TD 1 : Que peut-on faire d'un fichier source ? *

ED6 - Licence 3 - Université Paris Diderot

Séances du 8 et 15 février 2012

L'objectif de ces travaux dirigés est de s'initier à une manipulation efficace des fichiers sources, ces fichiers textes qui servent à stocker les programmes. À l'issue de cette séance, vous devrez être capables de :

- Utiliser les principales commandes de l'éditeur de texte Emacs, efficacemet à l'aide les raccourcis claviers.
- Archiver une arborescence, décompresser une archive
- Écrire un *script shell* Unix pour extraire des informations d'un fichier source ou pour le transformer.

Le sujet de ce TD est principalement centré sur l'utilisation et la vérification de conformité d'un programme C à une *norme d'écriture de programmes*. De nombreuses équipes de développement utilisent une telle norme de façon à rendre le code source plus lisible en augmentant son homogénéité, sa simplicité et la qualité de la mise en page. La norme que nous allons utiliser aujourd'hui, et durant tout ce cours, est décrite dans l'annexe A. Bien sûr, une norme ne doit pas forcément être suivie à la lettre : il s'agit de se donner des règles générales qui s'appliquent dans 99% des cas, le pourcent restant étant formé des exceptions qui les confirment !

En guise de support, nous allons utiliser un ensemble de fichiers sources écrits dans le langage C. Vous pouvez télécharger ces fichiers sur la page du cours : http://upsilon.cc/~zack/teaching/1112/ed6/. Vous organiserez votre répertoire de travail pour ce TD de la façon suivante :

`.`	Répertoire de travail
`|-- dictionary`	Bibliothèque de travail
`| '-- src`	Fichiers sources téléchargés sur le site
`'-- scripts`	Sources des scripts (exercice 3 et exercice bonus)

1 Édition

Préférer le clavier à la souris pour les commandes courantes

Un raccourci clavier permet de gagner du temps à condition de ne pas avoir à se le remémorer : son utilisation doit être « automatique ». Il faut donc faire un effort pour les utiliser de façon systématique et les assimiler avec le temps. Vous trouverez en annexe une fiche regroupant les principaux raccourcis clavier d'Emacs. Faites l'effort de l'avoir toujours avec vous lorsque vous programmez, vous progresserez alors significativement !

Nous allons (re)faire un tour d'horizon de l'éditeur Emacs, en nous focalisant sur les raccourcis clavier utiles à la programmation. Mettez donc votre souris de côté ! Elle est interdite à partir de maintenant. Voici comment lire la description des raccourcis de la fiche :

- `C-x` signifie qu'on doit appuyer sur la touche `Control` et la touche `x` simultanément.
- `M-x` signifie qu'on doit appuyer sur la touche `Meta` et la touche `x`. Dans Emacs, la touche `Meta` fait référence en général à la touche `Echap` ou `Alt`. Il s'agit d'une touche symbolique qu'on peut configurer à sa guise.

Un certain nombre de critères de la norme ne peuvent pas être traités automatiquement (ou bien difficilement). Leur vérification et leur correction sont donc faits manuellement. C'est ce que nous allons faire dans cette section.

Exercice 1

1. Récupérez l'archive http://upsilon.cc/~zack/teaching/1112/ed6/tp1-files.tar, décompressez l'archive et placez les fichiers dans `dictionnary/src`.
2. Ouvrez l'ensemble des fichiers contenus dans `dictionary/src`. À l'aide des commandes de la section **Buffers** de la fiche, naviguez à travers ces différents fichiers et à l'intérieur de chacun d'eux sans utiliser les flèches directionnelles du clavier (section **Motion**).

*Ce sujet de TD est (très) largement inspiré d'un sujet proposé par Yann Régis-Gianas

Dans EMACS, un *buffer* est une zone de travail. Il peut s'agir d'un fichier mais aussi d'une zone d'interaction entre l'utilisateur et l'éditeur (comme par exemple, le *buffer* contenant la liste des *buffers* ouverts). Un des *buffers* a un statut spécial : le *mini-buffer*. C'est une zone qui sert à afficher l'interaction courante entre l'éditeur et l'utilisateur.

