The Doomed Ship

2.0

Generated by Doxygen 1.8.15

Chapter 1

The Doomed Ship: an Adventure

1.1 Introduction

This project was started by a group of five students of Computer Science (the group **FSC**) as part of an exam and it developed into something bigger than it originally was.

1.2 This website

This website contains the documentation of both the code and the project itself. It has been generated using Doxygen and stylized using the M.CSS's Doxygen template.

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

Namespace Documentation

5.1 cryptography Namespace Reference

Various cryptography functions.

Functions

- std::string & decrypt (std::string &str, int key=5)
  Decrypts a string.

5.1.1 Detailed Description

Various cryptography functions.

This namespace contains some useful cryptographic functions.

5.1.2 Function Documentation

5.1.2.1 decrypt()

```
std::string & cryptography::decrypt (
            std::string & str,
            int key = 5 )
```

Decrypts a string.

This function decrypts a string that was encrypted using the Caesar Cypher. The string will be saved in the same object where the encrypted string is saved.

Parameters

in	*str*	The original string that will be decrypted. It's the same string where the result will be saved.
in	*key*	The key that will be used to decrypt the string (*optional*).

Chapter 6

Class Documentation

6.1 colored_string Class Reference

A colored string.

`#include <string_utilities.h>`

Public Types

- enum PrintColors : short {
  PrintColors::BLACK = 0, PrintColors::RED = 1, PrintColors::GREEN = 2, PrintColors::YELLOW = 3,
  PrintColors::BLUE = 4, PrintColors::MAGENTA = 5, PrintColors::CYAN = 6, PrintColors::WHITE = 7 }

  Printable colors.

Public Member Functions

- colored_string (const std::string &str, const PrintColors foreground=PrintColors::RED, const PrintColors background=PrintColors::BLACK)

  Colored string's constructor.
- **operator std::string** () const

Private Attributes

- std::string **m_str**
- std::string **m_color**

Friends

- std::ostream & operator<< (std::ostream &os, const colored_string &str)

  Output the colored string.

6.1.1 Detailed Description

A colored string.

This class allows to print a colored string to the stdout. This is a cross platform solution.

6.1.2 Member Enumeration Documentation

6.1.2.1 PrintColors

```
enum colored_string::PrintColors :  short  [strong]
```

Printable colors.

This enumerator contains all the possible colors that can be printed. Each color is identified by an integer.

Warning

The colors have various code based on the platform. This is because some operative systems (like Windows) don't support ASCII Escaped sequences.

Enumerator

BLACK	The black color.
RED	The red color.
GREEN	The green color.
YELLOW	The yellow color.
BLUE	The blue color.
MAGENTA	The magenta color.
CYAN	The cyan color.
WHITE	The white color.

6.1.3 Constructor & Destructor Documentation

6.1.3.1 colored_string()

```
colored_string::colored_string (
            const std::string & str,
            const PrintColors foreground = PrintColors::RED,
            const PrintColors background = PrintColors::BLACK )
```

Colored string's constructor.

This creates a new colored string ready to be printed.

Parameters

in	*str*	The normal string.
in	*foreground*	The foreground color (*optional*).
in	*background*	The background color (*optional*).

6.1.4 Friends And Related Function Documentation

6.1.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const colored_string & str )  [friend]
```

Output the colored string.

This outputs the colored string to a std::ostream (like `cout`).

Note

This function does *not* add a new line at the end of the output.

Parameters

in	*os*	The output stream.
in	*str*	The colored string.

Returns

The modified output stream.

Example

The following line of code will print `"Hello World!"` (with an ending new line character) to the stdout. The string will be printed in *red*, using the color *blue* as background.

```
std::cout « colored_string("Hello world!",
                colored_string::PrintColors::RED,
                colored_string::PrintColors::BLUE)
          « std::endl;
```

The documentation for this class was generated from the following files:

- utilities/string_utilities.h
- utilities/string_utilities.cpp

6.2 game Class Reference

The game.

`#include <game.h>`

Classes

- class languages

  An array of languages.

Public Member Functions

- void **begin** ()

Protected Types

- enum String_Resources : unsigned {
  GAME_TITLE = 0, ORIGINAL_AUTHOR, AUTHOR, COPYRIGHT,
  VERSION, INTRODUCTION, ERROR_STRING, MENU_FIRST_OPTION,
  MENU_SECOND_OPTION, MENU_EXIT, MENU_INPUT_PROMPT, LANGUAGE_SUBMENU_TITLE,
  RES_STRING_NUMBER }

  All the game's strings' codes.

Protected Member Functions

- void exec ()

  The main loop.

Protected Attributes

- std::string mStrings [RES_STRING_NUMBER]

  The array containing all the game strings.
- languages mLanguages = languages({{"it", "Italiano"}, {"en", "English"}})

  The object containing the array of languages.
- languages::AvailableLanguages mCurrentLang

  The current selected language's code.
- bool mEndGame

  Does the game have to end?

Private Member Functions

- void **end_game** ()
- void **change_language** ()
- unsigned **show_menu** ()
- void **show_intro** ()
- void **get_strings** ()

6.2.1 Detailed Description

The game.

This class contains all the core functions of the game. It's in this class that the main loop of the game can be found.

6.2.2 Member Enumeration Documentation

6.2.2.1 String_Resources

