Universidade Vigo

Telecommunication Engineering School

Telematics Engineering Department

Computer Programming I
Academic year 2019–2020
Project specification

Contents

1 Introduction **3**

2 Project Cartas specification **4**
2.1 Project goals . . . 4
2.2 Program functionalities . . . 4
2.3 Program header . . . 5
2.4 User interface . . . 5
2.5 Return to the program's main menu . . . 6
2.6 Program files . . . 7
2.6.1 Subscribers file . . . 7
2.6.2 Messages file . . . 7
2.6.3 Operating with the files . . . 8

3 Program's modules **9**
3.1 Main module . . . 9
3.2 Operation Manager module . . . 9
3.3 The Input/Output module . . . 9
3.4 Makefile . . . 9

4 Operation Manager module **10**
4.1 "Register a new subscriber" functionality . . . 10
4.2 "Write a message" functionality . . . 11
4.3 "List the messages for a subscriber" functionality . . . 14
4.4 "Erase a message" functionality . . . 16
4.5 "Unregister a subscriber" functionality . . . 18
4.6 "Exit" functionality . . . 19

5 The Input/Output module **20**
5.1 String input from the keyboard . . . 20
5.2 Integer numbers input from keyboard . . . 20
5.3 Action confirmation from keyboard . . . 21
5.4 Subscriber display in compact form . . . 22
5.5 Message display in extended form . . . 22
5.6 Message display in short form . . . 22

6 Project Design and Documentation **23**
6.1 Project Documentation Report . . . 23

7 Evaluation of the Project **23**
7.1 Submission deadlines 23
7.2 Submission requirements 23
7.3 Evaluation procedure 24
7.3.1 Code 24

A Program's list of strings **25**

1 Introduction

The project's goal if to help the student to put into practice the subject's theoretical concepts.

The Project program must be developed following a structured, modular and good styled programming framework. In order to guarantee that framework, it is essential that the Project exhibits a suitable modular (C language functions) structure, and that the algorithms implementing those functions make a clever use of C language control structures. These algorithms must optimize the management of every function's data, by building control structures as simple as possible, with the help of the mentioned good programming style: indentation, significant variable, constant and function identifiers, comments, etc.

As every program, the Project must provide results according to the expected: those required by the problem specifications. This agreement isn't achieved the easy way, except for small and simple problems. This is why, in general, an iterative trial and debugging process is necessary until the program results reach the expected level (for big and complex problems, the costs of the trial and debugging can represent more than 60% of the total cost).

All this leads us to insist in the idea that, in order to minimize the test/debugging process, programs must be developed following the structured and modular programming principles.

Another important aspect is the Project Documentation. It must contain all the information generated while developing (flowcharts and/or pseudocodes), coding (C source code listing), and testing/debugging.

The project proposed in order to achieve the mentioned goals is a simplified version of a Message Transfer Service among Subscribers (which, from this point on, will be called Cartas).

2 Project Cartas specification

2.1 Project goals

This Project's goal is the development of an application called **Cartas** which allows to manage a simple tool for transferring text *messages* among the service *subscribers*.

Such database can be implemented, at the programmer's will, in the form of two tables or two linked lists, each of whose nodes contain the following information:

- `Subscribers` table
  - **identity**: a `32-bit` integer, which contains the identifier assigned by the system to a person currently subscribed to the service. It must be a sequentially assigned integer, starting at `1`.
  - **count**: a `32-bit` integer, which contains the number of messages currently addressed to that subscriber (initially, `0`).
  - **name**: a character string consisting of between `1` and `15` non-white-space characters, which contains the subscriber's name. For example, it could be `Juan`, but it could be as well `Juan-Perez`, or `Juan_Perez`.
- `Messages` table
  - **sender**: a `32-bit` integer, which contains the identity of the subscriber who send the message.
  - **receiver**: a `32-bit` integer, which contains the identity of the subscriber to whom the message is addressed.
  - **text**: a character string consisting of between `1` and `100` non-white-space characters, which contains the message text. For example, it could be `Hello-how-are-you?`, but it could be as well `Whats_up`, or `Howdy`.

The character strings used in the project can have any character, except white-space[1] (space, tab, newline, etc.).

