

Huffman

0.1

Généré par Doxygen 1.8.11

Table des matières

1	Index des structures de données	1
1.1	Structures de données	1
2	Index des fichiers	3
2.1	Liste des fichiers	3
3	Documentation des structures de données	5
3.1	Référence de la structure Args	5
3.1.1	Description détaillée	5
3.1.2	Documentation des champs	5
3.1.2.1	compress	5
3.1.2.2	dest	5
3.1.2.3	source	5
3.1.2.4	verbose	6
3.2	Référence de la structure HufVertex	6
3.2.1	Description détaillée	6
3.2.2	Documentation des champs	6
3.2.2.1	child_l	6
3.2.2.2	child_r	6
3.2.2.3	count	7
3.2.2.4	value	7
3.3	Référence de la structure ReadBuffer	7
3.3.1	Description détaillée	7
3.3.2	Documentation des champs	7
3.3.2.1	data	7
3.3.2.2	next_bit	7
3.3.2.3	source_file	7
3.4	Référence de la structure WriteBuffer	8
3.4.1	Description détaillée	8
3.4.2	Documentation des champs	8
3.4.2.1	data	8
3.4.2.2	dest_file	8
3.4.2.3	flushed_count	8
3.4.2.4	pending_bits	8

4	Documentation des fichiers	9
4.1	Référence du fichier inc/arguments.h	9
4.1.1	Description détaillée	10
4.1.2	Documentation des fonctions	10
4.1.2.1	parseArgument(int key, char *arg, struct argp_state *state)	10
4.2	Référence du fichier inc/buffer.h	10
4.2.1	Description détaillée	11
4.2.2	Documentation des définitions de type	11
4.2.2.1	ReadBuffer	11
4.2.2.2	WriteBuffer	11
4.2.3	Documentation des fonctions	11
4.2.3.1	createReadBuffer(FILE *source)	11
4.2.3.2	createWriteBuffer(FILE *dest)	12
4.2.3.3	flushBuffer(WriteBuffer *buffer)	12
4.2.3.4	getBuffer(ReadBuffer *buffer)	12
4.2.3.5	getFlushedCount(WriteBuffer *buffer)	13
4.2.3.6	putBuffer(char bit, WriteBuffer *buffer)	13
4.3	Référence du fichier inc/common.h	13
4.3.1	Description détaillée	14
4.4	Référence du fichier inc/compress.h	14
4.4.1	Description détaillée	14
4.4.2	Documentation des fonctions	14
4.4.2.1	compress(FILE *source, FILE *dest)	14
4.5	Référence du fichier inc/decompress.h	15
4.5.1	Description détaillée	15
4.5.2	Documentation des fonctions	15
4.5.2.1	decompress(FILE *source, FILE *dest)	15
4.6	Référence du fichier inc/display.h	16
4.6.1	Description détaillée	16
4.6.2	Documentation des fonctions	16

4.6.2.1	<code>printCountsTable(bytecount *counts, bytecount total, size_t size)</code>	16
4.6.2.2	<code>printLabelsTable(char **labels, size_t size)</code>	17
4.6.2.3	<code>printTree(HufTree tree)</code>	17
4.6.2.4	<code>printVerbose(const char *format,...)</code>	17
4.6.2.5	<code>setVerbose(int enable)</code>	17
4.7	Référence du fichier <code>inc/huftree.h</code>	18
4.7.1	Description détaillée	19
4.7.2	Documentation des définitions de type	19
4.7.2.1	<code>HufVertex</code>	19
4.7.3	Documentation des fonctions	19
4.7.3.1	<code>createTree(bytecount *counts)</code>	19
4.7.3.2	<code>createTreeLabels(HufTree tree)</code>	19
4.7.3.3	<code>freeTree(HufTree tree)</code>	20
4.7.3.4	<code>freeTreeLabels(char **labels)</code>	20
4.7.3.5	<code>readTree(ReadBuffer *buffer)</code>	20
4.7.3.6	<code>writeTree(HufTree tree, WriteBuffer *buffer)</code>	20
Index		21

Chapitre 1

Index des structures de données

1.1 Structures de données

Liste des structures de données avec une brève description :

