Université Paris Diderot
L3 Informatique

Conduite de Projet
Année 2017-2018

VILAIN — (un) éditeur de texte

version 1.0

> *Un* ***éditeur de texte*** *est un logiciel destiné à la création et l'édition de fichiers textes. [...] Un éditeur de texte se distingue d'un traitement de texte par le fait qu'il est orienté lignes de code plutôt que paragraphes, et que les fichiers textes ne contiennent en général pas de mise en forme. [...] Les fonctionnalités les plus élémentaires d'un éditeur sont :*
> — *Ouvrir un fichier [...] ;*
> — *Ajouter du texte dans une ligne, ou des lignes dans un fichier ;*
> — *Ôter des caractères dans une ligne, ou des lignes d'un fichier ;*
> — *Rechercher/remplacer une chaîne texte [...] ;*
> — *Sauvegarder le fichier*
>
> — *https://fr.wikipedia.org/wiki/Éditeur_de_texte*

1 Introduction

Le but de ce projet est d'implémenter en langage C un éditeur de texte. Les éditeurs de texte, comme `vi`, `emacs` ou `nano`, sont des outils essentiels, notamment en développement logiciel. L'implémentation d'un tel logiciel est par ailleurs intéressante du point de vue de la conduite de projet, la diversité des fonctionnalités attendues d'un éditeur de texte imposant une certaine modularité.

Les éléments de spécification qui suivent se divisent en un *projet minimal* (dont on s'attend à ce qu'il soit implémenté en intégralité) et des suggestions d'extensions, dont l'implémentation pourra consolider votre note.

Le projet est à faire en trinômes. La progression régulière de votre travail au cours du semestre, de même que le bon usage des outils et des pratiques de conduite de projet présentés dans ce cours, seront au moins aussi importantes pour votre évaluation que le résultat final.

Des informations pratiques utiles vous seront fournies sur la page Moodle du cours. [1]

2 Projet minimal

Interface utilisateur On s'attend à ce que l'éditeur soit en *mode texte* (pas d'interface graphique). Vous aurez donc besoin d'une *bibliothèque externe* permettant de réaliser une telle interface, pouvant s'afficher dans tout terminal. On vous demande ici d'utiliser la bibliothèque `ncurses`, largement utilisée par des projets libres (`make menuconfig` du noyau Linux, `screen`, `mutt`, `irssi`, ...) et permettant d'implémenter des éléments tels que menus, barres, fenêtres... La présentation de cette bibliothèque est disponible sur la page du projet `ncurses` [2].

1. http://moodlesupd.script.univ-paris-diderot.fr/course/view.php?id=6660

Buffer L'exécution du projet affichera un *buffer* (structure de données contenant du texte). A l'initialisation du programme ce buffer pourra être vide, ou contenir le texte d'un fichier passé en argument.

Chargement et sauvegarde de fichier On doit pouvoir, même plusieurs fois pendant une session d'utilisation de l'éditeur, charger un fichier dans le buffer et réciproquement, sauvegarder le buffer.

Défilement (scrolling) Le contenu du buffer pouvant dépasser les dimensions de la fenêtre de terminal, on doit pouvoir faire défiler ce contenu au moyen du clavier (cf. également le *line wrapping* ci-dessous).

Curseur, position, édition Un curseur indiquera en surbrillance la position courante dans le buffer. L'utilisateur pourra insérer ou supprimer du texte à cette position. Le curseur doit bien sûr pouvoir être déplacé au moyen du clavier (et éventuellement au moyen de la souris, ce qui est possible avec `ncurses`). Le programme affichera à tout moment les numéros de ligne et de colonne courantes du curseur, par exemple dans une barre sous le buffer.

Sélection L'utilisateur doit pouvoir sélectionner une *région* du buffer (une zone délimitée par un début et une fin). On peut par exemple permettre à l'utilisateur de définir une *marque* : la région sélectionnée est le texte entre le curseur et la marque. Le texte de cette région pourra alors être supprimé ou écrasé en insérant de nouveaux caractères.

Copier, coller Le texte sélectionné doit pouvoir être copié (copy) ou coupé (cut). Un texte copié ou coupé doit pouvoir être inséré à la position courante du curseur (paste).

Raccourcis clavier Les fonctionnalités de chargement/sauvegarde, copier/coller, etc., seront mises en œuvre au moyen de raccourcis clavier choisis avec discernement.

Configuration L'éditeur doit être configurable. Au minimum, l'utilisateur doit pouvoir modifier depuis la session courante les raccourcis clavier ainsi que les options de *line wrapping*, et sauvegarder ces éléments dans un fichier de configuration.

Line wrapping Lorsqu'une ligne du buffer dépasse la largeur de la fenêtre du terminal, on doit pouvoir gérer la manière dont elle affichée. Sans que le buffer soit pour autant modifié, une telle ligne peut être affichée sur plusieurs lignes, ou encore interrompue : on doit alors pouvoir la faire défiler horizontalement. Le choix entre ces deux conventions est une option de configuration. Par ailleurs, il doit exister une commande pour réduire les lignes dépassant un certain nombre de caractères (column width) : le buffer est alors modifié. Les lignes seront réduites par insertion d'un saut de ligne après le dernier mot apparaissant dans les limites du column width, de même pour chaque nouvelle ligne créée par cette opération. Une option doit être disponible pour forcer ce comportement pendant l'insertion (cf. `auto-fill-mode` sur `emacs`).

3 Extensions possibles

Afin de consolider votre note, voici une liste d'extensions possibles pour votre éditeur. Cette liste n'est pas exhaustive, et vous pouvez nous soumettre vos propres idées d'améliorations ou

de nouvelles fonctionnalités. Les éléments que vous aurez traités devront être mentionnés dans votre rapport.

