Rapport de projet de Programmation Impérative

Quentin Hostettler - Joël Roman Ky

1 Introduction

L'objectif de ce rapport est de présenter le projet de compression de texte par le codage de Huffman. Il contiendra une brève présentation du problème traité, des modules utilisés, ainsi que des choix faits au cours de l'avancée du projet. Il y sera également exposé les différents problèmes rencontrés ainsi que la façon dont ils furent traités. Pour finir, un bilan sera donné sur le projet en lui-même ainsi que sur les différents membres y ayant participé.

2 Rapport

Le but de ce projet était de créer deux exécutables : l'un permettant de compresser un fichier texte, l'autre, de décompresser le fichier généré par compression du fichier texte ; les deux exécutables devant fonctionner à l'aide de la méthode du codage de Huffman (expliquée dans l'énoncé du projet).

2.1 Architecture de l'application des modules

En premier lieu, nous avons tout d'abord codé les différents modules qui selon nous allaient servir. Les modules réalisés sont les suivants :

- Liste Arbre : comportait les sous-programmes et types nécessaires à l'initialisation de l'arbre de Huffman. C'est-à-dire qu'il contient une liste chainée d'arbres (comme précisé à la figure 1 du sujet).

- Arbre : quant à lui comporte de quoi créer l'arbre à partir de la liste d'arbre (en utilisant également certaines opérations du module Liste Arbre), mais également de quoi créer l'arbre à partir de la table de Huffman et la description de l'arbre à partir de celui-ci.

- Chaine : contient la définition d'un type s'apparentant à un Unbounded String et les sous-programmes associés (afin d'être certain des opérations que l'on pouvait utiliser sur ce type et de son fonctionnement, nous avons préféré en créer un nouveau plutôt que d'utiliser celui déjà existant).

- Table Code : contient de quoi créer et utiliser la table de Huffman.

- Tab Freq : pour finir donne tous les sous-programmes nécessaires à la création du tableau de fréquence.

2.2 Principaux choix réalisés

Les principaux choix furent lors de la définition des différents types. Nous avions, au début, convenu d'utiliser un booléen (pour le type T Arb) afin de permettre à un sous-programme parcourant l'arbre de savoir par où il était déjà passé. Mais après en avoir parlé avec le professeur, nous avons convenu d'utiliser un parcours infixe pour parcourir l'arbre, simplifiant ainsi le type T Arb.

Nous comptions également utiliser le type prédéfini Unbounded String, seulement nous n'étions pas à l'aise avec l'utilisation d'un type que nous n'étions pas sûr de connaître parfaitement, d'autant plus que ce type allait être très utilisé. Nous avons donc décidé de définir nous-même un type T String ainsi que les sous-programmes associés.

2.3 Démarche adoptée pour tester les programmes

Nous nous sommes contenter de tester les modules avec des sous-programmes de tests, ce qui fut un peu compliqué, puisque nous avons dû rajouter certains sous-programmes au fur et à mesure de l'avancée du projet, nous rendant compte de leur nécessité des fois un peu plus tard.

Pour tester le programme compress, nous voulions au début tenter d'utiliser la commande xxd sur le .hf généré, mais ce n'était guère très lisible, nous nous sommes donc contentés d'utiliser les fonctions d'affichage de la table et de l'arbre. Quand au programme uncompress, nous avons pu le tester qu'avec le programme compress, et bien évidemment les fonctions d'affichage.

Afin de faciliter les tests notamment sur le fonctionnement des programmes finaux, il aurait pu être utile d'avoir un exécutable permettant de les tester. Puisque tant que l'uncompress n'a pas été fait, nous n'étions même pas sûr du bon fonctionnement du compress. Et même après, il est encore possible d'avoir fait une erreur dans l'un se compensant par une erreur similaire dans l'autre.

2.4 Difficultés rencontrées

L'une des plus grosses difficultés rencontrées fut lors de la création de certains sous-programmes récursifs. Nous avions besoin de conserver en mémoire des paramètres en « out » de procédures récursives. Seulement l'utilisation récursive ne le permettait, ou du moins pas à notre connaissance. Après y avoir longuement réfléchi, nous avons décidé de contourner le problème en passant le paramètre de « out » à « in out , en le créant vide avant d'appeler la procédure (on aurait aussi pu créer une procédure créant l'élément et appelant ensuite la fonction mais cela aurait grandement augmenté la quantité de procédures).

Nous avons aussi dû faire face à la gestion des retours chariots. Pour cela, plusieurs solutions étaient envisageables. Nous avons au final trouvé un moyen de les gérer comme des caractères (et de les rajouter dans le tableu de fréquence à chaque fin de ligne).

Mais de loin le plus gros soucis fut lors de la création de la table de codage. Nous avions un problème à cause du fait que l'élément « Code » des cases de la table est un pointeur. Ainsi une fois un élément placé, celui-ci ne devait pas être modifié, mais l'était quand même. Non seulement nous avons pris énormément de temps à nous rendre compte de la nature exacte du problème (grâce aux procedure d'affichage), mais même après avoir trouvé la nature du problème, nous n'avons pas pu le résoudre avant un certain temps.

3 Conclusion

3.1 Bilan technique

Le projet est à ce jour terminé, il reste toutefois perfectionnable en de nombreux points. Notamment dans la déclaration des types et des sous-programmes associés. En effet, la façon de coder ces sous-programmes manque parfois de régularité. Nous avons également constaté une taille anormalement élevé du fichier ".hf", apparemment dûe au fait que chaque binaire serait codé sur un octet, multipliant ainsi la taille "optimale" du fichier ".hf" par huit.

3.2 Bilan personnel

Un projet intéressant et motivant, bien que long et complexe, environ 25 à 30 heures chacun passées à la conception du projet (en comptant les horaires prévues à cette fin) même si cela reste difficile à évaluer précisément, environ 6 heures passés à la conception, 12 heures passés à l'implémentation, 8 heures passées à la mise au point et 3 heures au rapport. Le projet nous a appris à concevoir un programme nécessitant plusieurs modules, et à nous organiser pour un travail en binôme. Nous nous sommes rendu compte du manque de commentaires dans les codes, principalement dû au fait que nous avons essentiellement travailler ensemble sur les différentes étapes. Cependant, nous pensons tous deux qu'il serait préférable à l'avenir de commenter beaucoup plus et de façon plus concise les modules, types et sous-programmes.