FONCTIONNEMENT GÉNÉRAL :

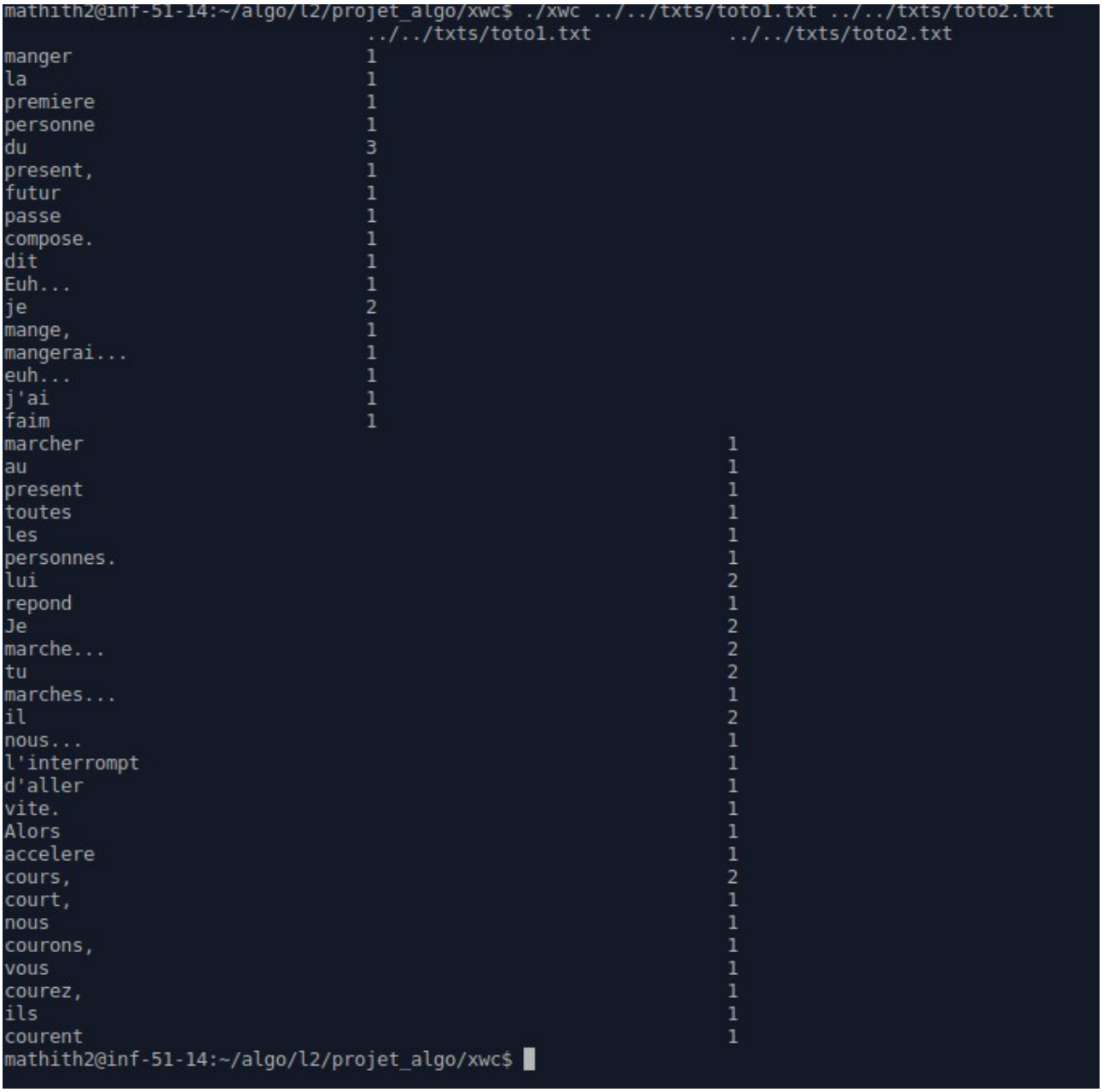
Les fichiers sont lus un à un avec « fgetc », chaque lettre est d'abord stockée dans le buffer "mot", quand le caractère lu est de type « isspace() », cela marque la fin du mot.
On cherche donc dans la table de hashage le mot en question.
Si la recherche est négative, on alloue une chaine de caractère s de la taille du mot qui a été lu, et on copie le contenu du buffer dans s.
On alloue aussi le couple nombre d'occurrence, numéro de fichier, qui va servir à donner les informations relatives à chaque mots.
Si lors de la lecture le mot dépasse la taille initiale du buffer, une réallocation est effectuée, la taille du buffer double.
Les chaines de caractères ainsi que les couples sont stockés dans des fourretouts séparés.
On ajoute à la table de hashage, la clé qui sera la chaine de caractère, et sa valeur qui sera le couple nombre d'occurrence du mot, numéro du fichier.
Ce couple permet de savoir très facilement à quel fichier un mot appartient et son nombre d'occurrence dans ce fichier.
La lecture des mots continue de se faire à chaque tour de boucle, le buffer mot est réinitialisé grâce à l'appel à la fonction « memset ».

