Huffman

0.1

Généré par Doxygen 1.8.11

Table des matières

Chapitre 1

Index des structures de données

1.1 Structures de données

Liste des structures de données avec une brève description :

Chapitre 2

Index des fichiers

2.1 Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

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

Compresse-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'occurences 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
- bytecount 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 bytecount 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

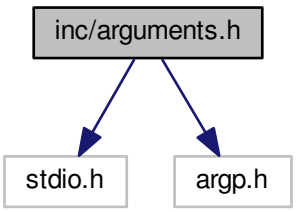
Documentation des fichiers

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

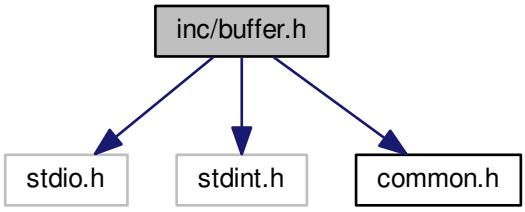
key	Clé de l'option, ou clé spéciale pour indiquer une opérande
arg	Valeur de l'option ou de l'opérande
state	État actuel de argp

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)
  Récupèrer 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

source	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

dest	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

buffer	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

buffer	Tampon depuis lequel lire un bit.

Renvoie

Valeur du bit qui est lu : 0 ou 1.

4.2.3.5 bytecount getFlushedCount (WriteBuffer * *buffer*)

Récupèrer 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

buffer	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.

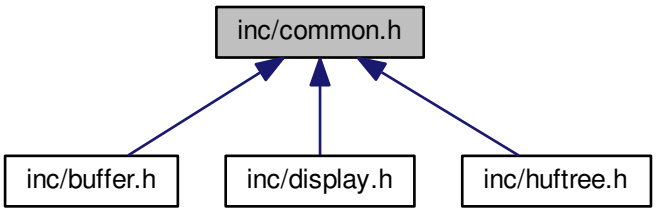
Paramètres

bit	Bit à ajouter au tampon (0 ou 1).
buffer	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

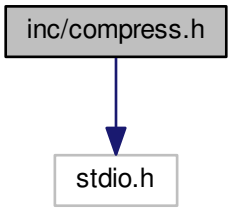
Définitions communes pour le reste du code.

4.4 Référence du fichier inc/compress.h

Compression des fichiers.

`#include <stdio.h>`

Graphe 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

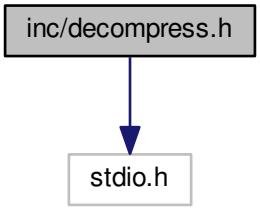
source	Fichier ouvert en lecture depuis lequel lire les données.
dest	Fichier ouvert en écriture dans lequel écrire les données compressées.

4.5 Référence du fichier inc/decompress.h

Décompression des fichiers.

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

Graphe 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

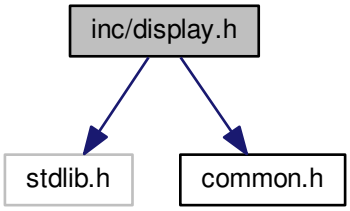
source	Fichier ouvert en lecture depuis lequel lire les données.
dest	Fichier ouvert en écriture dans lequel écrire les données décompressées.

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

counts	Tableau d'association des effectifs.
total	Nombre total de caractères.
size	Taille du tableau `counts`.

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

labels	Tableau d'association des étiquettes.
size	Taille du tableau `labels`.

4.6.2.3 void printTree (HufTree *tree*)

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

Paramètres

tree	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

format	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

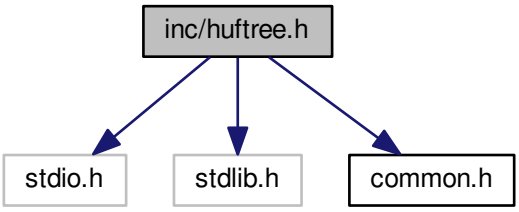
enable	Activer ou désactiver l'affichage.

4.7 Référence du fichier inc/huftree.h

Représentation mémoire des arbres binaires de Huffman.

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

Graphe 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'occurences 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 (bytecount * *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

counts	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

tree	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

tree	Arbre à libérer.

4.7.3.4 void freeTreeLabels (char ** *labels*)

Libérer la mémoire occupée par une clé de codage.

Paramètres

labels	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

tree	Arbre à représenter.
buffer	Tampon cible.

Index