Programming Practices
CAA1 - 20241

Delivery deadline: **13 / 10 / 2024**

Format and delivery deadline

The CAA must be delivered before **October 13, 2024** at 11:59 p.m.

It is necessary to deliver a file in ZIP format, which contains:

- A document in **PDF** format with the answers to exercises 1 and 3. The document must indicate on its first page the **name** and **surname** of the student submitting it.
- The files **taxation.h** and **taxation.c** requested in **exercise 2**.

It is necessary to deliver in the "**CAA1 Delivery**" activity in the theory classroom before the delivery date.

Goals

- Review basic programming concepts.
- Know how to interpret and follow the code of third parties.
- Know how to make small programs to solve problems based on a general description of the problem.
- Know how to specify an algorithm using pre and post conditions.
- Know how to include controls in programs to guarantee that preconditions are met.

Correction criteria

Each exercise has its associated score based on the total activity. It will be valued both that the answers are correct and that they are complete.

- Failure to follow the **delivery format**, both in terms of the **type** and **name of the files** and the requested content, will result in a **significant penalty** or a grade of **D for the activity**.
- In exercises in which algorithmic language is requested, the **format** must be respected.
- In the case of exercises in C language, these must **compile to be evaluated**. If they compile, it will be valued:
    - That they **work** as described in the statement.
    - That the **style criteria** are respected and that the code is **properly commented**.
    - That the **structures** used are correct.

Advice

We take this opportunity to remember that it is **totally prohibited to copy in the CAAs** of the subject. It is understood that there may be work or communication between students during the completion of the activity, but its delivery must be individual and differentiated from the rest. Submissions will be analyzed with **plagiarism detection tools**.

Thus, deliveries that contain any identical part with respect to other students' deliveries will be considered copies and all those involved (without the existing link between them being relevant) will suspend the delivered activity.

Citation guide: https://biblioteca.uoc.edu/es/contenidos/Como-citar/index.html
Monograph on plagiarism:
http://biblioteca.uoc.edu/es/biblioguias/biblioguia/Plagio-academico/

Observations

The following symbols are used in this document to refer to design and programming blocks:

 Indicates that the code shown is in algorithmic **language**.

 Indicates that the code shown is in C **language**.

 Shows the execution of a program in C **language**.

Statement

The Spanish Tax Agency is concerned because it suspects that many owners are not paying taxes according to the new Urban Leases Law. To try to improve the system they have asked us to make a small program that they will use to detect these errors (intentional or not). This program is to calculate the income for the year 2023.

For simplicity, for this exercise we will not apply the Urban Leases Law but rather an adaptation with these rules:

1) The system receives a flow of data from properties that are not used by their owners as a current apartment. For example, if an owner has three properties, one of which she lives in, data will only be received from the other two. This is so, because the property where you are living is exempt from paying taxes.

2) If an owner has an unrented apartment, he or she must pay €150 per month for it. When an apartment has been rented for any number of days in a month, for simplicity, we will consider that the tenant has paid for the entire month and that the owner will have to pay the taxes for the entire month. Thus, the owner will not have to pay the €150 if the apartment has been rented for at least one day that month.

3) A landlord will have to pay 20% of the rent he obtains from the rented property if the tenant is over 35 years old. If the tenant is up to and including 35 years old, the owner will only have to pay 10% of the rent.

Our application will connect with the Tax Agency system, and therefore the input and output data must follow the agreed format. To facilitate this connection, it has been agreed that all applications will receive a file in CSV (Comma Separated Values) format for the input data and will generate the output in the same format. In this type of files, each line corresponds to a record, and the different fields of each record are separated with a known separator character. Specifically, in our case we will use the semicolon (;) as the separation character, and the formats will be:

Input data

We have three input data streams into the system:

1) **Owner data declaring their income**: Here we do not receive all the data on a person's income but only a view of what they have declared/paid to the treasury for their properties that are not their habitual residence. This stream has three fields separated by (;) :
    a. **name**: Name (limited to 15 characters).
    b. **id**: National identity card of the owner (9 characters).
    c. **tax**: Value paid to the treasury for the properties in your name.

2) **Property data**: list of properties for which taxes must be paid (those that are habitual residences are excluded). It has these fields:
    a. **cadastral_ref**: Cadastral reference (7 characters).
    b. **street**: Street/square name (25 characters).
    c. **number**: Street/square number.
    d. **landlord_id**: National identity card of the owner (9 characters).

3) **Tenant data**: In our application we do not worry about whether the tenants' declaration is correct, we simply take the data from their declaration to check if the owners have declared the properties correctly. It has these fields:
    a. **start_date**: Initial date of the rental contract. If the contract began before 2023, the initial date will be 01/01/2023.
    b. **end_date**: End date of the rental contract. If the contract ends after 2023, the final date will be 12/31/2023.
    c. **tenant_id**: National identity document of the tenant (9 characters).
    d. **name**: Tenant name (limited to 15 characters).
    e. **rent**: Monthly rental income.
    f. **age**: Renters age.
    g. **cadastral_ref**: Cadastral reference (7 characters).

An example of an owner would be:

“Eric;42345671A;200.0”

An example of a property would be:

“ABC1234;Balmes;25;42345671A”

An example of a tenant would be:

“01/01/2023;31/12/2023;12345678A;Lucas;600.0;25;ABC1234”

Output data

The output of our program will be a list of the owners who have paid a lower amount on their tax return, with the amount they should have paid.