Si la recherche est positive, on teste si le mot provient du même fichier que l'on est en train de lire.
Si c'est le cas, on augmente l'occurrence de 1, sinon cela veut dire que le mot n'est pas exclusif au fichier, et on met l'occurrence à -1, ce qui le met de côté, le mot sera toujours présent dans la table de hashage mais il sera marqué comme "non exclusif".

Quand le caractère lu par « fgetc » est « EOF », on ferme le fichier, si il y en a d'autres ils sont ouverts puis lus de la même façon.
Une fois que tous les fichiers ont été lus, on applique au fourretout qui stocke les chaines alloués, la fonction « display », qui affiche à l'écran les chaines de caractères.
Cette fonction affiche le mot uniquement si son occurrence n'est pas -1, autrement dit seulement si il est exclusif.

Voici un exemple d'exécution sur 2 fichiers :

On constate que le programme liste les mots et leur nombre d'occurrence associés à leur fichier respectif lorsqu'ils sont exclusifs à ces derniers.



```
mathith2@inf-51-14:~/algo/l2/projet_algo/xwc$ ./xwc ../../txts/toto1.txt ../../txts/toto2.txt
                              ../../txts/toto1.txt            ../../txts/toto2.txt
manger                        1
la                            1
premiere                      1
personne                      1
du                            3
present,                      1
futur                         1
passe                         1
compose.                      1
dit                           1
Euh...                        1
je                            2
mange,                        1
mangerai...                   1
euh...                        1
j'ai                          1
faim                          1
marcher                                               1
au                                                    1
present                                               1
toutes                                                1
les                                                   1
personnes.                                            1
lui                                                   2
repond                                                1
Je                                                    2
marche...                                             2
tu                                                    2
marches...                                            1
il                                                    2
nous...                                               1
l'interrompt                                          1
d'aller                                               1
vite.                                                 1
Alors                                                 1
accelere                                              1
cours,                                                2
court,                                                1
nous                                                  1
courons,                                              1
vous                                                  1
courez,                                               1
ils                                                   1
courent                                               1
mathith2@inf-51-14:~/algo/l2/projet_algo/xwc$
```

