Practice 2: Question and answer manager

Programming 2

Academic Year 2023-2024

This practice of the course consists of developing a question and answer management system about your favorite subject. The concepts necessary to develop this practice are covered in *Unit 1*, *2*, and *3* of theory.

Delivery Conditions

- The deadline for this assignment is **Friday, April 19th**, by **23:59**
- You must submit two files: `preguntas.txt` with a set of example questions and `prac2.cc` with the code for all the functions

Code of Honor

If any form of copying (total or partial) is detected in your assignment, you will receive a **0** grade, and the Polytechnic School will be notified to take disciplinary action

It is acceptable to discuss possible solutions to assignments with your classmates
It is acceptable to enrol in an academy if it helps you study and complete the assignments

It is unacceptable to copy code from other classmates or ask ChatGPT to do the assignment for you
It is unacceptable to enrol in an academy to have the assignments completed for you

Seek help from your teacher if needed
Do not copy

General Rules

- You must submit the assignment exclusively through the practice server of the Department of Software and Computing Systems (DLSI). You can access it in two ways:
  - Main page of DLSI (`https://www.dlsi.ua.es`), option "HOMEWORK UPLOAD"
  - Directly at the address `https://pracdlsi.dlsi.ua.es`
- Things to consider when submitting:
  - The username and password for submitting assignments are the same as those used in UACloud
  - You can submit the assignment multiple times, but only the last submission will be evaluated
  - Submissions via other means, such as email or UACloud, will not be accepted
  - Late submissions will not be accepted

- Your assignment must compile without errors using the C++ compiler available in the Linux distribution of the practice labs
- If your assignment cannot be compiled, your grade will be 0
- The 70% of the grade for the assignment will depend on the automatic correction, so it is essential that you strictly adhere to the texts and output formats indicated in this statement. The other 30% will depend on the manual code review carried out by your practice professor, so you must also adhere to the style guide of the subject. The content of the requested question file will also be taken into account for the grade. As in previous assignments, it will also be evaluated if you have updated your code on *GitHub* every week
- At the beginning of all submitted source files, you must include a comment with your identification number (NIF or equivalent) and your name. For example:

prac2.cc

```
// ID 12345678X GARCIA GARCIA, JUAN MANUEL
...
```

- The calculation of the assignment grade and its relevance to the final grade of the course are detailed in the presentation slides of the course (*Unit 0*).

1 Practice Description

The goal of this practice is to develop a question and answer resolution system about the topics of the subject *Programming 2*. The goal of the system is for students to be able to make inquiries about a specific topic and for teachers registered in the program to be able to answer them. The tool will allow managing both questions and answers as well as teacher profiles.

2 Implementation Details

Several files necessary for the correct completion of the assignment will be published on the course's Moodle:

- `prac2.cc`. This file contains a program skeleton on which to base your assignment. Download it and add your code to it. This file contains the following information:
  - The necessary structures (`struct`) for the assignment
  - An enumerated type `Error` that contains all possible error classes that may occur in this assignment (e.g., `ERR_OPTION`)
  - A function `error` that displays the corresponding error message on the screen based on the parameter passed to it. For example, when the function receives the parameter `ERR_OPTION`, it will display the message `ERROR: wrong menu option`
  - A function `showMenu` that displays the program menu on the screen
  - A `main` function that implements the management of the main menu and calls the corresponding functions depending on the option chosen by the user
- `autocorrector-prac2.tgz`. It contains the files of the autocorrector to test the assignment with some input tests. The automatic correction of the assignment will be performed with a similar program, using these tests and others defined by the teachers
- `prac2`. Executable file of the assignment (compiled for 64-bit Linux machines) developed by the teachers, so you can test it with the inputs you want and see the expected correct output

- In the assignment correction, only data of the correct type will be provided, although the values may be incorrect. For example, if the rating of a teacher is requested, it will always be tested with an integer value, which could be `-1237` or `0`, but never with a value of another type (character, string, floating-point number, etc.)
- To avoid problems with character encoding, accents, diaeresis, or the letter ñ will not be used in any of the texts entered to evaluate this assignment. Only characters from the English alphabet will be used
- The rest of the decisions in the implementation of the assignment are up to you, but keep in mind that the source code will be reviewed by your teacher following the style guide published on the course's Moodle. Part of the assignment grade depends on this review

3 Program Workflow

The program you are going to develop consists of a tutoring system that will allow you to manage questions, answers, and teachers, storing them for later use using files. The users of the application you are going to develop will be students, who will register the questions, and teachers, who will answer those questions.