```
enum game::String_Resources : unsigned [protected]
```

All the game's strings' codes.

This enumerator is used to get the code associated with a string. This allows to write a more readable code.

Warning

Do *not* modify the first and last values: this enumerator is used to index an array.

Enumerator

GAME_TITLE	The game's title.
ORIGINAL_AUTHOR	The original game's authors.
AUTHOR	The modified game's authors.
COPYRIGHT	A copyright notice.
VERSION	The game's version.
INTRODUCTION	The game's introduction.
ERROR_STRING	The error message that will be printed if an input fails.
MENU_FIRST_OPTION	The first option of the menu.
MENU_SECOND_OPTION	The second option of the menu.
MENU_EXIT	The "Exit" option of the menu.
MENU_INPUT_PROMPT	The message that will be printed to wait a user input in the menu.
LANGUAGE_SUBMENU_TITLE	The title of the language selection sub-menu.
RES_STRING_NUMBER	A useful constant that indicates how many strings are being saved.

6.2.3 Member Function Documentation

6.2.3.1 exec()

```
void game::exec ( ) [protected]
```

The main loop.

This is the main loop of the game. In this loop, all the user's input (regarding actions in the game) and game events take place.

The documentation for this class was generated from the following files:

- game.h
- game.cpp

6.3 game::languages::language Struct Reference

a language

`#include <game.h>`

Public Attributes

- std::string ISO639_1

  The ISO 639-1 code of the language (two letters code).
- std::string name

  The name of the language. This is a name that can be printed and selected by the user.

6.3.1 Detailed Description

a language

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

- game.h

6.4 game::languages Class Reference

An array of languages.

`#include <game.h>`

Classes

- struct language

  a language

Public Types

- enum AvailableLanguages : unsigned { ITALIAN = 0, ENGLISH, NUMBER_OF_AVAILABLE_LANGUAGES }

  All the available languages.

Public Member Functions

- languages (const language(&pLang)[NUMBER_OF_AVAILABLE_LANGUAGES]) noexcept

  Languages' array's constructor.
- language & operator[ ] (AvailableLanguages lang) noexcept

  Get a language.

Private Attributes

- language mLanguages [NUMBER_OF_AVAILABLE_LANGUAGES]

  The underlaying array of languages.

6.4.1 Detailed Description

An array of languages.

This class is a wrapper for an array of languages. It is used to check data types and avoid errors.

6.4.2 Member Enumeration Documentation

6.4.2.1 AvailableLanguages

```
enum game::languages::AvailableLanguages : unsigned
```

All the available languages.

This enumerator is used to get a particular language by a costant and is useful to make the code more readable.

Warning

Do *not* modify the first and last values: this enumerator is used to index an array.

Enumerator

ITALIAN	The constant for the *Italian* language.
ENGLISH	The constant for the *English* language.
NUMBER_OF_AVAILABLE_LANGUAGES	A useful constant that indicates how many languages are available.

6.4.3 Constructor & Destructor Documentation

6.4.3.1 languages()

```
game::languages::languages (
            const language(&) pLang[NUMBER_OF_AVAILABLE_LANGUAGES] )  [noexcept]
```

Languages' array's constructor.

This construct a new languages' array.

Parameters

in	*pLang*	An array of languages.

6.4.4 Member Function Documentation

6.4.4.1 operator[]()

```
game::languages::language & game::languages::operator[] (
            AvailableLanguages lang )  [noexcept]
```

Get a language.

This gets a language using its code.

Parameters

in	*lang*	The language's code. It must be one defined in the AvailableLanguages enumerator.

The documentation for this class was generated from the following files:

- game.h
- game.cpp

Chapter 7

File Documentation

7.1 game.h File Reference

The main header of the project.

```
#include <string>
#include <array>
```

Classes

- class game

  The game.
- class game::languages

  An array of languages.
- struct game::languages::language

  a language

7.1.1 Detailed Description

The main header of the project.

This header contains the declaration of all the core functions of the game.

Copyright

GNU General Public License version 3.

7.2 utilities.h File Reference

Some utilities functions.

