Auteur : Pierre Boudes (mindsized.org)
sous licence ouverte, source : fichier org-mode

Emacs

Installer notre configuration d'Emacs (`.emacs.d`) ou au moins activer quelques raccourcis plus standards en tapant `Alt-x` puis `cua-mode` et entrée (et pour afficher les numéros de lignes `Alt-x` puis `linum-mode`).

Commandes générales

Commande	Effet
`C-g`	**Abandonner** une commande
`C-x C-f`	Ouvrir un fichier…
`C-x C-s`	Enregistrer le fichier
`C-x C-w`	Enregistrer sous un nouveau nom de fichier…
`C-x k`	Fermer le fichier
`C-espace`	Commencer une sélection au curseur
`C-x` ou `C-w`	Couper la sélection
`C-k`	Couper du curseur à la fin de la ligne
`C-c` ou `Alt-w`	Copier la sélection
`C-v` ou `C-y`	Coller
`Alt-y`	(après coller), coller l'élément précédent
`C-z` ou `C-_`	Annuler
`C-h`	Demander de l'aide…
`C-a`	Placer le curseur en début de ligne
`C-e`	Placer le curseur en fin de ligne
`C-t`	Intervertir les deux caractères autour du curseur
`Alt-t`	Intervertir les deux mots autour du curseur
`C-s`	Rechercher en avant
`C-r`	Rechercher en arrière
`C-%`	Rechercher/remplacer…

Les *vues*

Emacs permet de partager le cadre de ses fenêtres en plusieurs *tuiles* ou *vues* sur les fichiers ouverts (fermer une vue ne ferme pas le fichier).

Commande	Effet
`C-x 0`	Fermer la vue active (où se trouve le curseur)
`C-x 1`	Fermer les autres vues
`C-x 2`	Diviser la vue en deux, l'une au dessus de l'autre
`C-x 3`	Diviser la vue en deux, l'une à côté de l'autre
`C-x b`	Visiter un autre fichier ouvert dans la vue active…
`C-x o`	Passer le curseur dans une autre vue
`C-x C-+`	Augmenter la taille de police de la vue (`+`, `-`, `0`)

AMIL

Simulateur en ligne : `http:mindsized.org/amilweb/`

Instruction	Effet
`stop`	arrête l'exécution
`noop`	ne fait rien
`saut i`	met le compteur de programme à i
`sautpos ri j`	si la valeur du registre i est ≥ 0, effectue saut j
`valeur x ri`	stocke la valeur x dans le registre i
`lecture i rj`	charge le contenu de la mémoire i dans le registre j
`ecriture ri j`	écrit la valeur du registre i dans la mémoire j
`inverse ri`	inverse le signe du registre i
`add ri rj`	ajoute la valeur du registre i au registre j
`soustr ri rj`	soustrait la valeur du registre i au registre j
`mult ri rj`	multiplie le registre j par la valeur du registre i
`div ri rj`	divise le registre j par la valeur du registre i
`lecture *ri rj`	charge dans le registre j le contenu de la mémoire dont le numéro est dans le registre i
`ecriture ri *rj`	écrit la valeur du registre i dans la mémoire dont le numéro est dans le registre j

Programmation Bash

Cours 1 et cours 2 réalisés par Jean-Vincent Loddo.

Instruction	Effet
`#!/bin/bash`	première ligne d'un fichier script
`read X`	Lit une entrée de l'utilisateur et la place dans `X`
`age=$(wc -l <annees.txt)`	compte le nombre de lignes du fichier `annees.txt` et le place dans `age`
`test $age -gt 13`	réussit si la valeur de `age` est plus grande que 13, sinon échoue
`echo $?`	affiche le code de sortie de la dernière instruction
`if x; then y; else z; fi`	exécute x et si x réussit, exécute y, sinon z
`while x; do y; done`	exécute x et tant que x réussit, exécute y puis recommence (exécute x etc.)
`for X in *.c; do echo ${X%.c}; done`	Itération sur tous les noms de fichiers se terminant par `.c`

Programmation C

Auteur : Pierre Boudes (mindsized.org)
sous licence ouverte, source : fichier org-mode

