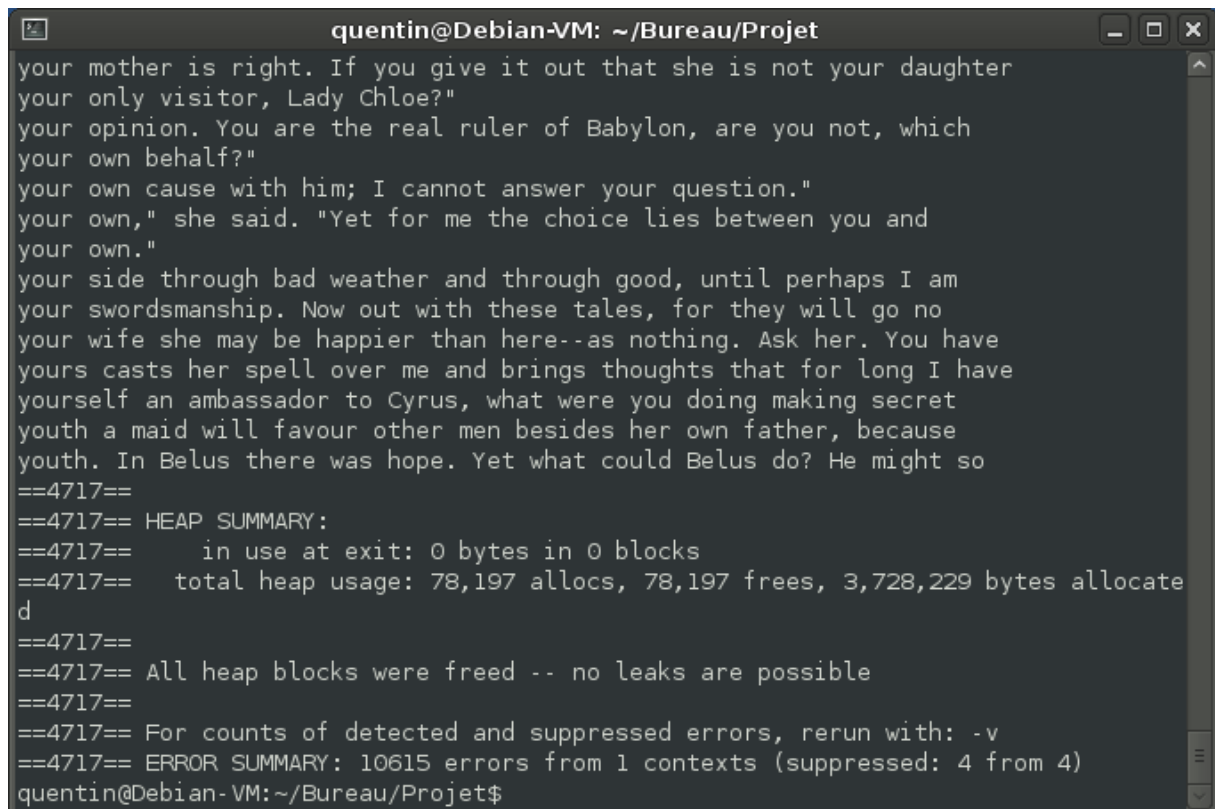


Projet Algorithmique : Rapport

Quentin Barrand – S2 – Printemps 2012

Ce rapport a pour but de décrire la réalisation du projet d'Algorithmique donné lors du deuxième semestre d'étude de la promotion 2013 du département Informatique de l'IUT de Belfort-Montbéliard. Pour rappel, il était demandé de créer un programme permettant de trier les lignes de l'entrée standard, à la manière de la commande UNIX `sort`. Le programme devait être écrit dans le langage C++.



```
quentin@Debian-VM: ~/Bureau/Projet
your mother is right. If you give it out that she is not your daughter
your only visitor, Lady Chloe?"
your opinion. You are the real ruler of Babylon, are you not, which
your own behalf?"
your own cause with him; I cannot answer your question."
your own," she said. "Yet for me the choice lies between you and
your own."
your side through bad weather and through good, until perhaps I am
your swordsmanship. Now out with these tales, for they will go no
your wife she may be happier than here--as nothing. Ask her. You have
yours casts her spell over me and brings thoughts that for long I have
yourself an ambassador to Cyrus, what were you doing making secret
youth a maid will favour other men besides her own father, because
youth. In Belus there was hope. Yet what could Belus do? He might so
==4717==
==4717== HEAP SUMMARY:
==4717==    in use at exit: 0 bytes in 0 blocks
==4717== total heap usage: 78,197 allocs, 78,197 frees, 3,728,229 bytes allocated
==4717==
==4717== All heap blocks were freed -- no leaks are possible
==4717==
==4717== For counts of detected and suppressed errors, rerun with: -v
==4717== ERROR SUMMARY: 10615 errors from 1 contexts (suppressed: 4 from 4)
quentin@Debian-VM:~/Bureau/Projet$
```

1. Structure du programme

1.1. Structure line

La structure `line` est destinée à constituer les maillons de la liste chaînée. Elle est composée d'une chaîne nommée `data` et d'un pointeur sur `line` vers le prochain maillon de la liste, nommé `next`.

1.2. Méthode `main()`

La méthode `main()` commence par lire l'intégralité de l'entrée standard. Dans une boucle `while(cin)`, la méthode `getline()` lit la ligne en cours et la stocke dans une chaîne `data`, membre d'une structure `line` nommée `current` préalablement dynamiquement allouée.

