Nombre: Tomas Mastrapasqua

[Dirección de correo electrónico]

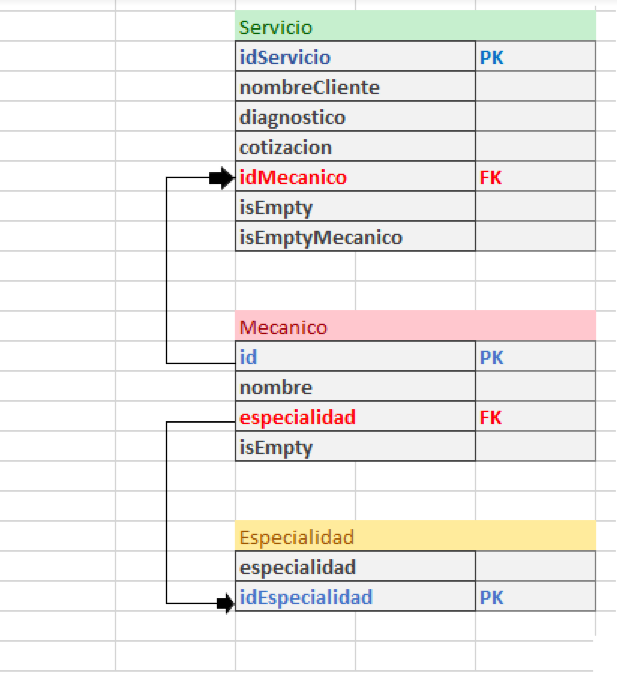
Descripción breve

DNI: 44668569  
División: 1ºB  
Materia: Laboratorio

Recuperatorio Primer Parcial Laboratorio

Tomasmastrapasqua3@gmail.com

Diagrama Entidad Relacion





Servicio.h

/\*\* \brief to indicate that all positions in the array are empty and the array don't have a mechanic

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \return void

\*

\*/

void InicializarServicios(eServicio servicio[], int tamServicio);

void HarcodearServicios(eServicio servicio[], int tamServicio);

/\*\* \brief esta funcion llama a la de BuscarLibre para buscar una posicion libre y poder llamar a la funcion para pedir los datos, le pasa la posicion de i

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param id int

\* \return void

\*

\*/

void AgregarServicio(eServicio servicio[], int tamServicio, int id);

/\*\* \brief to find a empty position to add a service

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \return int (index position)

\*

\*/

int BuscarLibre(eServicio servicio[], int tamServicio);

/\*\* \brief pide los datos del servicio y pregunta si los queres guardar llamando a una funcion para confirmar

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param i int

\* \return void

\*

\*/

void PedirServicio(eServicio servicio[], int tamServicio, int i);

/\*\* \brief para confirmar si queres guardar el servicio, si ingresas s (se valida que sea minuscula para preguntar en el if) se guarda

\*

\* \param message char\*

\* \return int ret (1 si ingresaste s)

\*

\*/

int ConfirmarProducto(char\* message);

/\*\* \brief to delete a service. The funcion change isEmpty to -1

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \return void

\*

\*/

void EliminarServicio(eServicio servicio[], int tamServicio);

/\*\* \brief muestra los servicios de la posicion i con la posicion isEmpty en 1

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \return void

\*

\*/

void ListarServicios(eServicio servicio[], int tamServicio);

/\*\* \brief muestra un servicio

\*

\* \param servicio eServicio

\* \return void

\*

\*/

void ListarServicio(eServicio servicio);

/\*\* \brief ordena por fecha los servicios. Primero compaar el año y si es igual compara el mes, despues compara el mes y si es igual hace lo mismo con el dia

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \return void

\*

\*/

void OrdenarPorFecha(eServicio servicio[], int tamServicio);

/\*\* \brief suma las cotizaciones de cambio de aceite para la fecha del 26/1/2023 y el diagnostico de "Cambio de aceite"

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \return void

\*

\*/

void SumarCotizacionesCambioAceite(eServicio servicio[], int tamServicio);

/\*\* \brief busca la posicion del producto, se le pasa el id para que busque y compare

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param id int

\* \return void

\*

\*/

int BuscarProductoID(eServicio servicio[], int tamServicio, int id);

/\*\* \brief Valida que haya un servicio. me fijo si hay un sercicio con el isEmpty en 1 (mayor a -1), en ese caso con que haya 1 servicio puede ingresar en algunas opciones

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \return int ret (0 si el isEmpty no es -1)

\*

\*/

int ValidarServicio(eServicio servicio[], int tamServicio);