```
#include <iostream>
#include <string>
#include <limits>
#include "utilities/string_utilities.h"
#include "utilities/cryptography.h"
```

Functions

- template<typename InputType >
  InputType get_value_in_range (const InputType &min, const InputType &max, const std::string &prompt, const std::string &error)

  Get a value in a range.
- void clear_screen ()

  Clear the screen.
- void press_any_key ()

  Press any key.

7.2.1 Detailed Description

Some utilities functions.

This file contains the declaration and definition of various miscellaneous functions that are useful in various parts of the project.

Copyright

GNU General Public License version 3.

Date

January 28, 2019

7.2.2 Function Documentation

7.2.2.1 get_value_in_range()

```
template<typename InputType >
InputType get_value_in_range (
            const InputType & min,
            const InputType & max,
            const std::string & prompt,
            const std::string & error )
```

Get a value in a range.

Given a range, this functions prompt the user for an input. In case of error (if the inserted value isn't in the range or other kind of errors that can set to true the `std::cin.fail()` flag), the user is asked to insert a new value.

Template Parameters

InputType	The type of object that must be inserted.

Parameters

in	*min*	The minimum value that can be accepted _(inclusive)_.
in	*max*	The maximum value that can be accepted _(inclusive)_.
in	*prompt*	The message that will be displayed to ask the user for an input.
in	*error*	The message that will be displayed in case of errors.

Note

The `error` message is prepended to the prompt value with the only addition of a space between the two values. So the resulting message will be like
`error + " " + prompt`

Returns

The inputed value that is in the range [`min`, `max`].

Precondition

In order to use this function on all kinds of object, there must be an ovveride of the `operator <`, `operator >` and `operator >> (std::istream&, InputType&)`.

7.2.2.2 press_any_key()

```
void press_any_key ( )
```

Press any key.

Prompts the user to press any key to continue and waits for a key press. The execution does *not* continue while this function is waiting for an input.

7.3 utilities/cryptography.h File Reference

Various cryptography functions.

```
#include <string>
```

Namespaces

- cryptography

  Various cryptography functions.

Functions

- std::string & cryptography::decrypt (std::string &str, int key=5)

  Decrypts a string.

7.3.1 Detailed Description

Various cryptography functions.

This file contains the declaration of various cryptographic functions. They are used in order to prevent the player from looking into the resources files getting an insight on the story behind the game.

7.4 utilities/string_utilities.h File Reference

Some utilities functions to be used with strings.

```
#include <list>
#include <string>
#include <iostream>
```

Classes

- class colored_string

  A colored string.

Functions

- std::string center_string (const std::string &str, unsigned width=44u)

  Center a string.
- std::list< std::string > split_string (const std::string &str, const std::string &delimiter="\n")

  Split a string based on another string.
- std::string wrap_string (const std::string &str, unsigned line_width=44u, const std::string &whitespace=" \t\r")

  Wrap a string based on a maximum length.

7.4.1 Detailed Description

Some utilities functions to be used with strings.

This file contains the declaration of various useful functions that work on `std::string`.

Copyright

GNU General Public License version 3.

Date

January 28, 2019.

7.4.2 Function Documentation

7.4.2.1 center_string()

```
std::string center_string (
            const std::string & str,
            unsigned width = 44u )
```

Center a string.

Given a string, this functions centers it with spaces.

Parameters

in	*str*	The string to be centered.
in	*width*	The total width of the final string. This is the total width on which the string has to be centered.

Returns

The centered string with trailing spaces (both at the beginning and the ending).

7.4.2.2 split_string()

```
std::list<std::string> split_string (
            const std::string & str,
            const std::string & delimiter = "\n" )
```

Split a string based on another string.

This function, given a string, splits it in a list of strings based on a `delimiter` string.

Parameters

in	*str*	The string to be splitted.
in	*delimiter*	The delimiter based on which the string has to be splitted (*optional*).

Returns

A list of string. If the `delimiter` is *not* found in the `str` string, the list will contain only a string that is equal to `str`. If the `delimiter` is found, instead, the list will contain multiple strings, to which the `delimiter` *has been removed*.

7.4.2.3 wrap_string()

```
std::string wrap_string (
            const std::string & str,
            unsigned line_width = 44u,
            const std::string & whitespace = " \t\r" )
```

Wrap a string based on a maximum length.

This function, given a string and a maximum linewidth, wraps it to get a better output format. The wrapping is based on a list of characters that will be considered whitespaces.

Parameters

in	*str*	The string to be wrapped.
in	*line_width*	The maximum line width on which the string has to be wrapped (*optional*).
in	*whitespace*	The list of character that will be considered as whitespaces (*optional*).

Returns

The wrapped string.

Index