Args	Valeurs des arguments (options et opérandes) passés au programme	5
HufVertex	Sommet d'un arbre binaire de Huffman	6
ReadBuffer	Tampon de lecture bit par bit	7
WriteBuffer	Tampon d'écriture bit par bit	8

Chapitre 2

Index des fichiers

2.1 Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

inc/arguments.h	Interprétation des arguments (options et opérandes) passés au programme	9
inc/buffer.h	Abstraction de l'écriture et la lecture d'un fichier bit à bit	10
inc/common.h	Définitions communes pour le reste du code	13
inc/compress.h	Compression des fichiers	14
inc/decompress.h	Décompression des fichiers	15
inc/display.h	Gestion de l'affichage	16
inc/huftree.h	Représentation mémoire des arbres binaires de Huffman	18

Chapitre 3

Documentation des structures de données

3.1 Référence de la structure Args

Valeurs des arguments (options et opérandes) passés au programme.

```
#include <arguments.h>
```

Champs de données

- int [verbose](#)
- int [compress](#)
- FILE * [source](#)
- FILE * [dest](#)

3.1.1 Description détaillée

Valeurs des arguments (options et opérandes) passés au programme.

3.1.2 Documentation des champs

3.1.2.1 int Args : :compress

Comprime-t-on la source ou bien la décompresse-t-on ?

3.1.2.2 FILE* Args : :dest

Fichier de sortie de l'opération effectuée

3.1.2.3 FILE* Args : :source

Fichier source de l'opération à effectuer

3.1.2.4 int Args : :verbose

Lance-t-on le programme en mode verbeux ? (informations de débogage)

La documentation de cette structure a été générée à partir du fichier suivant :
 — [inc/arguments.h](#)

3.2 Référence de la structure HufVertex

Sommet d'un arbre binaire de Huffman.

```
#include <hufTree.h>
```

Graphe de collaboration de HufVertex :



Champs de données

- unsigned int [value](#)
- [bytecount](#) count
- struct [HufVertex](#) * [child_l](#)
- struct [HufVertex](#) * [child_r](#)

3.2.1 Description détaillée

Sommet d'un arbre binaire de Huffman.

Sommet possédant éventuellement une valeur (caractère associé), un effectif d'apparition (nombre d'occurrences du caractère dans la source) et un sommet enfant à gauche et/ou à droite.

3.2.2 Documentation des champs

3.2.2.1 struct [HufVertex](#)* [HufVertex](#) : :child_l

Sommet enfant à gauche, ou NULL si ce sommet n'a pas d'enfant à gauche.

3.2.2.2 struct [HufVertex](#)* [HufVertex](#) : :child_r

Sommet enfant à droite, ou NULL si ce sommet n'a pas d'enfant à droite.

3.2.2.3 bytecount HufVertex : :count

Effectif du caractère associé au sommet, ou -1 si aucun caractère n'est associé.

3.2.2.4 unsigned int HufVertex : :value

Caractère associé au sommet, ou -1 si aucun caractère n'est associé (par exemple pour les sommets internes).

La documentation de cette structure a été générée à partir du fichier suivant :

— [inc/huftree.h](#)

3.3 Référence de la structure ReadBuffer

Tampon de lecture bit par bit.

```
#include <buffer.h>
```

Champs de données

- `int data`
- `size_t next_bit`
- `FILE* source_file`

3.3.1 Description détaillée

Tampon de lecture bit par bit.

Tampon permettant d'abstraire la lecture d'un fichier bit par bit au lieu d'octet par octet. Un octet est lu depuis le fichier seulement lorsque tous les bits du précédent octet ont été consommés.

3.3.2 Documentation des champs

3.3.2.1 int ReadBuffer : :data

Données du tampon de lecture en attente de lecture.

3.3.2.2 size_t ReadBuffer : :next_bit

Numéro du prochain bit de [ReadBuffer : :data](#) à lire.

3.3.2.3 FILE* ReadBuffer : :source_file

Fichier depuis lequel les données sont lues.

La documentation de cette structure a été générée à partir du fichier suivant :

— [inc/buffer.h](#)

3.4 Référence de la structure WriteBuffer

Tampon d'écriture bit par bit.

```
#include <buffer.h>
```

Champs de données

- int [data](#)
- size_t [pending_bits](#)
- bytcount [flushed_count](#)
- FILE * [dest_file](#)

3.4.1 Description détaillée

Tampon d'écriture bit par bit.