/\*\* \brief valida las fechas, que si ingresa el mes de febrero y el dia 29 vuelva a pedir los datos. No valido año bisiesto

\*

\* \param servicio eServicio[]

\* \param i int

\* \return void

\*

\*/

void PedirFechas(eServicio servicio[], int i);

/\*\* \brief pregunta por las fechas con las funcion de getint

\*

\* \param dia int\*

\* \param mes int\*

\* \param anio int\*

\* \return void

\*

\*/

void PreguntarFechas(int\* dia, int\* mes, int\* anio);

Mecanico.h

void InicializarMecanicos(eMecanico mecanico[], int tamMecanico);//CAMBIAR

/\*\* \brief harcodea los mecanicos. Primero se le asigna a unas variables y despues recorro un for y le asigno el valor a cada campo en la posicion de i

\*

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \return void

\*

\*/

void HarcodearMecanicos(eMecanico mecanico[], int tamMecanico);

/\*\* \brief Valida que haya un mecanico. No la uso porque los mecanicos estan harcodeados

\*

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \return int ret (0 si encontro 1 mecanico)

\*

\*/

int ValidarMecanico(eMecanico mecanico[], int tamMecanico);

/\*\* \brief

\*

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \return void

\*

\*/

void OrdenarPorEspecialidad(eMecanico mecanico[], int tamMecanico);

/\*\* \brief compara si el id que le pasas a la funcion es igual a el id de un mecanico, en caso de que sean iguales retorna 0, si son diferentes -1

\*

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \param id

\* \return int (0 si existe, -1 si no existe)

\*

\*/

int ValidarMecanicoExiste(eMecanico mecanico[], int tamMecanico, int id);

Especialidad.h

/\*\* \brief harcodea las especialidades. Las especialidades se relacionan con los mecánicos mediante un id (1,2 o 3)

\*

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void harcodeoEspecialidad(eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief Muestra el top 3 de especialidades mas estudiadas

\*

\* \param servicio eServicio[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void MostrarEspecialidadesMasEstudiadas(eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief Ordena las especialidades por cantidad

\*

\* \param servicio eServicio[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void ordenarPorCantidad(eEspecialidad especialidad[], int tamEspecialidad);

Input.h

/\*\*

\* @fn int getInt(int\*, char\*, char\*, int, int)

\* @brief

\*

\* @param integer recive un puntero a un int

\* @param message recive un mensaje para mostrar

\* @param messageError recive un mensaje de error para mostrar

\* @param min recive un numero indicando el valor minimo

\* @param max recive un numero indicando el valor maximo

\* @return ret 0 si salio bien o -1 si salio mal

\*/

int getInt(int\* integer, char\* message, char\* messageError, int min, int max);

/\*\*

\* @fn int getFloat(float\*, char\*, char\*, int, int)

\* @brief

\*

\* @param floating recive un puntero a un float

\* @param message recive un mensaje para mostrar

\* @param messageError recive un mensaje de error para mostrar

\* @param min recive un numero indicando el valor minimo

\* @param max recive un numero indicando el valor maximo

\* @return ret 0 si salio bien o -1 si salio mal

\*/

int getFloat(float\* floating, char\* message, char\* messageError, int min, int max);

/\*\*

\* @fn int getString(char\*, char\*, char\*, int)

\* @brief

\*

\* @param string recive un puntero a un char

\* @param message recive un mensaje para mostrar

\* @param messageError recive un mensaje de error para mostrar

\* @param max recive un numero indicando el tamaño maximo para el string

\* @return ret 0 si salio bien o -1 si salio mal

\*/

int getString(char\* string, char\* message, char\* messageError, int max);

/\*\*

\* @fn int getChar(char\*, char\*)

\* @brief

\*

\* @param character recive un puntero a un char

\* @param message recive un mensaje para mostrar

\* @return ret 0 si salio bien o -1 si salio mal

\*/

int getChar(char\* character, char\* message);

/\*\*

\* @fn int tolowerString(char\*)

\* @brief

\*

\* @param string recive un puntero a un char, lo cual deberia devolver el string con la primer letra mayuscula y el resto minuscula

\* @return ret 0 si salio bien o -1 si salio mal

\*/

int tolowerString(char\* string);

/\*\*

\* @fn int validateLetters(char\*)

\* @brief

\*

\* @param string recive un puntero a un char y determina si alguna de sus letras no es una letra minuscula, mayuscula o un espacio

\* @return ret 0 si salio bien o -1 si salio mal

\*/

int validateLetters(char\* string);

/\*\*

\* @fn int validateInt(char\*)

