Recuperatorio Laboratorio 1

Nombre: Lucas

Apellido Ferrini

Curso y division: 1ºF

Legajo / DNI: 37751976

Fecha: 05/07/2021

Teniendo en cuenta la consigna propuesta para la primera instancia del

parcial deberán:

1- Agregar una estructura más al proyecto. La misma deberá poder

relacionarse de alguna forma con las trabajadas en la primera instancia.

Deberá refactorizar el programa para utilizar también esta estructura. La

estructura debe tener sentido dentro del dominio que estamos manejando.

2- Crear por lo menos dos informes complejos que involucren las 3

estructuras trabajadas. No se podrán utilizar los informes ya solicitados,

debido a que formarían parte de la refactorización.

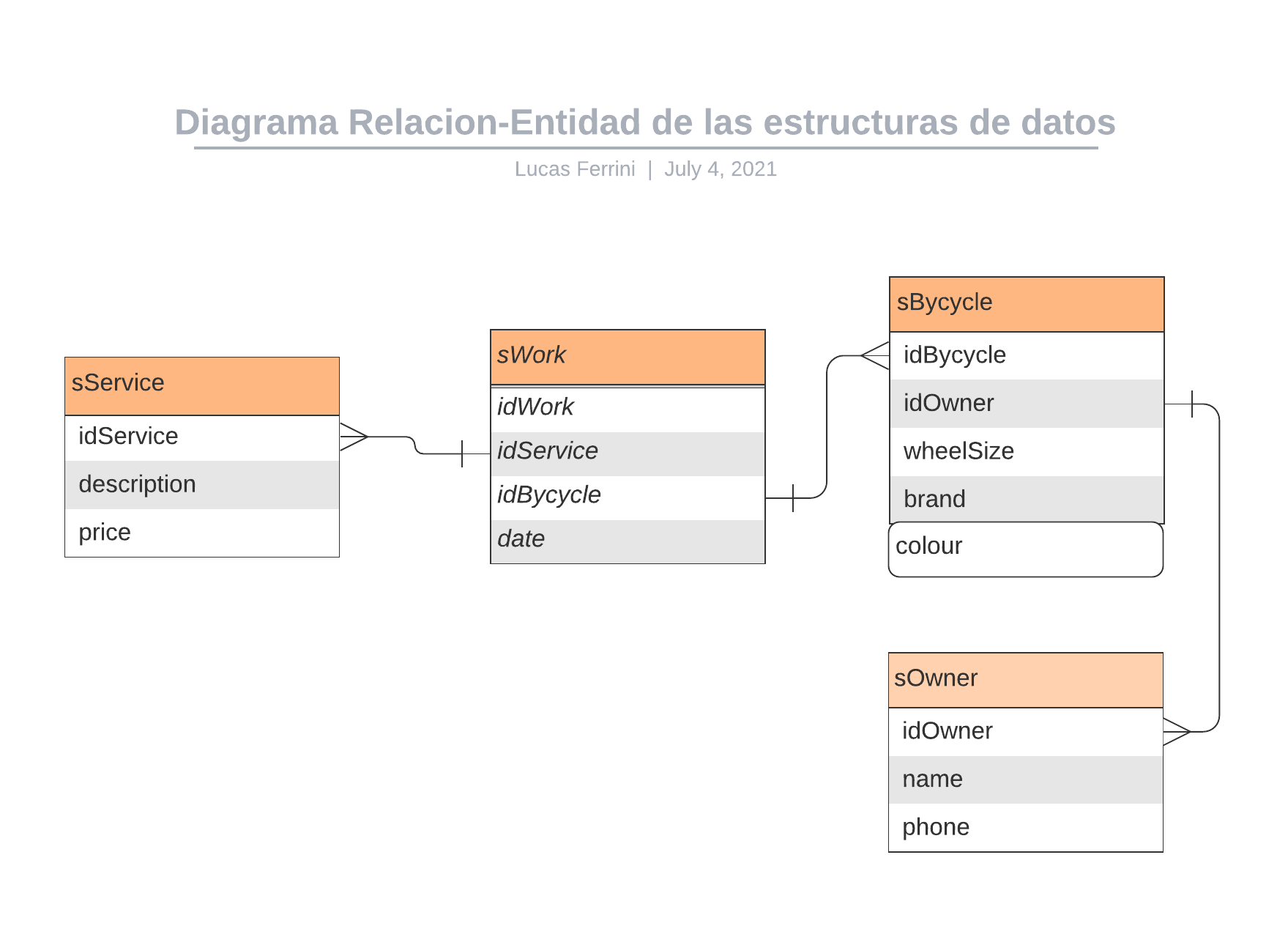
3- Entregar un documento con el prototipo y documentación de cada función

