# Practice 2: The BMI Laboratory

## Programming 2

Academic Year 2024-2025

This assignment involves developing a management system for lab test reports (body mass index reports, BMI). **To complete this assignment, you may use the C++ concepts covered in the lecture slides of *Unit 1*, *Unit 2*, and *Unit 3*.**

## Submission Guidelines

- The deadline for submission of this assignment is **Friday, March 28**, by **23:59**.
- You must submit a single file named `prac2.cc` containing the code for all functions.

## Honor Code

If plagiarism (total or partial) is detected in your Assignment, you will get a **0** for the submission, and the Polytechnic School (EPS) administration will be notified for disciplinary action.

Discussing possible solutions with your classmates is acceptable.
Enrolling in an academy to assist you study and complete the assignments is acceptable.

Copying code from other students or asking ChatGPT to complete the assignment for you is not acceptable.
Enrolling in an academy so they will implement your assignments is unacceptable.

If you need help, contact your professor.
Do not copy.

## General Rules

- You must submit your code exclusively through the assignment submission server of the Department of Languages and Computing Systems (DLSI). It can be accessed in two ways:
  - DLSI Main Page (`https://www.dlsi.ua.es`), option “ENTREGA DE PRÁCTICAS”.
  - Directly at `https://pracdlsi.dlsi.ua.es`.
- Important submission details:
  - The username and password for submission are the same as in UACloud.
  - You can submit the assignment multiple times, but only the last submission will be evaluated.
  - Submissions via email or UACloud will not be accepted.
  - Late submissions will not be accepted.

- Your assignment must compile without errors using the C++ compiler in the Linux distribution of the labs.
- If your assignment does not compile, the grade will be 0.
- At the beginning of all submitted source files, you must include a comment with your NIF (or equivalent) and name. For example:

prac1.cc

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

- Passing **all** the autograder tests qualifies you for the assignments exam. If any test fails, you will not be able to take the exam, and your grade for this assignment will be 0.
- The calculation of this assignment grade and its relevance in the final course grade are detailed in the course introduction slides (*Unit 0*).

# 1 Assignment Description

The goal is to develop a management system for lab test reports (in this case, body mass index reports) for a laboratory. The system will handle patient registrations and the results of each test performed. It will allow tracking of the entered data and will detect risk situations.

# 2 Implementation Details

Several files necessary for completing the assignment are posted on the course Moodle:

- `prac2.cc`. This file contains a skeleton program on which to base your assignment. Download it and add your code. This file contains the following:
  - The structs required to complete the assignment.
  - An enumerated type `Error` that contains all possible error classes that can occur in this assignment (for example, `ERR_OPTION`).
  - A function `error` that displays the corresponding error message on the screen based on the parameter passed. For example, when the function receives the parameter `ERR_OPTION`, it will show the message `ERROR: wrong option`.
  - A function `showMenu` that displays the program menu.
  - A `main` function that manages the main menu and calls the corresponding functions depending on the option chosen by the user.
- `autocorrector-prac2.tgz`. It contains the autograder files for evaluating the assignment with various input tests. The automatic grading of the assignment, after submission, will be performed using these same tests, and you must pass all of them to be eligible for the practical exam.
- `prac2`. Executable file for the assignment (compiled for 64-bit Linux machines) developed by the course instructors, so that you can test it with any inputs and see the expected correct output.

# 3 Program Operation

The program to be developed is a system for storing and managing the lab test reports done in a laboratory.

# 4 Components

The program must manage three fundamental elements: patients, the lab tests done for each patient, and the database where all patient and test information is stored. The following sections describe the structure of each of these components in more detail.

## 4.1 Patient

The structures of type `Patient` store information about a laboratory patient. Each patient is identified by their ID (NIF) (`nif`), their `name`, and their `telephone` number:

```
struct Patient{
    string nif;
    string name;
    string telephone;
};
```

We will also use a specific patient structure for storing patients in the binary file (`PatientBin`):

```
struct PatientBin{
    char nif[KMAXNIF];
    char name[KMAXNAME];
    char telephone[KMAXTELEPHONE];
};
```

The constants `KMAXNIF`, `KMAXNAME`, and `KMAXTELEPHONE` are defined as follows:

```
const int KMAXNIF=10;
const int KMAXNAME=50;
const int KMAXTELEPHONE=14;
```