4 Components

The program to be developed must be able to manage three fundamental elements: the questions asked by students, the teachers, who are each of the teachers responsible for answering the questions, and the database, which is where all the information of the questions and the teachers is stored. The following sections describe the structure of each of these components in more detail.

4.1 Question

Structures of type `Question` store information related to the questions asked by students. Each question will contain a numerical identifier (`id`), the unit number it corresponds to (`unit`), the text of the question (`question`), and the text of the answer (`answer`). This information will be stored in a record with the following format:

```
struct Question{
    unsigned int id;
    unsigned int unit;
    string question;
    string answer;
};
```

4.2 Teacher

Structures of type `Teacher` store information related to the teachers. Each teacher will have a name (`name`), their encrypted password (`password`), and the number of questions they have answered in the program (`answered`). This information will be stored in a record with the following format:

```
struct Teacher{
    char name[KMAXNAME];
    char password[KMAXPASSWORD];
    unsigned int answered;
};
```

`KMAXNAME` is an integer constant with the value `50` and `KMAXPASSWORD` is also an integer constant with the value `5`.

4.3 Database

The structure `Database` stores the entire collection of questions and teachers. It also contains a `nextId` field that will store the identifier that should be assigned to the `id` field of the next question (`Question`) to be created. This field will have the value `1` at the beginning of the program and will be incremented by one for each new question. That is, the first question created will have the identifier `1`, the second will have `2`, and so on.

```
struct Database{
    unsigned int nextId;
    vector<Question> questions;
    vector<Teacher> teachers;
};
```

5 Program start

The `main` of your practice will contain a variable of type `Database` that will store in memory all the information of the questions and teachers during the program's execution. When the program starts, the `nextId` field of this variable will be initialised to `1`.

The first thing you have to do is load the data of the teachers stored in the binary file `teachers.bin` into memory. This file contains structures of type `Teacher`, one after another, so that each one stores the information of a teacher. You should create a function `loadTeachers` that will be responsible for reading all the information of the teachers and loading it into the `teachers` field of the variable of type `Database` mentioned above.

- Your program will only manage a single variable of type `Database` that will contain all the data of questions and teachers
- If the file `teachers.bin` does not exist when the program starts, you should not show any type of error. The program will start normally, leaving the `teachers` vector of `Database` empty
- We will assume that the content of the file `teachers.bin` is correct. No verification of the content needs to be done during loading

6 Menu

Upon running the practice, and after loading the information of the teachers (if any), the main menu of the program will be displayed on the screen, waiting for the user to choose an option:

```
Terminal
1- Add question
2- Batch add questions
3- Delete question
4- Add teacher
5- Add answers
6- View answers
7- View statistics
8- Export questions
q- Quit
Option:
```

The valid options that the user can enter are the numbers from `1` to `8` and the letter `q`. If the chosen option is not any of these (for example, `9`, `x`, or `;`), the error `ERR_OPTION` will be emitted by calling the `error` function with that parameter. When the user chooses a correct option, the code associated with that option must be executed. Upon completion, it will return to display the main menu and prompt for another option, until the user decides to exit the program using the `q` option.

7 Options

The following sections describe the functionality that each of the options shown in the main menu of the program should have.

7.1 Add question

This option allows adding a new question to the program. It is activated when the user chooses option `1` from the main menu. A function `addQuestion` should be created to manage it. In this function, the user will be prompted to enter the unit number with the following message:

Terminal
```
Enter unit:
```

If the empty string is entered, the error `ERR_EMPTY` will be shown, and it will return to the main menu without saving any question information. Otherwise, it should be checked that the entered value is between 1 and 5. If not, the error message `ERR_UNIT` will be displayed, and the user will be asked again to enter the number showing the same message.

If the number is correct, the text of the question will be requested with the following message:

Terminal
```
Enter question:
```

It should be validated that the entered text does not contain any vertical bar characters (`|`) to avoid problems when reading and writing the text file, as explained in the next section. If a vertical bar is found, the error `ERR_CHAR` will be shown, and the question will be requested again displaying the same previous message. If the entered string is empty, the error `ERR_EMPTY` will be shown, and it will return to the main menu without saving anything.

Once the unit number and the question text are read correctly, a new question (`Question`) will be created with an empty `answer` field, as it will not yet have an associated answer.

Each new question will be automatically assigned a unique identifier, which will be stored in the `id` field of `Question`. This identifier will be given by the value stored in the `nextId` field of the `Database` structure that contains all the program's information at that time. As indicated in Section 4.3, this unique identifier will start with the value `1` and will be incremented by one for each new question. Therefore, each time a question is added, the `nextId` field should be incremented so that the next question is assigned a correct new identifier.