3. Visualisez un fichier `index.c` et son interface `index.h` en divisant la fenêtre principale d'EMACS en deux fenêtres verticales à l'aide des raccourcis de la section **Multiple Windows**.
4. La fonction `index_create` n'est pas exportée dans le fichier `index.h`. Toujours à l'aide de raccourcis claviers et sans utiliser les flèches directionnelles du clavier, placez-vous dans le fichier `index.c`, au début de l'en-tête de cette fonction et copiez-collez cette en-tête dans le fichier d'interface. Enregistrez vos modifications à l'aide du raccourcis idoine de la section **Files**

La plupart des éditeurs de texte possèdent un presse-papier (*clipboard*) qui sert de zone temporaire de stockage au mécanisme de copier/coller. EMACS fournit une **pile de presse-papiers**, appelée *kill-ring*. Elle permet de stocker un nombre non borné de portions de texte à copier/coller.

5. Effectuez la même transformation que dans la question précédente mais pour les deux fonctions `index_release` et `index_compare` en faisant deux opérations "copier" puis deux opérations "coller".
6. Dans le fichier `cast.h`, mettez en majuscule la macro `declare_as` à l'aide du raccourci clavier de la section **Case Change**.
7. Dans chacun des fichiers .c, remplacez toutes les occurrences de la macro `declare_as` en `DECLARE_AS` à l'aide d'un des raccourcis de la section **Query Replace**.
8. Une fonction de cette bibliothèque ne suit pas une convention de nommage homogène vis-à-vis du reste du projet. Saurez-vous la trouver (par exemple, en utilisant les raccourcis de la section **Incremental Search**) ? À l'aide d'un raccourci de la section **Transposing**, corrigez cette erreur.
9. Vérifiez et corrigez l'orthographe du prologue du fichier `index.c` à l'aide d'un raccourci de la section **Spelling Check**.
10. Le fichier `chunk.c` est mal indenté. Utilisez un raccourci de la section **Formatting** pour corriger cela.
11. Placez-vous dans le fichier `index.c`.
    - Utilisez le raccourcis `M-x`.
    - Vous pouvez remarquer que EMACS se met en attente d'une commande. Tapez `compile` et validez.
    - EMACS vous demande alors une commande de compilation. Fournissez-lui "`gcc -Wall index.c`" puis validez.
    - Que se passe-t-il ? En utilisant le raccourcis `M-g n`, transportez-vous sur la première erreur de compilation. Corrigez-là.
    - Transportez-vous sur la seconde erreur à l'aide de `M-g n`. Corrigez aussi cette erreur.
    - Recompilez votre programme à l'aide de la commande `M-x recompile`.

Toutes les commandes d'EMACS ne sont pas affectées à un raccourci clavier. On peut tout de même accéder à ces commandes à l'aide de la séquence `M-x nom-de-la-commande`. Un mécanisme de complétion sur le nom de la commande est accessible en appuyant sur la touche `[TAB]`. On peut affecter un raccourci clavier à toute commande en modifiant son fichier `.emacs` en rajoutant une ligne d'EMACS-LISP de la forme :

```
(global-set-key [f12] 'compile)
```

Cette petite série de manipulations n'a montré qu'une partie infime des possibilités de cet éditeur de texte. Pour continuer votre apprentissage, vous pouvez tout d'abord suivre le tutoriel intégré dans EMACS en faisant `C-h t` puis lire sa documentation (soit sa version en ligne à l'adresse http://www.gnu.org/software/emacs/manual/ soit sa version intégrée à l'éditeur `C-h r`).

2 Archivage

SAVOIR CAPTURER L'ÉTAT D'UN DÉVELOPPEMENT LOGICIEL

Tout au long du développement, l'ensemble des fichiers sources évolue. La façon la plus simple de capturer un état du développement, c'est-à-dire une version du logiciel, c'est d'archiver l'ensemble des sources du programme dans un fichier appelé **archive**.

Exercice 2

1. Lisez la documentation de l'outil `tar`.
2. Produisez un fichier `dictionary.tar` contenant l'arborescence dont la racine est le répertoire `dictionary`.
3. Compressez l'archive obtenue à l'aide de l'outil `gzip`.
4. Déterminez la commande permettant d'effectuer les deux étapes précédentes en un unique appel à `tar`.
5. Modifiez un fichier source du répertoire `src`.
6. Décompressez l'archive.
7. Calculez la différence entre l'arborescence et l'archive à l'aide de `tar`.
8. Créez un fichier `README` dans le répertoire `dictionary`.
9. Mettez à jour l'archive pour prendre en compte la nouvelle arborescence.
10. Calculez à nouveau la différence entre l'arborescence et l'archive. Comment interprétéz vous le résultat ?