Menus Des menus permettront également d'accéder à toutes ou partie des fonctionnalités de l'éditeur.

Langage de commandes Même lorsqu'il existe déjà des raccourcis clavier ou des menus pour accéder à certaines d'entre elles, les fonctionnalités de l'éditeur pourront aussi être mises en œuvre au moyen d'un mini-langage de commandes (par exemple `open fichier` chargera le fichier `fichier` dans le buffer). Ce mini-langage sera à tout moment invoquable à l'aide d'un raccourci clavier, son prompt étant affiché par exemple sous le buffer.

Tubes (pipes) L'utilisateur pourra invoquer une commande shell avec pour entrée standard le texte du buffer (ou une sélection), et insérer la sortie d'une commande shell dans le buffer – éventuellement celle de la première.

Recherche de chaîne L'utilisateur pourra rechercher une chaîne de texte dans le buffer, en respectant ou non sa casse (case-sensitiveness) selon son choix. Il devra pouvoir facilement passer d'une occurence à l'autre de la chaîne.

Recherche d'expression Outre la possibilité de rechercher une chaîne littérale, l'utilisateur pourra également rechercher dans le buffer une expression régulière (regex). Pour gérer de telles expressions, nous vous recommandons d'utiliser une bibliothèque externe telle que PCRE [3].

Remplacement de chaîne ou d'expression L'utilisateur pourra remplacer une ou toutes les occurences d'une chaîne ou d'une expression par une nouvelle chaîne.

Buffers multiples L'éditeur pourra gérer plusieurs buffers, les visualiser séparément ou simultanément (au moyen de splits horizontaux ou verticaux).

Coloration syntaxique Les fichiers source codés dans certains langages seront affichés avec une coloration syntaxique, de manière configurable. Il vous faudra implémenter un module associant des éléments de syntaxe à des catégories syntaxiques, ces catégories étant associées à des couleurs.

Undo/redo Une commande permettra d'annuler les dernières modifications, une autre permettra de les rétablir.

Undo/redo, variante persistante Outre la possibilité d'annuler ou de rétablir les dernières modifications, celles-ci seront sauvegardées en même temps que le buffer, dans un fichier séparé. Le chargement du fichier permettra alors de charger simultanément la liste des modifications effectuées à la session précédente.

Récupération Le programme s'efforcera de limiter les pertes de données qui pourraient résulter de son interruption inattendue. Une implémentation naïve de cette fonctionnalité est de sauvegarder automatiquement le buffer ou une copie de celui-ci à des intervalles de temps réguliers. Si le buffer est très gros, cette implémentation est cependant sous-optimale en matière d'espace disque. Une implémentation plus fine est d'enregistrer uniquement les modifications du buffer.

4 Évaluation de votre projet

La notation prendra en compte tous les points suivants.

Utilisation des outils standard Les cours magistraux présenteront tout au long du semestre des outils standard de développement (Makefile, git, etc.) qui devront être *effectivement* utilisés pendant le projet. Cette utilisation va de pair avec une séparation des différentes parties du projet en modules et librairies séparées.

Par exemple, une utilisation frauduleuse de *Makefile* pourrait être un appel unique à un script externe.

Modularité et tests Les différents modules du programme devront être séparés en différents fichiers-sources (`.c`) et en fichiers d'en-tête (`.h`) utilisés à bon escient. Vous préparerez aussi des tests unitaires de chaque fonction importante sur des exemples choisis, et des tests d'acceptation des fonctionnalités globales du projet. Ces tests seront inclus dans votre dépôt git.

Contrôle continu Le travail doit être réalisé tout au long du semestre grâce à un répertoire *git* pour chaque trinôme. Votre répertoire doit être `privé`, mais dès les premières semaines, vous donnerez accès à ce répertoire à tous les enseignants. Ceci permettra notamment de garder la trace et les dates de contributions de chaque étudiant. Vous pouvez choisir entre le service d'hébergement GitLab offert par l'université `http://moule.informatique.univ-paris-diderot.fr:8080/` ou tout autre service tiers, toujours à la condition de conserver votre répertoire privé et d'y donner accès à vos enseignants.

Diagramme d'architecture Vous devrez réaliser, le plus tôt possible, un diagramme décrivant l'architecture de votre projet – au moins le graphe de dépendance de ses modules, étiqueté par leurs noms et une description synthétique de leurs fonctionnalités respectives. Ce diagramme servira de base à votre soutenance.

Journal Chaque trinôme tiendra à jour un *journal de développement* inclus dans son répertoire git. Chaque semaine, vous indiquerez dans votre journal vos objectifs pour la semaine à venir, et vos résultats de la semaine passée. Dans la mesure du possible, faites en sorte que chaque commit corresponde à une tâche précise et soit effectué par la personne ayant effectué la tâche. Un dépôt git n'est pas limité à du code, n'hésitez pas à y inclure d'autres documents qui nous permettront de suivre votre travail, par exemple : des courts résumés de prise de décision, des notes sur le fonctionnement des bibliothèques que vous allez utiliser, la structure que vous projetez pour vos modules, etc.

Lisibilité et clarté Il est fortement recommandé d'indenter systématiquement, d'éviter les lignes trop longues (> 80 caractères), de choisir des noms parlants pour les variables et fonctions, et de commenter le code souvent, mais en restant concis. Vous supprimerez également toute duplication de code, suivant le principe DRY [1] ("Don't Repeat yourself").

Références

[1] Don't repeat yourself. https://en.wikipedia.org/wiki/Don%27t_repeat_yourself.

[2] Ncurses (page officielle). http://www.gnu.org/software/ncurses/ncurses.html.

[3] Pcre (wikipédia). https://en.wikipedia.org/wiki/Perl_Compatible_Regular_Expressions.