que utilizan en el programa según cada biblioteca. No es necesario hacerlo

con las funciones accesorias, solo las que forman parte del núcleo del

programa.

4- Grabar un video de no más de 10 minutos de duración, explicando las

funciones más importantes de su programa. La explicación tiene que ser clara

y deberá incluir las funciones refactorizadas y los nuevos informes. Las

funciones deben ser explicadas, no queremos la lectura del código.

Enlace al video: <https://drive.google.com/file/d/1o1eNCUetvS6hBCyX6-QEWZGO1qzhgQOJ/view>

Documentacion de las funciones principales:  
  
**Vector.h**

/\*\* \brief Creates a Vector struct, initialize it's value and returns a copy of itself

\*

\* \param pElement void\* Pointer to the adress of the array to stroe

\* \param len int Amount of elements in the array

\* \param size size\_t Size in bytes of an element of the array

\* \return Vector - a null vector {NULL,0,0} if any parameters are NULL

\* -a vector set with the given parametrs if ok

\*

\*/

Vector vector\_new(void\* pElement, int len, size\_t size);

/\*\* \brief Returns the amount of elements in the vector

\*

\* \param this Vector\* Pointer to vector

\* \return int -1 if vector is NULL

-Amount of elements if ok

\*

\*/

int vector\_len(Vector\* this);

/\*\* \brief Gets the size of the elements contained in a vector

\*

\* \param this Vector\* Pointer to the vector

\* \param size size\_t\* Output pointer for the size of the element

\* \return int -1 if any pointers are NULL

\* 0 if ok

\*

\*/

int vector\_getSize(Vector\* this, size\_t\* size);

/\*\* \brief Gets the element stored at index in a vector

\*

\* \param this Vector\* Pointer to the vector

\* \param index int Index of the element;

\* \return void\* NULL if vector is NULL or index minor to 0 or index bigger than vector's len

Adress of the element if ok

\*

\*/

void\* vector\_getElement(Vector\* this, int index);

/\*\* \brief Gets the element of a vector that has the same ID as the one given as a parameter

\*

\* \param this Vector\* Pointer to vector

\* \param idParameter int\* ID to search for

\* \param int(\*getId)(void\*,void\*) ID getter

\* \param int (\*isEmpty)(void\*) Pointer to function to check if the element has been removed previously

\* \return void\* NULL if couldn't find an element matching the ID or vector pointer is NULL or ID pointer is NULL

Address of element if ok

\*

\*/

void\* vector\_getElementById(Vector\* this, int\* idParameter, int(\*getId)(void\*,void\*), int (\*isEmpty)(void\*));

/\*\* \brief Sets an element in the given index of a vector

\*

\* \param this Vector\* Pointer to vector

\* \param index int Index of the element;

\* \param pElement void\* Pointer to element to store

\* \return int -1 if any pointers are NULL or index is minor to 0 or bigger than the amount of elements

0 if ok

\*

\*/

int vector\_setElement(Vector\* this, int index, void\* pElement);

