- BOUGHANEM Rayan
- BOU ZIED ELIE

Projet d'algorithmique et de structures de données

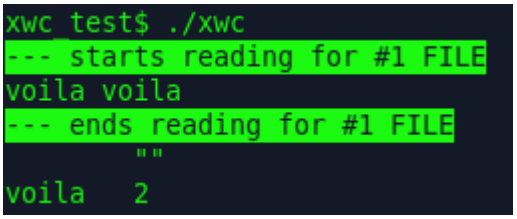
Comptage exclusif de mots

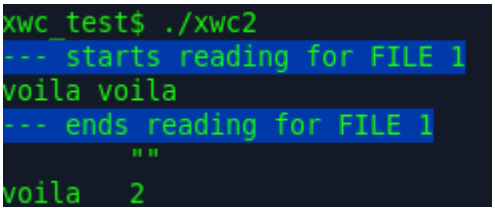
xwc

SOMMAIRE :

1) Généralités :

Ce rapport se concentre uniquement sur les exemples d'exécution des deux exécutables. La distinction entre les deux, dans le cas d'une comparaison, se fera à l'aide du nom de l'exécutable sur la ligne de commande : `xwc` pour l'original et `xwc2` pour la tentative de reproduction qui nous appartient.





Comme indiqué dans le support du sujet de TP, le projet consiste en un programme `xwc` qui compte les occurrences des mots apparaissant de façon exclusive dans les fichiers sur lesquels il est exécuté. Il n'y a aucune limite sur la taille des mots lus, tant que ces derniers n'appartiennent pas à la classe `isspace`. Avec un affichage fidèle à celui du modèle de l'exécutable fourni sur la plateforme UniversiTICE et une aide qui précise cela avec les bons détails.

Le projet consiste en un dossier nommé `boughray_projet` lui-même constitué de :

- xwc_test:

- makefile : fichier qui produit l'exécutable afin de lancer le programme.

- main.c : code source pour la gestion des fichiers, des options et l'utilisation d'autres modules.

- holdall :

- holdall.h : fourre-tout qui permet la mémorisation d'une liste de références d'objets quelconques, l'exécution de fonctions sur les objets repérés par les références dans la liste selon l'ordre dans lequel les références y figurent ainsi que la réorganisation de la liste.

- holdall.c : implémentation en C des fonctions spécifiées par le `holdall.h`.

- bst:

- bst.h : partie interface d'un module polymorphe pour la spécification ABINR du TDA AbinR(T) avec implémentation et gestion selon les arbres AVL suite aux TP3 et TP5.

- bst.c : implémentation en C des fonctions spécifiées par le `bst.h`.

Comme tout bon développeur, le programme fourni n'implémente pas toutes les fonctions qui ont été demandées (voir `./xwc --help` ou `./xwc -?`). Dans le cas des options, comme indiqué dans l'aide du programme, il est important de suivre la sémantique décrite. Autrement dit, les éventuelles options doivent impérativement précéder tout argument (fichiers) qui fait référence à un fichier. Voici quelques exemples précédés d'une clarification :

```
xwc_test$ ./xwc2 --usage
Usage: xwc [OPTION]... [FILE]...
```

[] pour dire que le champ est optionel

Exemple 1 : Même si `-l` est une option fonctionnelle, elle est automatiquement considérée comme un fichier.

```
xwc_test$ ./xwc2 x1.txt  x2.txt -l
xwc: Can't open for reading file '-l'.
```

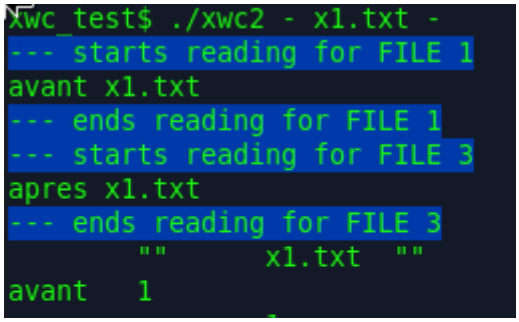
Exemple 2 : Cet exemple est équivalent à celui d'avant.