After assigning the identifier, the new question will be added to the `questions` vector of the `Database` record of the program, and it will return to the main menu.

- Duplicate questions will be allowed, so it is not necessary to check if a question already exists when entered

7.2 Batch add questions

This option allows loading questions from a text file. It is activated when the user chooses option `2` from the main menu. Its code should be included in a function called `batchAddQuestions`. Each line of the text file will have the following format:

```
Unit number|Question|Answer
```

First, the user will be prompted to enter the filename with the following message:

```
Terminal
Enter filename:
```

If the empty string is entered, the error `ERR_EMPTY` will be shown, and it will return to the main menu. If the file cannot be opened with the entered name, the error `ERR_FILE` will be shown, and the user will be asked again to enter the filename showing the same previous message.

If the file is successfully opened, the process of inserting the questions into the system will begin. It should be verified that the questions are correct by making the following validations:

- The unit number must be a value between 1 and 5. It cannot be empty
- The question text cannot be empty
- The answer text can be empty. In that case, the correct format of the line would be `Unit number|Question`. That is, it should be checked that there is no vertical bar after the question
- There cannot be more than two vertical bars as separators because it would imply that there are more fields than expected, and the format would be incorrect
- There must be at least one vertical bar for the format to be correct

If any of these validations fail, the message `Error line X` will be shown, where `X` is the line number of the file, and the content of the read line will be ignored, automatically moving on to read the next line without further checks. That is, only one error will be shown per question, even if there is more than one in the same line (for example, the unit may be incorrect and the question empty).

If the read line of the file is empty, it will be ignored without showing any error. For each correct line, a new question will be created in the database, incrementing the `nextId` identifier as explained in the previous section.

At the end of the process, a message will be displayed indicating how many questions the file had and how many have been successfully added to the system. For example, given the following file:

```
questions.txt
1|What is a matrix?|A bidimensional array
1|I don't understand what a boolean is|

2||The question is empty
1|What is the use of break?
5|How is inheritance implemented|Not studied in this subject
7|||
```

The following output would be obtained:

```
Terminal
Error line 2
Error line 4
Error line 7
Summary: 3/6 questions added
```

The error in line 2 occurs because there is an unnecessary vertical bar at the end. Line 3 is simply ignored because it is empty. Line 4 gives an error because the question text is empty. Line 7 gives an error because unit 7 does not exist. In addition, the question is empty, and there are more vertical lines than allowed, but only one error is shown. Therefore, in this example, 3 out of the 6 questions in the file would be added to the database.

If the file were empty, the following output would be obtained, without showing any errors:

```
Terminal
SUMMARY: 0/0 questions added
```

7.3 Delete question

This option allows deleting a question given its identifier. It is activated when the user chooses option 3 from the main menu. A function deleteQuestion should be created to handle this functionality. First, the identifier of the question will be requested with the message:

Terminal
```
Enter question id:
```

If there is no question with that identifier, the error ERR_ID will be shown, and the identifier will be requested again showing the same message. If the empty string is entered, the error ERR_EMPTY will be shown, and it will return to the main menu. If the identifier is correct, the question will be deleted from the questions vector of the Database record, and it will return to the main menu.

7.4 Add teacher

This option allows registering the teachers who will answer the students' questions. It is activated when the user chooses option 4 from the main menu. Its code should be included in a function called addTeacher. First, the name of the teacher will be requested with the following message:

Terminal
```
Enter teacher name:
```

The teacher's name must be validated following these rules:

- It can only contain uppercase letters (A-Z), lowercase letters (a-z), and whitespace characters. Again, only characters from the English alphabet will be used. One way to perform this check is using the isalpha function from the cctype library
- If it contains whitespace characters at the beginning or end of the name, they should be removed
- The minimum length of the name, after removing leading and trailing whitespace, will be 3 characters, and the maximum will be KMAXNAME-1 (remember to leave space for the '\0' character)

If any of these rules are not met, the error ERR_NAME will be shown, and the name will be requested again showing the same previous message. If the string is empty, the error ERR_EMPTY will be shown, and it will return to the main menu. If the name is valid, it will be checked that there are no other teachers with the same name. Otherwise, the error ERR_DUPLICATED will be shown, and the user will be asked again to enter the name showing the same previous message.