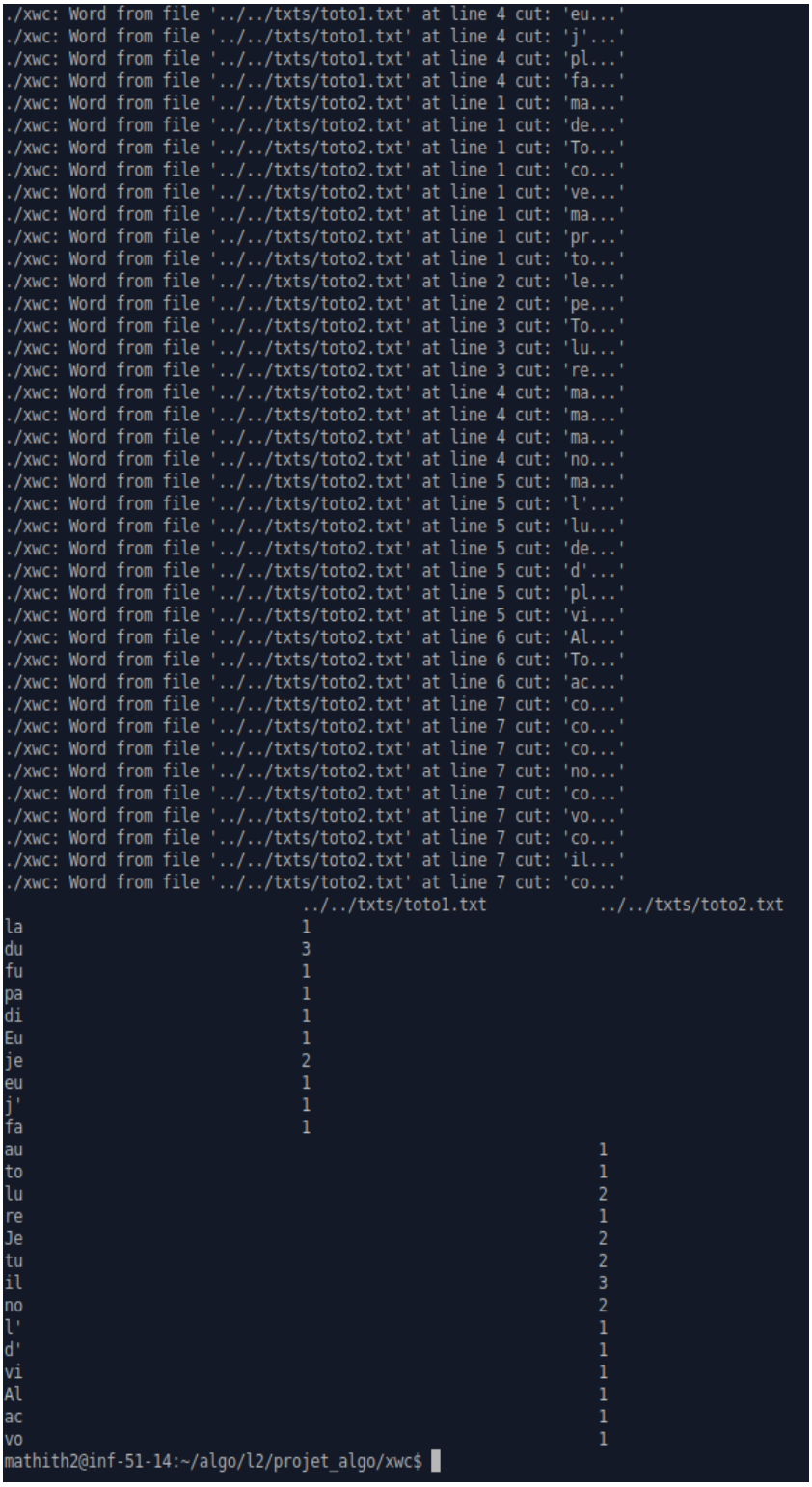
FONCTIONNEMENT DES OPTIONS :

La gestion des options se fait grâce à une structure qui sauvegarde les différentes valeurs des options entrées ou non et grâce à la fonction « get_options » qui initialise les champs de la structure, et qui détecte les options entrées.
Ces options sont ensuite traitées à l'intérieur de la fonction main, à l'aide de tests supplémentaires dans les boucles.
L'option -i rajoute un test en plus lors de la détection d'un espace qui symbolise la fin d'un mot.
Durant la lecture du mot, on teste si sa taille est égale au champs i de la structure. Si c'est le cas, on arrête la lecture du mot.

Un deuxième test est ensuite effectué afin de déterminer si le mot lu va être coupé ou non, puis le programme va se rendre au prochain espace du fichier permettant ainsi de limiter la lecture du mot à la taille indiquée par l'option -i.

Voici un exemple d’exécution de cette option :

On voit que les mots sont bien coupés par tranche de 2 comme convenus et l’option indique si les mots sont coupés avant la fin en précisant également la ligne du texte et nom du fichier où le mot est apparu.



L'option -p rajoute elle aussi un test supplémentaire lors de la détection d'un espace. Elle va tester si le caractère lu est une ponctuation. Si l'option est activée, alors cela marque la fin du mot.
Un test est rajouté dans la condition de la boucle while qui sert à ignorer tous les espaces qui séparent les mots, cette fois si on va aussi tester si le caractère est une ponctuation et si c'est le cas on l’ignore.
Cette boucle permet de traiter sans problèmes des fichiers qui pourraient contenir plusieurs espaces entre les mots.
Voici un exemple d’exécution de cette option :

