Universidad Tecnológica Nacional

Facultad Regional Avellaneda

Tecnicatura Universitaria en Programación

Recuperatorio Parcial I - Laboratorio

Alumno: Di Pino, Gonzalo

División F

Turno Noche

Profesor: Scarafilo, German

2021

Recuperatorio Laboratorio

Primer parcial

1. Alumnos que hayan obtenido un 4 y deseen levantar nota: deberán presentar el mismo trabajo que en la instancia anterior con las modificaciones que marcamos en las devoluciones (Parte 1 y 2 del parcial).

2. Alumnos que aún no hayan aprobado el parcial: deberán presentar el mismo trabajo que en la instancia anterior (parte 1 y 2) agregando lo siguiente:

a. Una estructura más, debidamente relacionada con las anteriores. La composición de la estructura deberá ser compleja: un id, un campo alfanumérico y uno entero o flotante (para utilizar en algún filtro).

b. Un filtro complejo distinto a los trabajados con anterioridad. Este filtro deberá utilizar la mayoría de las estructuras incluyendo la estructura agregada en el ítem anterior.

¿Qué material debo entregar?

1. Proyecto completo en eclipse.

2. Un video entre 10 y 15 minutos de duración, en donde deberán realizar la defensa del proyecto. Recuerden que deberán explicar, sin leer el código (pero si mostrándolo), las funciones más importantes del programa. Pueden tomarse unos minutos para demostrar cómo funciona la app. Tengan en cuenta que el video debe evidenciar las funciones más importantes del programa (Altas, Bajas, Modificaciones, Filtros complejos). Deberán estar con cámara prendida (Por ejemplo pueden presentar escritorio con zoom y estar con cámara prendida).

3. Un documento con:

a. Carátula con datos del alumno y la materia.

b. El detalle de la estructura agregada y su relación con las otras estructuras.

Realizar un diagrama entidad relación DER. (2b)

c. El enunciado del informe anexado (2b).

d. Prototipo de cada función del núcleo del programa con la documentación correspondiente.

e. Enlace a drive con el video explicativo.

Aclaraciones:

El trabajo deberá ser subido INTEGRAMENTE a un repositorio de Github que deberán informar mediante el formulario que está en el espacio del examen. No se corregirán parciales que sean entregados por otro medio. El video se debe ver y oír de manera clara. Por favor chequeen este punto antes de entregar.

Link a google drive con el video explicativo:

https://drive.google.com/drive/folders/1hIJYO3NhATq5Bc-mWXSZ18r1hcq2Lipt?usp=sharing