Chaque nouveau maillon de la liste chaînée est placé au début de la liste, pour éviter d'avoir à parcourir à chaque ajout. On profite de la boucle `while()` pour incrémenter `nbLinks`, un entier qui nous indiquera le nombre de maillons de la liste.

Une fois notre liste constituée, on fait appel à la fonction `quickSort()` qui va nous renvoyer un pointeur sur `line` vers le premier élément de la liste triée. Il nous suffit alors de dérouler une nouvelle boucle `while()` pour imprimer la chaîne sur la sortie standard et placer un opérateur `delete` à chaque tour (ainsi que deux autres pour les extrémités de la liste).

1.3. Autres méthodes du programme

1.3.1. Méthode `addToList()`

Prototype : `void addToList(line*, line*);`

Cette méthode prend en paramètres deux pointeurs sur `line` :

- Le premier pointeur redirige vers la première structure de la liste chaînée ;
- Le second pointeur redirige vers la structure que l'on cherche à ajouter à la fin de la liste chaînée.

Le mode opératoire est le parcours de la liste entière depuis le premier maillon. On utilise pour cela une boucle `while` qui tourne tant que le pointeur `next` de la structure courante n'est pas égal à `NULL`. Une fois qu'on est à la fin, il suffit d'initialiser le pointeur `next` du dernier maillon vers l'adresse de la structure à insérer.

1.3.2. Méthode `quickSort()`

Prototype : `line* quickSort(line*, int);`

Cette méthode prend en paramètres un pointeur sur `line` (le premier maillon de la liste chaînée) et un entier (le nombre de maillons de la liste chaînée).

Cette méthode permet de trier intégralement une liste chaînée. Les grandes lignes de son exécution sont les suivantes :

- Elle détermine un pivot en prenant le résultat de la division l'entier passé en paramètre par 2 et compare tous les éléments de la liste à ce pivot à l'aide de la fonction `string::compare()` ;
- Selon le résultat de cette comparaison, elle ajoute le maillon courant à l'une ou l'autre des deux listes ;

- Elle s'appelle elle-même (récursivité) une fois pour chaque liste ;
- Elle rejoint le pivot à la première liste et la deuxième liste au pivot ;
- Elle retourne un pointeur vers le premier maillon de la liste triée.