On voit que la ponctuation a bien été traitée comme un espace contrairement à l’exemple présent dans « FONCTIONNEMENT GENERAL » où l’on peut constater la présence de ponctuation.

```
mathith2@inf-51-14:~/algo/l2/projet_algo/xwc$ ./xwc -p ../../txts/toto1.txt ../../txts/toto2.txt
                              ../../txts/toto1.txt            ../../txts/toto2.txt
manger                        1
la                            1
premiere                      1
personne                      1
du                            3
futur                         1
passe                         1
compose                       1
dit                           1
Euh                           1
je                            2
mange                         1
mangerai                      1
euh                           1
j                             1
ai                            1
faim                          1
marcher                                               1
au                                                    1
toutes                                                1
les                                                   1
personnes                                             1
lui                                                   2
repond                                                1
Je                                                    2
marche                                                2
tu                                                    2
marches                                               1
il                                                    2
nous                                                  2
l                                                     1
interrompt                                            1
d                                                     1
aller                                                 1
vite                                                  1
Alors                                                 1
accelere                                              1
cours                                                 2
court                                                 1
courons                                               1
vous                                                  1
courez                                                1
ils                                                   1
courent                                               1
mathith2@inf-51-14:~/algo/l2/projet_algo/xwc$
```

- ? correspond à un simple appel à la fonction « print_help » qui affiche l'aide à l'utilisation et termine le programme, --version et --usage affichent respectivement la date de la dernière mise à jour du programme et la syntaxe à respecter pour l'appel du programme.

ENTRÉE STANDARD :

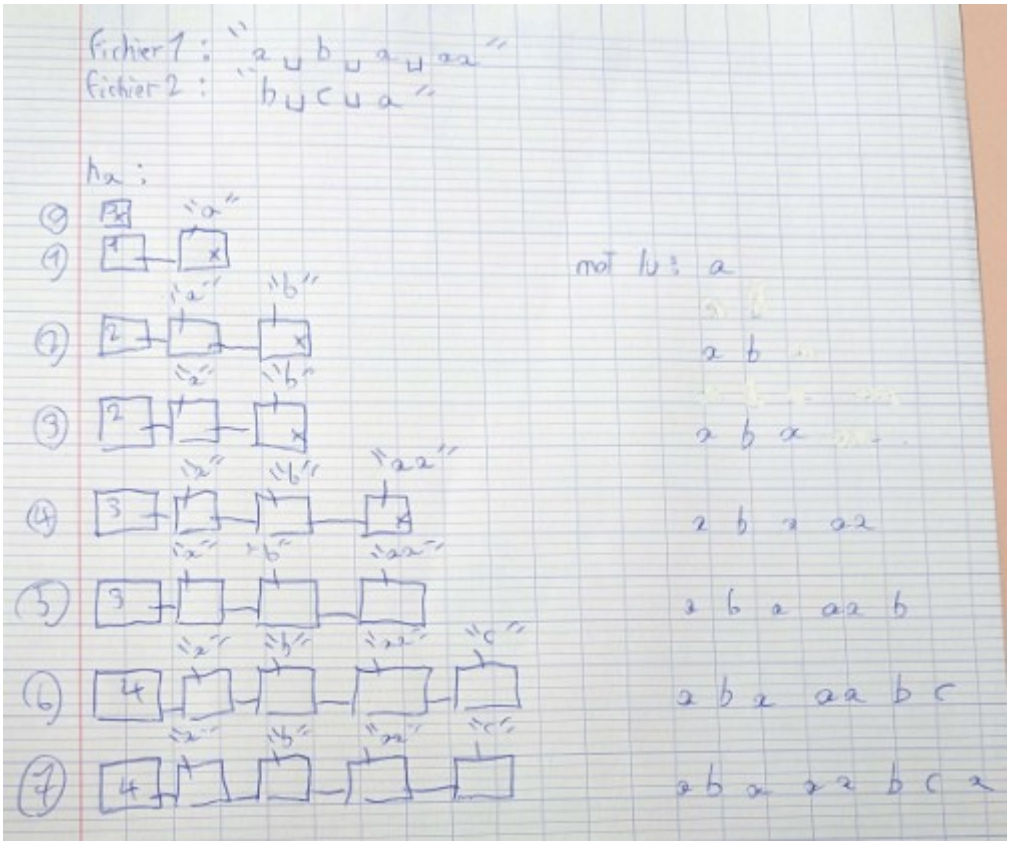
Le programme peut aussi prendre des mots directement au clavier, en entrant aucun fichier, ou en entrant un nombre de "-" qui correspond au nombre de fichiers simulés.
C'est la fonction « get_options » qui détecte le nombre de "-" entré.
Les fonctions « get_input_type » et « select_input » fonctionnent de paire pour respectivement récupérer le type d'entrée qui a été détecté et pour la sélectionner lors de l'ouverture d'un fichier.
Si c'est l'entrée standard qui a été détectée, le « FILE *f » sera initialisé à « stdin », sinon ce sera une ouverture classique de fichier.
Le type d'entrée est détecté lors de la lecture de la ligne de commande entrée lors de l'appel du programme xwc.