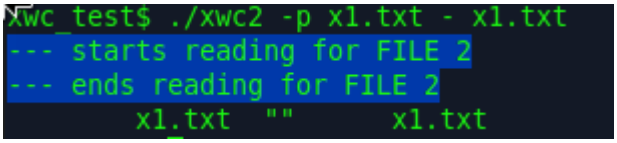
```
xwc_test$ ./xwc2 - - -l
--- starts reading for FILE 1
ca
marche
--- ends reading for FILE 1
--- starts reading for FILE 2
bien
mais
--- ends reading for FILE 2
xwc: Can't open for reading file '-l'.
```

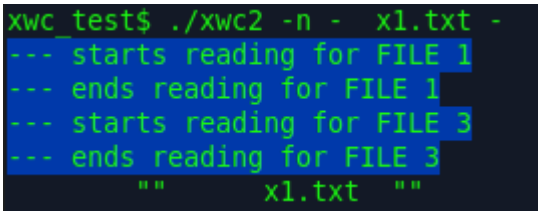
Autres possibilités : L'option « - » est certes une option qui peut être considérée comme un fichier, mais c'est aussi une option. On peut donc tenter de faire des combinaisons du genre. Voici les résultats suivants :

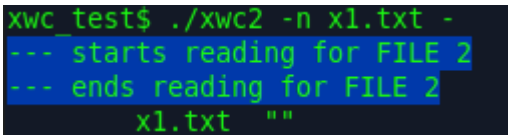
```
xwc_test$ ./xwc2 -n -a - - x1.txt
--- starts reading for FILE 1
--- ends reading for FILE 1
--- starts reading for FILE 2
--- ends reading for FILE 2
       ""      ""      x1.txt
a                      1
eee                    1
```

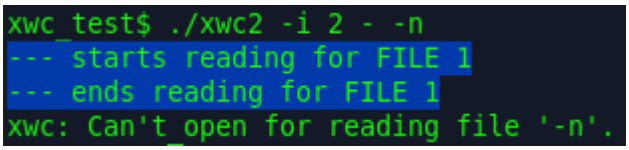
```
xwc_test$ ./xwc2 -p x1.txt - -l
--- starts reading for FILE 2
--- ends reading for FILE 2
xwc: Can't open for reading file '-l'.
```











Nous pensons donc avoir couvert toutes les possibilités de lecture valide entre un vrai fichier et l'option « - », qui fait référence à la lecture depuis l'entrée standard.

Aucune modification n'a été faite dans le dossier holdall et hashtable ; ils restent donc identiques à ce qui a été fourni dans l'espace de cours. En revanche, l'implémentation de bst a été en partie réalisée par les responsables pédagogiques et par nous-mêmes.

2) **Approche :**

Notre première approche de façon générale était la bonne, et notre première pensée s'est portée sur les arbres AVL dès lors que ces arbres représentent l'objet de 80 %* du programme pour le semestre courant.

L'idée était simple : nous avions besoin des occurrences, de la source (fichier) du mot et du mot lui-même. Ensuite, nous devions parcourir tous nos fichiers pour identifier chaque mot et vérifier si un mot identique avait déjà été stocké dans notre structure. Deux cas s'ensuivent automatiquement :

1. Nous n'avons jamais rencontré un mot pareil auparavant ; donc, nous allons le stocker dans notre structure, enregistrer sa provenance (source) et mettre son occurrence à 1.
2. Nous avons déjà rencontré le mot, et donc, nous le retirons de la structure s'il provient d'une source différente ou nous incrémentons ses occurrences autrement.

Cette solution fonctionne parfaitement pour un fichier ou pour deux. Les problèmes commencent lorsque nous avons 3 fichiers ou plus. En effet, si, par exemple, un mot est commun aux 3 fichiers, lors de la lecture du premier fichier, le mot est ajouté ; lors de la lecture du second, il

est retiré avant d'être ajouté une fois de plus lors de la dernière lecture. Cela pose donc un problème majeur, d'autant plus que le projet s'intitule « Comptage exclusif de mots ».