3 Un peu de shell-scripting

Les activités répétitives doivent être automatisées

Il vaut mieux passer une heure à écrire un script qui automatisera une tâche quotidienne demandant 10 minutes si elle est effectuée manuellement. Au bout de seulement 6 jours, vous gagnez 10 minutes de temps de travail par jour et par tâche !

Exercice 3

1. Écrivez un script `backup.sh` qui effectue la sauvegarde d'un fichier `foo.ext` est copiant son contenu dans un fichier dont le nom suit le format : `.foo.ext.backup`. Que se passe-t-il si le fichier `.foo.ext.backup` existe déjà ?
2. Modifiez `backup.sh` pour qu'il effectue la sauvegarde de `foo.ext` dans le fichier `.foo.backup`. Si le fichier existe déjà, le script échoue (le statut de retour est 1).
3. Modifiez `backup.sh` pour qu'il effectue la sauvegarde de `foo.ext` dans le fichier `.foo-jour-mois-annee-heure-minute-seconde.backup`. Si le fichier existe déjà, le script échoue (le statut de retour est 1).
   Indice : Consultez la page de manuel de la commande `date`.
4. Écrivez un script `clean-backup.sh` qui fait la liste de tous les fichiers du répertoire courant dont l'extension est `.backup` et les supprime après en avoir eu confirmation de la part de l'utilisateur.

Exercice Bonus

1. Parmi les points de la norme, lesquels vous semblent vérifiables automatiquement par un script (simple) ?
2. Pour chacun des critères de la norme que vous avez exhibés, écrivez un script nommé `check-C-norm-N-M.sh` qui vérifie que le critère `N-M` est bien vérifié. Si c'est bien le cas, le statut d'erreur sera 0, sinon ce sera 1. En cas de rejet, un message d'erreur de la forme `nom-du-fichier:numero-de-ligne:message` sera produit sur la sortie standard d'erreur. Vous stockerez ce script dans le répertoire `scripts`.
   Pour cette question, la collaboration inter-étudiants est encouragée ! Affectez un critère à chacun d'entre vous et échangez-vous vos scripts ! Bien sûr, vous devrez avoir compris le fonctionnement des scripts que vous utilisez.
3. Incluez l'ensemble des tests de critères dans un script nommé `check-C-norm.sh`, stocker dans votre répertoire `scripts`, et dont le comportement correspond à la spécification décrite plus haut.
4. Écrivez un script `install.sh` qui copie l'ensemble des fichiers `.sh` dans votre arborescence personnelle `$(HOME)/usr/bin`.
5. Déterminez l'ensemble des critères que vous pouvez corriger automatiquement dans un fichier non conforme à la norme.
6. Pour chacun des critères exhibés par la question précédente, écrivez un script `fix-C-norm-N-M.sh` qui corrige le fichier pris en argument. On n'oubliera pas d'utiliser le script `backup.sh` pour sauvegarder l'état du fichier avant modification.
   Pour cette question, la collaboration inter-étudiants est encouragée ! Affectez un critère à chacun d'entre vous et échangez-vous vos scripts ! Bien sûr, vous devrez avoir compris le fonctionnement des scripts que vous utilisez.
7. Modifiez le script `check-C-norm.sh` en lui rajoutant une option "-c" qui active la correction automatique des fichiers à l'aide des scripts précédents. Il devra vérifier que la correction des fichiers invalides est bien conforme à la norme.

A Norme d'écriture pour le langage C

A.1 Structure globale

1. Une ligne ne doit pas excéder 80 caractères.
2. Un bloc (ensemble d'instructions entre accolades) ne doit pas dépasser 15 lignes.
3. On alignera l'ensemble des identifiants intervenant dans une liste de déclarations.
4. Toute accolade ouvrante doit être suivie d'un retour à la ligne.
5. Tout point-virgule est suivi d'un retour à la ligne ou d'un espace.
6. Toute virgule est suivi d'un retour à la ligne ou d'un espace.
7. On sautera une ligne entre les déclarations de variables et les instructions.
8. Le code doit être indenté.