STRUCTURE DE DONNÉES :

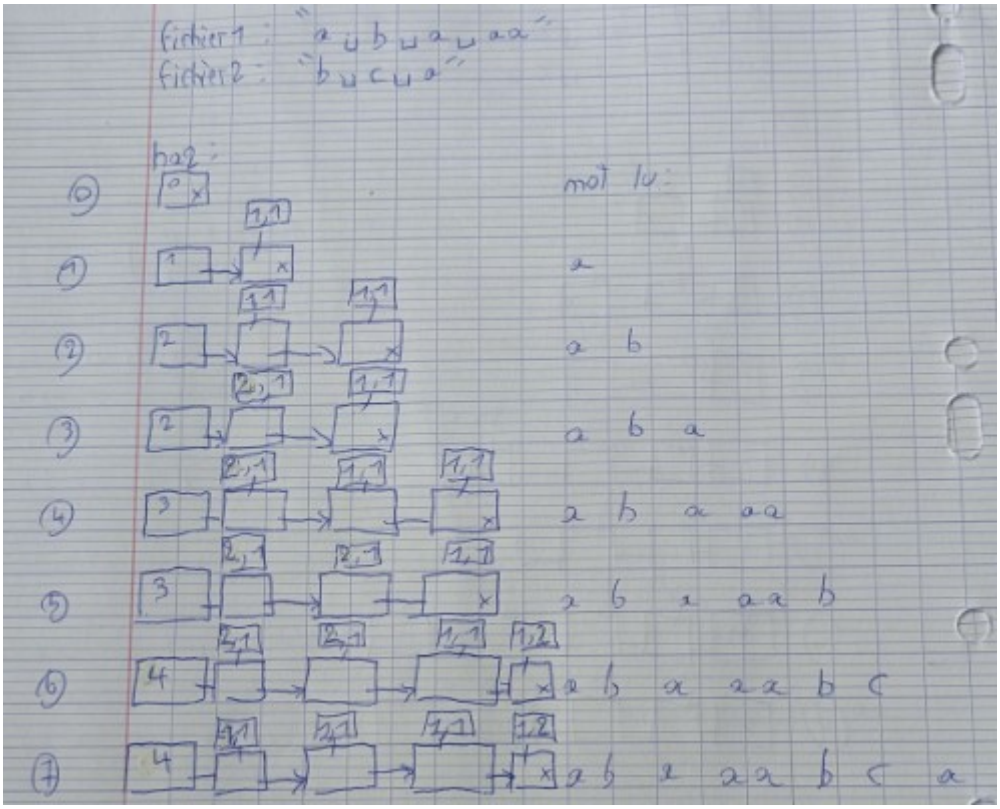
Nous utilisons deux fourretouts, un pour les allocations de chaînes et un pour les allocations des structures « occur_file », l'utilisation de 2 fourretouts nous permets de ranger efficacement et proprement les différentes allocations mais aussi de les récupérer et les afficher sans problème. (voir AFFICHAGE)
Ces allocations sont envoyées ensemble dans une table de hashage, la clé correspond à la chaine de caractère et sa valeur est la structure « occur_file » correspondante au mot.
Nous avons choisi la table de hashage plutôt que les AVL car nous avons estimé que la table de hashage était plus rapide et plus simple dans son utilisation.
Les fourretouts ne savent pas à quel mot appartient quel couple occurrence, numéro de fichier, c'est la table de hashage qui va permettre de connecter les deux.
Les adresses des chaînes et des couples sont envoyées ensemble dans la table de hashage. Lors de la recherche d'un mot dans la table, l'adresse du couple sera renvoyée. C'est comme cela que la table de hashage connecte ces deux informations