Tampon permettant d'abstraire l'écriture dans un fichier bit par bit au lieu d'octet par octet. Les bits sont vidés dans le fichier dès qu'un octet est complet, ou lorsque le vidage est forcé.

3.4.2 Documentation des champs

3.4.2.1 int WriteBuffer : :data

Données du tampon d'écriture en attente de vidage.

3.4.2.2 FILE* WriteBuffer : :dest_file

Fichier dans lequel le tampon est vidé.

3.4.2.3 bytcount WriteBuffer : :flushed_count

Nombre de vidages effectués par le tampon.

3.4.2.4 size_t WriteBuffer : :pending_bits

Nombre de bits utilisés dans [WriteBuffer : :data](#).

La documentation de cette structure a été générée à partir du fichier suivant :

- [inc/buffer.h](#)

Chapitre 4

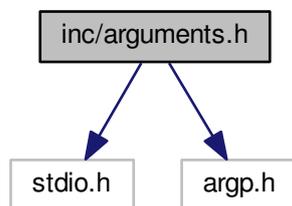
Documentation des fichiers

4.1 Référence du fichier inc/arguments.h

Interprétation des arguments (options et opérandes) passés au programme.

```
#include <stdio.h>
#include <argp.h>
```

Graphe des dépendances par inclusion de arguments.h :



Structures de données

- struct [Args](#)
Valeurs des arguments (options et opérandes) passés au programme.

Définitions de type

- typedef struct [Args](#) [Args](#)
Valeurs des arguments (options et opérandes) passés au programme.

Fonctions

- error_t [parseArgument](#) (int key, char *arg, struct argp_state *state)
*Interpréter l'argument *key* de valeur *arg* dans la structure *argp* d'état *state*.*

4.1.1 Description détaillée

Interprétation des arguments (options et opérandes) passés au programme.

4.1.2 Documentation des fonctions

4.1.2.1 `error_t parseArgument (int key, char * arg, struct argp_state * state)`

Interpréter l'argument `key` de valeur `arg` dans la structure `argp` d'état `state`.

Paramètres

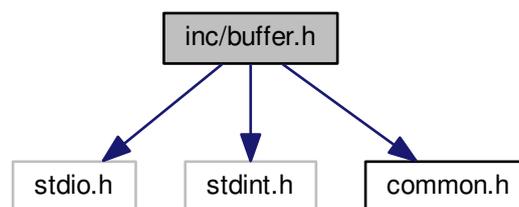
<code>key</code>	Clé de l'option, ou clé spéciale pour indiquer une opérande
<code>arg</code>	Valeur de l'option ou de l'opérande
<code>state</code>	État actuel de <code>argp</code>

4.2 Référence du fichier `inc/buffer.h`

Abstraction de l'écriture et la lecture d'un fichier bit à bit.

```
#include <stdio.h>
#include <stdint.h>
#include "common.h"
```

Graphe des dépendances par inclusion de `buffer.h` :



Structures de données

- struct [WriteBuffer](#)
Tampon d'écriture bit par bit.
- struct [ReadBuffer](#)
Tampon de lecture bit par bit.

Définitions de type

- typedef struct [WriteBuffer](#) [WriteBuffer](#)
Tampon d'écriture bit par bit.
- typedef struct [ReadBuffer](#) [ReadBuffer](#)
Tampon de lecture bit par bit.

Fonctions

- [WriteBuffer](#) [createWriteBuffer](#) (FILE *dest)
Initialiser un tampon d'écriture.
- void [putBuffer](#) (char bit, [WriteBuffer](#) *buffer)
Écrire un bit dans le tampon d'écriture.
- void [flushBuffer](#) ([WriteBuffer](#) *buffer)
Forcer le vidage du tampon d'écriture.
- [bytecount](#) [getFlushedCount](#) ([WriteBuffer](#) *buffer)
Récupérer le nombre de vidages effectués par ce tampon.
- [ReadBuffer](#) [createReadBuffer](#) (FILE *source)
Initialiser un tampon de lecture.
- char [getBuffer](#) ([ReadBuffer](#) *buffer)
Lire un bit depuis le tampon de lecture.