2.2 Program functionalities

The program provides the user with 6 functionalities:

- Register a new subscriber
- Write a message
- List the messages for a subscriber
- Erase a message
- Unregister a subscriber
- Exit the program

[1] Accented letters, or non-English characters as `ñ` or `¿` are not accepted either, although, as later noted, the student doesn't need to take care of these.

2.3 Program header

At start up, the program must display on the screen, just once, the following header, consisting of lines with 60 characters [2]:

```
1 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
2 +                          CARTAS                          +
3 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

The displaying of this header must be carried out by using the `headline()` and `flat()` functions, included in the Input/Output module (see section 5), as they have been defined in the Laboratory **Practice #6**.

2.4 User interface

After that, the program must offer the user the following main menu with the six functionalities stated in section 2.2:

```
1 R) Register a new subscriber
2 W) Write a message
3 L) List the messages for a subscriber
4 E) Erase a message
5 U) Unregister a subscriber
6
7 X) Exit the program
8
9 Choose an option:
```

Notice that, for the sake of simplicity, the user interface is free of accented letters and other non-English characters, as ñ, or ¿.

If the user selects `'R'` or `'r'` (that is, types `'R'` or `'r'` and then, presses the `ENTER` key), a new subscriber is **R**egistered for the service (see section 4.1).

If the user selects `'W'` or `'w'`, a message is **W**ritten from a sender to a receiver (see section 4.2).

If the user selects `'L'` or `'l'`, a **L**isting of the messages addressed to a receiver is displayed in short form (see section 4.3).

If the user selects `'E'` or `'e'`, an specific message addressed to a receiver is **E**rased from the database (see section 4.4).

If the user selects `'U'` or `'u'`, a subscriber is **U**nregistered from the service, and all the messages addressed to that subscriber are erased (see section 4.5).

If the user selects `'X'` or `'x'`, the program terminates (see section 4.6).

[2] Obviously, only the characters inside the box must be printed. Both in this and in the next samples of the program output, the numbers to the left of the box are set just to identify the successive lines. In occasions, a line is longer than the width of this document, so that line is displayed cut into two pieces; the number to the left points out which is the content of each line; and the cut out piece is marked with the symbol ↪.

Program's robustness from the user's point of view

In general, a program is considered robust if it responds correctly under any circumstance. The program's menu must be robust in the presence of any character typed by the user.

If the user selects `'R'`, ... `'X'` (or `'r'`, ... `'x'`), the program must respond performing the corresponding functionality.

However, if he or she provides any other input, the program must respond displaying the warning "`You have chosen an invalid option`", and displaying the menu once again. This behaviour must reiterate until one of `'R'`, ... `'X'` (or `'r'`, ... `'x'`) is selected.

Next figure shows a program running example:

```
R) Register a new subscriber
W) Write a message
L) List the messages for a subscriber
E) Erase a message
U) Unregister a subscriber

X) Exit the program

Choose an option: 0

You have chosen an invalid option

R) Register a new subscriber
W) Write a message
L) List the messages for a subscriber
E) Erase a message
U) Unregister a subscriber

X) Exit the program

Choose an option:
```

2.5 Return to the program's main menu

After one of the functionalities **`'R'`**, **`'W'`**, **`'L'`**, **`'E'`**, or **`'U'`** in the main menu is fulfilled, the program must go back to that menu (but it must not display the header ever again).

2.6 Program files

2.6.1 Subscribers file

The text file called **abonados.txt** preserves all the relevant information about the subscribers registered to the service. The different values regarding each subscriber are written to the subscribers file in one line, separated by spaces, with the following format:

```
identity count name
```

The following table describes the items in a subscriber's line:

identity	a `32-bit` integer number
count	a `32-bit` integer number
name	a character string consisting of between `1` and `15` characters

The file's character strings, as the database strings, can contain any character except white-space and special non-English characters.

For example, the subscribers file contents, if there are 6 subscribers registered to the service, could be:

```
1 1 Juan_Perez
2 0 Manolo_Martinez
3 1 Luisa_Costa
6 2 Angel_Garcia
9 1 Maria_Pelaez
12 1 Angela_Lopez
```

2.6.2 Messages file

The text file called **mensajes.txt** preserves all the relevant information about the messages exchanged among the subscribers registered to the service. The different values regarding each message are written to the messages file in one line, separated by spaces, with the following format:

```
sender receiver text
```

The following table describes the items in a message's line:

sender	a `32-bit` integer number
receiver	a `32-bit` integer number
text	a character string consisting of between `1` and `100` characters

The file's character strings, as the database strings, can contain any character except white-space and special non-English characters.

For example, the messages file contents, if there are 6 messages, could be:

```
6 1 Hola
9 12 Como_estas
1 6 Que_tal
3 6 Puedes_venir
6 3 Ahora-no
12 9 MuyBien
```

2.6.3 Operating with the files

The program must read from both files (**abonados.txt** and **mensajes.txt**) only at start-up, and must only write to both files just before finishing its execution (see section 4.6). This means that the program **can not access the files** in order to perform any of its functionalities.

At the time of finishing, when storing the database in the files, the program must make sure that the generated files are both correct.

A file is considered correct if all its lines abide to the described format. An empty file (that is, consisting of 0 lines) is considered correct. A file with empty lines[3] is **not** considered correct.

Moreover, in order to consider a subscribers file as correct, its successive lines must contain identities in ascending order. There could be *"gaps"*, that is, maybe the numbers are non-consecutive (as in the example displayed in section 2.6.1), but the numbers have to be in ascending order.

Additionaly, in order to consider a messages file as correct, all its lines must contain both sender and receiver identities corresponding to subscribers found in the subscribers file.

At program start-up, and before displaying the header:

- If the subscribers file exists and it is not empty:
  - It will be assumed correct[4], so the program must read its contents and copy them to the subscribers table[5]. The subscribers in the file must be stored in the subscribers table in exactly the same order they are in the file.
  - And:
    - If the messages file exists and it is not empty, it will be assumed correct, so the program must read its contents and copy them to the messages table[6]. The messages in the file must be stored in the messages table in exactly the same order they are in the file.
    - If the messages files doesn't exist or it is empty, the messages table must be empty by the moment.
- If the subscribers files doesn't exist or it is empty, both tables must be empty by the moment.

[3] Both if it consists only of empty lines and if it consists of correct lines and empty lines.

[4] All the files provided to the program in the test protocols will be correct: they could contain data, or be empty, but they will be correct, so **checking its correctness is not necessary**. While the student is testing the program with his or her own files, he or she must make sure that this rule is satisfied.

[5] From this point on, we will refer to the subscribers database as subscribers **table**, although, as previously noted, it could as well be implemented as a linked list.

[6] From this point on, we will refer to the messages database as messages **table**, although, as previously noted, it could as well be implemented as a linked list.

3 Program's modules

It is compulsory to build the program around a modular structure. It must consist of, at least, the following 3 modules, in none of which the use of either **global variables** or the **`goto`** statement is allowed:

3.1 Main module

Its mandatory name is `letters.c`, and, besides all the functions that the student consider necessary to develop the module, consists of the code necessary for the application initialization, and the user interface's command interpreter.

3.2 Operation Manager module

Its mandatory name is `operation.c`, and consists of a function library and its corresponding header file (`operation.h`).

Besides all the functions that the student consider necessary to develop the module, it must implement the six functions corresponding to the user interface operations. In other words:

- `s_register()`: registers a new subscriber
- `m_write()`: writes a message
- `m_list()`: lists the messages for a subscriber
- `m_erase()`: erases a message
- `s_unregister()`: unregisters a subscriber
- `p_exit()`: exits the program.

Each of these functions declaration must reflect, by means of the different **parameters** and result, the information exchange between the user interface and the manager.

3.3 The Input/Output module

Its mandatory name is `inout.c`, and, besides all the functions that the student consider necessary to develop the module, consists of a function library (at least, `headline()`, `confirm()` and `flat()`, and the functions described in section 5) and its corresponding header file (`inout.h`).

The definition of this module, its header file, and all its functions must be as specified in the successive laboratory practices, with the prototypes finally set in **Practice #6**.

3.4 Makefile

Jointly with the mentioned modules, every work group must create a `Makefile` suitable for generating an application whose mandatory name is "**letters**".

4 Operation Manager module

4.1 "Register a new subscriber" functionality

When the user selects `'R'` or `'r'` from the menu, the program proceeds to register a new subscriber to the service, so, it must inform on the operation to be performed with the line:

```
Register
```

In order to perform this functionality, the program first requires the user to enter the subscriber's name (by means of the function `get_string()`, see section 5.1), using the following prompt string:

```
Subscriber's name
```

And stating the value `15` as the maximum accepted length.

Once the user has entered a correct name (and notice that there is **NO** error if the entered name is repeated), the program:

1. Must assign a new `identity` to the new subscriber.

   Identities must be assigned in sequence, staring at `1`. The value assigned to a new subscriber must always be one unit more than the maximum currently assigned value.

   So, if the current subscriber identities are: 1, 2, 3 and 4, the new subscriber must be assigned the value 5. But, if the current identities are: 2, 8, 16 and 23, he or she must be assigned the value 24.

2. Must register the subscriber in the subscribers table.

3. And must inform on the insertion with the line:

   ```
   Subscriber registered:
   ```

   Followed by the information about the **subscriber in compact form** (see section 5.4).

An example of this functionality's outcome is provided in the following figure:

```
Choose an option: R

Register

Subscriber's name (1-15 char):

Null length

Subscriber's name (1-15 char): Hermenegildo_Garaicoechea

Excessive length

Subscriber's name (1-15 char): Pepe-Perez

Subscriber registered:
# 45:     Pepe-Perez

R) Register a new subscriber
...
```

4.2 "Write a message" functionality

When the user selects **`'W'`** or **`'w'`** from the menu, the program proceeds to accept a mesaage to be sent from a sender to a receiver, so, it must inform on the operation to be performed with the line:

```
Write
```

In case the subscribers table is empty, that is, there aren't any subscribers yet, the program must not ask the user to enter values; instead, it must display the warning "`No subscribers yet`", and then, immediately return to the main menu.

Otherwise, the program must require the user to enter the subscriber's identity (by means of the function `get_integer()`, see section 5.2), using the following prompt string:

```
Sender's identity
```

And stating the current greatest identity in the subscribers table as the maximum accepted value.

Once the user has entered a correct identity, the program must look up the subscribers table for a susbcriber with that identity.

If there are no subscribers whose identity matches the given one, the program must display the warning "`Subscriber not found`", and then, immediately return to the main menu.

If there is a subscriber with the given identity, the program must next require the user to enter the receiver's identity (by means of the function `get_integer()`, see section 5.2), using the following prompt string:

```
Receiver's identity
```

And stating the currently greatest identity in the subscribers table as the maximum accepted value.

Once the user has entered a correct identity, the program must look up the subscribers table for a susbcriber with that identity.

If there are no subscribers whose identity matches the given one, the program must display the warning "`Subscriber not found`", and then, immediately return to the main menu.

If there is a subscriber with the given identity, the program must next require the user to enter the text of the message (by means of the function `get_string()`, see section 5.1), using the following prompt string:

```
Message text
```

And stating the value `100` as the maximum accepted length.

Finally, once the user has entered a correct text (and notice that there is **NO** error if the entered text is repeated), the programa must register the message in the messages table, must update the count of messages for that receiver in the messages table, and must inform on the insertion with the line:

```
Message registered:
```

Followed by the full information on the given **message in extended form** (section 5.5).

An example of this functionality's outcome (assuming that the current greatest identity in the subscribers table is `12`) is provided in the following figure:

```
Choose an option: w

Write

Sender's identity [1-12]: Juan

Wrong value

Sender's identity [1-12]: 0

Wrong value

Sender's identity [1-12]: 15

Wrong value

Sender's identity [1-12]: 3

Receiver's identity [1-12]: 12

Message text (1-100 char):

Null length

Message text (1-100 char):
    ↪ I-am-sending-you-a-message-that-is-way-too-long-but-the-program-will-
    not-let-me-do-it-because-only-100-chars-are-accepted

Excessive length

Message text (1-100 char): Hola_amiga_como_estas?

Message registered:
> Sender: 3
> Receiver: 12
> Text: Hola_amiga_como_estas?

R) Register a new subscriber
...
```

Another possible outcome (in which the sender is unknown) would be:

```
Choose an option: W

Write

Sender's identity [1-12]: 10

Subscriber not found

R) Register a new subscriber
...
```

Another possible outcome (in which the receiver is unknown) would be:

```
Choose an option: w

Write

Sender's identity [1-12]: 3

Receiver's identity [1-12]: 11

Subscriber not found

R) Register a new subscriber
...
```

4.3 "List the messages for a subscriber" functionality

When the user selects `'L'` or `'l'` from the menu, the program proceeds to display a short form listing of all the messages addressed to a subscriber, so, it must inform on the operation to be performed with the line:

```
List
```

In case the subscribers table is empty, that is, there aren't any subscribers yet, the program must not ask the user to enter values; instead, it must display the warning "`No subscribers yet`", and then, immediately return to the main menu.

Otherwise, the program, must require the user to enter a subscriber's name (by means of the function `get_string()`, see section 5.1), using the following prompt string:

```
Receiver's name
```

And stating the value `15` as the maximum accepted length.

Once the user has entered a correct name, the program must look up the subscribers table for a susbcriber with that name.

If there are no subscribers whose name matches the given one, the program must display the warning "`Subscriber not found`", and then, immediately return to the main menu.

Finally, if there is a subscriber with the given name[7], the program must display the line:

```
Messages for GIVEN_RECEIVER:
```

Where `GIVEN_RECEIVER` is the subscriber's name entered by the user. After that, the program must display the data in all the messages addressed to that receiver, one message per line, **in short form** (see section 5.6).

If there aren't any messages for that subscriber, there is no error. The program will just display the first line, and nothing else.

An example of this functionality's outcome is provided in the following figure:

```
Choose an option: L

List

Subscriber's name (1-15 char): Luisa_Costa_Martinez

Excessive length

Subscriber's name (1-15 char): Angela_Lopez

Messages for Angela_Lopez:
# 9: 12:Como_estas
# 3: 12:Hola_amiga_c

R) Register a new subscriber
...
```

[7] If the name is repeated in the subscribers table, that is, if there are more than one subscriber with the same name, the program must display the first one found in the table, i.e., the one with the lowest identity.

Other example of this functionality's outcome (in which there are no subscribers in the subscriber's table), is provided in the following figure:

```
Choose an option: l

List

No subscribers yet

R) Register a new subscriber
...
```

Another possible outcome (in which the given name is not in the subscribers table) would be:

```
Choose an option: L

List

Subscriber's name (1-15 char): Antonio_Alvarez

Subscriber not found

R) Register a new subscriber
...
```

And yet another possible outcome (in which the subscriber exists, but there are no messages addressed to him or her yet) would be:

```
Choose an option: l

List

Subscriber's name (1-15 char): Manolo_Martinez

Messages for Manolo_Martinez:

R) Register a new subscriber
...
```

4.4 "Erase a message" functionality

When the user selects `'E'` or `'e'` from the menu, the program proceeds to remove a message from the messages table, so, it must inform on the operation to be performed with the line:

```
Erase
```

In case the subscribers table is empty, that is, there aren't any subscribers yet, the program must not ask the user to enter values; instead, it must display the warning "`No subscribers yet`", and then, immediately return to the main menu.

Otherwise, the program must require the user to enter the receiver's identity (by means of the function `get_integer()`, see section 5.2), using the following prompt string:

```
Receiver's identity
```

And stating the current greatest identity in the subscribers table as the maximum accepted value.

Once the user has entered a correct identity, the program must look up the subscribers table for a susbcriber with that identity.

If there are no subscribers whose identity matches the given one, the program must display the warning "`Subscriber not found`", and then, immediately return to the main menu.

If there is a subscriber with the given identity, the program must next:

- if there are no messages addressed to that subscriber (the number of messages currently addressed to that receiver is `0`), the program must display the warning "`No messages found`", and then, immediately return to the main menu.

- if there is at least one message addressed to that subscriber, the program must next require the user to enter the position[8] of the message to be removed (by means of the function `get_integer()`, see section 5.2), using the following prompt string:

  ```
  Message position:
  ```

  And stating the count of messages addressed to the given receiver as the maximum accepted value.

  Once the user has entered a correct position, the program must erase the given message from the database; that is, the message must be removed (from the messages table), and the count of messages addressed to the given receiver must be decremented (in the subscribers table).

  After that, the program must inform of the removal with the line:

  ```
  Message erased
  ```

[8] Position `1` refers to the first message found in the messages table and addressed to the given receiver; position `2` referes to the second message found in that table and addressed to that receiver, ...

An example of this functionality's outcome (in which there are 5 messages addressed to the given receiver) is provided in the following figure:

```
Choose an option: E

Erase

Receiver's identity [1-12]: 25

Wrong value

Receiver's identity [1-12]: 6

Message position [1-5]: 0

Wrong value

Message position [1-5]: 3

Message erased

R) Register a new subscriber
...

