Trabalho prático 1 – implementation of a struct vector library

1. General information

The first practical work consists of the implementation of changes and additional functions to include in a function library for the manipulation of struct vectors.

This work is expected to be done autonomously by each group until the established date. It is acceptable to consult diverse sources. However, the submitted code must be solely authored by the group members and any detected plagiarism will be duly penalized. The inability to explain the submitted code by any group member will also result in a penalization.

The submission should be sent via Moodle and the deadline is **March, the 28th, by 9pm**.

2. Concept

The USA have decided to create a seed vault, similar to that of the 2006's Svalbard initiative (https://en.wikipedia.org/wiki/Svalbard_Global_Seed_Vault). For that they require a database to catalogue and manage the stored species.

3. Work Implementation

The compressed archive PROG2_2021_T1.zip contains the necessary files for developing this work, namely:

- `plantas.h`: declarations of the library functions pertaining to colecao vector and planta elements
- `plantas.c`: requested functions implementation file, related with `plantas.h` library
- `colecao-teste.c`: includes the main program and calls upon and performs basic tests to the implemented functions
- db_small.txt: text file with informations about the existing collection

Important Notes:

1. **Only the plantas.c file should be altered and it will be the only file considered out of any work submitted.**
2. **Additional details about the functions (to develop) may be found nearing the respective function prototype in plantas.h**

The data structures `planta` e `colecao` are the library's framework and have the following declarations:

```
typedef struct
{
  /** Catalogue's unique identification **/
  char ID[10];

  /** Scientific name **/
  char  nome_cientifico[MAX_NAME];

  /** List of local names of the species, nicknames **/
  char **alcunhas;

  /** Total number of plant nicknames stored in alcunhas **/
  int n_alcunhas;

  /** Number of seeds stored in the vault for this plant **/
  int n_sementes;

} planta;
```

In this structure *planta* are stored: 1) the plant's identification (`ID`); 2) its scientific name (`nome_cientifico`); 3) a possible *array* of additional names for which the plant is known (`alcunhas`); 4) an integer with the total number of alternative names (`n_alcunhas`); 5) an integer with the number of stored seeds in the vault (`n_sementes`).

The data structure `colecao` points to a plant *vector*. Additionally, it includes the number of valid positions of the vector (`tamanho`) and the sorting criterion the collection currently follows (`tipo_ordem`).

```
typedef struct
{
  /** pointer to array of stored plants**/
  planta **plantas;


  /** plant vector size **/
  long tamanho;

  /** criterion that currently sorts vector in ascending order:
   'nome' - for nome_cientifico; 'id' - for the ID field **/
  char tipo_ordem[5];

} colecao;
```

The functions to implement in this work are available in the `plantas.c` file and are:

1. **planta *planta_nova** (const char *ID, const char *nome_cientifico, char **alcunhas, int n_alcunhas, int n_sementes);
   Create an instance of the struct `planta`*, copying each argument to the respective field. Every field must present valid values, with the exception of* `alcunhas`*, which may be* `NULL` *if for this plant no local names are available. The functions should return the pointer to the created* `planta` *instance or* `NULL` *if an error occurred.*
2. **colecao *colecao_nova**(const char *tipo_ordem);
   Create a new empty instance of the struct colecao*. You should return the pointer for this structure. Return NULL if an error occurred.*
3. **int planta_insere**(colecao *c, planta *p);
   Insert the `planta` *pointed by* `p` *in the* `colecao` *pointed by* `c`*, on the correct position, according to the* `tipo_ordem` *sorting criterion. If the plant already exists (has the same ID) it should add to that seed's counter and append the new local names, without duplication. Otherwise, a new instance should be added to the collection. Return zero if the plant is new and is successfully inserted, 1 if the plant already existed and the values were just updated or -1 if an error occurs.*
4. **int colecao_tamanho**(colecao *c);
   Return the number of different stored species in the provided `colecao` *structure. You should return an integer with the total or -1 in case of error.*
5. **colecao* colecao_importa**(const char *nome_ficheiro, const char *tipo_ordem);
   Imports a collection present in the file pointed by `nome_ficheiro`, returning the collection's pointer via output argument*. The returned collection should be sorted according to the criterion* `tipo_ordem` *passed on as input. If the function is unsuccessful it should return* `NULL`*. Each line of the file is a plant instance, and has the following format: ID, scientific name, number of seeds, nickname0, nickname1, …, nicknameN*
6. **planta* planta_remove**(colecao *c, const char *nomep);
   Remove a `planta` *with the scientific name* `nomep` *out of the* `colecao` *pointed by* `c` *and correct the collection on the position removed. If it's unsuccessful the function should return* `NULL`*.*
7. **int planta_apaga**(planta *p);
   Delete the data structure `planta` *pointed by* `p` *and free all of its previously allocated memory. If it's successful the function should return the value zero or -1 otherwise.*
8. **int colecao_apaga** (colecao *c);
   Delete the collection `c`*, and all its stored plants, releasing all its allocated memory. If it succeeds, the function should return zero, or -1 otherwise.*
9. **int *colecao_pesquisa_nome**(colecao *c, const char *nomep, int *tam);
   Return an indices array referring to the position of all plants that present the full scientific name or any of the local names equal to `nomep`*. Returns by reference the array's size (*`tam`*). If it's unsuccessful, the function should return* `NULL`*.*
10. **int colecao_reordena**(colecao *c, const char *tipo_ordem);
    Re-sort the collection `c` *according to the* `tipo_ordem` *criterion and update that field's value in the collection* `c`*. If it's unsuccessful the function should return -1, if it's unnecessary to re-sort return the value zero and, if it is necessary, return 1 after the re-sorting is performed.*

