

Huffman

Generated by Doxygen 1.9.6



<b>1 README</b>	<b>1</b>
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 huffman_node Struct Reference	7
4.1.1 Detailed Description	7
4.1.2 Member Data Documentation	7
4.1.2.1 frequency	7
4.1.2.2 right	7
4.1.2.3 symbol	7
<b>5 File Documentation</b>	<b>9</b>
5.1 args_handler.cpp File Reference	9
5.1.1 Detailed Description	9
5.1.2 Function Documentation	10
5.1.2.1 arguments_handler()	10
5.1.2.2 check_arg()	10
5.1.2.3 push_into_error_buffer()	11
5.1.3 Variable Documentation	11
5.1.3.1 error_messages	11
5.2 args_handler.h File Reference	11
5.2.1 Detailed Description	12
5.2.2 Function Documentation	12
5.2.2.1 arguments_handler()	12
5.2.2.2 check_arg()	12
5.2.2.3 push_into_error_buffer()	13
5.3 args_handler.h	13
5.4 error_handler.cpp File Reference	13
5.4.1 Detailed Description	14
5.4.2 Function Documentation	14
5.4.2.1 error_handler()	14
5.5 error_handler.h File Reference	14
5.5.1 Detailed Description	15
5.5.2 Function Documentation	15
5.5.2.1 error_handler()	15
5.6 error_handler.h	15
5.7 file_handler.cpp File Reference	15
5.7.1 Detailed Description	16
5.7.2 Function Documentation	16

5.7.2.1 read_file()	16
5.7.2.2 write_file()	17
5.8 file_handler.h File Reference	17
5.8.1 Detailed Description	17
5.8.2 Function Documentation	17
5.8.2.1 read_file()	18
5.8.2.2 write_file()	18
5.9 file_handler.h	18
5.10 huffman.cpp File Reference	19
5.10.1 Detailed Description	19
5.10.2 Function Documentation	20
5.10.2.1 clear_tree()	20
5.10.2.2 compress()	20
5.10.2.3 create_tree()	20
5.10.2.4 decompress()	21
5.10.2.5 escape_char()	21
5.10.2.6 is_leaf()	21
5.10.2.7 rebuild_tree()	22
5.10.2.8 unescape_char()	22
5.10.3 Variable Documentation	22
5.10.3.1 lowest_frequency	23
5.11 huffman.h File Reference	23
5.11.1 Detailed Description	23
5.11.2 Function Documentation	24
5.11.2.1 clear_tree()	24
5.11.2.2 compress()	24
5.11.2.3 create_tree()	24
5.11.2.4 decompress()	25
5.11.2.5 escape_char()	25
5.11.2.6 is_leaf()	25
5.11.2.7 rebuild_tree()	26
5.11.2.8 unescape_char()	26
5.12 huffman.h	26
5.13 main.cpp File Reference	27
5.13.1 Detailed Description	27

# Chapter 1

## README

Program do kompresji/dekompresji plików przy użyciu metody Huffmana.

Usage: `huffman -i <input_file> -o <output_file> -t <mode> -s <dictionary_file>`

Available parameters: `-i <input_file>` - path to the input file `-o <output_file>` - path to the output file `-t <mode>` - operation mode; `k` <- compression, `d` <- decompression `-s <dictionary_path>` - path to the dictionary file

Projekt na PPK.



## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">huffman_node</a>	Huffman node structure used later to create a tree of frequencies . . . . .	7
------------------------------	---	---





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">args_handler.cpp</a>	Implements functions defined in <a href="#">args_handler.h</a> . . . . .	9
<a href="#">args_handler.h</a>	Defines functions needed for parameters recognition and error handling . . . . .	11
<a href="#">error_handler.cpp</a>	Implements functions that are defined in <a href="#">error_handler.h</a> . . . . .	13
<a href="#">error_handler.h</a>	Simple header that holds the general error handler for the program . . . . .	14
<a href="#">file_handler.cpp</a>	Implements functions for interaction with the filesystem defined in <a href="#">file_handler.h</a> . . . . .	15
<a href="#">file_handler.h</a>	Defines functions for interaction with the filesystem . . . . .	17
<a href="#">huffman.cpp</a>	Implements functions defined in <a href="#">huffman.h</a> , contains the core functionality of this program . . .	19
<a href="#">huffman.h</a>	Defines core functions of the program . . . . .	23
<a href="#">main.cpp</a>	Main program file . . . . .	27



## Chapter 4

# Class Documentation

### 4.1 `huffman_node` Struct Reference

`huffman` node structure used later to create a tree of frequencies.

```
#include <huffman.h>
```

#### Public Attributes

- `char` [symbol](#)
- `unsigned int` [frequency](#)
- [huffman\\_node](#) \* `left`
- [huffman\\_node](#) \* `right`

#### 4.1.1 Detailed Description

`huffman` node structure used later to create a tree of frequencies.

