Projet d'algorithmique

Cours d'Informatique de Deuxième Année

—Deug Sciences M3-MI—

Editeur de Texte

Nous désirons éditer des textes, les modifier, les sauvegarder ou encore faire des substitutions de mots par d'autres dans un fichier de la manière la plus simple et la plus économique qu'il soit. L'édition des fichiers peut se faire en mode ligne (comme la commande `ex` par exemple) ou en mode pleine page (par exemple : `vi`, `ved`, `emacs`, `edit`, ...). Votre projet consistera à concevoir un éditeur en mode ligne (le mode pleine page n'étant pas obligatoire).

- ce travail doit être rendu au plus tard le lundi 06 janvier au secrétariat des DEUG Sciences, la soutenance aura lieu lors de la dernière séance de Travaux Pratiques ;
- vous devrez vous inscrire sur la liste de soutenance disponible au secrétariat en même temps que vous rendrez le projet ;
- la clarté et le soin apportés à la rédaction de votre travail, ainsi que la lisibilité de votre *listing* seront pris en compte ;
- ce travail peut être réalisé en binôme, chaque personne devant être présente à la soutenance et pouvant être interrogée individuellement ;
- la partie programmation devra être réalisée en *C ANSI*, compilable sous *HP–UX* (l'*UNIX* de *HP*).

---

1 Présentation du sujet

Tout utilisateur, lorsqu'il désire consulter un fichier ASCII, c'est-à-dire un fichier qui n'est pas dans un format propre à un logiciel, mais qui contient juste du texte entré au clavier, doit pouvoir le faire au travers d'un *éditeur de texte*.

Il existe deux catégories d'éditeurs : les *éditeurs pleine page* qui autorisent l'utilisateur à réaliser des opérations sur son fichier de manière interactive—les opérations se font relativement à la position courante du curseur et évoluent sur simple déplacement de ce dernier, de telle sorte que l'effet de chaque opération est visualisable instantanément ; les *éditeurs en ligne* implémentent les mêmes fonctionalités, à ceci près que le résultat des modifications n'est visualisable que sur demande de l'utilisateur. Dans les deux cas, les opérations de base que doit réaliser l'éditeur sont semblables et incluent notamment : l'ouverture d'un fichier, la sauvegarde, le déplacement du curseur (visible ou non), l'insertion, la suppression, la recherche d'une séquence de caractères, etc.

Le descriptif du travail à effectuer est détaillé plus bas. Toute amélioration doit se justifier dans le cadre de ce projet et nécessite une description exhaustive de l'opération implémentée. Vous trouverez un grand puits d'inspiration parmi des éditeurs en ligne comme `edlin` sous DOS ou encore `ex` sous UNIX.

1.1 Opérations de base

- Ouverture/création d'un fichier.
- Sauvegarde du fichier courant ou sous un nom donné.
- Sortie de l'éditeur.
- Déplacement du curseur (ligne + colonne).
- Insertion d'une ligne.
- Suppression d'une ligne.
- Insertion d'un mot.
- Effacement d'un mot.
- Remplacement d'un mot par un autre.
- Recherche d'une séquence de caractères de manière circulaire (position courante → fin de fichier → début de fichier → position courante).
- Possibilité d'annuler certaines commandes de base.

1.2 Algorithmes

La description des algorithmes que vous utilisez devra apparaître clairement dans votre rapport. Il sera bienvenu de reprendre des algorithmes généraux de manipulation de structures déjà étudiés en séance de TD ou TP.

Les structures de données sont à priori libres, mais vous avez tout intérêt à vous inspirer de celles qui sont données plus bas.

2 Description du travail à rendre

Le projet que vous devez rendre doit s'appuyer sur les grands axes définis ci-après.

2.1 Structures de données

Il est conseillé de choisir une structure de données très souple d'utilisation pour stocker le fichier en mémoire. On peut par exemple prendre une liste doublement chaînée des lignes du fichier (expliquer alors son intérêt), chaque ligne étant une chaîne de caractères au sens habituel :

```
typedef struct cellule {
        char * chaine;         /* une ligne = une chaine de caractere */
        struct cellule * prec; /* ligne precedente                    */
        struct cellule * suiv; /* ligne suivante                      */
        } * ListeDouble;
```

Ceci étant, il reste à rajouter des variables globales qui gèrent le curseur, le tampon, etc. :

```
unsigned c_ligne;              /* position en ligne du curseur        */
unsigned c_colonne;            /* position en colonne du curseur      */

unsigned t_ligne;              /* numero de la ligne mise dans le tampon */
unsigned t_colonne;            /* numero de la colonne correspondante    */
char * Tampon;                 /* tampon de ligne                        */
```

Décrivons l'utilité des structures introduites : `ListeDouble` implémente la structure de liste doublement chaînée ; `c_ligne` et `c_colonne` sont les variables globales qui indiquent à tout moment la position courante du curseur ; `Tampon` permet de sauvegarder la ligne en cours de modification pour une annulation éventuelle ; `t_ligne` et `t_colonne` sont les positions du curseur avant la modification courante.

2.2 Description d'une session interactive

Le programme que vous devez faire doit être interactif, au sens où l'utilisateur doit pouvoir faire des requêtes et obtenir des réponses. On pourra donc prévoir un petit langage de commandes permettant à l'utilisateur de passer ses demandes au programme. Le langage utilisé devra être très simple et pourra s'inspirer de l'exemple suivant (minimal).