“Eric;42345671A;3240.0”

Exercise 1: Defining data types [25%]

From the description of the input data of the statement, define in **algorithmic language**:

- A **tTenant** type that can save a tenant's information (contract start date, contract end date, ID, etc.).
- A **tProperty** type that can save all the data of a property (cadastral reference, address, owner's ID).
- A **tLandlord** type that can save all the data of an owner (name, ID, amount paid in tax) along with all their taxable properties.

For this you have the types **tDate** and **tProperties** already defined:

```
const
      MAX_PROPERTIES: integer = 80;
end const

type
      tDate = record
            day : integer;
            month : integer;
            year : integer;
      end record

      tProperties = record
            elems : vector [MAX_PROPERTIES] of tProperty;
            count : integer;
      end record

end type
```

Note: You can define additional types that you consider appropriate.

Exercise 2: Table manipulation [50%]

It is recommended to read and pay attention to the code provided in **main.c** before implementing these methods, as it will give you extra information to understand them better.

To perform this exercise, you have the types defined to store the owners (**tLandlords**) and properties (**tProperties**), and also some methods that work with them (**properties_init**, **properties_len**, **landlords_init** and **landlords_len**) in the file **taxation.c**.

Based on the data structures defined in exercise 1 and the structures and methods detailed in this exercise, define the following methods (actions or functions) in C language (use the files **taxation.h** and **taxation.c** to declare and implement the methods):

a) **landlord_parse**: given an entry in the CSV file (**tCSVEntry**) with the data of an owner (without property data), it initializes a structure of type **tLandlord** with that data.
b) **landlords_add**: given a structure of type **tLandlord**, adds it to the owners table of type **tLandlords**. If the owner is already at the table, it does nothing.
c) **landlords_cpy**: copies a structure of type **tLandlords** into another structure of the same type, except the amount to pay field which is initialized to zero in all owners.
d) **property_parse**: given an entry in the CSV file (**tCSVEntry**) with the data of a property, it assigns them to a structure of type **tProperty**.
e) **landlord_add_property**: adds to a structure of type **tLandlords**, the information of a new property (**tProperty**). If the property belongs to an owner who does not exist in the structure, nothing is done. If the owner exists and already has the property stored, nothing is done either. If the owner does not have the property in storage, the amount to be paid in taxes by the owner is added and updated considering that the new property is not rented.
f) **tenant_parse**: given an entry in the CSV file (**tCSVEntry**) with the data of a tenant, it assigns them to a structure of type **tTenant.**
g) **landlords_process_tenant**: given the property rented to a tenant (**tTendant**), it updates the amount to be paid by its owner (stored in the structure of type **tLandlords**). If the property does not belong to any owner, nothing is done.
h) **landlord_get**: returns a character string with the owner's data stored in the **index** position of the **tLandlords** structure. The result is only used to display it on standard output.

i) **property_get**: given an owner (**tLandlord**), returns a character string with the property data stored in the **index** position of the **tProperties** structure. The result is only used to display it on standard output.
j) **mismatch_tax_declaration**: returns true if the declared amount is less than what the owner would have to pay in taxes stored in the **index** position of the two **tLandlords** structures.

You must use the main routine that is provided in the **main.c** file without any modification. This routine executes the following tasks (and displays the data after each step):

1. Add 4 owners.
2. Add 6 properties.
3. Add 6 tenants.
4. Processes the data and notifies if there are owners who have paid a lower amount than they should.

To consider that the program works *correctly*, the final result on the screen must be the following:

```
Declarant landlords:
0;Eric;42345671A,200.0
1;Sandra;52345671B,970.0
2;Karol;43345671C,4200.0
3;Lucas;33345671D,1800.0

Expected landlords:
0;Eric;42345671A,3240.0
   0;ABC1234;Balmes,25;42345671A
   1;AAC2256;Villarroel,13;42345671A
1;Sandra;52345671B,970.0
   0;ZBC1234;Bonanova,51;52345671B
2;Karol;43345671C,4980.0
   0;QBC1234;Aribau,34;43345671C
   1;TBC1234;Aribau,44;43345671C
3;Lucas;33345671D,1800.0
   0;YBC1234;Casanova,66;33345671D

Tax declaration mismatch: Eric;42345671A,3240.0
Tax declaration mismatch: Karol;43345671C,4980.0
```

Note: To carry out this exercise, in the files **csv.h**, **csv.c**, **taxation.h** and **taxation.c** you will find the declaration and implementation of the following methods:

tCSVEntry	Type of data that saves a line of a CSV file.
csv_numFields	Given a tCSVEntry structure, returns the number of fields it contains.
csv_getAsInteger	Given a tCSVEntry structure and the position of a field (0 indicates the first), returns its value as an integer.
csv_getAsString	Given a tCSVEntry structure and the position of a field (0 indicates the first), returns its value as a string.
csv_getAsReal	Given a tCSVEntry structure and the position of a field (0 indicates the first), returns its value as real.
tDate	Structure that allows saving date information.
date_parse	Given a tDate structure and a string with a date (in the format of the input data), it initializes it with date.

Exercise 3: Formal specification [25%]

Defines the declaration (**not the implementation**) of the **property_parse** and **landlord_add_property** methods from the previous exercise in algorithmic language.

a) Add pre and post conditions in algorithmic language.
b) Add to the C language code from the previous exercise the necessary "asserts" to ensure that the pre-conditions specified in these methods are met.