## 4.2 Analysis

The structures of type `Analysis` store information related to a test performed on a patient. Each test is identified by a number and contains a unique identifier (`id`), the patient's id (`nif`), the test date (`dateAnalysis`), `weight`, and `height`:

```
struct Analysis{
    unsigned int id;
    char nif[KMAXNIF];
    Date dateAnalysis;
    float weight;
    float height;
};
```

The `Date` structure is defined as follows:

```
struct Date{
    unsigned int day;
    unsigned int month;
    unsigned int year;
};
```

## 4.3 Database

The `Database` structure stores the entire collection of patients (`patients`) and tests performed (`analysis`). It also contains a field `nextId` that will store the identifier to be assigned to the `id` field of the next test created. This field will initially have the value 1 and will be incremented for each new test. That is, the first test created will have the identifier 1, the second 2, and so on.

```
struct Database{
    unsigned int nextId;
    vector <Patient> patients;
    vector <Analysis> analysis;
};
```

# 5 Program Initialization

The `main` function of your assignment should contain a variable of type `Database` that stores all patient and test information in memory during the program execution. When the program starts, the `nextId` field of this variable is initialized to 1.

The first thing to do is load into memory the patient data stored in the binary file `patients.bin`. This file contains structures of type `PatientBin`, one after the other, each storing the information of a patient. You must create a function `loadPatients` that reads all the patient information and loads it into the `patients` vector of the aforementioned `Database` variable.

- Your program will manage only one variable of type `Database` that will contain all patient and test data.
- If the file `patients.bin` does not exist when the program starts, no error should be shown. The program will start normally with the `patients` vector in `Database` empty.
- We will assume that the contents of the file `patients.bin` are correct. No verification of the content is required during loading.

# 6 Menu

When the assignment is executed, and after loading the patient information (if any), the program's main menu will be shown, waiting for the user to choose an option:

Terminal

```
1- Add patient
2- View patient
3- Delete patient
4- Save patients
5- Add analysis
6- Export analysis
7- Import analysis
8- Statistics
q- Quit
Option:
```

The valid options that the user may enter are the numbers `1` through `8` and the letter `q`. If an invalid option is entered (for example, `9`, `x`, or `;`), the error `ERR_OPTION` will be issued by calling the function `error` with that parameter. When the user selects a valid option, the corresponding code is executed. After finishing, the main menu is shown again and the program waits for another option, until the user decides to exit using the option `q`.

# 7 Options

The following sections describe the behavior that each option in the main menu must have.

## 7.1 Add patient

This option allows you to register patients for whom you can later enter tests in the application. Its code should be included in a function called `addPatient`. The first piece of information requested is the patient's NIF with the prompt:

```
Terminal
  Enter NIF:
```

If a blank NIF is entered, the program should return to the main menu. It must be ensured that exactly nine characters are entered for the NIF, consisting of eight digits and one letter. Otherwise, the error `ERR_WRONG_NIF` will be shown and the NIF will be requested again.

The function `searchPatient` will be used to check if the patient exists. This function must return -1 if the patient does not exist; otherwise, it should return the position of the patient within the `patients` vector. If a patient with that NIF already exists, the error `ERR_PATIENT_EXISTS` will be raised and the NIF will be requested again. If not, the program will ask for the rest of the information and add a new patient record. First, it will request the name:

```
Terminal
  Enter name:
```

It must be verified that the name field contains at least three characters. Otherwise, it will show the error `ERR_WRONG_NAME` and the name will be requested again. Next, the telephone number will be requested:

```
Terminal
  Enter telephone:
```

It must be verified that the phone number starts with the symbol + (for the country code) and is followed by between 10 and 12 digits. If not, the error `ERR_WRONG_TELEPHONE` must be displayed and the telephone number should be requested again.

- To check if a character is a digit, you can use the function `isdigit()` from the `cctype` library.
- To check if a character is a letter, you can use the function `isalpha()` from the `cctype` library.

## 7.2 View patient

This option allows you to view a patient's information. A function called `viewPatient` must be implemented. It will first request the NIF with the prompt:

```
Terminal
  Enter NIF:
```

If there is no patient with that NIF (using the function `searchPatient` to locate it), then the error `ERR_PATIENT_NOT_EXISTS` will be raised and the program must ask again for the NIF. If an empty string is entered, the program will return to the main menu without doing anything.