#### 4.1.2 Member Data Documentation

##### 4.1.2.1 `frequency`

```
unsigned int huffman_node::frequency
```

The frequency of that letter it has been appearing in data

##### 4.1.2.2 `right`

```
huffman\_node * huffman_node::right
```

Pointers to the left (1) and right (0) child of a node.

##### 4.1.2.3 `symbol`

```
char huffman_node::symbol
```

The symbol, letter that's stored in a node, later it's just a null character

The documentation for this struct was generated from the following file:

- [huffman.h](#)



## Chapter 5

# File Documentation

### 5.1 args\_handler.cpp File Reference

Implements functions defined in [args\\_handler.h](#).

```
#include <iostream>
#include <map>
#include <vector>
#include "args_handler.h"
#include "error_handler.h"
```

#### Functions

- bool [check\\_arg](#) (std::string &value, std::string arg)  
*Checks if parameters exist.*
- bool [push\\_into\\_error\\_buffer](#) (std::vector< std::string > &target, std::string &error)  
*Pushes errors into an error buffer so if more errors occur in one go the program will list them line-by-line.*
- bool [arguments\\_handler](#) (int argc, char \*argv[], std::string &input, std::string &output, std::string &mode, std::string &dictionary)  
*Handles the arguments passed to the program.*

#### Variables

- std::map< std::string, std::string > [error\\_messages](#)

#### 5.1.1 Detailed Description

Implements functions defined in [args\\_handler.h](#).

Author

Michał Czyż

## 5.1.2 Function Documentation

### 5.1.2.1 arguments\_handler()

```
bool arguments_handler (
    int argc,
    char * argv[],
    std::string & input,
    std::string & output,
    std::string & mode,
    std::string & dictionary )
```

Handles the arguments passed to the program.

#### Parameters

<i>argc</i>	The number of arguments passed to the program.
<i>argv</i>	An array of pointers to the arguments passed to the program.
<i>input</i>	A reference to a string that will contain the file name of the input.
<i>output</i>	A reference to a string that will contain the file name of the output.
<i>mode</i>	A reference to a string that will contain program mode (compression/decompression).
<i>dictionary</i>	A reference to a string that will contain the file name of the dictionary file.

#### Returns

Will return true if the arguments were successfully handled. Otherwise, it will return false.

### 5.1.2.2 check\_arg()

```
bool check_arg (
    std::string & value,
    std::string arg )
```

Checks if parameters exist.

#### Parameters

<i>value</i>	Value of an argument.
<i>arg</i>	Passed argument.

#### Returns

Returns false if there is a problem i.e. arg is empty or there is no dash.

### 5.1.2.3 push\_into\_error\_buffer()

```
bool push_into_error_buffer (
    std::vector< std::string > & target,
    std::string & error )
```

Pushes errors into an error buffer so if more errors occur in one go the program will list them line-by-line.

#### Parameters

<i>target</i>	A reference to the target vector of string holding all errors.
<i>error</i>	A reference to a string holding an error message.

#### Returns

Will always return false to signal that an error occurred.

## 5.1.3 Variable Documentation

### 5.1.3.1 error\_messages

```
std::map<std::string, std::string> error_messages
```

#### Initial value:

```
{
    {"not_enough", "Too few arguments.\n"},
    {"input", "-i argument requires <input_file> path.\n"},
    {"output", "-o argument requires <output_file> path.\n"},
    {"mode", "-t argument requires <mode>; k <- compression, d <- decompression.\n"},
    {"dictionary", "-s argument requires <dictionary_file> path.\n"},
}
```

## 5.2 args\_handler.h File Reference

Defines functions needed for parameters recognition and error handling.

```
#include <iostream>
#include <map>
#include <vector>
#include <string>
```

### Functions

- bool [check\\_arg](#) (std::string &value, std::string arg)  
*Checks if parameters exist.*
- bool [push\\_into\\_error\\_buffer](#) (std::vector< std::string > &target, std::string &error)  
*Pushes errors into an error buffer so if more errors occur in one go the program will list them line-by-line.*
- bool [arguments\\_handler](#) (int argc, char \*argv[], std::string &input, std::string &output, std::string &mode, std::string &dictionary)  
*Handles the arguments passed to the program.*

## 5.2.1 Detailed Description

Defines functions needed for parameters recognition and error handling.