---

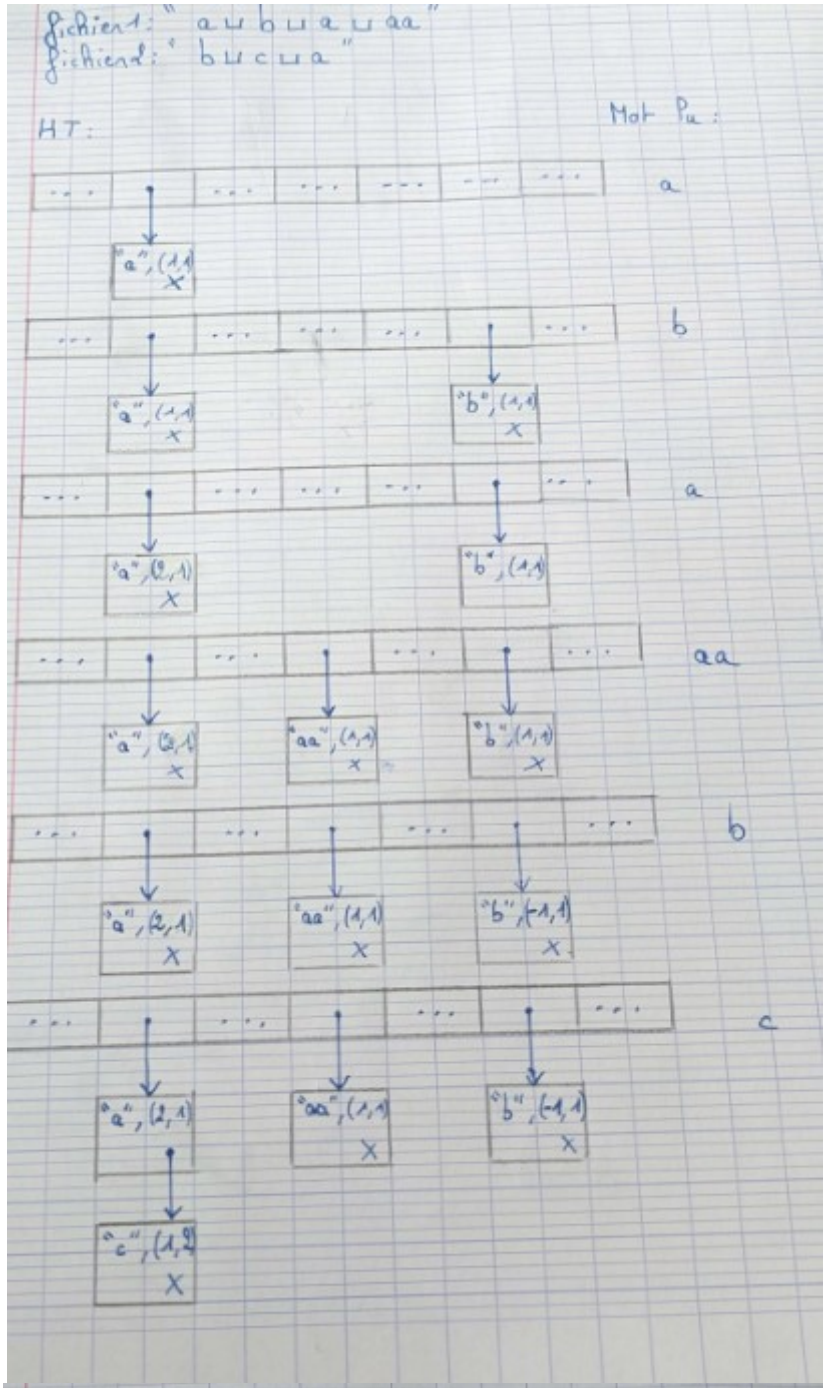
Voici un schéma de l’ajout de différents mots au fourretout « ha »
Les chaines ne sont allouées que si la recherche dans la table de hashage est négative. Dans ce cas, on alloue une chaîne de caractère, on copie les lettres récupérées dans le buffer et on ajoute la chaîne au fourretout.



Voici un schéma de l’ajout des structures « occur_file » au fourretout « ha2 »
De même que pour le fourretout « ha » les structures ne sont allouées uniquement lorsque la recherche dans la table de hashage est négative.
On voit que le nombre d’occurrence pour le mot « a » passe à -1 à l’étape 7 après la lecture du mot « a » présent dans le fichier 2, le mot « a » est donc marqué comme « non exclusif », son nombre d’occurrence est mis à -1 et le mot ne sera pas affiché par la fonction « display ».