La solution au problème a donc été de simplement attribuer une valeur spéciale à la source (qui normalement avait le numéro du fichier lu). Le choix s'est porté sur -1 comme valeur spéciale (nous aurions très bien pu mettre 0 ou un autre champ qui permet de savoir si c'est exclusif ou pas). Cela nous permet donc d'avoir une façon de distinguer les mots déjà vus et simplifie le programme. À la lecture d'un mot, on peut se permettre de vérifier en premier temps si le mot est présent dans notre structure ou pas, avec donc deux possibilités :

1. Le mot n'est pas dans la structure ; on peut donc l'ajouter comme dans la première approche sans se poser de question.
2. Le mot est déjà dans la structure ; on va donc vérifier sa source. Si elle est égale au numéro du fichier en cours de lecture, on incrémente ses occurrences ; dans le cas contraire, on affecte notre valeur spéciale (-1) au champ afin d'anticiper la prochaine fois qu'on le rencontre et de gérer notre affichage.

Un exemple est réalisé sur les fichiers suivants :
On voit très clairement que les mots exclusifs à chaque fichier sont, au final : « a 1 2 d 3 ».
Cela se fait comme suit sur les structures BST, par exemple :

```
xwc_test$ cat < exemple1.txt
a b 1
xwc_test$ cat < exemple2.txt
b c 2
xwc_test$ cat < exemple3.txt
c d 3
```

Début

On commence avec un arbre vide.

Lecture du premier fichier `exemple1.txt` (source = 1) :

Ajout de {mot = a, occurrence = 1, source = 1}
Ajout de {mot = b, occurrence = 1, source = 1}
Ajout de {mot = 1, occurrence = 1, source = 1}

Lecture du second fichier `exemple2.txt` (source = 2) :

Modification du champ source de b, car il est déjà présent et a une source différente de la courante : {mot = b, occurrence = 1, source = -1}
Ajout de {mot = c, occurrence = 1, source = 2}
Ajout de {mot = 2, occurrence = 1, source = 2}

Lecture depuis le fichier `exemple3.txt` (source = 3) :

Modification du champ source de c, car il est déjà présent et a une source différente de la courante : {mot = c, occurrence = 1, source = -1}
Ajout de {mot = d, occurrence = 1, source = 3}
Ajout de {mot = 3, occurrence = 1, source = 3}

Fin

Au final, on a donc :

- En vert : les mots qui sont dans l'arbre et qui sont affichés.
- En rouge : les mots qui sont dans l'arbre et qui ne sont pas affichés.

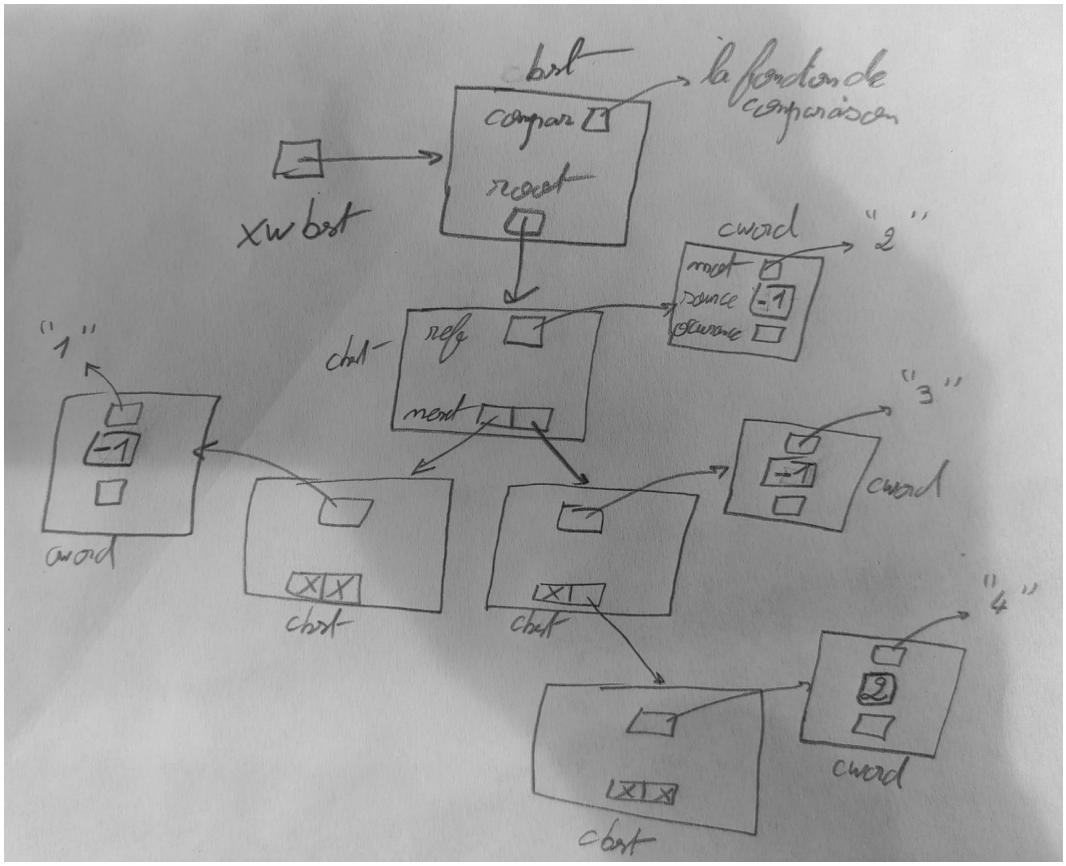
Une précision doit être apportée concernant les éléments en rouge. En effet, dire que le mot est dans l'arbre mais qu'il n'est pas affiché pourrait laisser penser que tous les mots lus seront forcément au final dans l'arbre mais ne seront pas affichés. Ceci est vrai uniquement dans le sens où un mot de `exemple1.txt` comme « b » représente le même mot que le « b » qui est dans `exemple2.txt`. Cependant, techniquement dans la structure, c'est le « b » de `exemple1.txt` qui est présent, et non celui de `exemple2.txt`. Donc, le « b » du second fichier n'est pas lui-même inséré dans la structure, mais on a son équivalent au sens linguistique. Des exemples plus clairs suivront après la partie implémentation.