Author

Michał Czyż

## 5.2.2 Function Documentation

### 5.2.2.1 arguments\_handler()

```
bool arguments_handler (
    int argc,
    char * argv[],
    std::string & input,
    std::string & output,
    std::string & mode,
    std::string & dictionary )
```

Handles the arguments passed to the program.

#### Parameters

<i>argc</i>	The number of arguments passed to the program.
<i>argv</i>	An array of pointers to the arguments passed to the program.
<i>input</i>	A reference to a string that will contain the file name of the input.
<i>output</i>	A reference to a string that will contain the file name of the output.
<i>mode</i>	A reference to a string that will contain program mode (compression/decompression).
<i>dictionary</i>	A reference to a string that will contain the file name of the dictionary file.

#### Returns

Will return true if the arguments were successfully handled. Otherwise, it will return false.

### 5.2.2.2 check\_arg()

```
bool check_arg (
    std::string & value,
    std::string arg )
```

Checks if parameters exist.



## Parameters

<i>value</i>	Value of an argument.
<i>arg</i>	Passed argument.

## Returns

Returns false if there is a problem i.e. arg is empty or there is no dash.

## 5.2.2.3 push\_into\_error\_buffer()

```
bool push_into_error_buffer (
    std::vector< std::string > & target,
    std::string & error )
```

Pushes errors into an error buffer so if more errors occur in one go the program will list them line-by-line.

## Parameters

<i>target</i>	A reference to the target vector of string holding all errors.
<i>error</i>	A reference to a string holding an error message.

## Returns

Will always return false to signal that an error occurred.

## 5.3 args\_handler.h

[Go to the documentation of this file.](#)

```
00001
00007 #ifndef ARGS_H
00008 #define ARGS_H
00009
00010 #include <iostream>
00011 #include <map>
00012 #include <vector>
00013 #include <string>
00014
00019 bool check_arg(std::string &value, std::string arg);
00020
00025 bool push_into_error_buffer(std::vector<std::string> &target, std::string &error);
00026
00035 bool arguments_handler(int argc, char *argv[], std::string &input, std::string &output, std::string
    &mode, std::string &dictionary);
00036
00037 #endif
```

## 5.4 error\_handler.cpp File Reference

Implements functions that are defined in [error\\_handler.h](#).

```
#include <iostream>
#include "error_handler.h"
```

## Functions

- bool [error\\_handler](#) (std::string program\_name, std::string message="")  
*Generates an error output in the console and help, with an instruction of how to use a program.*

### 5.4.1 Detailed Description

Implements functions that are defined in [error\\_handler.h](#).

#### Author

Michał Czyż

### 5.4.2 Function Documentation

#### 5.4.2.1 error\_handler()

```
bool error_handler (
    std::string program_name,
    std::string message )
```

Generates an error output in the console and help, with an instruction of how to use a program.

#### Parameters

<i>program_name</i>	A string that holds a name of a program to printout.
<i>message</i>	A string containing an error message to printout.

#### Returns

Always returns false.

## 5.5 error\_handler.h File Reference

Simple header that holds the general error handler for the program.

```
#include <iostream>
```

## Functions

- bool [error\\_handler](#) (std::string program\_name, std::string message)  
*Generates an error output in the console and help, with an instruction of how to use a program.*

### 5.5.1 Detailed Description

Simple header that holds the general error handler for the program.

#### Author

Michał Czyż

### 5.5.2 Function Documentation

#### 5.5.2.1 error\_handler()

```
bool error_handler (
    std::string program_name,
    std::string message )
```

Generates an error output in the console and help, with an instruction of how to use a program.

#### Parameters

<i>program_name</i>	A string that holds a name of a program to printout.
<i>message</i>	A string containing an error message to printout.

#### Returns

Always returns false.

