Manuel Utilisateur

Utilisation en ligne de commande **1**
Compresser **1**
Option 1
Nom fichier 1
Exemples sans option 1
Exemples avec l’option bavarde 1
Décompresser **3**
Option bavarde 3
Nom du fichier 3
Exemples sans option 3
Exemples avec l’option bavarde 4
Utilisation des modules pour les développeurs **4**
Module Arbre 4
Module Liste_Chainee 6
Module Flux_Binaire 8

Utilisation en ligne de commande

Compresser

Compresser est un programme exécutable, que l'on utilise en entrant "./compresser" dans dans l'invite de commande. Il lui faudra 1 ou 2 arguments, explicités ci-après.
Les guillemets "" ne servent dans ce manuel qu'à mettre en valeur les arguments à passer au programme, et ne sont pas à réécrire lors de l'utilisation.

Option

Il est possible de mettre en argument après "./compresser", mais avant l'argument suivant. Les possibilités d'option actuelles sont "-b" ou "--bavard". Entrer ces arguments fait que le programme affiche les fréquences de chaque caractère dans le fichier à compresser, ainsi que l'arbre de Huffman généré et la table de Huffman (qui indique pour chaque caractère le code associé).
Lorsque vous entrez une option, il faut en entrer une seule : que "-b" ou que "--bavard", mais pas "-b –bavard".
Cet argument est optionnel donc pas ce n'est pas obligatoire de l'utiliser à chaque fois.

Nom fichier

L'argument obligatoire pour exécuter le programme est le nom du fichier à compresser qui doit être indiqué. Le fichier doit se trouver dans le même répertoire que le programme exécutable, sinon l'exécution ne sera pas faite. Cet argument doit être renseigné en dernier et non avant l'option précédente.

Exemples sans option

Exemple de commande sans option avec nom du fichier renseigné :

```
./compresser exemple.txt
```

Exemple de commande sans nom de fichier renseigné et résultat obtenu :

```
rpeyremo@n7-ens-lnx090:~/Documents/S5/UE_PIM/Projet/EF08/src$ ./compresser
Vous n'avez pas entré d'arguments.
```

Exemple de commande où le fichier renseigné n'est pas présent et le résultat retourné :

```
rpeyremo@n7-ens-lnx090:~/Documents/S5/UE_PIM/Projet/EF08/src$ ./compresser inconnu.txt
Fichier à compresser inexistant.
```

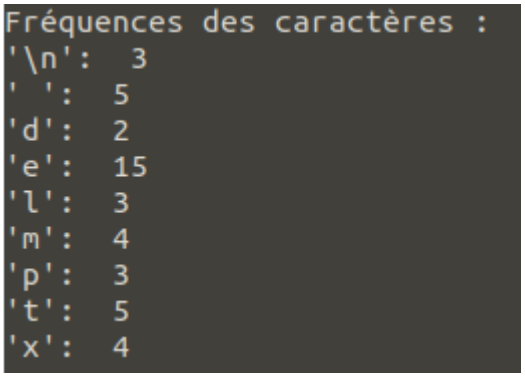
Exemples avec l'option bavarde

Exemples de commandes avec option sur un fichier exemple :

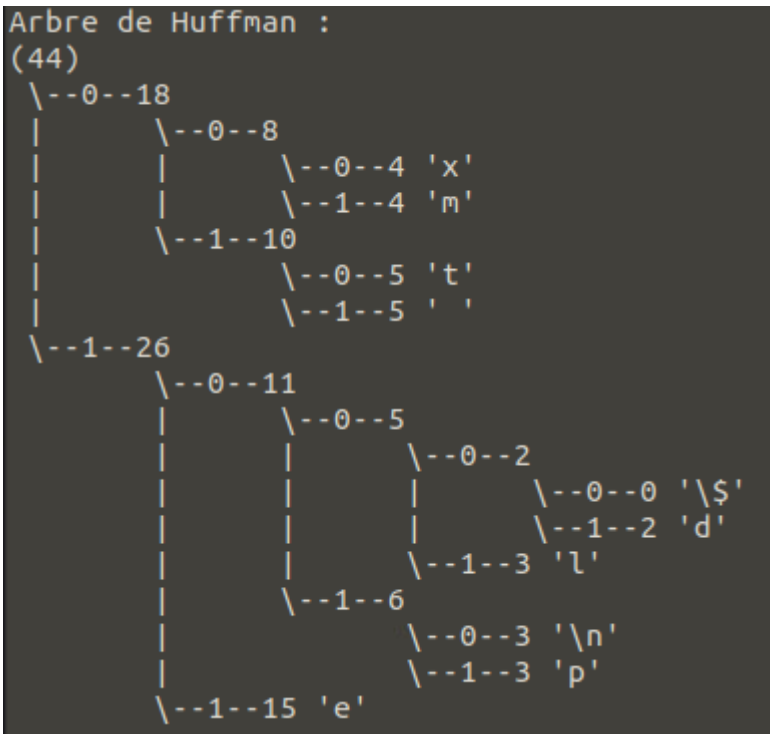
```
./compresser -b exemple.txt
```

```
./compresser --bavard exemple.txt
```