2. Problématiques rencontrées

2.1. Manipulation de la liste chaînée

La manipulation de la liste chaînée et des pointeurs en général a été assez difficile à appréhender, tout comme la notion d'allocation dynamique. Il m'a fallu relire plusieurs fois le cours pour comprendre comment parcourir une liste chaînée et y ajouter ou supprimer des éléments.

2.2. Méthode de tri de la liste

L'algorithme de la fonction `quickSort()` m'a également donné beaucoup de fil à retordre. Il faut que la fonction puisse gérer tous les différents cas (déterminer le pivot dans une chaîne vide ou d'un seul caractère, etc...). Les différents tests à chaque tour de boucle ou appel par récursivité (pour déterminer quand il est nécessaire de retourner `NULL`) m'ont permis de connaître la fameuse frustration du programmeur face à une « segfault »...

2.3 Destructeurs et élimination des fuites mémoire

Comme nous l'avons vu dans le descriptif du programme, lors de la lecture de chaque ligne, une structure `line` est créée, via l'opérateur `new`. Ces emplacements mémoire doivent être désalloués, sous peine d'obtenir une fuite mémoire : contrairement à Java et son Garbage Collector, il est nécessaire d'utiliser des destructeurs pour chaque objet créé.

J'utilise ces destructeurs via l'opérateur `delete` dans la boucle finale d'affichage de la fonction `main()`. Un contrôle avec l'outil Valgrind m'a permis de constater que tous les blocs alloués sont détruits après l'exécution du programme ; il n'engendre aucune fuite mémoire.

3. Résultats

Après compilation et exécution du programme, j'ai pu constater les résultats suivants :

- Le programme fonctionne pour des lignes arbitrairement grandes. Le type `std::string` utilise l'allocation dynamique pour créer des chaînes de caractères aussi longues que nécessaire ;
- Le programme n'a montré aucune erreur lors de la lecture de textes relativement longs. Le plus gros test que j'ai fait comportait plus de 10 000 lignes ; j'imagine que la seule limitation est la quantité de mémoire vive du système ;
- Le programme présente un résultat identique à celui de la commande `sort` dans le cas d'un texte sans caractères spéciaux. La comparaison des sorties avec la commande `diff` ne renvoie aucun message.

- Le programme ne provoque aucune fuite mémoire.

4. Développements futurs

Mon programme suivant l'ensemble des directives de l'énoncé, j'ai réfléchi à plusieurs axes de développement.

4.1. Arguments en ligne de commande

Aujourd'hui, pour tester mon programme, il est nécessaire d'utiliser les redirections dans la console.

Exemple : `$ cat texteatester | ./projet`

Je souhaite pouvoir le lancer sans utiliser ces commandes système, en utilisant les arguments de la ligne de commande.

Exemple : `$./projet texteatester`

Il faudrait dans ce cas utiliser les arguments de la méthode `main()` et ajouter la ligne `#include <fstream>` au début du programme, pour ouvrir et écrire dans le fichier `texteatester`.

4.2. Caractères spéciaux et accentués

En l'état, mon programme place les caractères accentués à la fin de la sortie. J'ai pourtant correctement utilisé la fonction `std::setlocale()` qui applique les paramètres régionaux de mon système au programme, mais il semble que la fonction `string::compare()` ne les prenne pas en compte. Je dois continuer à chercher pour déterminer la cause de ce dysfonctionnement.

5. Conclusions

J'ai compris que dans tout projet, il est indispensable de consulter toute la documentation possible, dans toutes ses formes. En l'occurrence, les sites [StackOverflow](#) et [CPlusPlus](#) m'ont clairement aidé à appréhender le fonctionnement des pointeurs et des listes chaînées, tout comme le cours et certains ouvrages de la bibliothèque.

L'utilisation du débogueur `gdb` m'aura également été d'une grande aide dans la réalisation de ce projet. Il m'aura en effet souvent permis de situer les erreurs dans mon code, là où l'exécution du programme renvoyait dans la console une simple `Segmentation fault`.