## 5.6 error\_handler.h

[Go to the documentation of this file.](#)

```
00001
00007 #ifndef ERROR_H
00008 #define ERROR_H
00009
00010 #include <iostream>
00011
00016 bool error_handler(std::string program_name, std::string message);
00017
00018 #endif
```

## 5.7 file\_handler.cpp File Reference

Implements functions for interaction with the filesystem defined in [file\\_handler.h](#).

```
#include <iostream>
#include <map>
#include <sstream>
#include <sys/stat.h>
#include <algorithm>
#include "file_handler.h"
#include "error_handler.h"
```

## Functions

- bool [read\\_file](#) (std::string program\_name, std::string &data, std::string &file\_name, bool binary)  
*Reads data from the provided file.*
- bool [write\\_file](#) (std::string program\_name, std::string &data, std::string &file\_name, bool binary)  
*Writes data to the provided file.*

## Variables

- const int **chunk\_size** = 8

### 5.7.1 Detailed Description

Implements functions for interaction with the filesystem defined in [file\\_handler.h](#).

#### Author

Michał Czyż

### 5.7.2 Function Documentation

#### 5.7.2.1 read\_file()

```
bool read_file (
    std::string program_name,
    std::string & data,
    std::string & file_name,
    bool binary )
```

Reads data from the provided file.

#### Parameters

<i>program_name</i>	A string containing the program_name which in case of an error is passed down to the error_handler.
<i>data</i>	A reference to a string that will hold the data read from the file.
<i>file_name</i>	A reference to a string that holds the file name of a desired file.
<i>binary</i>	A method in which the function should read the file (true for binary or false for normal filestream).

#### Returns

Returns true if operation completed successfully, false if it failed.

### 5.7.2.2 write\_file()

```
bool write_file (
    std::string program_name,
    std::string & data,
    std::string & file_name,
    bool binary )
```

Writed data to the provided file.

#### Parameters

<i>program_name</i>	A string containing the program_name which in case of an error is passed down to the error_handler.
<i>data</i>	A reference to a string that holds the data.
<i>file_name</i>	A reference to a string that holds the file name of a desired file.
<i>binary</i>	A method in which the function should write into the file (true for binary or false for normal filestream).

#### Returns

Returns true if operation completed successfully, false if it failed.

## 5.8 file\_handler.h File Reference

Defines functions for interaction with the filesystem.

```
#include <iostream>
#include <fstream>
```

### Functions

- bool [read\\_file](#) (std::string program\_name, std::string &data, std::string &file\_name, bool binary)  
*Reads data from the provided file.*
- bool [write\\_file](#) (std::string program\_name, std::string &data, std::string &file\_name, bool binary)  
*Writed data to the provided file.*

### 5.8.1 Detailed Description

Defines functions for interaction with the filesystem.

#### Author

Michał Czyż

### 5.8.2 Function Documentation

### 5.8.2.1 read\_file()

```
bool read_file (
    std::string program_name,
    std::string & data,
    std::string & file_name,
    bool binary )
```

Reads data from the provided file.

#### Parameters

<i>program_name</i>	A string containing the program_name which in case of an error is passed down to the error_handler.
<i>data</i>	A reference to a string that will hold the data read from the file.
<i>file_name</i>	A reference to a string that holds the file name of a desired file.
<i>binary</i>	A method in which the function should read the file (true for binary or false for normal filestream).

#### Returns

Returns true if operation completed successfully, false if it failed.

### 5.8.2.2 write\_file()

```
bool write_file (
    std::string program_name,
    std::string & data,
    std::string & file_name,
    bool binary )
```

Writed data to the provided file.

#### Parameters

<i>program_name</i>	A string containing the program_name which in case of an error is passed down to the error_handler.
<i>data</i>	A reference to a string that holds the data.
<i>file_name</i>	A reference to a string that holds the file name of a desired file.
<i>binary</i>	A method in which the function should write into the file (true for binary or false for normal filestream).

#### Returns

Returns true if operation completed successfully, false if it failed.

