Advanced algorithm project
Evaluated work

To read before starting work

- This work is to achieve in pairs (02 students per group)
- The deadline is due to May 6th, 2019 at 10 pm
- A .zip file should be submitted on Moodle. It must contain:
  - **Source code** (.c & .h files)
  - A **manuscript** of 4 pages at most.
- 00/20 will be attributed for each work submitted without a report.
- A code with Warnings leads mallus
- Code that does not compile or run will be noted 00/20.
- Any cheat leads in a score of 00/20 for the whole class.

Preamble:

To manage a (very) large number of people - for example, a directory - it may seem sensible to the first idea is to store the information associated with those people in the same set sorted alphabetically on the names. Indeed, access will be faster (eg with a dichotomy search algorithm).

However, since the alphabet contains only 26 letters, there is necessarily a lot of people associated to the same first letters. Finally, the alphabetic order does not satisfy a good organization allowing a quick access to all items.

It is therefore proposed in this project to organize the items in a set, called “Table", where the position of an element is provided by the function index.

This function is defined as follows:
Let’s Consider N the size of the "table" and B an integer strictly positive called "base", s a string so that $s="c_0c_1c_2\dots c_K"$ $K\geq 0$ (where c_i is a character) and ***ascii*** is a function returning the ascii code of one character:

$$index(s, B, N) = \left(\sum_{i=0}^{k} ascii(c_i).B^i \right) \text{ modulo } N$$

- The modulo ensures that the position provided by the function index is an integer in the interval [0, N[;
- Function index ensures a good distribution of the elements when the base B is a prime number.

For this project, manipulated data will be those corresponding to the employees of Chicago city in the USA (cf. Chicago.txt file).

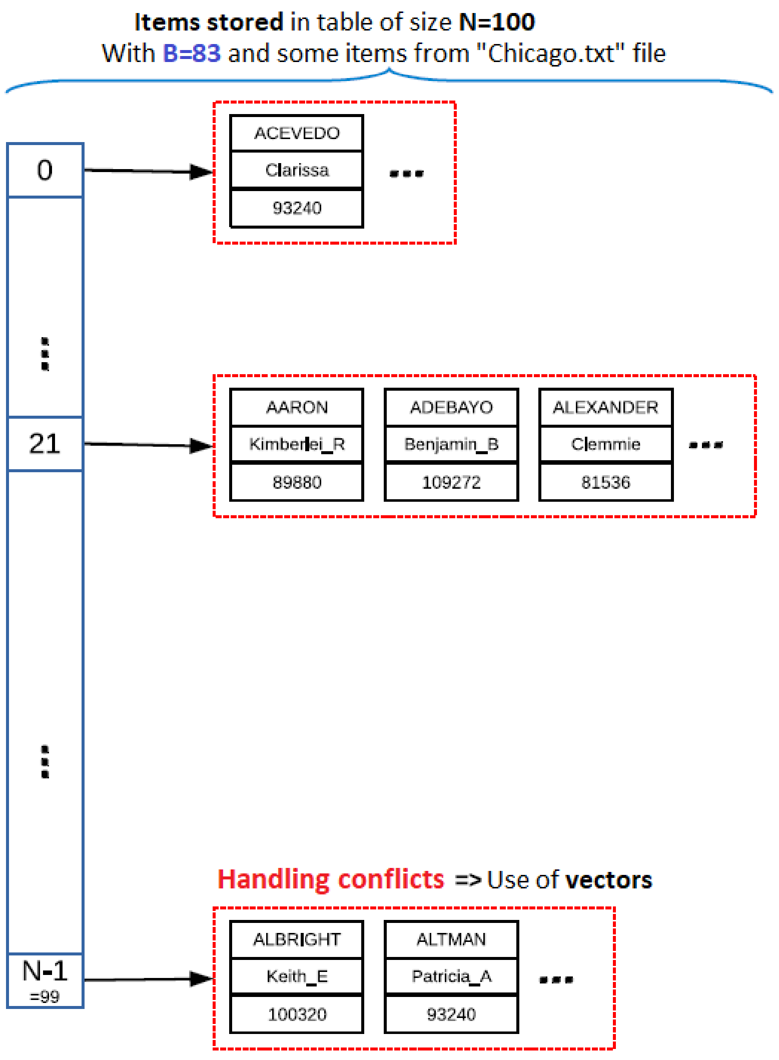
For the purposes of the problem, the string to be considered for the function index will be the concatenation of the first four letters of the last name and the first 2 letters of the first name of an employee. ⇒ therefore, the index function will have 4 parameters instead of 3.