4.2.1 Description détaillée

Abstraction de l'écriture et la lecture d'un fichier bit à bit.

Définir des structures et des fonctions de manipulation liées à un fichier et permettant d'accéder à son contenu ou de le modifier bit par bit au lieu d'octet par octet.

4.2.2 Documentation des définitions de type

4.2.2.1 typedef struct [ReadBuffer](#) [ReadBuffer](#)

Tampon de lecture bit par bit.

Tampon permettant d'abstraire la lecture d'un fichier bit par bit au lieu d'octet par octet. Un octet est lu depuis le fichier seulement lorsque tous les bits du précédent octet ont été consommés.

4.2.2.2 typedef struct [WriteBuffer](#) [WriteBuffer](#)

Tampon d'écriture bit par bit.

Tampon permettant d'abstraire l'écriture dans un fichier bit par bit au lieu d'octet par octet. Les bits sont vidés dans le fichier dès qu'un octet est complet, ou lorsque le vidage est forcé.

4.2.3 Documentation des fonctions

4.2.3.1 [ReadBuffer](#) [createReadBuffer](#) (FILE * source)

Initialiser un tampon de lecture.

Initialiser un tampon vide avec le fichier `source` comme source.

Paramètres

<i>source</i>	Fichier source.
---------------	-----------------

Renvoie

Tampon créé.

4.2.3.2 WriteBuffer createWriteBuffer (FILE * *dest*)

Initialiser un tampon d'écriture.

Initialiser un tampon vide avec le fichier *dest* pour destination.

Paramètres

<i>dest</i>	Fichier de destination.
-------------	-------------------------

Renvoie

Tampon créé.

4.2.3.3 void flushBuffer (WriteBuffer * *buffer*)

Forcer le vidage du tampon d'écriture.

Forcer le vidage du tampon dans son fichier et sa réinitialisation. Si le tampon est vide, pas de vidage (pas d'octet superflu écrit). Si le tampon n'est pas rempli, complétion à droite par des 0.

Paramètres

<i>buffer</i>	Tampon à vider.
---------------	-----------------

4.2.3.4 char getBuffer (ReadBuffer * *buffer*)

Lire un bit depuis le tampon de lecture.

Lire le prochain bit depuis le tampon de lecture, et lire un octet depuis la source si besoin est.

Paramètres

<i>buffer</i>	Tampon depuis lequel lire un bit.
---------------	-----------------------------------

Renvoie

Valeur du bit qui est lu : 0 ou 1.

4.2.3.5 bytecount getFlushedCount (WriteBuffer * buffer)

Récupérer le nombre de vidages effectués par ce tampon.

Permet de savoir combien d'octets ont été écrits dans le fichier par le tampon `buffer`.

Paramètres

<code>buffer</code>	Tampon à vérifier.
---------------------	--------------------

Renvoie

Nombre de vidages effectués.

4.2.3.6 void putBuffer (char bit, WriteBuffer * buffer)

Écrire un bit dans le tampon d'écriture.

Ajouter le bit `bit` aux données du tampon `buffer`. Si le tampon est plein, vidage dans le fichier associé.

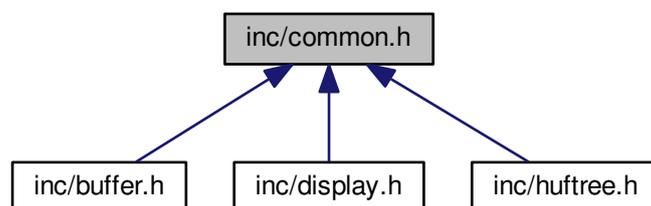
Paramètres

<code>bit</code>	Bit à ajouter au tampon (0 ou 1).
<code>buffer</code>	Tampon dans lequel ajouter le bit.

4.3 Référence du fichier inc/common.h

Définitions communes pour le reste du code.

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Macros

— #define TRUE 1

- *Macro correspondant à la valeur booléenne vraie (1).*
#define `FALSE` 0
- *Macro correspondant à la valeur booléenne fausse (0).*
#define `NUM_CHARS` 256
- *Macro correspondant au nombre de caractères possibles.*

Définitions de type

- typedef long long unsigned int `bytecount`
Alias du type d'entier non-signé 64 bits pour compter les longueurs de fichiers.