## 5.9 file\_handler.h

[Go to the documentation of this file.](#)

```

00001
00007 #ifndef FILE_HANDLER
00008 #define FILE_HANDLER
00009
00010 #include <iostream>
00011 #include <fstream>
00012
00019 bool read_file(std::string program_name, std::string &data, std::string &file_name, bool binary);
00020
00027 bool write_file(std::string program_name, std::string &data, std::string &file_name, bool binary);
00028
00029 #endif

```

## 5.10 huffman.cpp File Reference

Implements functions defined in [huffman.h](#), contains the core functionality of this program.

```

#include <iostream>
#include <queue>
#include <map>
#include <sstream>
#include <vector>
#include <string>
#include "huffman.h"

```

### Functions

- bool [is\\_leaf](#) ([huffman\\_node](#) \*node)  
*Checks if the provided node has no children. It is the leaf of a tree.*
- void [compress](#) ([huffman\\_node](#) \*node, std::string data, std::map< char, std::string > &frequency)  
*Compresses data recursively to a binary stream.*
- void [decompress](#) (std::map< std::string, char > &frequency, std::string &data, std::string &result)  
*Decompresses binary stream back to the normal format.*
- [huffman\\_node](#) \* [create\\_tree](#) (std::map< char, int > &frequency)  
*Creates a huffman binary tree out of a map of frequencies.*
- std::map< std::string, char > [rebuild\\_tree](#) (std::string &dictionary)  
*Rebuilds an inverted (easier to process later) frequency map from a string.*
- void [clear\\_tree](#) ([huffman\\_node](#) \*node)  
*Clears a heap of a huffman tree so the program doesn't leak any data.*
- std::string [escape\\_char](#) (char character)  
*Escapes difficult character that might break something later.*
- char [unescape\\_char](#) (std::string character)  
*Unescapes characters and prints them in a normal form.*

### Variables

- auto [lowest\\_frequency](#)

#### 5.10.1 Detailed Description

Implements functions defined in [huffman.h](#), contains the core functionality of this program.

#### Author

Michał Czyż

## 5.10.2 Function Documentation

### 5.10.2.1 clear\_tree()

```
void clear_tree (
    huffman_node * node )
```

Clears a heap of a huffman tree so the program doesn't leak any data.

#### Parameters

<i>node</i>	A pointer to a node that has to be cleared. By default it should be the entire tree.
-------------	--

### 5.10.2.2 compress()

```
void compress (
    huffman_node * node,
    std::string data,
    std::map< char, std::string > & frequency )
```

Compresses data recursively to a binary stream.

#### Parameters

<i>node</i>	A pointer to the huffman tree root.
<i>data</i>	A string of data to be compressed.
<i>frequency</i>	A reference to a frequency map.

### 5.10.2.3 create\_tree()

```
huffman_node * create_tree (
    std::map< char, int > & frequency )
```

Creates a huffman binary tree out of a map of frequencies.

#### Parameters

<i>frequency</i>	A reference to a map of frequencies.
------------------	--------------------------------------



### Returns

Returns a pointer to a huffman binary tree.

#### 5.10.2.4 decompress()

```
void decompress (
    std::map< std::string, char > & frequency,
    std::string & data,
    std::string & result )
```

Decompresses binary stream back to the normal format.

### Parameters

<i>frequency</i>	A reference to an inverted frequency map.
<i>data</i>	A reference to the string of encoded data.
<i>result</i>	A reference to the string that will contain decompressed data.

#### 5.10.2.5 escape\_char()

```
std::string escape_char (
    char character )
```

Escapes difficult character that might break something later.

### Parameters

<i>character</i>	A char that holds a normal character to escape i.e '\n'.
------------------	--

### Returns

Returns an escaped version of a character.

#### 5.10.2.6 is\_leaf()

```
bool is_leaf (
    huffman_node * node )
```

Checks if the provided node has no children. It it the leaf of a tree.

**Parameters**

<i>node</i>	A pointer to the node that has to be checked.
-------------	---

**Returns**

Returns true if the node is a leaf node.

**5.10.2.7 rebuild\_tree()**

```
std::map< std::string, char > rebuild_tree (
    std::string & dictionary )
```

Rebuilds an inverted (easier to process later) frequency map from a string.

**Parameters**

<i>dictionary</i>	A reference to a string that holds a map.
-------------------	---

**Returns**

Returns an inverted frequency map.

**5.10.2.8 unescape\_char()**

```
char unescape_char (
    std::string character )
```

Unescapes characters and prints them in a normal form.

**Parameters**

<i>character</i>	A string holding an encoded character to escape i.e '\n'.
------------------	---

**Returns**

Returns a raw, unescaped character.