Cycle opérationnel de création d'un programme

Début		
1	`emacs foo.c &`	Éditer foo.c en tâche de fond
2	`C-x C-s` puis `Alt-tab`	Sauvergarder, passer dans le terminal
3	`gcc -Wall foo.c -o foo.exe`	Compiler et **lire les messages**
4	`./foo.exe`	Exécuter pour **tester**
5	`Alt-tab`	Retour à l'éditeur pour **améliorer**

Instructions impératives

Instruction	Effet
`int a;`	déclare une variable de nom `a` et de type entier
`int b = 3;`	déclare une variable `b` de type entier et fixe sa valeur à 3
`double a = 1.2;` `char b;`	déclare deux variables : `a` de type nombre à virgule (en double précision) initialisée à 1.2, et `b` de type caractère.
`a = expression;`	évalue `expression` et affecte sa valeur à `a` (variable,...)
`a += expression;`	évalue `expression` et ajoute sa valeur à `a`
`a *= 3;`	multiplie `a` par 3

Structures de données

Instruction	Effet
`int t[42];`	déclare 42 *variables* `t[0]`, `t[1]`, ..., `t[41]` (un *tableau* de 42 entiers)
`struct a_s toto;`	déclare une variable `toto` de type structure `a_s`
`struct a_s {` `int b; char c; };`	déclare un type structure `a_s` englobant un entier b et un caractère c
`struct a_s toto =` `{.c = 'w', .b = 2};`	(à partir de C99) déclare et intialise une variable `toto` de type structure `a_s`
`toto.b += 40;`	accès aux *champs* d'une variable de type struct

Expressions booléennes

Instruction	Effet
`#include <stdbool.h>` (C99)	définit `true` (alias pour 1) et `false` (pour 0)
`(x && y)`, `(x || y)`, `!x`	`x` et `y`, `x` ou `y`, non `x`
`x == y`, `x != y`, `x >= y`, etc.	test d'égalité, de différence, sup. ou égal etc.

Préprocesseur

Instruction	Effet
`#include <stdlib.h>`	charge la description de la bibliothèque standard
`#include <stdio.h>`	idem pour la bibliothèque d'entrées-sorties
`#define N 10`	définir une constante symbolique
`return EXIT_SUCCESS;`	retourne 0 le code de succès (def. dans `stdlib.h`).

Instructions de contrôle

Instruction	Effet
`if ( x ) { y }`	évalue `x` et si cette condition est vraie (valeur différente de 0), exécute `y`
`else { z }`	lorsque la condition du if juste avant était fausse, exécute `z`
`while ( x ) { y }`	tant que la condition `x` est vraie, exécute `y` (en boucle)
`for (i = 0; i < n; i += 1)` `{ x }`	exécute n fois x, pour i, la *variable de boucle*, allant de 0 à $n-1$ par pas de 1
`for (x; y; z) { t }`	exécute `x` une fois, puis, tant que `y` est vraie, exécute `t`, puis `z`, en boucle

Fonctions et procédures

Instruction	Effet
`int main () { ... }`	définit la fonction principale du programme (son point d'entrée).
`double puissance(double base, int exposant);`	**déclare** une fonction nommée puissance et prenant en paramètres d'entrées un double et un int
`double puissance(double base, int exposant)` `{ x }`	**définit** puissance comme devant exécuter la suite d'instructions x. Les **paramètres formels** `base` et `exposant` sont des variables déclarées dans `x`, initialisées aux valeurs des paramètres effectifs à chaque appel.
`...puissance(3.2, 4)...`	dans un expression, **appelle** la fonction `puissance` avec les **paramètres effectifs** 3.2 et 4. L'expression prend pour valeur la valeur de sortie de l'appel.
`return expression`	Évalue `expression` et retourne sa valeur comme valeur de sortie de la fonction courante
`void afficher(...);`	Déclare une fonction sans valeur de retour (une *procédure*).
`printf("%d %c %g?", 7 * 6, '-', 0.00001);`	Appel à la fonction d'affichage formaté de `stdio.h`. Affiche `42 - 1e-05?`