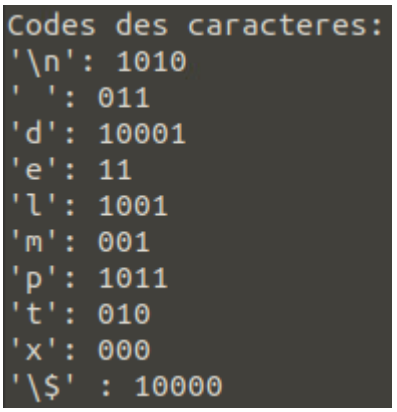
Exemple d’affichage des fréquences des caractères :



Exemple d’affichage d’arbre de Huffman :



Exemple d’affichage de la table de Huffman :



Exemple de commande avec une option inconnu (ni “-b” ni “--bavard”) et le résultat affiché :

```
rpeyremo@n7-ens-lnx090:~/Documents/S5/UE_PIM/Projet/EF08/src$ ./compresser -a exemple.txt
Option entrée non reconnue.
```

Exemple de commande avec une option en trop et le résultat affiché :

```
rpeyremo@n7-ens-lnx090:~/Documents/S5/UE_PIM/Projet/EF08/src$ ./compresser -b --bavard exemple.txt
Vous avez entré trop d'arguments.
```

Décompresser

Décompresser est un programme exécutable, que l'on utilise en entrant "./decompresser" dans l'invite de commande. Il lui faudra 1 ou 2 arguments, explicités ci-après.
Petite précision, les guillemets "" ne servent dans ce manuel qu'à mettre en valeur les arguments à passer au programme, et ne sont pas à réécrire lors de l'utilisation.

Option bavarde

Il est possible d'ajouter l'option "-b" ou "--bavard" juste après "./decompresser". Cette option permet d'afficher l'arbre de huffman reconstruit à partir du fichier compressé. Attention, il faut bien entrer "-b" OU "--bavard", et non les deux à la fois. Ainsi il serait incorrect d'écrire "-b --bavard". Cette option reste totalement optionnelle par définition, donc il n'est pas nécessaire de l'indiquer à chaque fois.

Nom du fichier

L'argument obligatoire à donner est le nom du fichier. Il doit se trouver après l'option bavarde si elle est présente : "./decompresser ex.txt" ou "./decompresser -b ex.txt" mais pas "./compresser ex.txt -b". Attention le nom du fichier doit se terminer par ".hff" sinon le décompresseur pensera qu'il n'a pas été compressé avec Huffman et vous l'indiquera avec un message.

Exemples sans option

Avec un fichier effectivement compressé par le compresseur

```
vfontain@n7-ens-lnx008:~/pim/projet/src$ ./decompresser exemple.txt.hff
vfontain@n7-ens-lnx008:~/pim/projet/src$
```

Sans aucun argument

```
vfontain@n7-ens-lnx008:~/pim/projet/src$ ./decompresser
Vous n'avez pas entré d'arguments.
```