4.3.1 Description détaillée

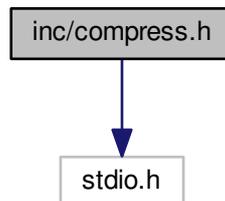
Définitions communes pour le reste du code.

4.4 Référence du fichier inc/compress.h

Compression des fichiers.

```
#include <stdio.h>
```

Grappe des dépendances par inclusion de `compress.h` :



Fonctions

- void `compress` (FILE *source, FILE *dest)
Compresser le fichier source vers dest.

4.4.1 Description détaillée

Compression des fichiers.

4.4.2 Documentation des fonctions

4.4.2.1 void `compress` (FILE * source, FILE * dest)

Compresser le fichier `source` vers `dest`.

Lit les caractères de `source`, applique l'algorithme de Huffman pour déterminer une représentation compacte et la stocke dans `dest`.

Paramètres

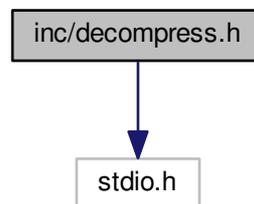
<i>source</i>	Fichier ouvert en lecture depuis lequel lire les données.
<i>dest</i>	Fichier ouvert en écriture dans lequel écrire les données compressées.

4.5 Référence du fichier inc/decompress.h

Décompression des fichiers.

```
#include <stdio.h>
```

Graphes des dépendances par inclusion de decompress.h :



Fonctions

- void `decompress` (FILE **source*, FILE **dest*)
Décompresser le fichier source vers dest.

4.5.1 Description détaillée

Décompression des fichiers.

4.5.2 Documentation des fonctions

4.5.2.1 void `decompress` (FILE * *source*, FILE * *dest*)

Décompresser le fichier *source* vers *dest*.

Lit le fichier *source* ayant été compressé avec l'algorithme de Huffman et écrit les données décompressées dans *dest*.

Paramètres

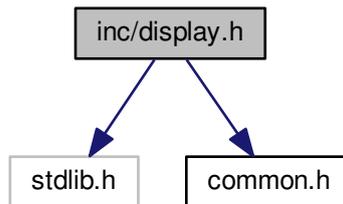
<i>source</i>	Fichier ouvert en lecture depuis lequel lire les données.
<i>dest</i>	Fichier ouvert en écriture dans lequel écrire les données décompressées.

4.6 Référence du fichier inc/display.h

Gestion de l'affichage.

```
#include <stdlib.h>
#include "common.h"
```

Graphe des dépendances par inclusion de display.h :



Définitions de type

- typedef struct [HufVertex](#) **HufVertex**
- typedef [HufVertex](#) * **HufTree**

Fonctions

- void [printVerbose](#) (const char *format,...)
Écrire un message de débogage sur la sortie d'erreurs.
- void [setVerbose](#) (int enable)
Activer ou désactiver l'affichage des messages de débogage.
- int [isVerbose](#) ()
Vérifie si les messages de débogage sont activés ou non.
- void [printTree](#) ([HufTree](#) tree)
Afficher sur la sortie d'erreurs une représentation de l'arbre passé en paramètre.
- void [printCountsTable](#) ([bytecount](#) *counts, [bytecount](#) total, [size_t](#) size)
Afficher sur la sortie d'erreurs le tableau associant les caractères à leur effectif d'apparition passé en argument.
- void [printLabelsTable](#) (char **labels, [size_t](#) size)
Afficher sur la sortie d'erreurs le tableau associant les caractères à leur étiquette passé en argument.

4.6.1 Description détaillée

Gestion de l'affichage.

4.6.2 Documentation des fonctions

4.6.2.1 void printCountsTable ([bytecount](#) * counts, [bytecount](#) total, [size_t](#) size)

Afficher sur la sortie d'erreurs le tableau associant les caractères à leur effectif d'apparition passé en argument.

Paramètres

<i>counts</i>	Tableau d'association des effectifs.
<i>total</i>	Nombre total de caractères.
<i>size</i>	Taille du tableau <i>counts</i> .

4.6.2.2 void printLabelsTable (char ** labels, size_t size)

Afficher sur la sortie d'erreurs le tableau associant les caractères à leur étiquette passé en argument.