A.2 Expression

1. Un opérateur binaire doit toujours être précédé et suivi par un espace.
2. Il doit toujours y avoir un espace avant une parenthèse.
3. Un opérateur unaire est toujours collé à la sous-expression sur laquelle il s'applique.

A.3 Instruction

1. Il ne doit y avoir qu'une instruction par ligne.
2. Les affectations au sein des expressions sont interdites.
3. Le mot-clé **return** est toujours suivi d'une expression entre parenthèses.

A.4 Déclaration

1. Les déclarations multiples de pointeurs sont interdites.
2. Les étoiles servant à dénoter les types de pointeur doivent être collées au type des données pointées.
3. Toute variable dont la valeur initiale est importante doit être initialisée au moment de sa déclaration.
4. Les noms de type déclarés par l'utilisateur doivent se terminer par "_t".
5. Les noms de structure déclarés par l'utilisateur doivent se terminer par "_s".
6. Les noms d'énumération déclarés par l'utilisateur doivent se terminer par "_e".
7. Les macros doivent être définies en lettres majuscules.
8. Les identifiants doivent être le plus informatifs possible.
9. On limitera le plus possible l'utilisation de variables globales.

A.5 Commentaires

1. Les commentaires doivent suivre le format du logiciel Doxygen décrit ici :

   http://www.stack.nl/~dimitri/doxygen/docblocks.html
2. Une unique langue doit être utilisée dans les commentaires.
3. L'orthographe des commentaires doit être correcte.
4. Une documentation n'est pas une paraphrase du code : elle doit contenir une information qui n'est pas évidente à la lecture de ce dernier (condition d'utilisation de la fonction, propriété de la sortie, invariants des structures de données utilisés, ...).
5. Un fichier d'en-tête (.h) commence par un prologue décrivant l'utilisation des fonctions et des types qu'il déclare.
6. Un fichier d'implémentation (.c) commence par un prologue décrivant les détails de fonctionnement du code.

A.6 Unités de compilation

1. Un fichier `foo.c` doit inclure un fichier `foo.h`.
2. Un fichier d'en-tête `foo.h` doit être protégé contre la double inclusion par un identifiant du préprocesseur de la forme `FOO_H` (éventuellement préfixé par le nom de la bibliothèque).
3. Seules les inclusions strictement nécessaires aux prototypes des fonctions doivent apparaître dans les fichiers d'en-tête.
4. La liste des inclusions débute par l'inclusion des modules nécessaires du système puis des modules de l'utilisateur.

GNU Emacs Reference Card

(for version 23)

Starting Emacs

To enter GNU Emacs 23, just type its name: `emacs`

Leaving Emacs

suspend Emacs (or iconify it under X)	`C-z`
exit Emacs permanently	`C-x C-c`

Files

read a file into Emacs	`C-x C-f`
save a file back to disk	`C-x C-s`
save **all** files	`C-x s`
insert contents of another file into this buffer	`C-x i`
replace this file with the file you really want	`C-x C-v`
write buffer to a specified file	`C-x C-w`
toggle read-only status of buffer	`C-x C-q`

Getting Help

The help system is simple. Type `C-h` (or `F1`) and follow the directions. If you are a first-time user, type `C-h t` for a **tutorial**.

remove help window	`C-x 1`
scroll help window	`C-M-v`
apropos: show commands matching a string	`C-h a`
describe the function a key runs	`C-h k`
describe a function	`C-h f`
get mode-specific information	`C-h m`

Error Recovery

abort partially typed or executing command	`C-g`
recover files lost by a system crash	`M-x recover-session`
undo an unwanted change	`C-x u`, `C-_` or `C-/`
restore a buffer to its original contents	`M-x revert-buffer`
redraw garbaged screen	`C-l`

Incremental Search

search forward	`C-s`
search backward	`C-r`
regular expression search	`C-M-s`
reverse regular expression search	`C-M-r`
select previous search string	`M-p`
select next later search string	`M-n`
exit incremental search	`RET`
undo effect of last character	`DEL`
abort current search	`C-g`

Use `C-s` or `C-r` again to repeat the search in either direction. If Emacs is still searching, `C-g` cancels only the part not matched.