...
```

Another possible outcome (in which the given receiver is not in the subscribers table) would be:

```
Choose an option: e

Erase

Receiver's identity [1-12]: 11

Subscriber not found

R) Register a new subscriber
...
```

And yet another possible outcome (in which the receiver exists, but there are no messages addressed to him or her yet) would be:

```
Choose an option: e

Erase

Receiver's identity [1-12]: 2

No messages found

R) Register a new subscriber
...
```

4.5 "Unregister a subscriber" functionality

When the user selects `'U'` or `'u'` from the menu, the program must unregister a subscriber from the service and remove all the messages addressed to him or her, so, it must inform on the operation to be performed with the line:

```
Unregister
```

In case the subscribers table is empty, that is, there aren't any subscribers yet, the program must not ask the user to enter values; instead, it must display the warning "`No subscribers yet`", and then, immediately return to the main menu.

Otherwise, the program must require the user to enter the subscriber's identity (by means of the function `get_integer()`, see section 5.2), using the following prompt string:

```
Subscriber's identity
```

And stating the current greatest identity in the subscribers table as the maximum accepted value.

Once the user has entered a correct identity, the program must look up the subscribers table for a susbcriber with that identity.

If there are no subscribers whose identity matches the given one, the program must display the warning "`Subscriber not found`", and then, immediately return to the main menu.

If there is a subscriber with the given identity, the program must next:

1. Remove all the messages (if any) addressed to that subscriber currently in the messages table.
2. Remove the given subscriber from the subscribers table.
3. And inform of the unsubscription with the line:

   ```
   Subscriber unregistered
   ```

An example of this functionality's outcome is provided in the following figure:

```
Choose an option: U

Unregister

Receiver's identity [1-12]: 11

Subscriber not found

Receiver's identity [1-12]: 6

Subscriber unregistered

R) Register a new subscriber
...
```

4.6 "Exit" functionality

When the user selects `'X'` or `'x'` from the menu, the program must inform on the operation to be performed with the line:

```
Exit
```

Next, the program must require the user to confirm the exit (using the function `confirm()`, see section 5.3) using the following prompt string:

```
Are you sure you want to exit the program? (y/n):
```

If the user gives:

- An affirmative response:
  - The program must save all the information stored in the subscribers table back to the subscribers file, overwriting any information previously written to it. The subscribers currently in the subscribers table must be written in the subscribers file in exactly the same order they are in the subscribers table.
  - The program must save all the information stored in the messages table back to the messages file, overwriting any information previously written to it. The messages currently in the messages table must be written in the messages file in exactly the same order they are in the messages table.
  - And the program must immediately finish its execution.
- A negative response, the program must immediately return to the main menu, instead of exiting.

An example of this functionality's outcome is provided in the following figure:

```
Choose an option: x

Exit

Are you sure you want to exit the program? (y/n): x

Are you sure you want to exit the program? (y/n): n

R) Register a new subscriber
W) Write a message
L) List the messages for a subscriber
E) Erase a message
U) Unregister a subscriber