**5.10.3 Variable Documentation**

### 5.10.3.1 lowest\_frequency

```
auto lowest_frequency
```

#### Initial value:

```
= [] (huffman_node *left, huffman_node *right)
{
    return left->frequency > right->frequency;
}
```

## 5.11 huffman.h File Reference

Defines core functions of the program.

```
#include <iostream>
#include <map>
#include <string>
```

### Classes

- struct [huffman\\_node](#)  
*huffman node structure used later to create a tree of frequencies.*

### Functions

- bool [is\\_leaf](#) (huffman\_node \*node)  
*Checks if the provided node has no children. It is the leaf of a tree.*
- void [compress](#) (huffman\_node \*node, std::string data, std::map< char, std::string > &frequency)  
*Compresses data recursively to a binary stream.*
- void [decompress](#) (std::map< std::string, char > &frequency, std::string &data, std::string &result)  
*Decompresses binary stream back to the normal format.*
- huffman\_node \* [create\\_tree](#) (std::map< char, int > &frequency)  
*Creates a huffman binary tree out of a map of frequencies.*
- std::map< std::string, char > [rebuild\\_tree](#) (std::string &dictionary)  
*Rebuilds an inverted (easier to process later) frequency map from a string.*
- void [clear\\_tree](#) (huffman\_node \*node)  
*Clears a heap of a huffman tree so the program doesn't leak any data.*
- std::string [escape\\_char](#) (char character)  
*Escapes difficult character that might break something later.*
- char [unescape\\_char](#) (std::string character)  
*Unescapes characters and prints them in a normal form.*

### 5.11.1 Detailed Description

Defines core functions of the program.

#### Author

Michał Czyż

## 5.11.2 Function Documentation

### 5.11.2.1 clear\_tree()

```
void clear_tree (
    huffman_node * node )
```

Clears a heap of a huffman tree so the program doesn't leak any data.

#### Parameters

<i>node</i>	A pointer to a node that has to be cleared. By default it should be the entire tree.
-------------	--

### 5.11.2.2 compress()

```
void compress (
    huffman_node * node,
    std::string data,
    std::map< char, std::string > & frequency )
```

Compresses data recursively to a binary stream.

#### Parameters

<i>node</i>	A pointer to the huffman tree root.
<i>data</i>	A string of data to be compressed.
<i>frequency</i>	A reference to a frequency map.

### 5.11.2.3 create\_tree()

```
huffman_node * create_tree (
    std::map< char, int > & frequency )
```

Creates a huffman binary tree out of a map of frequencies.

#### Parameters

<i>frequency</i>	A reference to a map of frequencies.
------------------	--------------------------------------

### Returns

Returns a pointer to a huffman binary tree.

#### 5.11.2.4 decompress()

```
void decompress (
    std::map< std::string, char > & frequency,
    std::string & data,
    std::string & result )
```

Decompresses binary stream back to the normal format.

### Parameters

<i>frequency</i>	A reference to an inverted frequency map.
<i>data</i>	A reference to the string of encoded data.
<i>result</i>	A reference to the string that will contain decompressed data.

#### 5.11.2.5 escape\_char()

```
std::string escape_char (
    char character )
```

Escapes difficult character that might break something later.

### Parameters

<i>character</i>	A char that holds a normal character to escape i.e '\n'.
------------------	--

### Returns

Returns an escaped version of a character.

#### 5.11.2.6 is\_leaf()

```
bool is_leaf (
    huffman_node * node )
```

Checks if the provided node has no children. It is the leaf of a tree.

**Parameters**

<i>node</i>	A pointer to the node that has to be checked.
-------------	---

**Returns**

Returns true if the node is a leaf node.

**5.11.2.7 rebuild\_tree()**

```
std::map< std::string, char > rebuild_tree (
    std::string & dictionary )
```

Rebuilds an inverted (easier to process later) frequency map from a string.

**Parameters**

<i>dictionary</i>	A reference to a string that holds a map.
-------------------	---

**Returns**

Returns an inverted frequency map.

**5.11.2.8 unescape\_char()**

```
char unescape_char (
    std::string character )
```

Unescapes characters and prints them in a normal form.

**Parameters**

<i>character</i>	A string holding an encoded character to escape i.e '\n'.
------------------	---

**Returns**

Returns a raw, unescaped character.