Avec un fichier qui n'existe pas

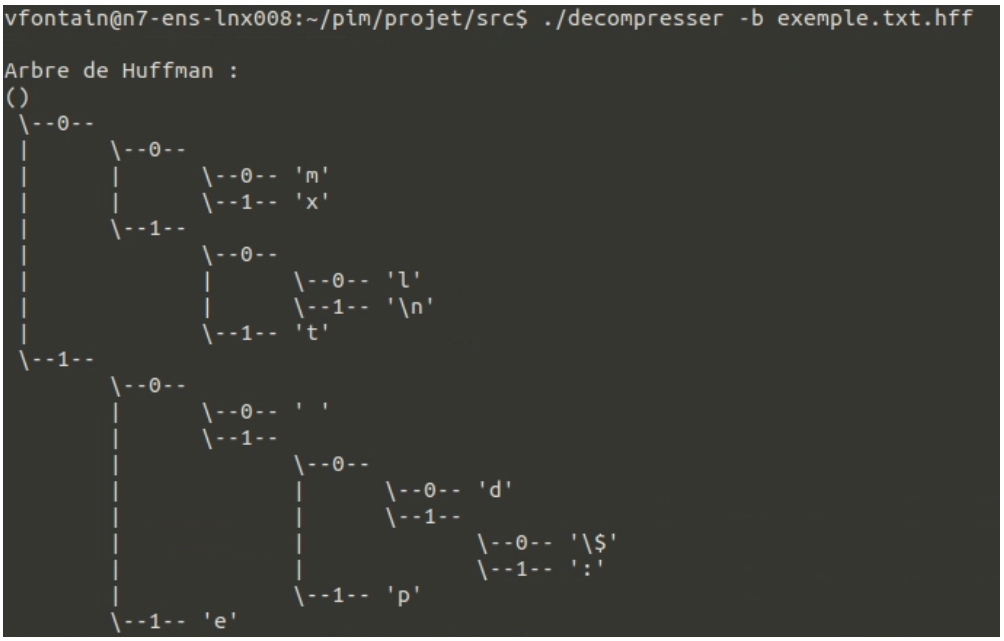
```
vfontain@n7-ens-lnx008:~/pim/projet/src$ ./decompresser inconnu.txt.hff
Fichier à décompresser inexistant.
```

Avec un fichier ne se terminant pas par ".hff"

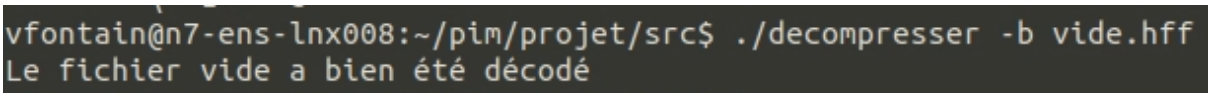
```
vfontain@n7-ens-lnx008:~/pim/projet/src$ ./decompresser exemple.txt
Le fichier à décompresser n'a pas été compressé par Huffman.
```

Exemples avec l'option bavarde

Avec un fichier classique



Avec le fichier vide



Utilisation des modules pour les développeurs

Si vous souhaitez réutiliser certains de nos modules dans vos propres applications, c'est tout à fait possible, voici leurs interfaces.

Module Arbre

Le module Arbre défini le type `T_Arbre` qui est un pointeur sur un enregistrement. Cet enregistrement contient deux autres pointeurs de type `T_Arbre` . Il contient aussi deux variables de types génériques.

Types génériques

Les types génériques à instancier lors de l'utilisation de ce module sont `T_Valeur` et `T_Clé`. Ils correspondent aux types des variables qui sont enregistrées dans ce que pointe `T_Arbre`.

Initialiser

La procédure `Initialiser` est la procédure permettant de créer une variable de type `T_Arbre` . Elle est nécessaire lors de l'utilisation de ce type dans votre programme. Elle prend en argument une variable out de type `T_Arbre` .

Est_Vide

La fonction `Est_Vide` prend en entrée une variable de type `T_Arbre` et renvoie un booléen. Le booléen est True si la variable entrée est vide, False sinon.

Enregistrer

La procédure `Enregistrer` prend en argument dans l'ordre :

- une variable in out de type `T_Arbre` correspondant à l'arbre où on veut enregistrer les éléments
- une variable in de type `T_Clé` qui est la clé que l'on veut mettre dans l'arbre
- une variable in de type `T_Valeur` qui est la valeur que l'on veut mettre dans l'arbre
- une variable in de type `T_Arbre` qui est l'arbre qui va être pointé à gauche
- une variable in de type `T_Arbre` qui est l'arbre qui va être pointé à droite

Cette procédure permet d'associer de compléter l'arbre pris en entrée avec les autres éléments en arguments.