Next, the password that the teacher will use to identify themselves will be requested with the following message:

Terminal
```
Enter password:
```

Only passwords consisting exclusively of 4 digits (0-9) will be considered valid. In any other case, the error ERR_PASSWORD will be shown, and the password will be requested again showing the same message. You can use the isdigit function from the cctype library to perform this check. If the string is empty, the error ERR_EMPTY will be shown, and it will return to the main menu.

If the password is valid, before saving it in the teacher's information, it will be encrypted by transforming the digits into characters using the following equivalence table:

For example, the password 0375 will be encrypted as TAFM. Once the data is validated and the password is encrypted, the counter of questions answered by the teacher will be initialised to 0, and it will be saved at the end of the teachers vector of the Database record.

Digit	0	1	2	3	4	5	6	7	8	9
Letter	T	R	W	A	G	M	Y	F	P	D

7.5 Add answers

This option allows teachers to add answers to questions that have not yet been answered. It is activated when the user chooses option `5` from the main menu. Its code should be included in a function called `addAnswers` which you have to implement.

To add answers, teachers must log in using their name and password. First, the teacher's name will be requested with the following message:

Terminal
```
Enter teacher name:
```

It should be checked if the teacher's name exists in the database (it is not necessary to validate it before). If it is not found, the error `ERR_NAME` will be shown, and the name will be requested again showing the same message. If the empty string is entered, the error `ERR_EMPTY` will be shown, and it will return to the main menu. If the name is correct, the password will be requested with the following message:

Terminal
```
Enter password:
```

Remember that the password was saved encrypted! If the password does not match the one stored, the error `ERR_PASSWORD` will be shown, and it will be requested again showing the same message. If the entered string is empty, the error `ERR_EMPTY` will be shown, and it will return to the main menu.

If the teacher's credentials are correct, a repetitive process of answering questions will start. To do this, the questions that do not yet have an answer will be displayed on the screen with the format:

```
ID. (Unit) Question
```

and then the following message will be displayed requesting the identifier of the question to answer:

Terminal
```
Enter question id:
```

In the following example, you can see a possible output with two pending questions to answer, with identifiers 2 and 5, belonging to units 1 and 3, respectively:

Terminal
```
2. (1) What is the use of break?
5. (3) How to open a text file?
Enter question id:
```

If there are no unanswered questions, the error `ERR_NO_QUESTIONS` will be shown, and it will return to the main menu.

If the entered identifier does not correspond to one of the listed questions, the error `ERR_ID` will be shown, and the identifier will be requested again showing the message `Enter question id:`, but without showing the list of questions again. If the empty string is entered, the error `ERR_EMPTY` will be shown, and it will return to the main menu. If the identifier is correct, the answer will be requested with the following message:

```
Terminal
Enter answer:
```

The answer cannot contain the vertical bar character ('|'), otherwise the error `ERR_CHAR` will be shown, and the previous message will be shown again to request the answer again. If the empty string is entered, the error `ERR_EMPTY` will be shown, and it will return to the main menu.

If the answer is valid, it will be saved in the corresponding question, and the list of pending questions to answer will be shown again, and an identifier will be requested. This process will end when the character `'b'` is entered instead of a valid identifier or when there are no more questions to answer. In both cases, it will return to the main menu.

7.6 View answers

This option allows you to view all the questions that have already been resolved with their corresponding answers. It is activated when the user chooses option `6` from the main menu. You should create a function `viewAnswers` with the code for this option.

When this option is initiated, the questions will be displayed with the following format:

```
Id. (Unit) Question: Answer
```

For example:

```
Terminal
2. (1) What is the use of break?: For example, to exit a loop
5. (3) How to open a text file?: Look at slide 8 of Unit 3
6. (3) How to close a text file?: Slide 11 of Unit 3
7. (1) What does int mean?: Integer type, dummy
12. (2) What is a string?: A class to handle strings easily
```

After this, it will return to the main menu.

7.7 View statistics

This option allows you to view statistics about the number of questions and how many each teacher has answered. It is activated when the user chooses option `7` from the main menu. You should create a function `viewStatistics` to handle this option. Below is an example of the output:

```
Terminal
Total number of questions: 10
Number of questions answered: 7
Grace Hopper: 2
Ada Lovelace: 4
Kim Dotcom: 0
Tim Berners Lee: 1
```

The teachers will be displayed in the order they were introduced into the system. If there are no questions, or there are questions but they are not answered, the corresponding lines will display the value `0`. In this example, there are no questions, answers, or teachers:

```
Terminal
Total number of questions: 0
Number of questions answered: 0
```

7.8 Export questions

This option allows you to export all the questions from the database to a text file. It is activated when the user chooses option 8 from the main menu. You have to include its code in a function called `exportQuestions`, which you will need to implement. First, it will prompt for the filename with the following message:

Terminal
```
Enter filename:
```

If an empty filename is entered, it will display the error `ERR_EMPTY` and return to the main menu. If the entered filename cannot be opened, it will display the error `ERR_FILE` and ask for the filename again, showing the same message.