Note: The provided data and test files consist of an example set. The files which will be used to evaluate your work may be others and contain more systematic testing, while including extreme cases not currently present: unforeseen argument values. As such, it is your responsibility to guarantee that the arguments are duly tested in order to only accept values within the expected range.

4. Function library test

The library may be tested by running the program within `colecao-teste.c`. There is a basic test for each function you must implement which determines if the function presents the expected behaviour. Please note that these tests are not exhaustive. As such these tests should be viewed only as an indicator of the general correctness of the desired functionalities.

When the functions are correctly implemented, the program `colecao-teste` should present the following results when run:

```
INICIO DOS TESTES

...verifica_planta_nova: planta nova nao e' NULL (ok)
...verifica_planta_nova: ID coincide com o esperado (= SALAM2) (ok)
...verifica_planta_nova:  Nome cientifico coincide com o esperado (= Sagittaria
lancifolia L. ssp. media (Micheli) Bogin) (ok)
...verifica_planta_nova: Numero de sementes coincide com o esperado (= 60) (ok)
...verifica_planta_nova: Numero de alcunhas coincide com o esperado (= 2) (ok)
...verifica_planta_nova: O conteudo do vetor alcunhas coincide com o esperado (ok)
OK: verifica_planta_nova passou

(...)

...verifica_colecao_pesquisa_nome (teste de um nome que nao existe): retornou NULL
(ok)
...verifica_colecao_pesquisa_nome (teste de um nome que existe): numero de indices
encontrados coincide com o esperado (= 2) (ok)
OK: verifica_colecao_pesquisa_nome passou

...verifica_colecao_reordena: 1ª e 'ultima plantas coincidem com o esperado (= ACSP5
e YUSM) (ok)
OK: verifica_colecao_reordena passou

FIM DOS TESTES: Todos os testes passaram
```

5. Development tools

The use of an IDE or Visual Studio Code is advised when developing the work, as it enables debugging in an efficient way. You may find additional information on how to use Visual Studio Code in a short tutorial on the Moodle platform.

6. Evaluation

The work's grade is calculated over the implementation submitted by the students. The final grade for the first work (T1) is given by:

T1 = 0.8 Implementation + 0.2 Memory

The Implementation grade is determined by additional automatic tests (namely, using more extensive test files). If the submitted work does not compile, this implementation component will be graded as zero.

Memory management will also be evaluated at a minor degree, and 3 grading steps will be considered: 100% no memory leaks, 50% a few, poorly significant memory leaks, 0% large quantity or significant memory leaks.

7. Server test

Soon a server will become available for you to test your code during development. The code sent to the server **WILL NOT BE CONSIDERED FOR YOUR GRADE**. Only the version submitted to Moodle is eligible for grading purposes.

8. Submitting your work

The submission is only possible via Moodle and before the previously stated date on the top of this document. A *zip* file should be submitted with the following content:

- the edited implementation file plantas.c
- a text file named autores.txt stating the name and number of each of the group's elements

Important Note: only the submission with the following naming scheme will be accepted: `T1_G<group_number>.zip`. For instance: "`T1_G999.zip`"