/\*\* \brief Sorts a vector using the criteria given through a function

\*

\* \param this Vector\* Pointer to vector

\* \param int (\*pFunc)(void\*, void\*) Compare function to sort the vector

\* \param order int 1 for ascending order, 0 for descending

\* \return int -1 if couldn't sort the vector

0 if ok

\*

\*/

int vector\_sort(Vector\* this, int (\*pFunc)(void\*, void\*),int order);

/\*\* \brief Searchs the current vector to find any empty slots

\*

\* \param this Vector\* Pointer to vector

\* \param int (\*checkSpace)(void\*) Function to check if the current element is available to use

\* \return int -1 if vector is full

index of the first empty space if ok

\*

\*/

int vector\_searchEmptySpace(Vector\* this, int (\*checkSpace)(void\*));

/\*\* \brief Applies the given function to every element in the vector

\*

\* \param this Vector\* Pointer to vector

\* \param int(\*pFunc)(void\*) Pointer to function to apply

\* \return int -1 if either pointers are NULL

amount of elements mapped if ok

\*

\*/

int vector\_map(Vector\* this, int(\*pFunc)(void\*));

/\*\* \brief Searchs a vector for an element that metchs the given ID

\*

\* \param Vector\* this Pointer to vector

\* \param int(\*compare)(void\*, void\*) ID Comparison function

\* \param int(\*isEmpty)(void\*) Function to check if the current element is available to use

\* \param int(\*getId)(void\* pElement, void\*pParameter)

\* \param int\* targetId

\* \return int vector\_searchById(Vector\* this, int(\*compare)(void\*,

\*

\*/

int vector\_searchById(Vector\* this, int(\*compare)(void\*, void\*),int(\*isEmpty)(void\*),

int(\*getId)(void\* pElement, void\*pParameter), int\* targetId);

/\*\* \brief Filters a vector given a criteria. Stores the filtered elements in buffer

\*

\* \param this Vector\* Pointer to vector

\* \param int(\*filter)(void\*,void\*) Pointer tu function to apply criteria to every element

\* \param ciretia void\* Criteria to filter

\* \param buffer void\* Adress to store the filtered elements

\* \return int vector\_filter(Vector\* this,

\*

\*/

int vector\_filter(Vector\* this, int(\*filter)(void\*,void\*),void\* criteria, void\* buffer);

**Controller.h**

/\*\* \brief Initializes and hardcore the data inside the given vectors

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \param owners Vector\* Pointer to owners array

\* \return int 0 if ok

-1 if any pointers are NULL

\*

\*/

int controller\_initProgram(Vector\* works, Vector\* services, Vector\* bycycles, Vector\* owners);

/\*\* \brief Adds a new work to the list

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \return int -1 if any pointers are NULL or the user cancels the input

0 if ok

\*

\*/

int controller\_addWork(Vector\* works, Vector\* services, Vector\* bycycles);

/\*\* \brief Asks the user to input the data related to the new work

\*

\* \param new sWork\* Pointer to the new work

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \return int 0 if ok

-1 if any pointers are NULL

\*

\*/

int controller\_inputWork(sWork\* new, Vector\* services, Vector\* bycycles);

/\*\* \brief Modifies the selected work

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \return int -1 if any pointers are NULL or the user cancels the modification

0 if ok

\*

\*/

int controller\_editWork(Vector\* works, Vector\* services, Vector\* bycycles);

/\*\* \brief Removes a work from the list

\*

\* \param works Vector\* Pointer to works array

\* \return int 0 if ok

-1 if the user cancels the removal or pointers are NULL

\*

\*/

int controller\_removeWork(Vector\* works);

/\*\* \brief Lists all the current works

\*

\* \param works Vector\* Pointer to works array

\* \return int -1 if pointer is NULL

Amount of elements printed if ok

\*

\*/

int controller\_listWorks(Vector\* works);

