2023

RAPPORT PROJET MYGREP

PROGRAMMATION ALTERNANTE

LINA MERAZGA

UNIVERSITE JEAN MONNET | FACULTE DES SCIENCES ET TECHNIQUES

INTRODUCTION

Dans le cadre du programme en « L2 Informatique Alternant » de l'Université Jean Monnet, ma mission était de réaliser un projet pour le cours de programmation.

Le but de ce projet étant de faire notre propre version de la commande grep en langage C. La commande grep cherche la chaîne de caractère « <expression> » à l'intérieur des fichiers ou des flux de données et affiche les lignes correspondante. Ce programme doit notamment implémenter les options suivante : -c -l -L -i -v -h -H -A -B -n -AB -e.

Nous allons dans un premier temps faire une rétrospective sur l'ensemble des itérations, puis dans une seconde partie, nous verrons le choix des structures utilisées et pourquoi.

TABLE DES MATIERES

RETROSPECTIVE SUR L'ENSSEMBLES DES ITERATIONS

PREMIERE ITERATION

Au commencement, pour ce qui est de la première itération après analyse de la fonction grep, j'ai élaboré une liste de tâches à réaliser, une liste non-exhaustive, car au fur et à mesure des itérations, de nouvelles tâches, auxquelles je n'avais pas pensée au départ, devaient être rajouté où supprimer pour le bon fonctionnement du programme.

C'est ainsi, que j'ai décidé, de réaliser tout d'abord des fonctions de recherche simple sans option sur l'entré standard et dans un fichier afin d'avoir une base pour la suite du projet.

Dans un même temps quelques recherches et tests de la fonction getotp ainsi que optind et optarg afin de mieux appréhender la suite.

SECONDE ITERATION

Après la première itération, j'ai pu améliorer les fonctionnalités de base en améliorant ma recherche de motif sur un ou plusieurs fichiers.

En effet grâce à la gestion de la ligne de commande, j'ai pu traiter quatre cas diffèrent, dont deux fonctionnels : recherche de motif sur l'entré standard, recherche de motif sans option en parcourant plusieurs fichiers, et deux cas non-fonctionnels : recherche sur l'entrée standard avec option et recherche de motif avec option et plusieurs fichiers.

De cela a pu en émerger la fonction de recherche avec option. En effet j'ai pu implémenter quelques options tel que l'option -c, -l, -n, -h et -v.

Cependant ces options fonctionnaient indépendamment les unes des autres.

Par suite en a émaner une fonction de récupération (verif_option) des options qui n'étais pas fonctionnelle lors de cette itération car l'utilisation de getopt, optind, et optarg n'étais pas acquis totalement.

Enfin, cela a pu me donner une idée des structures que j'aurais potentiellement besoin lors de la suite du programme.

Ainsi, j'ai créé une structure Option regroupant les options sous forme de tableau d'options regroupant le caractère de l'option et son argument si nécessaire. Puis une structure Correspondance prenant en compte le nom du fichier un numéro de ligne et la ligne.

TROISIEME ITERATION

A la fin de la seconde itération, mon programme fonctionnait pour le cas de recherche de motif avec une option sans option et sur l'entrée standard.

Pour donner suite à cela, j'ai dû revoir ma structure option. Cette structure est accessible plus facilement que le tableau d'option et toute option ne possède pas un argument d'où le changement.

Cela m'a permis de gérer autrement la ligne de commande en récupérant les options grâce à getopt et en mettant à jour la structure option_g

Lorsque tous ces objectifs ont été atteint cela m'a permis de me poser des questions sur la compatibilité des options et comment les traiter en cas de fonction non-compatible.

A ce moment-là, j'ai pu réaliser des tests sur la commande grep et au fur et à mesure de cela en a découler la fonction verif_compatibiliter qui mettais à jour la structure option en désactivant les options activé et non-compatible avec les options activé qui sont prioritaire.