Once the file is opened, you should save all the questions from the database, both answered and unanswered, each on a separate line following the correct format described in Section 7.2. They will be saved in the same order as they were introduced.

7.9 End of the program

When selecting the option `q`, the program will terminate, saving the updated list of teachers in the binary file. For this, you should overwrite the content of the file `teachers.bin` with the content of the vector `teachers`, in the same order they are in the vector. If the file cannot be opened, the error `ERR_FILE` should be displayed, and the program should exit anyway.

This action should be performed every time the program ends, even if no new teachers have been added during the execution.

8 Program arguments

In this practice, the program must allow the user to indicate some actions to perform by passing command-line arguments. You should use the parameters `int argc` and `char *argv[]` of the `main` function to manage them. The program will accept the following command-line arguments:

- `-f <text file>`: Allows importing questions from a text file, whose name should be indicated after the `-f` argument, starting the program afterwards. It will behave the same as if the user selected the `Batch add questions` option from the menu, also when handling errors if any of the read data is incorrect. For example, if a question with empty text is read from the file, the `ERR_EMPTY` error will be emitted, the data of that question will be ignored, and the rest of the file will be processed. If the file cannot be opened, the `ERR_FILE` error will be emitted, and the program will start normally without loading any data. If the file data is loaded correctly, a message will be displayed showing how many questions have been imported, just like in Section 7.2. The following example will import the data stored in the text file `questions.txt`:

  Terminal
  ```
  $ prac2 -f questions.txt
  ```

- `-s`: Shows statistics, just like if the user selected the `View statistics` option from the menu, and then exits the program without showing the main menu

Both arguments can appear at the same time in a program call and also in any order. Regardless of the order in which they appear on the command line, the order in which the arguments will be processed is as follows:

1. First, the `-f` option will be processed to load data from the file
2. Second, the `-s` option will be executed to show statistics

In the following example, first the data stored in the file `questions.txt` will be loaded, and then the statistics will be shown, after which the program will exit.

```
Terminal
$ prac2 -s -f questions.txt
```

Another example, with valid parameters, would be the following:

```
Terminal
$ prac2 -f -f -s
```

Although at first glance the parameters may seem incorrect, in reality they are not. The program receives a first parameter `-f` followed by a file named `-f`. Next, it receives the parameter `-s` to show the statistics. This sequence of parameters is therefore correct.

If an error occurs in the input arguments (for example, if an option that does not exist is entered or a duplicate one is entered), the `ERR_ARGS` error should be emitted, and the program will terminate without showing the main menu. If all arguments are correct, the user-entered options will be processed, and then the main program menu will be displayed as usual (if the `-s` option has not been passed).

- Regardless of whether program arguments are passed or not, the `teachers.bin` file must always be loaded at the beginning of the program, before loading questions and before showing statistics, if applicable
- Both arguments are optional and may not appear in the program call
- A specific argument can only appear once in the call; otherwise, the arguments will be considered incorrect
- If an argument other than those mentioned (e.g., `-n`) appears, the arguments will be considered incorrect
- If the filename is not provided after `-f`, the arguments will also be considered incorrect
- Files do not necessarily have to have the extension `.txt`. We use these extensions in the examples to make it clearer that we are working with text files

9 The `preguntas.txt` file

As part of this assignment, you must submit a `preguntas.txt` file that should include 10 questions created by you about the contents of the `Programming 2` subject. **Put effort into it, as these questions will be reviewed by the teaching staff and will be taken into account for manual correction scoring**. In other words, do not include trivial questions or example questions taken from this document.

Specifically, you should create two questions for each of the five units covered in the subject. The format of the file will be as follows:

```
Unit|Question
```

The questions will be sorted by unit, meaning that the first two questions will be from Unit 1, followed by the two from Unit 2, and so on. An example of the first five lines of the file could be:

```
1|¿Para qué sirve un break?
1|¿En qué orden se ejecutan los parámetros de un bucle for?
2|¿Cómo se copia el contenido de un string en un array de caracteres?
2|¿Para qué sirve srtcat?
3|¿Qué pasa si abro un fichero de escritura con el flag ios::in?
```

- The questions must be written in Spanish
- In this file, the question can contain any characters you want (ñ, accents, opening question mark...) except the vertical bar