/\*\* \brief Lists all the current services

\*

\* \param services Vector\* Pointer to services array

\* \return int -1 if pointer is NULL

Amount of elements printed if ok

\*

\*/

int controller\_listServices(Vector\* services);

/\*\* \brief Prints the list of works and shows the total amount of earnings from all the works

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \param owners Vector\* Pointer to owners array

\* \return float -1 if any pointers are NULL

Total earnings if OK

\*

\*/

float controller\_TotalEarnings(Vector\* works, Vector\* services ,Vector\* bycycles, Vector\* owners);

/\*\* \brief Sorts the array of works using the bycycle's brands as criteria

\*

\* \param works Vector\* Pointer to works array

\* \param bycycles Vector\* Pointer to bycycles array

\* \return int -1 if either pointers are NULL

0 if ok

\*

\*/

int controller\_sortWorksByBrand(Vector\* works, Vector\* bycycles);

/\*\* \brief Compares two given works using their associated bycycle's brands as criteria

\*

\* \param firstWork sWork\* Pointer to first work

\* \param secondWork sWork\* Pointer to second work

\* \param bycycles Vector\* Pointer to array of bycycles

\* \return int -2 if any pointers are NULL

-1 if first work's brand is closer to A than second work's one

0 if brands are the same

1 if first work's brand is closer to z than second work's one

\*

\*/

int controller\_compareWorksByBrand(sWork\* firstWork, sWork\* secondWork, Vector\* bycycles);

/\*\* \brief Sorts, compares and list all works using the bycycle's brand as criteria

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \param owners Vector\* Pointer to owners array

\* \return int -1 if any pointers are NULL

Amount of elements printed if OK

\*

\*/

int controller\_listWorksByBrand(Vector\* works, Vector\* bycycles, Vector\* services, Vector\* owners);

/\*\* \brief Gets a bycycle from the array using a work as the criteria to get its ID

\*

\* \param work sWork\* Pointer to work

\* \param bycycles Vector\* Pointer to array of bycycles

\* \return sBycycle\* NULL if any pointers are NULL

Bycycle that match the work's bycycle ID if ok

\*

\*/

sBycycle\* controller\_getBycycleByWork(sWork\* work, Vector\* bycycles);

/\*\* \brief Gets and prints the most done services

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \param owners Vector\* Pointer to owners array

\* \return int -1 if any pointers are NULL

Amount of printed services if ok

\*

\*/

int controller\_listMostDoneServices(Vector\* works, Vector\* services, Vector\* bycycles, Vector\* owners);

/\*\* \brief Prints all the works associated with the given service

\*

\* \param service sService\* Pointer to service

\* \param works Vector\* Pointer to works array

\* \param bycycles Vector\* Pointer to bycycles array

\* \param owners Vector\* Pointer to owners array

\* \return int 0 if ok

-1 if any pointers are NULL

\* \return int

\*

\*/

int controller\_printWorksByService(sService\* service, Vector\* works, Vector\* bycycles, Vector\* owners);

/\*\* \brief Counts how many times each service was performed

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param counters int\* Output pointer for the counters

\* \return int -1 if either pointers are NULL

Amount of different services counted if ok

\*

\*/

int controller\_countServices(Vector\* works, Vector\* services, int\* counters);

/\*\* \brief Prints the information of a work using the data of bycycles and services whenever is possible

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \param owners Vector\* Pointer to owners array

\* \return int Amount of works listed if ok

-1 if any pointers are NULL

\*

\*/

int controller\_listServicesWithBycycles(Vector\* works, Vector\* services, Vector\* bycycles, Vector\* owners);

/\*\* \brief Gets a service using a given work as criteria to get it's ID

\*

\* \param work sWork\* Pointer to given work

\* \param services Vector\* Pointer to services array

\* \return sService\* NULL if any pointers are NULL

Service provided if ok

\*

\*/

sService\* controller\_getServiceByWork(sWork\* work, Vector\* services);