Pour finir, la suite me paraissait compliqué, car ayant avancé dans mon code et n'ayant pas pensé à certain cas cela a ralentie ma progression et toute idée de ma part bloquait mon avancée. Je n'arrivais pas à voir la suite du code avec mon code actuel.

QUATRIEME ITERATION

Pour débuter cette itération j'ai dû revoir une partie de mon code et faire davantage de recherche sur la commande grep.

De ce fait j'ai pu constater que les options étaient divisées en plusieurs catégories : motif sélection et interprétation, divers, control de sortie, contrôle de contexte.

C'est pourquoi j'ai décidé de ne pas implémenter les options dans une seul et même fonctions et d'éclatée mon code plus ou moins en fonction de ces catégories.

J'ai décidé de gérer dans un premier temps les options qui agissent sur le motif et les lignes (catégorie interprétation et manipulation des motifs) telles que le -i et le -e ainsi que la recherche en début, fin, au à l'intérieur de la ligne. En effet, de cela en a découler le module liste qui contient une structure StringList et différentes fonctions associées.

Cette structure va aussi être bénéfique pour stocker les différents fichiers et pour stocker les lignes correspondantes de l'option B.

Par la suite, grâce à cela, j'ai pu gérer ma recherche de motif dans une ligne et m'a permis d'intégrer à ma fonction de recherche de motif, les options v, A et B.

Du fait de tous ces changements quelques fonctions ont dû être supprimées où améliorer. En effet, aux lieux de gérer 4 fonctions de recherche motif, je n'en gère plus qu'une, à la place nous avons 3 fonctions de recherche : une fonction de recherche dans le fichier, une fonction pour le motif qui aura besoin d'une fonction de recherche dans une ligne.

A cette étape, on a uniquement l'affichage de l'option -A -B -v et le nombre de matchs des lignes correspondantes, ainsi que l'option -i et -e dans la recherche de ligne. Ces 3 fonctions s'imbriquent et permettent le bon fonctionnement de la recherche.

Par conséquent la Structure option_g évolue, prenant en compte les listes de fichiers et de motifs en paramètre, mais au aussi un paramètre affiche_ligne qui permettra d'indiquer si nous allons avoir l'affichage des lignes correspondante ou autre (nom de fichier, nombre de matchs).

Pour finir la fonction verif_option évolue en même temps et gèrent à présent tous les paramètres de la ligne de commande.

CINQUIEME ITERATION

Pour finir lors de cette dernière itération le traitement de toutes les options et de l'affichage du programme mygrep fonctionne.

Après avoir traité un certain nombre d'options qui ne concernait pas l'affichage des lignes, j'ai pu implémenter les options d'affichage.

Pour cela, je me suis inspiré de la catégorie contrôle de sortie. J'ai ensuite divisé en deux les type d'options : celle qui affichait les lignes avec le nom de fichier et le numéro de ligne (-h -n) et celle qui n'affichais pas de ligne, mais uniquement les noms des fichiers ou le nombre de correspondances (-L -l -c), ainsi de les coder dans leur fonction respective : recherche_motif et recherche_fichier

Pour finir quelques tests et correction si besoin. En effet, il a fallu rajouter la recherche de motif pour les lignes contenant uniquement le motif. Ainsi que l'option -H qui affiche une aide d'utilisation du programme. Ainsi que la documentation du code.

CHOIX DES STRUCTURES UTILISEES

Tout d'abord, le code a été divisé en différents modules : option, fonction, liste, recherche, main.

1) Liste

Le module liste définit une structure de données StringList qui gère une liste de chaînes de caractères. Ainsi, le module fournit plusieurs fonctions pour manipuler les listes : supprimer, ajouter, afficher, vider, libérer de la mémoire.

2) Option

Le module option définit une structure option_g et une fonction permettant de vérifier et traiter les options spécifiées en ligne de commande ainsi que de stocker les fichiers et motifs.

