOOLEAN	Booléen (TRUE or FALSE)
STRING_CONSTANT	Constante de chaîne de caractère
STRING	Chaîne de caractère extensible (`create` *<capacity>*)
FAST_ARRAY[*<type>*]	Tableau extensible que par le haut (`lower = 0`, `create_with_capacity` *<capacity>*)
FAST_ARRAY2[*<type>*]	Tableau FAST_ARRAY à 2 dimensions (`create` *<cap1>,<cap2>*)
ARRAY[*<type>*]	Tableau extensible (`create` *<idx_min>* `to` *<idx_max>*)
ARRAY2[*<type>*]	Tableau ARRAY à 2 dimensions (`create` *<idx_min1>,<idx_min2>* `to` *<idx_max1>,<idx_max2>*)
LINKED_LIST[*<type>*]	Liste simplement chaînée (`lower = 1`, `create`)
LINKED2_LIST[*<type>*]	Liste doublement chaînées (`lower = 1`, `create`)
SET[*<type>*]	Ensemble d'éléments *hashable* (`create`)
DICTIONARY[*<type_value>,<type_key>*]	Dictionnaire associatif (`create`)

Conditionnelle

(*<cond>*).if { } [else { }]	Contionnelle de type "`if then else`"
(*<cond>*).if_false { }	Evalutation, si la condition est fausse
().if { }.elseif {*<Cond>*} then { } [else { }]	Contionnelle de type "`elseif then else`"
<expr>.when *<value>* then { }	Contionnelle de type "`switch case`"
<expr>.when *<val1>* to *<val2>* then { }	Contionnelle de type "`switch case`" avec intervalle

Boucle

{*<cond>*}.while_do { }	Boucle de type "`while`", test au début
{*<cond>*}.until_do { }	Boucle de type "`repeat until`", test au début
{ }.do_while {*<cond>*}	Boucle de type "`do while`", test en fin
{ }.do_until {*<cond>*}	Boucle de type "`repeat until`", test en fin
<low>.to *<up>* [by *<step>*] do { *<decl_counter>* }	Itération incrémentale
<up>.downto *<low>* [by *<step>*] do { *<decl_counter>* }	Itération décrémentale

Message de base

<object>.print	Affiche l'objet
<collection>.append *<collection>*	Ajouter une collection à la fin de la collection
<collection>.prepend *<collection>*	Ajouter une collection au début de la collection
<collection>.lower	Renvoi le premier index d'une collection
<collection>.upper	Renvoi le dernier index d'une collection
<collection>.count	Renvoi le nombre d'éléments d'une collection
<collection>.put *<elt>* to *<index>*	Entrer un élément dans une collection
<collection>.add_last *<elt>*	Entrer un élément en fin de collection
<collection>.add_first *<elt>*	Entrer un élément en début de collection
<collection>.remove *<index>*	Supprimer un élément d'une collection
<collection>.clear	Supprimer tous les éléments d'une collection
<collection>.item *<index>*	Renvoyer un élément d'une collection
<collection>.last	Renvoyer le dernier élément d'une collection
<collection>.first	Renvoyer le premier élément d'une collection

Exemples

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;
    };
  );
```