/\*\* \brief Lists all bycycles whose colour matches "Rojo" (Red)

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \param owners Vector\* Pointer to owners array

\* \return int Amount of bycycles if ok

-1 if any pointers are NULL

\*

\*/

int controller\_listRedBycycles(Vector\* works, Vector\* bycycles, Vector\* services, Vector\* owners);

/\*\* \brief Lists the owners of bycycles that spent the most in services

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \param owners Vector\* Pointer to owners array

\* \return int Amount of bycycles if ok

-1 if any pointers are NULL

\*

\*/

int controller\_OwnersThatSpentMost(Vector\* works, Vector\* services, Vector\* bycycles, Vector\* owners);

/\*\* \brief Compares if the owner id of a bycycle matchs the id of an owner

\*

\* \param bycycle sBycycle\* Pointer to bycycle

\* \param owner sOwner\* Pointer to owner

\* \return int -2 if either pointers are NULL

-1 if ids don't match

0 if ids match

\*

\*/

int controller\_compareBycycleAndOwner(sBycycle\* bycycle, sOwner\* owner);

/\*\* \brief Counts how much every owner spent in services

\*

\* \param works Vector\* Pointer to works array

\* \param services Vector\* Pointer to services array

\* \param bycycles Vector\* Pointer to bycycles array

\* \param owners Vector\* Pointer to owners array

\* \param totals float\* Output pointer to store the accumulated values

\* \return -1 if any pointers are NULL

Amount of owners counted if ok

\*

\*/

int controller\_countEarningsPerOwner(Vector\* works, Vector\* services, Vector\* bycycles, Vector\* owners, float\* totals);

/\*\* \brief Compares if a given work was done to a given owner using the bycycle's ids to make a match

\*

\* \param owner sOwner\* Pointer to owner

\* \param work sWork\* Pointer to work

\* \param bycycles Vector\* Pointer to bycycles array

\* \return int -2 if any pointers are NULL

-1 if the given work wasn't done to the given owner

0 if the given work was done to the given owner

\*

\*/

int controller\_compareOwnerByWork(sOwner\* owner, sWork\* work, Vector\* bycycles);

/\*\* \brief Sorts the works array using the bycycle's owner phone number as a criteria

\*

\* \param works Vector\* Pointer to works array

\* \param bycycles Vector\* Pointer to bycycles array

\* \param owners Vector\* Pointer to owners array

\* \return int -1 if any pointers are NULL

0 if the array was sorted

\*

\*/

int controller\_sortWorksByPhoneNumber(Vector\* works, Vector\* bycycles, Vector\* owners);

/\*\* \brief Sorts and prints the works using the bycycle's owner phone number as a criteria

\*

\* \param works Vector\*

\* \param bycycles Vector\*

\* \param owners Vector\*

\* \param services Vector\*

\* \return int

\*

\*/

int controller\_ListWorksByPhoneNumber(Vector\* works, Vector\* bycycles, Vector\* owners, Vector\* services);

/\*\* \brief Gets an owner using a bycycle as a criteria to get its ID

\*

\* \param bycycle sBycycle\* Pointer to bycycle

\* \param owners Vector\* Pointer to owners array

\* \return sOwner\* NULL if either pointers are NULL

Pointer to the owner of the bycycle if ok

\*

\*/

sOwner\* controller\_getOwnerByBycycle(sBycycle\* bycycle, Vector\* owners);

/\*\* \brief Prints all data related to a given work, including the data contained in its related structs

\*

\* \param work sWork\* Pointer to the work to print

\* \param service sService\* Pointer to its related service

\* \param bycycle sBycycle\* Pointer to its related bycycle

\* \param owner sOwner\* Pointer to the bycycle's owner

\* \return int -1 if any pointers are NULL

0 if the data was printed

\*

\*/

int controller\_printInform(sWork\* work, sService\* service, sBycycle\* bycycle, sOwner\* owner);