\* @brief determina si se ingreso un numero

\*

\* @param integer recive un puntero a un char

\* @return ret 0 si salio bien o -1 si salio mal

\*/

int validateInt(char\* integer);

/\*\*

\* @fn int getName(char\*, char\*, char\*, int)

\* @brief Pide un nombre y a diferencia de getString esta no es alfanumerica, esta funcion valida numeros y simbolos

\*

\* @param string recive un puntero a un char

\* @param message recive un mensaje para mostrar

\* @param messageError recive un mensaje de error para mostrar

\* @param min recive un numero indicando el tamaño maximo para el string

\* @return ret 0 si salio bien o -1 si salio mal

\*/

int getName(char \*string, char \*message, char \*messageError, int max);

Nexo.h

/\*\* \brief Muestra un submenu con mas opciones. Para ingresar debe poner la opcion 5 en el menu anterior

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void MostrarMenu(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief Muestra un menu con opciones. Para ingresar a la 2,3 y 4 debe ingresar un servicio

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void MostrarSubMenu(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief to modify the service. The funcion allow you to change name, date abd mechanic

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void ModificarServicio(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief se le asigna un mecanico a un servicio. Primero pregunto por el mecanico, despues por el servicio en la funcion de AsignarIdMecanico y ahi al idMecanico del es igual al id del mecanico

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void Diagnosticar(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief muestra un mecanico

\*

\* \param servicio eServicio

\* \return void

\*

\*/

void ListarMecanico(eMecanico mecanico, eEspecialidad especialidad);

/\*\* \brief muestra los mecanicos de la posicion con mas servicios

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void ListarMecanicos(eMecanico mecanico[], int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief muestro un servicio y un mecanico

\*

\* \param servicio eServicio

\* \param mecanico eMecanico

\* \return void

\*

\*/

void ListarServicioyMecanico(eServicio servicio, eMecanico mecanico, eEspecialidad especialidad);

/\*\* \brief muestra los servicios junto con los mecanicos, para eso llamo a la funcion de ListarServicioyMecanico en la posicion de i para el mecanico y j para el servicio

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \return void

\*

\*/

void ListarServiciosyMecanicos(eServicio servicio[], int tamServicio, eMecanico mecanico[], int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief // busca si el mecanico tiene un servicio o no, en caso de que no tenga lo muestra. Uso un contador

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \return void

\*

\*/

void ListarMecanicosSinServicio(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief busca la cantidad de servicios que tiene cada mecanico y busca el mecanico con mas cantidad. Despues llama a la funcion de abajo y le pasa la cantidad de servicios de cada mecanico y el mecanico con mas cantidad

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \return void

\*

\*/

void DeterminarMecanicosMasServicios(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief muestra los mecanicos con mas servicios. Compara si la cantidad de servicios de un mecanico es igual a la mayor cantidad que tiene un mecanico

\*

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \param cantidad int[]

\* \param masCaro int

\* \return void

\*

\*/

void MostrarMecanicosMasServicios(eMecanico mecanico[], int tamMecanico, int cantidad[], int masCaro, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void CalcularPromedioCotizacionesMecanico(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief muestra los servicios con una fecha entre marzo y mayo del 2020. Para la especialidad hago un strcmp y comparo la especialidad con "Suspencion" para ver si son iguales, en ese caso seria 0

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void ListarServiciosEntreMayoyMarzo2022(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief calcula el porcentaje de cada diagnostico 1, 2 o 3

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \return void

\*

\*

\*/

void CalcularPorcentajeDiagnostico(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief calcula la cantidad de servicios diagnosticados. Lo que hago es comparar si idMecanico de servicio es igual a la id del mecanico, ahi el contador se suma 1 numero

\*

\* \param servicio eServicio[]

\* \param tamServicio int

\* \param mecanico eMecanico[]

\* \param tamMecanico int

\* \return int contador (cantidad de servicios diagnosticados)

\*

\*/

int CalcularServiciosDiagnosticados(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico);

/\*\* \brief Compara un mecanico que le pasamos con un los servicios hasta buscar que sean iguales. En ese caso retorna la posicion del servicio y si no se cumple retorna -1

\*

\* \param servicio eServicio[]

\* \param mecanico eMecanico[]

\* \param tamServicio int

\* \param index int

\* \return int auxIndex (posicion del servicio)

\*

\*/

int CompararMecanicoServicio(eServicio servicio[], eMecanico mecanico[], int tamServicio, int index);

/\*\* \brief Compara la especialidad de un mecanico que le pasamos con un los los ids de las especialidades hasta buscar que sean iguales. En ese caso retorna la posicion del la especialidad y si no se cumple retorna -1