Motion

entity to move over	**backward**	**forward**
character	`C-b`	`C-f`
word	`M-b`	`M-f`
line	`C-p`	`C-n`
go to line beginning (or end)	`C-a`	`C-e`
sentence	`M-a`	`M-e`
paragraph	`M-{`	`M-}`
page	`C-x [`	`C-x ]`
sexp	`C-M-b`	`C-M-f`
function	`C-M-a`	`C-M-e`
go to buffer beginning (or end)	`M-<`	`M->`

scroll to next screen	`C-v`
scroll to previous screen	`M-v`
scroll left	`C-x <`
scroll right	`C-x >`
scroll current line to center of screen	`C-u C-l`

Killing and Deleting

entity to kill	**backward**	**forward**
character (delete, not kill)	`DEL`	`C-d`
word	`M-DEL`	`M-d`
line (to end of)	`M-0 C-k`	`C-k`
sentence	`C-x DEL`	`M-k`
sexp	`M-- C-M-k`	`C-M-k`

kill **region**	`C-w`
copy region to kill ring	`M-w`
kill through next occurrence of *char*	`M-z` *char*
yank back last thing killed	`C-y`
replace last yank with previous kill	`M-y`

Marking

set mark here	`C-@` or `C-SPC`
exchange point and mark	`C-x C-x`
set mark *arg* **words** away	`M-@`
mark **paragraph**	`M-h`
mark **page**	`C-x C-p`
mark **sexp**	`C-M-@`
mark **function**	`C-M-h`
mark entire **buffer**	`C-x h`

Query Replace

interactively replace a text string	`M-%`
using regular expressions	`M-x query-replace-regexp`

Valid responses in query-replace mode are

replace this one, go on to next	`SPC`
replace this one, don't move	`,`
skip to next without replacing	`DEL`
replace all remaining matches	`!`
back up to the previous match	`^`
exit query-replace	`RET`
enter recursive edit (`C-M-c` to exit)	`C-r`

Multiple Windows

When two commands are shown, the second is a similar command for a frame instead of a window.

delete all other windows	`C-x 1`	`C-x 5 1`
split window, above and below	`C-x 2`	`C-x 5 2`
delete this window	`C-x 0`	`C-x 5 0`
split window, side by side	`C-x 3`	
scroll other window	`C-M-v`	
switch cursor to another window	`C-x o`	`C-x 5 o`
select buffer in other window	`C-x 4 b`	`C-x 5 b`
display buffer in other window	`C-x 4 C-o`	`C-x 5 C-o`
find file in other window	`C-x 4 f`	`C-x 5 f`
find file read-only in other window	`C-x 4 r`	`C-x 5 r`
run Dired in other window	`C-x 4 d`	`C-x 5 d`
find tag in other window	`C-x 4 .`	`C-x 5 .`
grow window taller	`C-x ^`	
shrink window narrower	`C-x {`	
grow window wider	`C-x }`	

Formatting

indent current **line** (mode-dependent)	`TAB`
indent **region** (mode-dependent)	`C-M-\`
indent **sexp** (mode-dependent)	`C-M-q`
indent region rigidly *arg* columns	`C-x TAB`
insert newline after point	`C-o`
move rest of line vertically down	`C-M-o`
delete blank lines around point	`C-x C-o`
join line with previous (with arg, next)	`M-^`
delete all white space around point	`M-\`
put exactly one space at point	`M-SPC`
fill paragraph	`M-q`
set fill column to *arg*	`C-x f`
set prefix each line starts with	`C-x .`
set face	`M-o`

Case Change

uppercase word	`M-u`
lowercase word	`M-l`
capitalize word	`M-c`
uppercase region	`C-x C-u`
lowercase region	`C-x C-l`

The Minibuffer

The following keys are defined in the minibuffer.

complete as much as possible	`TAB`
complete up to one word	`SPC`
complete and execute	`RET`
show possible completions	`?`
fetch previous minibuffer input	`M-p`
fetch later minibuffer input or default	`M-n`
regexp search backward through history	`M-r`
regexp search forward through history	`M-s`
abort command	`C-g`

Type `C-x ESC ESC` to edit and repeat the last command that used the minibuffer. Type `F10` to activate menu bar items on text terminals.