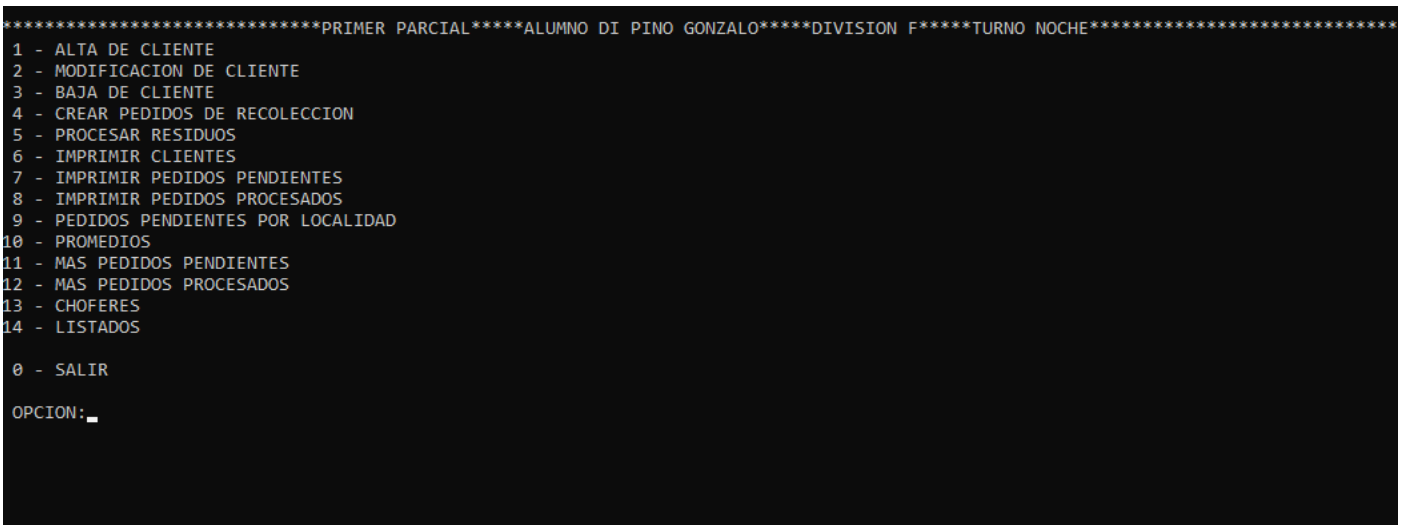
Detalle de la estructura agregada

A demás de finalizar el examen como se pide, cree una estructura que se relaciona con las demás, pero para ello tuve que agregar un campo más en una de las estructuras principales.

Esta nueva estructura a la que llame **“eChofer”** manipula datos de choferes que se encargan de recolectar los pedidos que se dan de alta y están en estado Pendiente hasta que se procesen. Como dije antes, para llevar a cabo una relación de esta estructura con las otras agregué un campo en la estructura **“ePedido”** al cual llame **IDdelChofer** para que, a la hora de realizar los nuevos filtros estos se relacionen con todas o la mayoría de las estructuras.

```
typedef struct {
    int id;
    char nombre[50];
    int salario;
    float horasTrabajadas;
    int isEmpty;
    int descuento;
}eChofer;
```

Esta nueva estructura está formada por un **id** único que representa a cada chofer, el **nombre** el cual se guarda en una cadena de caracteres, el **salario** que ganan, las **horas** que trabajan, las cuales se modifican a medida que se realizan pedidos, se elige el chofer y se determina cuanto tiempo tardo en dicho pedido, el campo **isEmpty** el cual determina si los choferes se encuentran activos o no y el campo **descuento** el cual determina si un chofer obtuvo una bonificación.



En el menú de opciones se agregaron dos opciones nuevas. La opción 13 – CHOFERES permite la manipulación de los choferes mientras que la opción 14 – LISTADOS muestra las nuevas relaciones realizadas para el recuperatorio.

Desarrollo de la nueva entidad

Detalle de la opción 13- CHOFERES

```
void SubMenuChoferes(eChofer* choferes, int tam_Choferes,int *idChofer) {

    int opcion;
    int retorno = -1;
    do {
        system("cls");
        printf("\n************************ C H O F E R E S ************************\n\n ");
        printf("1- ALTA DE CHOFER\n 2- BAJA DE CHOFER\n 3- MODIFICACION DE CHOFER\n 4 - LISTA DE CHOFERES\n\n 0 - VOLVER\n\n");
        opcion = LoadInt("OPCION", 0, 4);

        switch(opcion) {

            case 1:
                retorno = AltaDeChofer(choferes, tam_Choferes, idChofer);
                AlertMessageAndEnter(retorno, "Datos de chofer cargados con exito.","No se pueden agregar mas choferes, elimine alguno.");

                break;
            case 2:
                if(ValidarListaChoferes(choferes, tam_Choferes) == 0) {

                    retorno = BajaDeChofer(choferes, tam_Choferes);
                    AlertMessageAndEnter(retorno, "Chofer eliminado con exito.","No se elimino al chofer.");
                }else {
                    AlertMessageAndEnter(retorno, "","Cargue al menos un chofer.");
                }
                break;
            case 3:
                if(ValidarListaChoferes(choferes, tam_Choferes) == 0) {

                    retorno = ModificacionDeChofer(choferes, tam_Choferes);
                    AlertMessageAndEnter(retorno, "Datos de chofer modificados con exito.","No se modifico el chofer.");
                } else {
                    AlertMessageAndEnter(retorno, "","Cargue al menos un chofer.");
                }
                break;
            case 4:
                if(ValidarListaChoferes(choferes, tam_Choferes) == 0) {

                    ImprimirListaDeChoferes(choferes, tam_Choferes);
                    AlertMessageAndEnter(retorno, "","");
                } else {
                    AlertMessageAndEnter(retorno, "","Cargue al menos un chofer.");
                }
                break;
        }
        if(opcion != 0) {
            opcion = -1;
        }
        } while(opcion != 0);
    }
```