Le processus décrit ci-dessus est le même de façon générale avec l'approche des tables de hachage (chose dont on s'est rendu compte quelques jours avant le rendu du projet). Les seules petites différences sont les appels aux fonctions des deux modules dans le main.c, en fait au lieu de faire une recherche dans les AVL, celle-ci est faite sur la table de hachage et pareillement pour l'appel d'ajout.

Laissons place à un exemple des deux ajouts, pour simplification on prendra deux fichiers dont le contenu est comme suit :

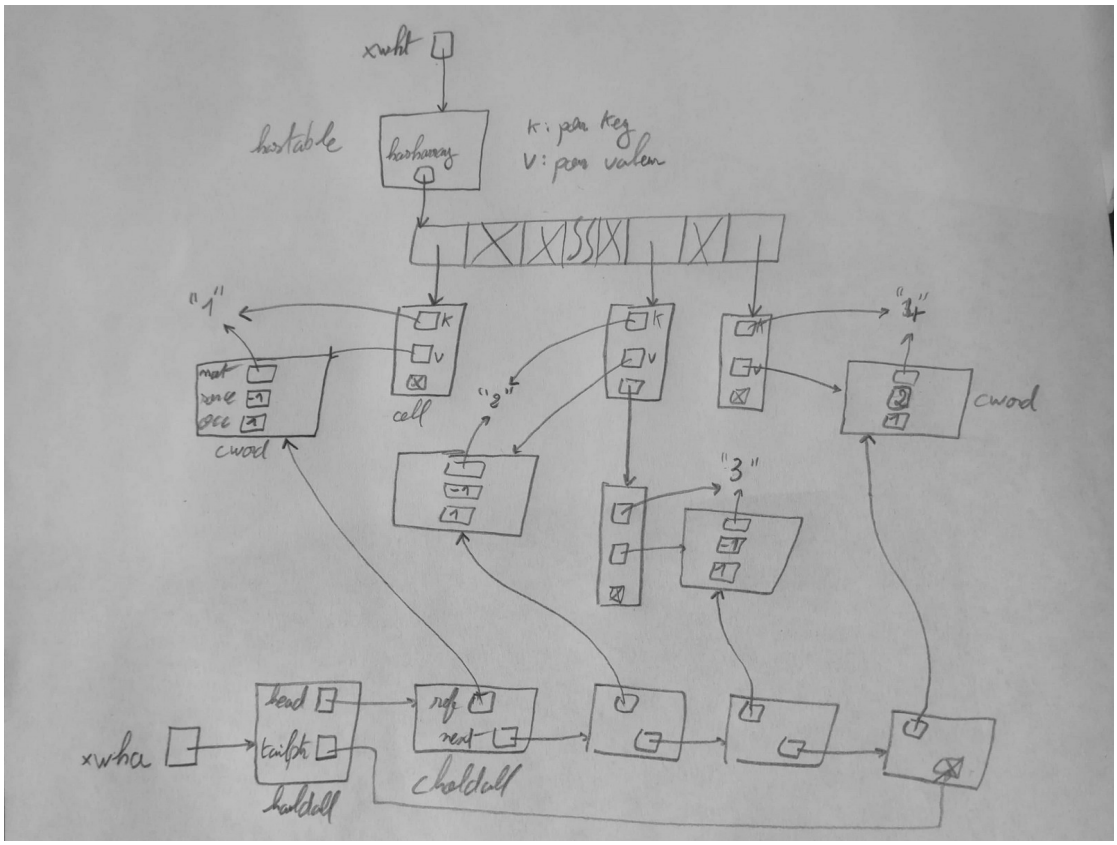
```
xwc_test$ cat > avl.txt
1 2 3 4
xwc_test$ cat > avl2.txt
1 2 3
```