*

Paramètres

<i>labels</i>	Tableau d'association des étiquettes.
<i>size</i>	Taille du tableau <i>labels</i> .

4.6.2.3 void printTree (HufTree tree)

Afficher sur la sortie d'erreurs une représentation de l'arbre passé en paramètre.

Paramètres

<i>tree</i>	L'arbre à afficher.
-------------	---------------------

4.6.2.4 void printVerbose (const char * format, ...)

Écrire un message de débogage sur la sortie d'erreurs.

Cette fonction n'a aucun effet si l'affichage des messages de débogage ne sont pas activés, sinon le message est affiché dans `stderr`.

Paramètres

<i>format</i>	Format d'affichage à la printf.
...	Variables pour l'affichage.

4.6.2.5 void setVerbose (int enable)

Activer ou désactiver l'affichage des messages de débogage.

Si le paramètre est vrai, les appels suivants à `printVerbose` seront affichés sur la sortie d'erreurs. Sinon, ils n'auront aucun effet.

Paramètres

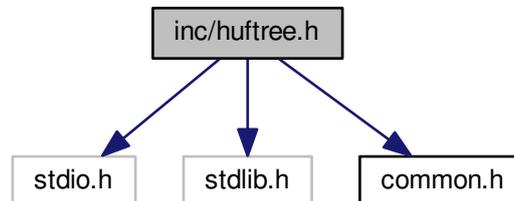
<code>enable</code>	Activer ou désactiver l'affichage.
---------------------	------------------------------------

4.7 Référence du fichier `inc/huftree.h`

Représentation mémoire des arbres binaires de Huffman.

```
#include <stdio.h>
#include <stdlib.h>
#include "common.h"
```

Grphe des dépendances par inclusion de `huftree.h` :



Structures de données

- struct `HufVertex`
Sommet d'un arbre binaire de Huffman.

Définitions de type

- typedef struct `HufVertex` `HufVertex`
Sommet d'un arbre binaire de Huffman.
- typedef `HufVertex *` `HufTree`
Arbre binaire de Huffman (pointeur vers un `HufVertex`)
- typedef struct `WriteBuffer` `WriteBuffer`
- typedef struct `ReadBuffer` `ReadBuffer`

Fonctions

- `HufTree` `createTree` (`bytecount` *`counts`)
Construire un arbre de Huffman.
- void `writeTree` (`HufTree` `tree`, `WriteBuffer` *`buffer`)
Écrire un arbre dans le tampon d'écriture donné.
- `HufTree` `readTree` (`ReadBuffer` *`buffer`)
Lire un arbre depuis le tampon de lecture donné.
- void `freeTree` (`HufTree` `tree`)
Libérer la mémoire occupée par un arbre.
- char ** `createTreeLabels` (`HufTree` `tree`)
Créer la clé de codage à partir d'un arbre de Huffman.
- void `freeTreeLabels` (char **`labels`)
Libérer la mémoire occupée par une clé de codage.

4.7.1 Description détaillée

Représentation mémoire des arbres binaires de Huffman.

Définir une structure permettant de représenter en mémoire les arbres binaires de Huffman et des fonctions permettant de manipuler ces arbres.

4.7.2 Documentation des définitions de type

4.7.2.1 typedef struct HufVertex HufVertex

Sommet d'un arbre binaire de Huffman.

Sommet possédant éventuellement une valeur (caractère associé), un effectif d'apparition (nombre d'occurrences du caractère dans la source) et un sommet enfant à gauche et/ou à droite.

4.7.3 Documentation des fonctions

4.7.3.1 HufTree createTree (bytcount * counts)

Construire un arbre de Huffman.

Lire les effectifs de caractères dans `counts` et en déduire un arbre binaire de Huffman. Le résultat doit être libéré avec la fonction `freeTree`.

Paramètres

<code>counts</code>	Comptes des caractères.
---------------------	-------------------------

Renvoie

Arbre construit.

4.7.3.2 char** createTreeLabels (HufTree tree)

Créer la clé de codage à partir d'un arbre de Huffman.

Associer à chaque feuille de l'arbre une étiquette unique basée sur sa position dans l'arbre. Aucune étiquette n'est ainsi préfixe d'une autre. Le tableau renvoyé associe chaque caractère ASCII à son préfixe ou à NULL s'il n'est pas présent dans l'arbre. Le résultat doit être libéré avec `freeTreeLabels`.

