

oprintanswers

# Utilisation des tables de *tags* avec Emacs

## 1 Étiquetage d'un programme

Les *tags* sont des étiquettes associées aux constructions d'un langage de programmation. On peut construire une table de *tags* associée à un ensemble de fichiers sources au moyen de la commande *etags*.<sup>1</sup> En C, les *tags* sont par défaut associés

- aux fonctions ;
- aux définitions de noms de types : *typedef* ;
- aux définitions de types : *struct*, *union* et *enum* ;
- aux macro-définitions : *#define* ... ;
- aux constantes énumérées (définies au moyen d'*enum*) ;
- aux variables globales ;

Les déclarations de fonctions et de variables externes peuvent également être étiquetées.

EXERCICE 1 – Allez dans le répertoire *bcb* et construisez le fichier *TAGS* en lançant la commande *etags* avec en argument tous les fichiers suffixés par *.c* et *.h* ; examinez le contenu du fichier ainsi créé.

Reconstruisez le fichier en produisant également les définitions externes (recherchez l'option dans la documentation).

EXERCICE 2 – On peut aussi lancer la commande *etags* avec l'argument *-* ; dans ce cas, elle prend la liste des fichiers sources à examiner sur son entrée standard. Revenez dans le répertoire *src* et construisez le fichier *TAGS* (incluant les définitions externes) pour l'ensemble des fichiers *.c* et *.h* de cette arborescence (on utilisera la commande *find* pour rechercher les fichiers et le mécanisme de *pipe*). Examinez le contenu du fichier ainsi créé.

EXERCICE 3 – On peut sélectionner une table de tags au moyen de la commande *visit-tags-table*. Sélectionnez la table du répertoire *src* (le nom *TAGS* est choisi par défaut). Recherchez au moyen de la clé `M-.`  un *tag* correspondant à la chaîne *memoire\_allouer*.

EXERCICE 4 – Allez dans le fichier *test-chaine-1.c* et positionnez-vous sur le mot *vext*. Essayez `M-. <RET>`. Recherchez les autres occurrences possibles en itérant la clé `C-u M-.` .<sup>2</sup> Que constatez-vous ? Essayez également les clés `C-u - M-.` , et `M-*`  (voir la documentation Info pour plus de précision).

**Remarque.** On peut également rechercher les *tags* correspondant à une expression régulière au moyen des clés `C-M-.`  et `C-u C-M-.` .

<sup>1</sup> Voir la liste des langages reconnus dans la section *Tags* de la documentation Info d'Emacs ou dans le message d'usage de la commande *etags*.

<sup>2</sup> On obtient le même résultat en passant n'importe quel argument numérique à la commande *find-tag* associée à la clé `M-.` . Il est donc plus pratique de taper, par exemple, `M-0 M-.`  — équivalent à `C-u 0 M-.`  — à la place de `C-u M-.` .

## 2 Recherche et remplacement au moyen de la table de *tags*

EXERCICE 5 – La table de *tags* peut être utilisée pour spécifier une liste de fichiers à parcourir, soit pour rechercher un motif spécifié par une expression régulière (*tags-search*), soit pour remplacer un motif par une chaîne (*tags-query-replace*). La recherche ou le remplacement en cours peuvent être relancés au moyen de la clé `M-,`. Essayez par exemple d’itérer une recherche à partir de

```
M-x tags-search <RET> chaine.*; <RET>
```

EXERCICE 6 – Remplacer toutes les définitions et utilisations de *memoire\_trace* par *memoire\_tracer*.

EXERCICE 7 – Défaire ces modifications au moyen de la commande *tags-search* et des clés `M-,` et `C-_`.

## 3 Utilisation de *find-tag* dans une fonction

EXERCICE 8 – Écrire en Emacs Lisp une fonction interactive (*f tag*) qui reçoit un *tag* en paramètre sous la forme d’une chaîne de caractères et insère un commentaire vide (*/\* \*/*) sur la ligne précédent la déclaration correspondant à chaque *tag* trouvé.

**Conseils.** Consultez la documentation de la fonction *find-tag*. Pour itérer la recherche des *tags*, on pourra utiliser une construction de la forme

```
(while (progn
        (...))
      :
      (...))
```

EXERCICE 9 – On pourra modifier cette fonction de manière à insérer le commentaire

```
/*! \fn ligne
*/
```

où *ligne*<sup>3</sup> est la ligne du *tag*.<sup>3</sup> Exemple :

```
/*! \fn vext_ajouter(vext self, void *valeur)
*/
void
vext_ajouter(vext self, void *valeur)
{
    ....
}
```

**Conseil.** Utiliser pour récupérer la ligne la fonction *thing-at-point* (voir documentation).

---

<sup>3</sup>Nous verrons plus tard que ce type de commentaire permet de produire de la documentation automatique.