selon la logique décrite en haut on devrait uniquement avoir le mot 4 comme mot exclusif.



On voit clairement que le seul survivant qui aura l'honneur d'être affiché est le mot « 4 », dû au fait que c'est le seul qui a une source différente de la valeur spéciale. De plus, il est à noter que le module BST effectue un traitement selon les spécifications sur les AVL, ce qui implique donc diverses rotations afin de garder récursivement un équilibre entre la hauteur des deux sous-arbres gauche et droit, d'où le fait que l'arbre résultant n'est pas un peigne droit. Dans notre cas, une rotation gauche a été effectuée. Le développement de notre projet a été fait comme si on n'avait pas accès aux fichiers bst.c (de même pour hashtable.c et prochainement holdall.c), c'est-à-dire qu'on était uniquement des utilisateurs des fonctions fournies sur bst.h, hashtable.h et holdall.h.

Le même exemple est fait selon la gestion par les tables de hachage, cette fois-ci avec le holdall qui permet de comprendre le résultat de l'affichage (on va omettre les détails de quelques champs pour la visibilité des dessins) :



Au final, le rôle du holdall est assez simple, grâce à une fonction d'affichage qui est appliquée à toute la chaîne du holdall. Seuls les mots dont la source est différente de la valeur spéciale sont affichés sur la sortie standard, dans notre cas le mot « 4 » du fichier 1.

```
xwc_test$ ./xwc2 -w avl-binary-tree avl.txt avl2.txt
        avl.txt avl2.txt
4       1
xwc_test$ ./xwc2 -w hash-table  avl.txt avl2.txt
        avl.txt avl2.txt
4       1
```

3) Implementation :