Paramètres

<code>tree</code>	Arbre source de la clé de codage.
-------------------	-----------------------------------

Renvoie

Clé générée.

4.7.3.3 void freeTree (HufTree tree)

Libérer la mémoire occupée par un arbre.

Paramètres

<i>tree</i>	Arbre à libérer.
-------------	------------------

4.7.3.4 void freeTreeLabels (char ** labels)

Libérer la mémoire occupée par une clé de codage.

Paramètres

<i>labels</i>	Clé à libérer.
---------------	----------------

4.7.3.5 HufTree readTree (ReadBuffer * buffer)

Lire un arbre depuis le tampon de lecture donné.

Reconstruire un arbre de Huffman à partir du tampon de lecture *buffer*. Le résultat doit être libéré avec la fonction *freeTree*.

Renvoie

L'arbre reconstruit.

4.7.3.6 void writeTree (HufTree tree, WriteBuffer * buffer)

Écrire un arbre dans le tampon d'écriture donné.

Écrire une représentation binaire de l'arbre *tree* dans le tampon *buffer*.

Paramètres

<i>tree</i>	Arbre à représenter.
<i>buffer</i>	Tampon cible.

Index

- Args, 5
 - compress, 5
 - dest, 5
 - source, 5
 - verbose, 5
- arguments.h
 - parseArgument, 10
- buffer.h
 - createReadBuffer, 11
 - createWriteBuffer, 12
 - flushBuffer, 12
 - getBuffer, 12
 - getFlushedCount, 12
 - putBuffer, 13
 - ReadBuffer, 11
 - WriteBuffer, 11
- child_l
 - HufVertex, 6
- child_r
 - HufVertex, 6
- compress
 - Args, 5
 - compress.h, 14
- compress.h
 - compress, 14
- count
 - HufVertex, 6
- createReadBuffer
 - buffer.h, 11
- createTree
 - hufree.h, 19
- createTreeLabels
 - hufree.h, 19
- createWriteBuffer
 - buffer.h, 12
- data
 - ReadBuffer, 7
 - WriteBuffer, 8
- decompress
 - decompress.h, 15
- decompress.h
 - decompress, 15
- dest
 - Args, 5
- dest_file
 - WriteBuffer, 8
- display.h
 - printCountsTable, 16
 - printLabelsTable, 17
 - printTree, 17
 - printVerbose, 17
 - setVerbose, 17
- flushBuffer
 - buffer.h, 12
- flushed_count
 - WriteBuffer, 8
- freeTree
 - hufree.h, 20
- freeTreeLabels
 - hufree.h, 20
- getBuffer
 - buffer.h, 12
- getFlushedCount
 - buffer.h, 12
- HufVertex, 6
 - child_l, 6
 - child_r, 6
 - count, 6
 - hufree.h, 19
 - value, 7
- hufree.h
 - createTree, 19
 - createTreeLabels, 19
 - freeTree, 20
 - freeTreeLabels, 20
 - HufVertex, 19
 - readTree, 20
 - writeTree, 20
- inc/arguments.h, 9
- inc/buffer.h, 10
- inc/common.h, 13
- inc/compress.h, 14
- inc/decompress.h, 15
- inc/display.h, 16
- inc/hufree.h, 18
- next_bit
 - ReadBuffer, 7
- parseArgument
 - arguments.h, 10
- pending_bits
 - WriteBuffer, 8
- printCountsTable

- display.h, 16
- printLabelsTable
 - display.h, 17
- printTree
 - display.h, 17
- printVerbose
 - display.h, 17
- putBuffer
 - buffer.h, 13
- ReadBuffer, 7
 - buffer.h, 11
 - data, 7
 - next_bit, 7
 - source_file, 7
- readTree
 - hufree.h, 20
- setVerbose
 - display.h, 17
- source
 - Args, 5
- source_file
 - ReadBuffer, 7
- value
 - HufVertex, 7
- verbose
 - Args, 5
- WriteBuffer, 8
 - buffer.h, 11
 - data, 8
 - dest_file, 8
 - flushed_count, 8
 - pending_bits, 8
- writeTree
 - hufree.h, 20