**5.12 huffman.h**

[Go to the documentation of this file.](#)

00001

```
00007 #ifndef HUFFMAN_H
00008 #define HUFFMAN_H
00009
00010 #include <iostream>
00011 #include <map>
00012 #include <string>
00013
00018 struct huffman_node
00019 {
00020     char symbol;
00021     unsigned int frequency;
00022     huffman_node *left, *right;
00023 };
00024
00028 bool is_leaf(huffman_node *node);
00029
00034 void compress(huffman_node *node, std::string data, std::map<char, std::string> &frequency);
00035
00040 void decompress(std::map<std::string, char> &frequency, std::string &data, std::string &result);
00041
00045 huffman_node* create_tree(std::map<char, int> &frequency);
00046
00050 std::map<std::string, char> rebuild_tree(std::string &dictionary);
00051
00054 void clear_tree(huffman_node *node);
00055
00059 std::string escape_char(char character);
00060
00064 char unescape_char(std::string character);
00065
00066 #endif
```

## 5.13 main.cpp File Reference

Main program file.

```
#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <sstream>
#include "args_handler.h"
#include "file_handler.h"
#include "error_handler.h"
#include "huffman.h"
```

### Functions

- int **main** (int argc, char \*\*argv)

#### 5.13.1 Detailed Description

Main program file.

Author

Michał Czyż





# Index

- args\_handler.cpp, [9](#)
  - arguments\_handler, [10](#)
  - check\_arg, [10](#)
  - error\_messages, [11](#)
  - push\_into\_error\_buffer, [10](#)
- args\_handler.h, [11](#)
  - arguments\_handler, [12](#)
  - check\_arg, [12](#)
  - push\_into\_error\_buffer, [13](#)
- arguments\_handler
  - args\_handler.cpp, [10](#)
  - args\_handler.h, [12](#)
- check\_arg
  - args\_handler.cpp, [10](#)
  - args\_handler.h, [12](#)
- clear\_tree
  - huffman.cpp, [20](#)
  - huffman.h, [24](#)
- compress
  - huffman.cpp, [20](#)
  - huffman.h, [24](#)
- create\_tree
  - huffman.cpp, [20](#)
  - huffman.h, [24](#)
- decompress
  - huffman.cpp, [21](#)
  - huffman.h, [25](#)
- error\_handler
  - error\_handler.cpp, [14](#)
  - error\_handler.h, [15](#)
- error\_handler.cpp, [13](#)
  - error\_handler, [14](#)
- error\_handler.h, [14](#)
  - error\_handler, [15](#)
- error\_messages
  - args\_handler.cpp, [11](#)
- escape\_char
  - huffman.cpp, [21](#)
  - huffman.h, [25](#)
- file\_handler.cpp, [15](#)
  - read\_file, [16](#)
  - write\_file, [16](#)
- file\_handler.h, [17](#)
  - read\_file, [17](#)
  - write\_file, [18](#)
- frequency
  - huffman\_node, [7](#)
- huffman.cpp, [19](#)
  - clear\_tree, [20](#)
  - compress, [20](#)
  - create\_tree, [20](#)
  - decompress, [21](#)
  - escape\_char, [21](#)
  - is\_leaf, [21](#)
  - lowest\_frequency, [22](#)
  - rebuild\_tree, [22](#)
  - unescape\_char, [22](#)
- huffman.h, [23](#)
  - clear\_tree, [24](#)
  - compress, [24](#)
  - create\_tree, [24](#)
  - decompress, [25](#)
  - escape\_char, [25](#)
  - is\_leaf, [25](#)
  - rebuild\_tree, [26](#)
  - unescape\_char, [26](#)
- huffman\_node, [7](#)
  - frequency, [7](#)
  - right, [7](#)
  - symbol, [7](#)
- is\_leaf
  - huffman.cpp, [21](#)
  - huffman.h, [25](#)
- lowest\_frequency
  - huffman.cpp, [22](#)
- main.cpp, [27](#)
- push\_into\_error\_buffer
  - args\_handler.cpp, [10](#)
  - args\_handler.h, [13](#)
- read\_file
  - file\_handler.cpp, [16](#)
  - file\_handler.h, [17](#)
- rebuild\_tree
  - huffman.cpp, [22](#)
  - huffman.h, [26](#)
- right
  - huffman\_node, [7](#)
- symbol
  - huffman\_node, [7](#)

unescape\_char

huffman.cpp, [22](#)

huffman.h, [26](#)

write\_file

file\_handler.cpp, [16](#)

file\_handler.h, [18](#)