Example:

`index("AARON","Karina",83,100) = 77`

In case of conflict(s), i.e. when two or more items meet in the same position, these items are sorted in alphabetic order according to the last name in the vector of the same index.

- To facilitate the code, even if it is only one item at a given index, it will still be stored in the vector associated to that index.



Problem statement:

Create a program:

- That starts by asking the user size and the base wished for the "table" and initializes this "table";

  - It is necessary to define a structured type suitable for such a "table."

- Which displays a menu offering the following functionalities:
  1. **Index**:
     - The user must provide a last name and a first name
     - The program returns the index associated to the first and last names (regardless of the items present in the “table”)

  2. **Add**
     - The user must provide the last name, the first name, and the salary of the employee to add.
     - The program adds the employee to the table, except the case where the employee is already stored. In any case, a message should be displayed to indicate whether the addition to the table is done or not.

  3. **Load**
     - The user must provide the number of employees to load from the file “Chicago.txt”
     - The program will add - without duplication - the selected employees in the "table".

  4. **Display salary**
     - The user must provide the last name and the first name of a given employee.
     - The program displays the salary if the employee is present in the table, -1 otherwise.

  5. **Display between**
     - The user must provide the desired start and end indices.
     - The program displays the employees lists of each index, sorted in alphabetic order according to the last name.

  6. **Conflicts number**
     - The user has nothing to provide.
     - The program returns the number of conflicts present in the “table”.

  7. **Conflict average size**
     - The user has nothing to provide
     - The program returns the average number of items having the same index value.
-

  8. **Delete**
     - The user must provide the last name and the first name of the employee to delete.

- The program deletes the employee from the table, except the case where the employee is not present. In any case, a message should be displayed to indicate whether the deletion is done or not

9. **Delete between**
   - The user must provide the desired start and end indices.
   - The program will remove the employees present in the "table" between the two indices.
10. **Exit**
    - The user has nothing to provide.
    - The program will stop.

> Except for "Exit", the menu must be re proposed to the user after each functionality choice, in order to select another one.

Chicago.txt file:

The file Chicago.txt is formatted as follows:

- The first line contains a single integer indicating the number of lines in the fi le.
- Next, each line corresponds to an employee information, it is composed of:
  - the last name of the employee, a space, its first name, a space, and the annual salary in USD (integer).

> Therefore, both last name and first name do not contain space (If separation is necessary, the character "_" is used.

Note that data in this file is not sorted.

Constraints:

Particular attention should be given to the quality of the code:

- Division into (sub) functions.
- Use significative names for variables and functions.
- Organize the code so that to have .c and .h files.
- Do not forget to comment your code.

Only those libraries can be used: <stdlib.h>, <stdbool.h>, <string.h> et <stdio.h>, Therefore, <math.h> is not allowed

Recommendations:

It is recommended before starting your code, to prepare the library "vector". This one may contain the structure of your vector and some of or all the following functions:

- vector* create_vector();
- bool is_null_or_empty(vector const * p_vec);
- bool add(double element, vector * p_vec);
- unsigned int size(vector const * p_vec);
- double* element_at(unsigned int index, vector const * p_vec);
- bool remove_at(unsigned int index, vector * p_vec);
- void delete_vector(vector * p_vec);

About manuscript:

A mini-report up to 4 pages is to provide with your code. It must contain:

- The full names of the students who did the work.
- An accurate explanation, in English, about the chosen structure to represent the table.
- An explanation, in English, of how each functionality is coded and its corresponding number in the menu.
- A small scheme can possibly used to illustrate the general idea of your conception.