X) Exit the program

Choose an option: X

Exit

Are you sure you want to exit the program? (y/n): y

user@host (path)>
```

5 The Input/Output module

Both the definition of this module, and its header file and the functions it contains must be as specified in the laboratory practices, with the prototypes finally set in **Practice #6** for the `headline()` and `flat()` functions, plus the functions described below.

5.1 String input from the keyboard

Any time the program has to accept a string from the keyboard (both in the case of a subscriber's name and in the case of an message text), it must use the function `get_string()`, which must have 3 parameters:

- the address of the memory location where the string is to be stored,
- the maximum accepted length of the string,
- the prompt string to be displayed.

The function `get_string()` must proceed as follows:

1. First, the program must display a line formed by:
   - the corresponding prompt string,
   - followed by the string:

   ```
   (1-MAX char):
   ```

   where `MAX` is the maximum accepted length of the string.
2. If the user enters an empty string (consisting of 0 characters), the program must notify that the input is invalid with the warning "`Null length`", and must display the prompt string again.
3. If the string entered by the user is too long (that is, contains more characters than maximum accepted length) the program must notify that the input is invalid with the warning "`Excessive length`", and must display the prompt string again.

   This process is repeated until the user enters a correct string[9].

5.2 Integer numbers input from keyboard

Any time the program hast to accept an integer number from the keyboard, it must use the function `get_integer()`, which must have 2 parameters:

- the maximum accepted value for the integer, and
- the prompt string to be displayed.

Furthermore, this funtion must return as a result the entered number.
The function `get_integer()` must proceed as follows:

[9] As previously mentioned, this strings must not contain any white-space, or special non-English characters, but the program **must not** check for this. In the test protocols, those characters will never be entered.

1. First, the program must display a line formed by:

   - the corresponding prompt string,
   - followed by the string:

   ```
   [1-MAX]:
   ```

   where `MAX` is the maximum accepted value for the integer.

2. If the user enters anything but an integer number from `1` to the maximum accepted value, the program must notify that the input is invalid with the warning "`Wrong value`", and must display the prompt string again.

This process is repeated until the user enters a correct value.

5.3 Action confirmation from keyboard

Any time the user hast to confirm a program's action, it must be carried out by means of the `confirm()` function, as it has been defined in the Laboratory **Practice #6**.

That is, the function `confirm()` must have 1 parameter:

- the prompt string to be displayed.

Furthermore, this funtion must return as a result an integer value (`0` or `1`).
The function `confirm()` must proceed as follows:

1. First, the program must display the corresponding prompt string.
2. If the user enters:

   - The character `'Y'` or `'y'`, it will be considered as an affirmative response, thus the `confirm()` function must return a `1`.
   - The character `'N'` or `'n'`, it will be considered as an negative response, thus the `confirm()` function must return a `0`.
   - Any other input, it will be considered as an invalid response, thus the `confirm()` function must display the prompt string again, until the user enters a valid input.

5.4 Subscriber display in compact form

All the functionalities that have to display a subscriber's information in compact form must use the function `display_subscriber()`, which must have 1 parameter:

- a structure containing the information to be displayed.

The function `display_subscriber()` must show that information in a single line with the following arrangement:

```
#IDENTITY:NAME
```

Where:

- `IDENTITY`: means the subscriber's identity in a 3-character wide field.
- `NAME`: means the subscriber's name in a 15-character wide field.

Notice the initial symbol `'#'`.

5.5 Message display in extended form

All the functionalities that have to display the contents of a message in extended form must use the function `display_extended()`, which must have 1 parameter:

- a structure containing the information to be displayed.

The function `display_extended()` must show that information in three lines with the following arrangement:

```
> Sender: the sender's identity
> Receiver: the receiver's identity
> Text: the text of the message
```

Notice the initial symbol `'>'`.

5.6 Message display in short form

All the functionalities that have to display the contents of a message in short form must use the function `display_short()`, which must have 1 parameter:

- a structure containing the information to be displayed.

The function `display_short()` must show that information in a single line with the following arrangement:

```
#SENDER:RECEIVER:TEXT
```

Where:

- `SENDER`: means the sender's identity in a 3-character wide field.
- `RECEIVER`: means the receiver's identity in a 3-character wide field.
- `TEXT`: means (at most) the first 12 characters in the text of the message.

Notice the initial symbol `'#'`.

6 Project Design and Documentation

6.1 Project Documentation Report

The students attending the Final Practice Test must hand in a Report containing the final Project Documentation, which must include, at least, a final version of the Project's division in modules, as required in section 3.

Specifically, for each module, the Report must consist of the following:

- the purpose of the module, that is, a (brief, yet clear) description of its role,
- and, for each function included in the module,
  - a (brief, yet clear) description of the purpose of the function,
  - its prototype,
  - the involved information process, that is, the program data the function needs to know in its input, and which data modifies at its output. In other words, the function's parameters and/or result,
  - and its pseudocode or its flowchart.

Those pseudocodes or flowcharts must contain a high-level description of the activities of the given function.

High-level means it is not necessary to descend to the detail of "a value 0 is assigned to variable k", but, for example:

- In the main() function, one can write lines as: "displays the header", "displays the menu", ... or "if the user selects `'R'`, calls the s_register() function".
- And, in the s_register() function, one can write lines as "requires the user to enter a string with the subscriber's name", ...

7 Evaluation of the Project

7.1 Submission deadlines

- The code must be uploaded to Faitic not later than Wednesday, January 8, 2020.
- A printed version of the Project Documentation Report must be handed in at the end of the Final Practice Test, on Monday, January 13, 2020.

7.2 Submission requirements

All the work groups must upload to Faitic:

- All the C code ("*.c") and header ("*.h") files, comprising their project.
- The corresponding Makefile suitable for generating the application's executable.

Each person in the work group must upload:

- individually, using his or her own Faitic account,
- the required files,
- as a single compressed[10] file,
- whose name must coincide with his or her *ID Card number*[11]. Thus, if using `zip` compression, and if the ID card number is `12345678A`, the uploaded file must be "**`12345678A.zip`**".

Each student must take responsibility for making sure that the uploaded code:

1. compiles without any errors, and
2. exhibits a behaviour which conforms to this Document's specifications.

With the goal of facilitating the second, the instructors will publish a test protocol in Faitic (consisting of a test battery, and the results every test must produce), so that every work group can autonomously [12] assess the proper operation of their program.

7.3 Evaluation procedure

7.3.1 Code

As stated in the subject's Educational Guide, the Project will be evaluated by means of the Final Practice Test (PPF).

At the PPF, the student will be required to perform several simple modifications in his or her code. What this means is that several small changes to the Project Specification will be defined, and the student will be required to adapt his or her code to the new Specification.

The (in this way modified) code must be capable to satisfactorily pass a new test protocol which verifies its conformance to the new specification.

[10] `zip`, `rar`, `tgz`, ...

[11] For Spanish people, the *DNI*.

[12] Obviously, counting on the necessary advice by the subject's instructors, both at the laboratory (in the practice sessions), and at tutorials.

A Program's list of strings

- Application's name in the header: "CARTAS"
- Main menu's strings:
  - "R) Register a new subscriber"
  - "W) Write a message"
  - "L) List the messages for a subscriber"
  - "E) Erase a message"
  - "U) Unregister a subscriber"
  - "X) Exit the program"
  - Request to enter an option: "Choose an option: "
  - Selection error: "You have chosen an invalid option"

- "Register a new subscriber" function strings:
  - Operation label: "Register"
  - Request to enter the name: "Subscriber's name (1-15 char): "
  - Empty string error: "Null length"
  - Length error: "Excessive length"
  - Success report:
    - Line #1: "Subscriber registered:"
    - Line #2: "`#IDENTITY:NAME`"

- "Write a message" function strings:
  - Operation label: "Write"
  - Empty subscribers table error: "No subscribers yet"
  - Request to enter the sender: "Sender's identity [1-MAX]: "
  - Request to enter the receiver: "Receiver's identity [1-MAX]: "
  - Value error: "Wrong value"
  - Unknown subscriber error: "Subscriber not found"
  - Request to enter the text: "Message text (1-100 char): "
  - Empty string error: "Null length"
  - Length error: "Excessive length"
  - Success report:
    - Line #1: "Message registered:"
    - Line #2: "> Sender: `the sender's identity`"
    - Line #3: "> Receiver: `the receiver's identity`"
    - Line #4: "> Text: `the text of the message`"

- "List the messages for a subscriber" function strings:
  - Operation label: "List"
  - Empty subscribers table error: "No subscribers yet"
  - Request to enter the receiver: "Receiver's name (1-15 char): "
  - Empty string error: "Null length"
  - Length error: "Excessive length"
  - Unknown subscriber error: "Subscriber not found"
  - Success report:
    * Line #1: "Messages for `receiver's name`:"
    * Line #2, if at least one message: "> `#SENDER:RECEIVER:TEXT`"
    * Line #3, if at least two messages: "> `#SENDER:RECEIVER:TEXT`"
    * ...

- "Erase a message" function strings:
  - Operation label: "Erase"
  - Empty subscribers table error: "No subscribers yet"
  - Request to enter the receiver: "Receiver's identity [1-MAX]: "
  - Request to enter the position: "Message position [1-MAX]: "
  - Value error: "Wrong value"
  - Unknown subscriber error: "Subscriber not found"
  - Success report:
    * Line #1: "Message erased"

- "Unregister a subscriber" function strings:
  - Operation label: "Unregister"
  - Empty subscribers table error: "No subscribers yet"
  - Request to enter the receiver: "Receiver's identity [1-MAX]: "
  - Value error: "Wrong value"
  - Unknown subscriber error: "Subscriber not found"
  - Success report:
    * Line #1: "Subscriber unregistered"

- "Exit the program" function strings:
  - Operation label: "Exit"
  - Request to decide: "Are you sure you want to exit the program? (y/n): "