3) Recherche

Le module recherche contient les fonctions permettant de rechercher les motifs spécifiés en parcourant les lignes d'un fichier ou sur l'entrée standard.

4) Fonction

Le module fonction contient des fonctions auxiliaires aux modules de recherche et agissant sur les lignes.

5) Utilitaire

Le module utilitaire contient une fonction d'utilisation du programme mygrep.

6) Main

Le module main est le programme principal qui va permettre la recherche à un ou plusieurs fichiers ou sur l'entrée standard et affiche les correspondances sur l'entrée standard.

STRUCTURES UTILISEES DANS LE CODE

STRUCTURE STRINGLIST

La structure StringList contient un tableau de pointeur de caractère et sa taille. Elle va nous permettre de regrouper plusieurs chaînes de caractères en une seul entité. Cela va faciliter la gestion et la manipulation des données dans le code.

Cette structure offre plusieurs fonctions pour manipuler les listes telles que : la libérer de la mémoire, ajouter, supprimer et vider une liste

En effet, cette structure va servir pour stocker les noms de fichier et les motifs (si plusieurs) qui vont être intégrés à la structure option_g afin de pouvoir y avoir accès facilement lors de la recherche.

De plus, l'option B est aussi gérée par une StringList. Elle est utilisée pour stocker temporairement les lignes correspondantes avant d'être affichée lorsqu' une correspondance est trouvée.

GETOPT ET SWITCH CASE

Afin de pouvoir gérer les options passées en ligne de commande, j'ai opté pour l'utilisation de la fonction getopt de la bibliothèque standard. Elle est utilisée pour analyser les arguments passés par l'utilisateur lors de l'exécution et facilite le traitement des options et de leurs arguments associés.

Le switch case va permettre de gérer les différentes options détectées par getopt et exécuter le code correspondant à chaque option. Ici, le switch case va mettre à jour la structure option_g afin d'activer les options présente en ligne de commande.

STRUCTURE OPTION_G

La structure option_g est utilisée pour stocker les paramètres spécifiés en ligne de commande. Elle contient les différentes options et des variables pour leurs arguments (si besoin). Elle continent aussi la StingListe des motifs à rechercher et la StringListe des fichiers à parcourir.

J'ai choisi cette structure, car elle permet de regrouper toutes les informations nécessaires concernant les paramètres du programme.

Les options sont de type booléen, cela permet de les représenter de manière plus simple et claire et rend le code plus compréhensif. En effet, les booléens permettent de représenter des état true ou false qui vont permettre d'indiquer l'option est activé ou désactivé.

GETLINE

Afin de parcourir les lignes d'un fichier, j'ai préféré utiliser la fonction getline de la bibliothèque standard. Elle est utilisée pour lire une ligne de texte depuis un flux (stdin, un fichier) et la stocker dans un tampon.

Contrairement à la fonction fgets, getline alloue dynamiquement la mémoire nécessaire pour stocker la ligne, ce qui permet de gérer des lignes de taille variable sans se soucier de la limite de la taille fixe d'un tampon.

CONCLUSION

Pour conclure nous pouvons constater que le projet a mal été abordé dès le départ, il aurait fallu faire davantage de recherche afin de traiter le sujet différemment afin de ne pas revenir sur la construction du code au fur et à mesure des itérations.

Cependant cela a bien été gérer après une connaissance plus approfondie du sujet et a permis une réorganisation du code, afin qu'il soit le plus optimisé possible et fonctionnels avec toutes les options. En effet en fonction des types d'option et de leur fonctionnalité le code a pu être découper et codé en conséquence. De plus cela a apporté une logique dans le programme.

De plus les structures utilisées sont accessibles facilement, et permette une meilleure lisibilité du code. En effet grâce à des structures simple et précise cela a diminué le nombre de ligne et a permis d'optimiser le code.

En résulte un programme ressemblant à la commande grep, contenant différents modules avec des taches précise ainsi qu'un code est claire et commenté notamment grâce à doxygen.