If the patient exists in the database, their data and the results of all tests performed will be displayed, one per line, following this format:

Terminal

```
NIF: 11111111D
Name: Pedro Pi
Telephone: +34612345689
Id  Date        Height  Weight
23  11/11/2027  188     95
45  17/02/2028  188     93
```

The columns `Id`, `Date`, `Height`, and `Weight` are separated by tab characters (the `'\t'` character). If there are no analyses for that patient, only their data will be displayed, without showing `Id`, `Date`, etc.

## 7.3 Delete patient

This option allows you to delete a patient's information. For this, implement a function called `deletePatient` that will first request the NIF with the following prompt:

Terminal

```
Enter NIF:
```

If a blank NIF is entered, the program should return to the main menu. If no patient with that NIF is found, the error `ERR_PATIENT_NOT_EXISTS` will be raised and the NIF will be requested again.

If the patient exists, they will be removed from the `patients` vector in the database, and all tests for that patient stored in the `analysis` vector will be deleted.

## 7.4 Save patients

This option saves all patients from the database to the file `patients.bin`. If a file with that name already exists, it will be overwritten with the current database information. A function `savePatients` must be implemented for this purpose.

To save a patient's information (of type `Patient`) in the binary file, you must first convert it into a structure of type `PatientBin`. This may require truncating the patient's name so that it does not exceed the maximum size `KMAXNAME`.

## 7.5 Add analysis

This option allows you to add the results of a test for a patient. It must be implemented in a function called `addAnalysis`. First, it will request the patient's NIF:

Terminal

```
Enter NIF:
```

If a blank NIF is entered, the program should return to the main menu. If no patient exists with that NIF, the error `ERR_PATIENT_NOT_EXISTS` will be raised and the NIF will be requested again. If the patient exists, the test date will be requested with the following prompt:

Terminal

```
Enter date (day/month/year):
```

The user must enter the date in the specified format. For example: `12/5/2027`. We assume that the user always enters the correct format. There is no need to verify the format.

However, you must verify that the date is valid. For this, the day must be between 1 and 31, the month between 1 and 12, and the year between 2025 and 2050, included. It is not necessary to check that the

number of days is correct for a specific month (i.e., assume all months can have 31 days—even February 30 would be considered correct). In case of an error, the message `ERR_WRONG_DATE` will be issued and the date will be requested again. Next, the weight (in kilograms) will be requested with the prompt:

```
Terminal
Enter weight:
```

It must be verified that the weight is positive. Otherwise, the error `ERR_WRONG_NUMBER` will be issued and the value will be requested again. Finally, the height (in centimeters) will be requested:

```
Terminal
Enter height:
```

Again, it must be verified that the height is positive. Otherwise, the error `ERR_WRONG_NUMBER` will be issued and the value will be asked again.

Finally, the field `id` will be assigned the value of `nextId` from the database, and this value will be incremented for the next test.

## 7.6 Export analysis

This option stores all test records from the database in a file called `analysis.bin`. A function `exportAnalysis` must be implemented for this purpose. If the file already exists, it should be overwritten.

## 7.7 Import analysis

This option reads the information from the file `analysis.bin`. It must be implemented in a function called `importAnalysis`. If the file does not exist, the error `ERR_FILE_NOT_EXISTS` should be raised and the program will return to the main function.

For each test record stored in the file, you must first check that a patient with the given NIF exists in the `patients` vector of the database. If not, write the NIF of that patient to a text file called `wrong_patients.txt` (one per line) and do not load that test into the system. This file should not be overwritten if it already exists; rather, new information should be appended. If the file `wrong_patients.txt` cannot be opened, the error `ERR_FILE_NOT_EXISTS` will also be displayed, and the program will return to the main menu without doing anything else.

For each test loaded into the system, assign the appropriate `id` as if it were being inserted from the `Add analysis` option, based on the value stored in the `Database`. That is, ignore the `id` that the test had when stored in the binary file. Tests that may already exist in the system will not be deleted.

## 7.8 Statistics

This option will iterate through the test records and show the degree of obesity for each test of the patients. A patient may have multiple tests, so all of them will be displayed. It will be implemented in a function called `statistics`.