Vider

La procédure `Vider` s'occupe de supprimer tous les éléments d'un arbre mis en entrée (de type `T_Arbre`) ainsi que les sous-arbres de cet arbre. À la fin, l'arbre sera vide. Il faut utiliser cette procédure pour éviter les fuites de mémoire.

Pour_Chaque

La procédure `Pour_Chaque` est une procédure qu'il faut instancier car elle utilise une procédure `Traiter` générique. `Pour_Chaque` prend en argument une variable in de type `T_Arbre` qui appliquera à toutes les Clés et Valeurs de l'arbre et des sous-arbres la procédure `Traiter`.

Traiter

`Traiter` est une procédure générique à définir dans votre programme qui prend en entrée deux arguments (dans l'ordre) :

- une variable in de type `T_Clé`
- une variable in de type `T_Valeur`

Module Liste_Chainee

Le module Liste_Chainee défini le type `T_Liste_Chainee` qui est une liste dynamique de taille variable.

Types génériques

Le type générique à instancier lors de l'utilisation de ce module est `T_Valeur`, le type des valeurs qui composent la liste.

Initialiser

La procédure `Initialiser` est la procédure permettant de créer une variable de type `T_Liste_Chainee`. Elle est nécessaire lors de l'utilisation de ce type dans votre programme. Elle prend en argument une variable out de type `T_Liste_Chainee`.

Est_Vide

La fonction `Est_Vide` prend en entrée une variable in de type `T_Liste_Chainee` et renvoie un booléen. Le booléen est True si la variable entrée est vide, False sinon.

Taille

La fonction `Taille` prend en entrée une variable in de type `T_Liste_Chainee` et renvoie un entier. Cet entier correspond au nombre d'éléments dans la liste

Ajouter

La procédure `Ajouter` permet d'ajouter à une liste chaînée un nouvel élément dans la liste. Elle prend en argument (dans l'ordre) :

- une variable in out de type `T_Liste_Chainee` qui est la liste où l'on va ajouter le nouvelle élément
- une variable in de type `T_Valeur` qui est la valeur de la nouvelle liste ajoutée
- une variable in de type booléen qui, si elle est True, ajoute l'élément au début de la liste. Dans le cas False, l'élément sera ajouter à la fin de la liste

Enregistrer

La procédure `Enregistrer` prend en argument dans l'ordre :

- une variable in out de type `T_Liste_Chainee` correspondant à la liste chainee où l'on veut enregistrer l'élément
- une variable in de type `T_Valeur` qui est la valeur que l'on veut enregistrer à l'indice dans la liste
- une variable in de type entier qui est l'indice dans la liste où l'on veut enregistrer l'élément (la valeur)

Cette procédure permet de modifier une valeur dans la liste chaînée à l'indice voulu.

Inserer_Avant

La procédure `Inserer_Avant` est similaire à celle `Ajouter`, cependant elle diffère car elle ajoute une liste et sa valeur à un indice dans la liste. Elle prend donc en entrée les arguments :

- une variable in out de type `T_Liste_Chainee` correspondant à la liste chainee où l'on veut ajouter l'élément
- une variable in de type entier qui est l'indice dans la liste où l'on veut ajouter l'élément (la valeur)
- une variable in de type `T_Valeur` qui est la valeur que l'on veut ajouter à l'indice dans la liste.

Supprimer

La procédure `Supprimer` permet de supprimer une ou plusieurs fois une Valeur entrée en arguments d'une liste. Elle a donc en arguments :

- une variable in out de type `T_Liste_Chainee` correspondant à la liste chainee où l'on veut supprimer la valeur
- une variable in de type `T_Valeur` qui est la valeur à supprimer
- une variable in de type booléen qui si elle est True, ajoute supprime la première itération de la Valeur dans la liste, si elle False, supprimer toutes les itérations existantes dans la liste

Supprimer_Indice

La procédure `Supprimer_Indice` supprime l'élément se trouvant à l'indice dans la liste. Elle prend en arguments :

- une variable in out de type `T_Liste_Chainee` correspondant à la liste chainee où l'on veut supprimer l'indice ;
- une variable in de type entier qui est l'indice que l'on veut supprimer.

