L3 ING Project - Huffman coding

The objective of this project is to propose a code allowing to apply a Huffman coding on ASCII text files to compress their data without loss of information

Getting Started

These instructions will help you to get an executable file, as well as the method to use this executable

Prerequisites

Make

To compile the project you will need to use **make**. This command uses the file named Makefile and is used to compile the project. It is normally on all GNU Operating Systems

If it is not installed on your machine you will need to install it. This command can be found for example on Ubuntu in the build-essential package. You can install it as follows:

```
sudo apt-get install make
```

To learn how to install **make** on other operating systems, please check the correct documentation

Learn more about make

Valgrind

Valgrind is a tool used to know with details the state of the memory at the exit of the program. It is used in the makefile and by using the corresponding command (see *commands.md*) you can execute the program with valgrind

To install it you need to use this command:

```
sudo apt-get install valgrind
```

To learn how to install **valgrind** on other operating systems, please check the correct documentation

Learn more about valgrind

Compilation

Executable

To compile you will have to open a terminal at the root of the project folder and type in the terminal:

```
make
```

> *Note: You can use the -j{x} option with make and replace {x} with the number of jobs you want to run simultaneously. To summarize if your machine has 4 cores add -j4 to compile faster using the 4 cores instead of one*

Quite simple isn't it? Now the project is ready to run

Library

You can compile a library libhuffman containing functions that can be used to encrypt and decrypt a string of character or a file by using the command:

```
make lib
```

The library will be generated in the directory **lib/**. Two versions will be generated, a static library and a dynamic library.

> *Note: The static one end by ".a" and the dynamic one end with ".so"*

Use of the executable

Start the test

The project is easy to use. You can execute this command to run the default test of the executable:

```
make run
```

You can also write directly:

```
./bin/huffman_exec
```

This test encrypt and decrypt a string of characters wrote in **src/huffman_exec.c**

A section to do your personnalized tests is present in the file **src/huffman_exec.c**, you can modify this section and tests every function of the project you want to test

You can modify the tests strings stored in the variable *TESTS_V* in **src/huffman_exec.c** to make your own tests

If you want to run the program with valgrind you can execute:

```
make memory_run
# OR
valgrind ./bin/huffman_exec
```

More precise execution

Previously we only used the test of huffman, but it is impractical to compile each time

To overcome this problem, it is possible to encrypt files directly by giving the file to be encrypted as execution parameter of the executable

For this you need to type as follows:

```
./bin/huffman_exec encrypt {pathFileInput} {pathFileOut} {pathFileKey}
```

1. `{pathFileInput}` : Path of the file to encrypt
2. `{pathFileOut}` : Path of the output file *(not obligatory)*
3. `{pathFileKey}` : Path of the file used to save the key *(not obligatory)*

> *Note: Remember that you cannot change the order of the arguments. For example if you want to put {pathFileKey} you must have put {pathFileOut}*

To decrypt the order of the argument is not exactly the same:

```
./bin/huffman_exec decrypt {pathFileInput} {pathFileKey} {pathFileOut}
```

1. `{pathFileInput}` : Path of the file to encrypt
2. `{pathFileKey}` : Path of the file used to save the key *(not obligatory)*
3. `{pathFileOut}` : Path of the output file *(not obligatory)*

> *Note: it is possible not to specify files other than the one of pathFileInput, because the program will determine the filenames (pathFileOut = pathFileInput + ".hfm" and pathFileKey = pathFileInput + ".hfm.key")*

It goes without saying that you can put **valgrind** before **./bin/huffman_exec** to use it

We encourage you to always use only the **1.** parameter, and let the program calculate what's left, because it's more fast to test

Console Interface

An interface for console have been made, you can use it with:

```
make run_interface
# OR
./bin/huffman_exec interface
```

It is also possible to use valgrind to check the memory at the end of the interface execution with:

```
make memory_run_interface
# OR
valgrind ./bin/huffman_exec interface
```

Java Interface

> **Attention!** the java interface uses the linux executable generated by the makefile, it doesn't w ork on Window s or MACOSX or operating systems other than GNU

In the folder **app/** you can found a Java executable **.jar** w hich is an interface to use the executable w ithout using manually the commands show ed before. To run this executable you w ill need Java 8 or up. First you w ill need to open a terminal in the folder **app/** and execute this command:

```
app/$: java -jar HuffmanCoding.jar
```

There is a difference betw een the encrypt and the decrypt inputs

- **Encryption:** w hen you encrypt the first file is the {pathFileInput}, and the second is {pathFileOut}
- **Decryption:** w hen you decrypt the first file is the {pathFileInput}, and the second is {pathFileKey}

To sum up the argument needed for the interface correspond to the arguments **1.** and **2.** described in the previous part for each option (*Encrypt* and *Decrypt*)

Learn more

You can learn more about the commands you can use w ith **make** by looking at the **commands.md** file

Generation of the documentation

The documentation of this project can be generated using the softw are Doxygen

You can learn about how to install Doxygen here

Use Doxygen w ith the **Doxyfile** file in the **doc/** folder to generate w eb-format documentation (html). You can use Doxygen w ith the Doxyw izard, or if you can by typing in a terminal *(by being in the directory doc /)* the command:

```
doxygen Doxyfile
```

If the version of Doxyfile is too old for your version of Doxygen, you can upgrade it w ith:

```
doxygen -u Doxyfile
```

Do not hesitate to consult the documentation of Doxygen for any problem

Standards used

- C99 - The international standard used for C language
- Doxygen - The comments of our files are based on the file documentation described by the documentation generator Doxygen

Versioning

We used git for versioning. An online version is hosted on gitlab, how ever the directory is private and only visible to developers

Authors

- **Clément GUICHARD** - *Developer*
- **Thàng long CAMA** - *Developer*

License

This project is subject to French law s concerning IT projects carried out w ithin the framew ork of a University project