GNU Emacs Reference Card

Buffers

select another buffer	`C-x b`
list all buffers	`C-x C-b`
kill a buffer	`C-x k`

Transposing

transpose **characters**	`C-t`
transpose **words**	`M-t`
transpose **lines**	`C-x C-t`
transpose **sexps**	`C-M-t`

Spelling Check

check spelling of current word	`M-$`
check spelling of all words in region	`M-x ispell-region`
check spelling of entire buffer	`M-x ispell-buffer`

Tags

find a tag (a definition)	`M-.`
find next occurrence of tag	`C-u M-.`
specify a new tags file	`M-x visit-tags-table`
regexp search on all files in tags table	`M-x tags-search`
run query-replace on all the files	`M-x tags-query-replace`
continue last tags search or query-replace	`M-,`

Shells

execute a shell command	`M-!`
run a shell command on the region	`M-|`
filter region through a shell command	`C-u M-|`
start a shell in window `*shell*`	`M-x shell`

Rectangles

copy rectangle to register	`C-x r r`
kill rectangle	`C-x r k`
yank rectangle	`C-x r y`
open rectangle, shifting text right	`C-x r o`
blank out rectangle	`C-x r c`
prefix each line with a string	`C-x r t`

Abbrevs

add global abbrev	`C-x a g`
add mode-local abbrev	`C-x a l`
add global expansion for this abbrev	`C-x a i g`
add mode-local expansion for this abbrev	`C-x a i l`
explicitly expand abbrev	`C-x a e`
expand previous word dynamically	`M-/`

Regular Expressions

any single character except a newline	`.` (dot)
zero or more repeats	`*`
one or more repeats	`+`
zero or one repeat	`?`
quote regular expression special character *c*	`\`*c*
alternative ("or")	`|`
grouping	`\( ... \)`
same text as *n*th group	`\`*n*
at word break	`\b`
not at word break	`\B`

entity	**match start**	**match end**
line	`^`	`$`
word	`\<`	`\>`
buffer	`\‘`	`\’`
class of characters	**match these**	**match others**
explicit set	`[ ... ]`	`[^ ... ]`
word-syntax character	`\w`	`\W`
character with syntax *c*	`\s`*c*	`\S`*c*

International Character Sets

specify principal language	`C-x RET l`
show all input methods	`M-x list-input-methods`
enable or disable input method	`C-\`
set coding system for next command	`C-x RET c`
show all coding systems	`M-x list-coding-systems`
choose preferred coding system	`M-x prefer-coding-system`

Info

enter the Info documentation reader	`C-h i`
find specified function or variable in Info	`C-h S`

Moving within a node:

scroll forward	`SPC`
scroll reverse	`DEL`
beginning of node	`.` (dot)

Moving between nodes:

next node	`n`
previous node	`p`
move **up**	`u`
select menu item by name	`m`
select *n*th menu item by number (1–9)	*n*
follow cross reference (return with `l`)	`f`
return to last node you saw	`l`
return to directory node	`d`
go to top node of Info file	`t`
go to any node by name	`g`

Other:

run Info **tutorial**	`h`
look up a subject in the indices	`i`
search nodes for regexp	`s`
quit Info	`q`

Registers

save region in register	`C-x r s`
insert register contents into buffer	`C-x r i`
save value of point in register	`C-x r SPC`
jump to point saved in register	`C-x r j`

Keyboard Macros

start defining a keyboard macro	`C-x (`
end keyboard macro definition	`C-x )`
execute last-defined keyboard macro	`C-x e`
append to last keyboard macro	`C-u C-x (`
name last keyboard macro	`M-x name-last-kbd-macro`
insert Lisp definition in buffer	`M-x insert-kbd-macro`

Commands Dealing with Emacs Lisp

eval **sexp** before point	`C-x C-e`
eval current **defun**	`C-M-x`
eval **region**	`M-x eval-region`
read and eval minibuffer	`M-:`
load from standard system directory	`M-x load-library`

Simple Customization

customize variables and faces	`M-x customize`

Making global key bindings in Emacs Lisp (example):

```
(global-set-key (kbd "C-c g") 'search-forward)
(global-set-key (kbd "M-#") 'query-replace-regexp)
```

Writing Commands

```
(defun command-name (args)
  "documentation" (interactive "template")
  body)
```

An example:

```
(defun this-line-to-top-of-window (line)
  "Reposition current line to top of window.
With ARG, put point on line ARG."
  (interactive "P")
  (recenter (if (null line)
                0
              (prefix-numeric-value line))))
```

The `interactive` spec says how to read arguments interactively. Type `C-h f interactive` for more details.

For copies of the GNU Emacs manual, see:

`http://www.gnu.org/software/emacs/#Manuals`