Valeur_Presente

La fonction `Valeur_Présente` prend en entrée une variable in de type `T_Liste_Chainee`, ainsi qu'une variable in de type `T_Valeur` et renvoie un booléen. Le booléen est True si la variable entrée est présente dans l'ensemble de la liste chaînée, False sinon.

La_Valeur

La fonction `La_Valeur` prend en entrée une variable in de type `T_Liste_Chainee`, ainsi qu'une variable in de type entier et renvoie une variable de type `T_Valeur` qui est la Valeur se trouvant à l'indice entré.

L_Indice

La fonction `L_Indice` prend en entrée une variable in de type `T_Liste_Chainee`, ainsi qu' une variable in de type `T_Valeur` et renvoie une variable de type entier qui est l'indice où se trouve la Valeur d'entrée. La fonction retourne -1 si la Valeur n'est pas présente dans la liste chaînée.

Vider

La procédure `Vider` s'occupe de supprimer tous les éléments d'une liste chaînée donnée en entrée (flux de type in, variable de type `T_Liste_Chainee`). À la fin, la liste sera vide. Il faut utiliser cette procédure pour éviter les fuites de mémoire.

Pour_Chaque

La procédure `Pour_Chaque` est une procédure qu'il faut instancier car utilise une procédure `Traiter` générique. `Pour_Chaque` prend en argument une variable in de type `T_Liste_Chainee` qui appliquera à tous les éléments de la liste la procédure `Traiter`

Traiter

`Traiter` est une procédure générique à définir dans votre programme qui prend en entrée une variable in de type `T_Valeur`.

Module Flux_Binaire

Le module Flux_Binaire défini trois types : `T_Bit` un entier de taille 1 (sa valeur étant soit 0 soit 1), `T_Octet` un entier de taille 8 (valeurs entre 0 et 255) et `T_Flux_Binaire`, qui représente une suite de bits.

Initialiser

La procédure `Initialiser` est la procédure permettant de créer une variable de type `T_Flux_Binaire`. Elle est nécessaire lors de l'utilisation de ce type dans votre programme. Elle prend en argument une variable out de type `T_Liste_Chainee`.

Est_Vide

La fonction `Est_Vide` prend en entrée une variable in de type `T_Flux_Binaire` et renvoie un booléen. Le booléen est True si la variable entrée est vide, False sinon.

Taille

La fonction `Taille` prend en entrée une variable in de type `T_Flux_Binaire` et renvoie un entier. Cet entier correspond au nombre d'octets présent dans le flux.

Le_Bit

La fonction `Le_Bit` prend en entrée un `T_Octet` et un indice i entier. Elle renvoie le i-ème bit de l'octet. Attention l'indice doit être entre 0 et 7.

L_Octet

La fonction `L_Octet` prend en entrée un `T_Flux_Binaire` et un indice i entier. Elle renvoie le i-ème octet du flux. Attention l'indice doit être entre 0 et 7.

Ajouter_Bit

La procédure `Ajouter_Bit` prend en entrée un `T_Flux_Binaire` ainsi qu'un bit de type `T_Bit`. Elle ajoute le bit au flux.

Ajouter_Octet

La procédure `Ajouter_Octet` prend en entrée un `T_Flux_Binaire` ainsi qu'un bit de type `T_Octet` Elle ajoute l'octet au flux.

Ajouter_Flux

La procédure `Ajouter_Bit` prend en entrée deux `T_Flux_Binaire`, nommés Flux et Flux_Deux. Elle ajoute tous les bits de Flux_Deux dans Flux.

Vider

La procédure `Vider` s'occupe de supprimer tous les éléments d'un flux donnée en entrée (flux de type in, variable de type `T_Flux_Binaire`). À la fin, le flux sera vide. Il faut utiliser cette procédure pour éviter les fuites de mémoire.

Une_Chaine

La fonction Une_Chaine prend en entrée un `T_Flux_Binaire` et renvoie une représentation de ce flux sous la forme d'une chaîne de caractères (`String`).

Pour_Chaque

La procédure `Pour_Chaque` est une procédure qu'il faut instancier car utilise une procédure `Traiter` générique. `Pour_Chaque` prend en argument une variable in de type `T_Flux_Binaire` qui appliquera à tous les octets du flux la procédure `Traiter`.

Traiter

`Traiter` est une procédure générique à définir dans votre programme qui prend en entrée une variable in de type `T_Octet`.