Au lancement de `a.out` (par exemple), le programme affiche un *prompt*—nous avons choisi `edith>`—et se met en attente d'un ordre de la part de l'utilisateur. La première commande utile sera la commande `h` fournissant de l'aide sur les commandes disponibles. L'utilisateur peut désormais, au choix, visualiser le fichier en cours d'édition, le modifier, le sauvegarder, etc., comme dans l'exemple qui suit.

```
$a.out
edith> h
```

Commande	*Effet*
h	Pour obtenir cette aide
e <nom>	Chargement du fichier (fichier nom en option)
w <nom>	Sauvegarde du fichier (sous le nom en option)
l	Visualisation du fichier en cours d'édition
g x <y>	Positionnement du curseur à la ligne x (éventuellement à la colonne y) $ référence la fin, . la position courante
i texte	Insertion de texte à la position courante
a texte	Ajout de texte après la position courante
x <n>	Efface le (n) caractère(s) à la position courante
o texte	Ajouter du texte après la ligne courante

```
O texte     Ajouter du texte à la ligne courante
d           Effacer la ligne courante
p           Insertion du tampon de ligne après la ligne courante
P           Insertion du tampon de ligne à la ligne courante
y           Remplissage du tampon de ligne avec la ligne courante
u           Restitution de la dernière ligne modifiée
?           Donne la position courante du curseur
q           Quitter
```

```
edith> e myfic
"myfic" No such file or directory
edith> l
edith> ?
1, 1
edith> g 2 3
Not that many lines in file myfic (0 lines)!
edith> i mademoiselle...
edith> ?
1, 15
edith> g . 1
edith> i Bonjour,
edith> l
Bonjour, mademoiselle...
edith> o Che derange ?
edith> g . 1
edith> x 3
edith> i Je
edith> l
Bonjour, mademoiselle...
Je derange ?
edith> g . $
edith> a Il fait beau...
edith> q
No write since last change
edith> g 1
edith> y
edith> g $
edith> p
edith> g . 9
edith> x 12
edith> i monsieur
edith> g . $
edith> a  Comment allez-vous?
```

```
edith> l
Bonjour, mademoiselle...
Je derange ?
Bonjour, monsieur... Comment allez-vous?
edith> w
"myfic" 3 lines, 79 characters
edith> q
```

2.3 Travail minimal

Le travail minimal à effectuer est *grosso-modo* résumé dans l'exemple ci-dessus, Vous avez tout intérêt à décomposer au maximum les fonctions ou procédures à écrire pour que votre travail soit *modulaire*.

En résumé, il est demandé au minimum de réaliser :

- la déclaration des différents types, constantes et variables globales ;
- un interpréteur de commandes (voir exemple ci-dessus) ;
- la réalisation des commandes présentées dans l'aide de l'exemple minimal ;
- un compte rendu détaillé de votre travail incluant un rapport décrivant vos choix de structures et d'algorithmes, un *listing* suffisamment commenté.

2.4 Quelques détails sur les commandes

Les commandes à programmer ne nécessitent aucune explication plus détaillée, si ce n'est l'annulation de la dernière commande. En effet, cette opération peut être relativement complexe à réaliser en fonction de la puissance recherchée : si nous nous limitons cependant à annuler uniquement les opérations qui ont modifié (mais non supprimé) la ligne courante, alors la variable `Tampon` permet de restituer instantanément cette ligne avant modification. Si vous désirez néanmoins implémenter une annulation plus puissante, il faut savoir que sa conception nécessitera certainement beaucoup de réflexion et de temps...

2.5 Améliorations et options

Les améliorations et les options ne sont pas obligatoires, mais constituent un *plus* pour l'évaluation finale de votre travail. Vous êtes invités à faire part de vos propositions aux enseignants pour savoir dans quelle mesure elles sont réellement pertinentes (à moins que vous soyez un crac des éditeurs...).

Les options les plus simples à programmer sont celles qui autorisent la combinaison de fonctions précédemment écrites. Par exemple, si vous êtes capables de repérer la fin du mot sur lequel le curseur est positionné, il est relativement aisé, à partir des fonctions d'effacement et d'insertion de changer la fin de ce mot par un autre. Si vous savez rechercher un mot dans la ligne courante, vous pouvez alors enchaîner d'autres fonctions comme la substitution d'un mot (voire d'une chaîne de caractères quelconques) dans une ligne donnée, puis le remplacement d'un mot dans tout le fichier.

Une autre option particulièrement utile est la visualisation de la position du curseur à l'affichage du fichier (le mettre par exemple en inverse vidéo). La commande `?` affichera alors la ligne courante et son numéro.

Il peut arriver que l'on veuille insérer un fichier au milieu d'un autre en cours d'édition. Vous pouvez facilement réaliser cette option par de simples opérations d'insertion sur les listes.

Lorsque vous éditez un petit programme en C, il est pratique de disposer d'une commande qui, en supposant que le curseur soit positionné sur une accolade ouvrante, vous envoie directement sur l'accolade fermante correspondante (et vice-versa). Si le curseur n'est pas sur une accolade, alors cette commande l'envoie directement sur la première accolade ouvrante.

Enfin, l'option la plus intéressante (et qui est normalement compatible avec tout ce qui a été écrit auparavant) est le mode édition pleine page : vous trouverez une aide essentielle à cette "touche finale" dans le fichier `/usr/include/curses.h` (taper `man curses`, préférer l'option `-DMINICURSES` pour `cc`, afin d'améliorer les performances). Le mode pleine page fait que vous pourrez utiliser agréablement votre éditeur (et le distribuer ?).