le main se repartie en 3 partie majeure : l

-La gestion des options

-Le traitement des lectures

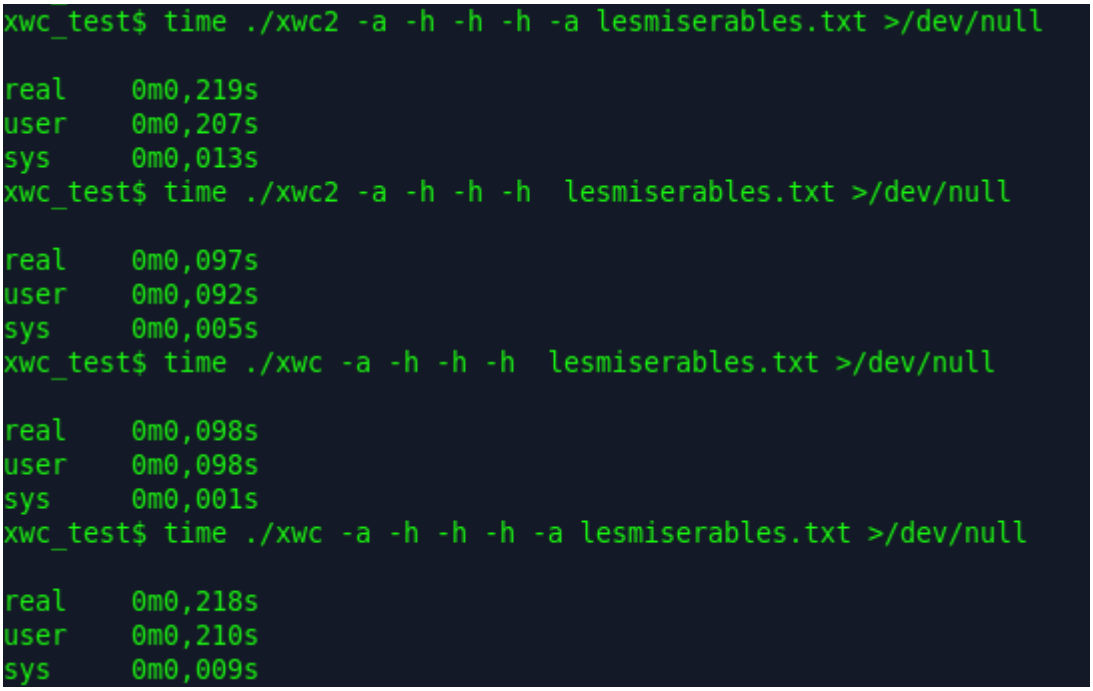
-Le post-traitement

3.1) Les options :

Le traitement des options se fait avec une simple boucle `for` qui parcourt `argv` afin de chercher les options et leurs éventuels arguments. La boucle parcourt les arguments saisis sur l'entrée standard à la recherche de toute chaîne de caractères qui commence par un « - » suivi d'un caractère; elle est donc considérée comme une option. Dans le cas contraire, l'arrêt de l'exécution de la boucle est effectué grâce à l'instruction `break`, également valable pour la lecture depuis l'entrée standard avec uniquement « - ».

```
137   //--------------------GESTION DES OPTIONS----------------------
138   for (int i = 1; i < argc; i++) {
139     char *arg = argv[i];
140     if (arg[0] != '-' || options_ended) {
143     if (strcmp(arg, SHORT_HELP) == 0 || strcmp(arg, LONG_HELP) == 0) {
146     } else if (strcmp(arg, USAGE) == 0) {
149     } else if (strcmp(arg, VERSION) == 0) {
152     } else if (strcmp(arg, HASH_TABLE) == 0) {
155     } else if (strcmp(arg, BINARY_TREE) == 0) {
158     } else if (strcmp(arg, LEX_SORT) == 0) {
161     } else if (strcmp(arg, NUM_SORT) == 0) {
164     } else if (strcmp(arg, REVERSE_SORT) == 0) {
167     } else if (strcmp(arg, CAP_SORT) == 0) {
170     } else if (strcmp(arg, PUNCTUATION) == 0) {
173     } else if (strcmp(arg, CUT) == 0) {
203     } else if (strcmp(arg, SORT) == 0) {
230     } else if (strcmp(arg, WORD_PROCES) == 0) {
251     } else if (strcmp(arg, NO_FILE) == 0) {
256     } else {
262   }
```

Chaque condition sert à activer les booléens associés aux options ou à les désactiver en fonction des cas, mais aussi à gérer les priorités entre les différentes options, comme celle du tri par exemple, mais aussi l'enchaînement des options comme suit :



De façon générale, c'est la dernière option qui l'emporte en cas de conflit entre options.

3.2) Le traitement des lectures :

La partie de traitement commence par une boucle qui parcourt les fichiers un par un, en les ouvrant s'il s'agit de fichiers, ou en lisant depuis l'entrée standard si cela est demandé. Pour cela, une variable joue le rôle d'intercepteur du flux de lecture demandé afin de pouvoir commencer la seconde partie.

La seconde partie consiste en une autre boucle qui lit les caractères un par un depuis le flux qui lui est associé et constitue par la suite un mot selon les critères demandés dans l'énoncé et aussi selon certaines options si elles sont activées. Notre première approche pour cela était juste de recycler le code du premier TP avec la fonction scanf, tout en oubliant que cela va justement entraîner une limitation fixe maximale de la taille des mots, ce qui va fortement à l'encontre de ce qui est demandé. Donc, une lecture caractère par caractère s'est vite imposée et, chose bien au final, les options comme -i et -p sont beaucoup plus faciles à gérer comme ça, et aussi la mise en place d'une taille illimitée pour les mots est désormais possible grâce aux réallocations en cas de dépassement de la taille initiale arbitrairement définie. Cela se fait tout en rajoutant une certaine taille qui elle aussi est arbitraire.