A measure of obesity is determined by the body mass index (BMI), which is calculated using the following formula:

$$BMI = \frac{\text{weight (kg)}}{[\text{height (m)}]^2}$$

For example, a person who is 1.85 meters tall and weighs 94 kilograms would have a BMI of $94/1.85^2 = 27.5$. According to the U.S. National Heart, Lung, and Blood Institute (NHLBI), overweight is defined as a BMI of 25 or more, and a person is considered obese if their BMI is 30 or higher. The following table shows the different body compositions and their corresponding BMI:

For each analysis, display the NIF of the patient, test date (in `dd/mm/yyyy` format), weight, height, and body composition (`Underweight`, `Healthy`, `Overweight`, or `Obesity`). All these fields will be separated by a `';'`:

| Body Composition | Body Mass Index (BMI) |
|---|---|
| Underweight | Less than 18.5 |
| Healthy | 18.5 – 24.9 |
| Overweight | 25.0 – 29.9 |
| Obesity | 30.0 or greater |

Terminal

```
11111111D;02/05/2025;110;180;Obesity
11111111D;04/07/2025;115;180;Obesity
22222222G;13/11/2025;75;186;Healthy
```

Only information for patients who have at least one test will be displayed. The same information shown on the screen (with each field separated by `';'`) will be saved in a text file called `statistics.txt`. If the file already exists, it will be overwritten.

- When showing the information on the screen and saving it to a file, if the day or month number is less than 10, a 0 must be added in front so that the format has two digits for the day and two for the month. For example: `05/07/2026`.

# 8 Program Arguments

In this assignment, the program must allow the user to specify certain actions via command line arguments. You should use the parameters `int argc` and `char *argv[]` in the `main` function to manage them. The program will accept the following command line arguments:

- `-f <text file>`. Allows importing tests from a text file, whose name must be provided after the `-f` argument, before the program starts. The format of the text file must be the same as that generated by the `Statistics` option (no need to verify it). The BMI value read from the file should be ignored, and all the remaining information should be stored in records of type `Analysis` in the `analysis` vector of the database. For each test loaded into the system, assign the appropriate `id` as if it were being inserted from the `Add analysis` option. As in the `Import analysis` menu option, if a NIF does not exist in the database, the NIF of that patient will be written to a text file called `wrong_patients.txt`.

  If the file passed as an argument does not exist or the file `wrong_patients.txt` cannot be opened, the error `ERR_FILE_NOT_EXISTS` will be issued and the program will start normally without loading any tests. Note that before processing the test file, the program will load into memory the information from `patients.bin` so that there are patients to associate the tests with.

  The following example will import the data stored in the text file `blood_tests.txt`:

  Terminal

  ```
  $ prac2 -f blood_tests.txt
  ```

- `-s`. Displays the statistics, just as if the user had selected the `Statistics` option from the menu, and then exits the program without showing the main menu. This option only makes sense if the user has also provided the `-f` argument, since otherwise there would be no information to show. For this reason, the following call would be incorrect:

  Terminal

  ```
  $ prac2 -s
  ```

Both arguments can appear simultaneously in a call to the program and in any order. Regardless of the order in which they appear on the command line, the arguments will be processed in the following order:

1. First, process the `-f` option to load data from the file.
2. Second, execute the `-s` option to display the statistics.

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

Terminal

```
$ prac2 -s -f blood_tests.txt
```

Another valid example is:

Terminal

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

Although at first glance the parameters might seem incorrect, they are actually valid. The program receives a first parameter `-f` that is followed by a file named `-f`. Then it receives the `-s` parameter to display 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 provided or a duplicate is present), the error `ERR_ARGS` should be raised and the program will terminate without showing the main menu. If all the arguments are correct, the options provided by the user will be processed and then the main menu will be shown as usual (unless the `-s` option is provided, which displays the information and terminates the program).



- Remember, regardless of whether command line arguments are provided or not, the file `patients.bin` must always be loaded at the start of the program.
- Both arguments are optional and may be omitted in the program call.
- Remember, if the `-s` parameter is present, the `-f` parameter must also be present. If it appears alone, the error `ERR_ARGS` should be raised.
- A given argument can only appear once in the call; otherwise, the arguments will be considered incorrect.
- If any argument other than those mentioned (for example, `-n`) is provided, the arguments will be considered incorrect.
- If no file name appears after `-f`, the arguments will also be considered incorrect.
- Text files do not necessarily have to have the `.txt` extension, nor do binary files necessarily have to have the `.bin` extension. We use these extensions in the examples to clarify that we are working with text and binary files.