\*

\* \param especialidad eEspecialidad[]

\* \param mecanico eMecanico[]

\* \param tamEspecialidad int

\* \param index int

\* \return int auxIndex (posicion de la especialidad)

\*

\*/

int CompararMecanicoEspecialidad(eEspecialidad especialidad[], eMecanico mecanico[], int tamEspecialidad, int index);

/\*\* \brief Compara un servicio que le pasamos con un los mecanicos hasta buscar que sean iguales. En ese caso retorna la posicion del mecanico y si no se cumple retorna -1

\*

\* \param mecanico eMecanico[]

\* \param servicio eServicio[]

\* \param tamMecanico int

\* \param index int

\* \return int auxIndex (posicion del mecanico)

\*

\*/

int CompararServicioMecanico(eMecanico mecanico[], eServicio servicio[], int tamMecanico, int index);

/\*\* \brief Muestra los mecanicos ordenados por especialidad. Para eso recorre el array de mecanicos. Previamente lo ordeno por especialidad del mecanico (int) y recorro el array de mecanicos

\*

\* \param servicio eServicio[]

\* \param mecanico eMecanico[]

\* \param tamServicio int

\* \param tamMecanico int

\* \param especialidad eEspecialidad[]

\* \param tamEspecialidad int

\* \return void

\*

\*/

void ListarOrdenadoPorEspecialidad(eServicio servicio[], eMecanico mecanico[], int tamServicio, int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief Asigna el id del mecanico al servicio

\*

\* \param servicio eServicio[]

\* \param mecanico eMecanico[]

\* \param tamServicio int

\* \param tamMecanico int

\* \param id int

\* \return void

\*

\*/

void AsignarIdMeacnico(eServicio servicio[], eMecanico mecanico[],int tamServicio, int tamMecanico, int id);

/\*\* \brief Valida que la estadia no tenga un id de un mecanico igual a la de otro mecanico

\*

\* \param servicio eServicio[]

\* \param mecanico eMecanico[]

\* \param index int

\* \param tamMecanico int

\* \return int ret -1 si esta diagnosticado y 1 si no lo esta

\*

\*/

int validarMecanicoDiagnosticado(eServicio servicio[], eMecanico mecanico[], int index, int tamMecanico);

/\*\* \brief busca cuantos servicios tiene un mecanico. Recorre los servicios y si el id del mecanico de la posición que le paso es igual al id mecanico de un servicio sumo el contador

\*

\* \param servicio eServicio[]

\* \param mecanico eMecanico[]

\* \param index int

\* \param tamServicio int

\* \return int contador (cantidad de servicios diagnosticados que tiene un mecanico)

\*

\*/

int BuscarCantidadServiciosPorMecanico(eServicio servicio[], eMecanico mecanico[],int tamServicio ,int index);

/\*\* \brief Suma las cotizaciones de un mecanico en especifico, recorre un for con los servicios y busca si un id mecanico de un servicio es igual al id del mecanico de la posición index que le paso

\*

\* \param servicio eServicio[]

\* \param mecanico eMecanico[]

\* \param index int

\* \param tamServicio int

\* \return int acumulador (suma de cotización de \* \un mecanico)

\*

\*/

Int SumarCotizacionesServiciosPorMecanico(eServicio servicio[], eMecanico mecanico[],int tamServicio ,int index);

/\*\* \brief la cantidad Busca de mecanicos que estudiaron cada especialidad y las ordena de manera que la mas estudiada sea la primera y la menos estudiada la ultima

\*

\* \param mecanico eMecanico[]

\* \param especialidad eEspecialidad[]

\* \param tamMecanico int

\* \param tamEspecialidad int

\* \return void

\*

\*/

void CalcularEspecialidadesEstudiadas(eMecanico mecanico[], int tamMecanico, eEspecialidad especialidad[], int tamEspecialidad);

/\*\* \brief Busca cuantos mecanicos estudiaron esa especialidad, le paso la especialidad, comparo y voy sumando un contador

\* \param servicio eServicio[]

\* \param mecanico eMecanico[]

\* \param index int

\* \param tamMecanico int

\* \return int contador (cantidad de mecanicos que estudiaron esa especialidad)

\*

\*/

int BuscarCantidadEspecialidad(eMecanico mecanico[],int tamMecanico, eEspecialidad especialidad[],int index);

Video explicativo:

https://drive.google.com/file/d/17i2im9oudTNMbGM1ElvXqs\_e6U884ey8/view?usp=sharing