

# Projet de Programmation avancée : Compresseur

Borges Daniel ( [dborges@etudiant.univ-mlv.fr](mailto:dborges@etudiant.univ-mlv.fr) )

21 mai 2006

# 1 Description générale

Ce logiciel est un compresseur/décompresseur de textes utilisant une méthode de compression/décompression vue en cours.

## 2 Manuel de l'utilisateur

`compress` est la commande de compression, elle doit être utilisée comme suit :

```
$ compress <fichier texte> <fichier compressé>
```

`<fichier texte>` doit être un fichier texte, peut importe l'encodage, du moment que les caractères ASCII se trouvent au même endroit dans l'encodage du fichier texte et dans le code ASCII. `<fichier compressé>` sera le nom du fichier une fois compressé.

`decompress` est la commande de décompression qui doit être utilisée comme suit :

```
$ decompress <fichier compressé> <fichier décompressé>
```

`<fichier compressé>` doit être un fichier compressé par la commande `compress` fournie. `<fichier décompressé>` contiendra le texte décompressé.

Ces deux commandes supportent les options `-b` et `-o`. Ce sont deux méthodes de codage des entiers utilisés lors de la compression, la méthode la plus efficace et la méthode utilisée par la première option, elle est choisie par défaut. Une dernière option, `-v`, empêche l'affichage de la barre de progression lors de la compression ou de la décompression et affiche une liste de mots utilisés par le programme, ceci à des fins de débogage.

Il n'existe aucun moyen de connaître si le fichier a été compressé par l'une ou l'autre des deux méthodes de codage des entiers, autrement dit : il faut savoir quel fut la méthode utilisée pour décompresser un fichier correctement, aucune garantie n'est donnée par le programme si un fichier est décompressé avec la mauvaise méthode de codage des entiers.

## 3 Manuel du programmeur

Les sources du programme se trouvent dans les répertoires `src` et `include`. Comme leur nom l'indique, le premier contient les fichiers de code C et l'autre, les en-têtes de ces fichiers.

### 3.1 `ecriture.c`

Ce fichier contient toutes les fonctions relatives à l'écriture et la lecture d'entiers selon les deux méthodes de codage vu en cours :

**bit à bit** : le nombre de bits nécessaire à l'enregistrement d'un entier est codé de manière unaire puis un bit de séparation - 0 - est enregistré et enfin l'entier est codé avec uniquement ses bits utiles.

Exemple : l'entier 5, dont la représentation sur 32 bits est 00000000 00000000 00000000 00000101 sera codé 111 0 101.

Les fonctions utilisées sont :

```
int ecrirebitabit(FileBAB * fichier, unsigned int n)
int lecturebitabit(FileBAB * fichier, unsigned int * n).
```

**octet par octet** : les entiers sont codés sur des paquets de 7 bits, le huitième sert à indiquer si l'écriture se poursuit sur le paquet suivant. On invite ainsi d'écrire des paquets pleins de 0.

Exemple : toujours pour l'entier 5, 0 0000101

Les fonctions utilisées sont :

```
int ecrireoctetparoctet(FileBAB * fichier, unsigned int n)
int lectureoctetparoctet(FileBAB * fichier, unsigned int * n)
```

Des fonctions de lecture/écriture de chaînes de caractères et de caractères dans des fichiers des structures de type `FileBAB` sont aussi fournies.

La structure `FileBAB` permet d'écrire de manière bufferisée, des bits dans un fichier. En effet il est impossible d'écrire directement des bits dans un fichier. L'unité minimale pour lire/écrire dans un fichier est l'octet. La structure `FileBAB` contient donc un octet qui est rempli au fur et à mesure de l'utilisation des fonctions de `ecriture.c` et son écriture s'effectue lorsqu'il est plein.

### 3.2 `liste.c`

Ce fichier contient des fonctions pour la gestion de listes simplement chaînées de chaînes de caractères et de caractères, utilisées lors de la compression. Ce sont des listes simplement chaînées classiques.

### 3.3 `parseur.c`

Les mots lus par le programme se font à l'aide de ce parseur de fichier texte bufferisé. Suivant l'idée de la structure `FileBAB`, une structure `Mots` a été créée. Le programme

lit une ligne complète et la fonction `lire_mot` s'occupe de retourner les mots au fur et à mesure de la lecture de la ligne lue et lit une nouvelle ligne si elle a été entièrement parcourue. Le caractère de ponctuation sur lequel s'est arrêté la fonction `lire_mot` est retourné, 0 indique la fin du fichier.

### 3.4 `progress.c`

Ce fichier contient des fonctions permettant l'affichage d'une barre de progression, utilisée pendant la compression pour voir son avancement. `largeur_terminal` utilise différentes méthodes pour connaître la largeur du terminal, selon le système utilisé. Cette valeur est utilisée par la fonction `progressbar` pour afficher une barre de progression de la largeur du terminal.

### 3.5 `compress.c`

Ce fichier contient les deux fonctions principales permettant la compression et la décompression de fichiers textes.

La compression est faite à l'aide de deux listes chaînées. La première contient les mots du texte, la seconde, les caractères de ponctuation séparant les mots.

#### 3.5.1 Compression

La compression s'effectue en lisant mot à mot le texte à l'aide du parseur. Lorsqu'un nouveau mot est trouvé, il est ajouté en fin de liste de mots et codé dans le fichier compressé par un 0 suivi du nouveau mot en clair. Puis le caractère de ponctuation est écrit en suivant la même méthode ( s'il est nouveau, il est codé par 0 et écrit en clair dans le fichier et ajouté en fin de liste de ponctuation ).

Lorsqu'un mot est lu et qu'il est déjà présent dans la liste, il est codé à l'aide de sa position dans la liste, il n'est donc pas nécessaire de coder le mot en clair dans le texte. De la même façon, on code les différents caractères de ponctuation séparant les mots.

#### 3.5.2 Décompression

La décompression se fait en faisant exactement l'opération inverse de la compression : un entier est lu, s'il s'agit d'un 0, le mot est lu, s'il s'agit d'un autre nombre, on cherche le mot à cette position dans la liste de mots. On fait de même pour connaître le caractère de ponctuation suivant ce mot.