S'ensuit la partie du traitement du mot lu : une petite vérification est d'abord faite pour savoir quel traitement choisir entre AVL et table de hachage, et un appel à la fonction appropriée est effectué puis un ajout vers le holdall comme ce qui a été montré avec les dessins précédents pour les deux cas. Après la boucle de lecture depuis les fichiers et/ou l'entrée standard, la fermeture de ces deux est requise. Cependant, on a cherché un équivalent de fclose() pour stdin et on avait beaucoup de choix ; les plus communs que l'on a trouvés étaient clearerr() et rewind(), mais elles nous restent très ambiguës. La seule chose qui nous importe pour le moment était le fait que ça marchait bien. Il y a très probablement une façon plus simple et plus correcte de le faire, mais par manque de temps, on a préféré prioriser d'autres choses bien plus importantes.

3.3) le post traitement :

Par la suite, le traitement de l'output est fait en affichant d'abord la liste des fichiers selon les spécifications de `display_files()`, et la gestion du potentiel tri demandé grâce aux fonctions de comparaison définies dans le main et selon ce qui est aussi demandé par le sujet du projet. Et finalement, une libération des ressources allouées, si tout s'est bien déroulé (ou pas), afin d'espérer passer le jugement de Valgrind.

4) Des comparaison :

```
xwc_test$ ./xwc -h -n -p -R -i 7 lesmiserables.txt fr_.txt > h.txt 2> h1.txt
xwc_test$ ./xwc2 -h -n -p -R -i 7 lesmiserables.txt fr_.txt > m.txt 2> m1.txt
xwc_test$ diff h.txt m.txt -s
Les fichiers h.txt et m.txt sont identiques
xwc_test$
xwc_test$
xwc_test$ time ./xwc -h -n -p -R -i 7 lesmiserables.txt fr_.txt > h.txt 2> h1.txt

real    0m1,555s
user    0m0,345s
sys     0m1,175s
xwc_test$ time ./xwc2 -h -n -p -R -i 7 lesmiserables.txt fr_.txt > m.txt 2> m1.txt

real    0m1,639s
user    0m0,488s
sys     0m1,120s
xwc_test$
```

```
xwc_test$
xwc_test$ valgrind  ./xwc -h -n -p -R  lesmiserables.txt fr_.txt >/dev/null
==25518== Memcheck, a memory error detector
==25518== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==25518== Using Valgrind-3.21.0 and LibVEX; rerun with -h for copyright info
==25518== Command: ./xwc -h -n -p -R lesmiserables.txt fr_.txt
==25518==
==25518==
==25518== HEAP SUMMARY:
==25518==     in use at exit: 0 bytes in 0 blocks
==25518==   total heap usage: 1,363,474 allocs, 1,363,474 frees, 36,925,123 bytes allocated
==25518==
==25518== All heap blocks were freed -- no leaks are possible
==25518==
==25518== For lists of detected and suppressed errors, rerun with: -s
==25518== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
xwc_test$ valgrind  ./xwc2 -h -n -p -R  lesmiserables.txt fr_.txt >/dev/null
==25525== Memcheck, a memory error detector
==25525== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==25525== Using Valgrind-3.21.0 and LibVEX; rerun with -h for copyright info
==25525== Command: ./xwc2 -h -n -p -R lesmiserables.txt fr_.txt
==25525==
==25525==
==25525== HEAP SUMMARY:
==25525==     in use at exit: 0 bytes in 0 blocks
==25525==   total heap usage: 1,380,622 allocs, 1,380,622 frees, 31,677,126 bytes allocated
==25525==
==25525== All heap blocks were freed -- no leaks are possible
==25525==
==25525== For lists of detected and suppressed errors, rerun with: -s
==25525== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
xwc_test$
```

5) Conclusion :

On est très fiers de notre tout premier projet en algorithmique, qui nous a permis de consolider un bon nombre de thématiques à aborder en cours, TD et TP, notamment l'organisation d'un tel projet, la répartition des tâches et bien d'autres. On pense que notre travail a été au moins à la hauteur du minimum qui a été demandé, malgré certains manques de notre part.