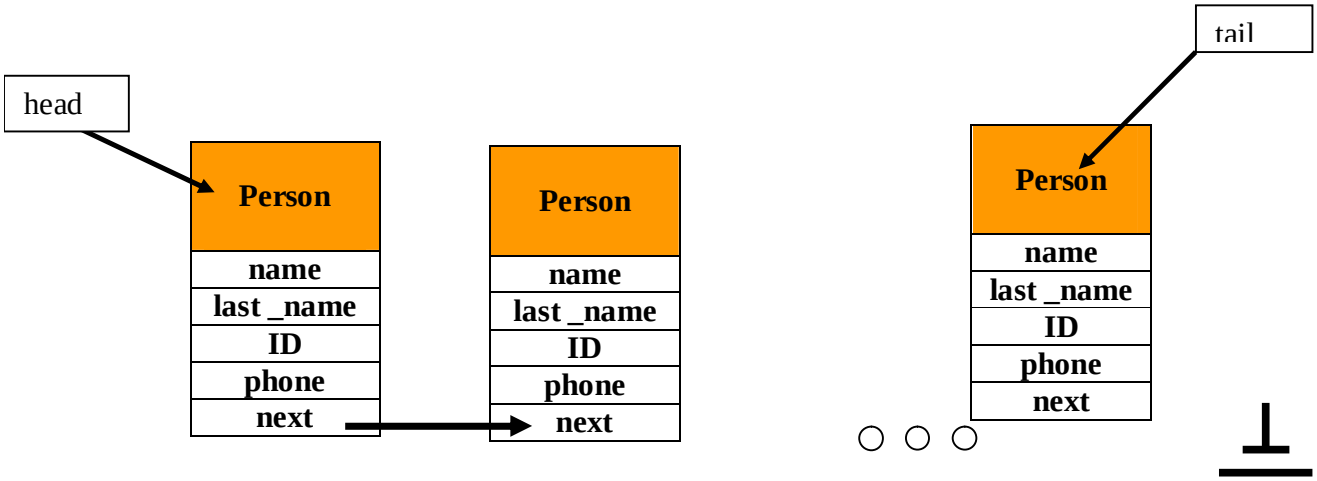
Phonebook Project

Goal

We'd like to create a simple phonebook, which will contain a list of persons and their details. The phonebook will be arranged and kept in some sorting order, requested by the user .

Basic elements& Supported features

The phonebook will be implemented by a linked list as shown:



Part I: *A person*

Person's components:

- **`name`** – dynamically allocated string up to MAX_BUFF chars.
- **`last_name`** - dynamically allocated string up to MAX_BUFF chars.
- **`ID`** - dynamically allocated string up to MAX_BUFF chars. It should be confirmed that ID is 9 chars long, all of which are numeric.
- **`phone`** - dynamically allocated string up to MAX_BUFF chars.
- **`next`** – pointer to next person.

Supported features:

- **`create_person`** : allocates a new person .
- **`read_person`** : given some person, initializes from the user all necessary fields.
  This refers to an existing person as well as a reference to a new person.
  At the end of the update process, ID must hold a legal format.

- **`reset_person`** : given some person, assigns 00 to all it fields.
  This refers to an existing person as well as a reference to a new person.

- **`disp_person`** : displays all a person's details.
- **`free_entries`**: frees all fields of the person.
- **`free_person`**: frees a person structure.

Part II *The phonebook list*

List's components:

- **`head`** – pointer to head of list
- **`tail`** – pointer to tail of list
- **`len`** – current number of items in the list.

These variables should be declared globally in one implementation file and `extern` in all the rest of the implementation files of the project.

Supported features:

- **`is_empty`** – check if the list is empty of any persons.

- **`add_item_by_ID`** – create a new person, initialize its details from user and add the new item to the list in the right position, by ID ascending order .

- **`add_item_by_name`** – create a new person, initialize its details from user and add the new item to the list in the right position, by *"< last_name><name>"*, ascending order. (policy: spaces insensitive.)

- **`find_name`** - find a person by the name of: *"< last_name><name>"*, and display item's details. (policy: spaces insensitive.)

- **`remove_name`** - find a person by the name of: *"< last_name><name>"*, and remove it.
- **`update_person`** – update a person details with new details, received from user.

- **`sort_by_id`** – sorts an unsorted list by ID ascending order.
- **`sort_by_name`** – sorts an unsorted list by *"< last_name><name>"* ascending order. (policy: spaces insensitive)
- **`clear_list`** – remove all of the list's people.
- **`disp_list -`** display all of the list's people on the screen.

User Interface

All the operations will be conducted through an interface that will enable user to choose between the following options:
1. add item by #ID
2. add item by Name
3. find a name
4. remove a name
5. update person
6. sort by ID
7. sort by Name
8. display list
9. Quit

- Invalid selections should be notified and ignored.
- Unless user has chosen to quit, the program will run the selected option and re-enable these options.
- If user chose to quit , the program will terminate in an organized manner.

General Coding Instructions

1. The code should be divided into **5** source files:
   - ***person.h*** – header file with all functions prototypes, #defines, #macros, tytpedefs etc. for the person's elements and features.
   - ***person.cpp*** – implementation file of the above.
   - ***phone_list.h-*** header file with all functions prototypes, #defines, #macros, tytpedefs etc. for the phone list elements and features.
   - ***phone_list.cpp-*** implementation file of the above.
   - ***phone_list _main.cpp*** – main( ) file, that will communicate with the user, invoke all the operations and manage all errors.

2. The functions mentioned are the minimal must. You might, of course, implement additional ones to facilitate stages and include them in other functions.
3. You are free to supply each function with any arguments you choose as well as its return values.
4. Please provide comments that explain what u do (See some solutions on site to follow similar patterns)

5. Don't "re- invent the wheel". Use conventional libraries/macros for popular manipulations: swap, strcmp, atoi…
6. Make sure to check status after any "system" manipulation : malloc(),realloc() free(), buffer overflows etc.
7. Good luck!