Voici un schéma représentant les ajouts des chaînes et couples nombre d’occurrence, numéro de fichier à la table de hashage. On remarque que l’ajout à la table se fait en fonction du résultat de la fonction de hashage. Dans le cas d’une collision, l’ajout se fait en queue de la liste.
Si un mot est marqué comme « non exclusif » il est toujours présent dans la table de hashage et dans le fourretout, il sera tout simplement ignoré lors de l’affichage.



AFFICHAGE :

L'affichage se fait grâce à la fonction « holdall_apply_context » fourni dans le module « holdall ».
La fonction prend en paramètres le fourretout des mots (ha) , la table de hashage, la fonction de recherche dans la table de hashage et la fonction « display ».
Cette fonction « display » prend en paramètres une chaîne de caractère ainsi qu'une structure « occur_file ».
Elle va afficher la chaîne de caractère seulement si le champs « occur » de la structure est différent de -1.
Les paramètres de cette fonction « display » vont être donnés grâce aux autres paramètres de la fonction « holdall_apply_context ».
Cette fonction applique à chaque élément du fourretout
«int r = fun2(p->ref, fun1(context, p->ref));»

Dans notre cas,
fun2 correspond à « display »,
p→ref est notre chaine de caractère,
fun1 correspond au retour de la fonction de recherche de la table de hashage (cette fonction renvoie la valeur de la clé trouvée, donc dans notre cas un couple la structure « occur_file »),
context correspond à la table de hashage dans laquelle on va effectuer la recherche.

PROBLÈMES RENCONTRES

- Le buffer doit être réinitialisé après chaque fin de mot trouvé, nous avons commencé par créer une fonction « strcpy_and_delete » qui, quand elle copie le buffer dans la chaine allouée, supprime par la même occasion les lettres du buffer. Nous avons fini par remplacer cette fonction par « memset » qui réinitialise « proprement » le buffer et par un appel à « strcpy », un gain de 0,1 seconde à l'exécution en moyenne a été constaté.

- Le buffer était initialement limité par une constante « WORD_MAX_LEN », ce qui produisait des erreurs lorsqu'un mot dépassant cette constante était lu. De plus, le programme se devait de pouvoir lire des mots potentiellement infini.
Ce problème a été facilement résolu.
Le buffer était initialement un simple tableau de caractères déclaré localement. Nous avons transformé ce tableau en un tableau alloué. De ce fait, une réallocation est devenu possible si la taille max était atteinte.
La taille du buffer double donc lorsque la limite est atteinte.

- Un décalage se produisait lors de la lecture de la ligne de commande entrée. Quand des options étaient entrées, les fichiers n'étaient plus à la même place. Il fallait donc ajouter le nombre d'options entrées pour retrouver leur indice dans le tableau « argv ». Mais la lisibilité du code était réduite.

Ce problème a ainsi été résolu grâce à la séparation en deux variables «arg» et «nfile». «arg» correspond au numéro de l'argument traité sur la ligne de commande, il sert d'indice pour le tableau argv.
Il vaut 1 + le nombre d'options entrées – le nombre de fichiers directement entré.
Ce calcul sert à positionner « arg » sur le premier argument autre que le nom de l'exécutable, à le décaler sur le premier fichier en ignorant le nombre d'options entré et enfin, comme les entrées directes étant considérées comme des options, il faut par conséquent retirer le nombre de « fichiers » à entrée directe à ce calcul afin de placer l'indice « arg » au bon endroit dans argv.
« nfile » correspond au numéro du fichier traité, cette variable sert à initialiser le champ « nfile » de la structure « occur_file ».
Ces deux variables augmentent de 1 après chaque tour de boucle.

- L'entrée standard ne fonctionnait pas après la lecture d'un fichier, elle s'arrêtait.
Ce problème a été résolu grâce à l'appel à la fonction « clearerr » sur le flux, qui permet de lever l'exception « EOF » afin de continuer à lire le flux.

- Les options -l, -n et -R n'ont pas été intégrées au rendu final de part le fait du manque de temps et de la difficulté à programmer ces dernières. Nous avions toutefois envisagé de procéder à un tri fusion car nous estimions que ce tri était le plus optimisé pour trier des listes chaînées.

- L'option -r n'a pas été réalisée à cause du manque de temps mais la non-réalisation de cette dernière est également dû au fait qu'intégrer cette option aurait nécessité une modification importante du code.