Esta opción abre un nuevo submenú el cual permite la manipulación del alta, baja y modificación de los choferes y además una opción que muestra un listado con todos los choferes que se encuentran activos en el momento. El menú tiene las correspondientes validaciones que no permiten modificar, borrar o ver la lista si al menos no se cargó un chofer. A la hora de dar de alta un chofer se asigna automáticamente un id, Se toma el nombre del chofer, y se asigna un sueldo. Las horas trabajadas se inicializan en 0 y se verán afectadas a medida que se realicen pedidos de recolección. No obstante, ante cualquier carga de datos errónea, se permite desde la modificación del chofer cambiar el valor de las horas trabajadas.

Detalle de la opción 14 – LISTADOS

```
void Listados(ePedido* pedido,int tam_Pedido, eLocalidad* localidad, int tam_Localidad, eCliente* cliente, int tam_CLiente, eChofer* chofer, int tam_Choferes) {

    int retorno = 0;
    int opcion;

    do {
        system("cls");
        printf("\n**************************************** L I S T A S D O S   Y   B O N O S  ****************************************\n\n");

        printf(" 1 - ORDENAMIENTO POR ID DE CHOFERES \n 2 - VIAJES REALIZADOS POR CADA CHOFER\n");
        printf(" 3 - LOCALIDADES RECORRIDAS POR CADA CHOFER\n 4 - CHOFERES CON MAS VIAJES \n 5 - DAR BONO POR 3 VIAJES REALIZADOS\n 0 - VOLVER\n ");
        opcion = LoadInt("OPCION = ",0,5);

        switch (opcion) {
            case 1:
                retorno = OrdenamientoPorId(chofer, tam_Choferes);
                AlertMessageAndEnter(retorno, "", "");
                break;
            case 2:
                retorno = ViajesRealizadosPorCadaChofer(pedido, tam_Pedido, cliente, tam_CLiente, chofer, tam_Choferes);
                AlertMessageAndEnter(retorno, "", "Los choferes no tienen horas trabajadas.");
                break;
            case 3:
                retorno = LocalidadesRecorridasPorCadaChofer(pedido, tam_Pedido, localidad, tam_Localidad, cliente, tam_CLiente, chofer, tam_Choferes);
                AlertMessageAndEnter(retorno, "", "");
                break;
            case 4:
                retorno = ChoferConMasViajes(pedido, tam_Pedido, cliente, tam_CLiente, chofer, tam_Choferes);
                AlertMessageAndEnter(retorno, "", "");
                break;
            case 5:
                retorno = BonoPorViajes(pedido, tam_Pedido, cliente, tam_CLiente, chofer, tam_Choferes);
                AlertMessageAndEnter(retorno, "Se realizaron modificaciones en los salarios", "No hay choferes a los que bonificar.");
                break;


        }

        if(opcion != 0) {
            opcion = -1;
            retorno = 0;

        }
    } while(opcion != 0);

}
```

Esta opción también abre un nuevo submenú el cual nos da la opción de ver 5 nuevos filtros desarrollados para el recuperatorio. Estos filtros manipulan la mayoría de las estructuras:

1- **Ordenamiento por ID:**
Este filtro es simple, solo realiza un ordenamiento del ID de los choferes. Sirve para ver las modificaciones que se realizan luego de ejecutar la opción 5.

2- **Viajes realizados por cada chofer**:
En este filtro veremos una relación entre el cliente, el pedido y el chofer. Se muestra un chofer y en los siguientes renglones, todos los viajes que este realizó. Solo se muestra al chofer cuando este tiene en el campo “horasTrabajadas” un número mayor a 0. Cuando se encuentra un chofer activo, se muestra el nombre de este. A continuación, recorremos la lista de clientes, si se encuentra un cliente activo, recorremos la lista de pedidos y en caso que encontremos un pedido activo con el mismo id del cliente y el mismo id del chofer lo imprimimos. Se recorre toda la lista de choferes.

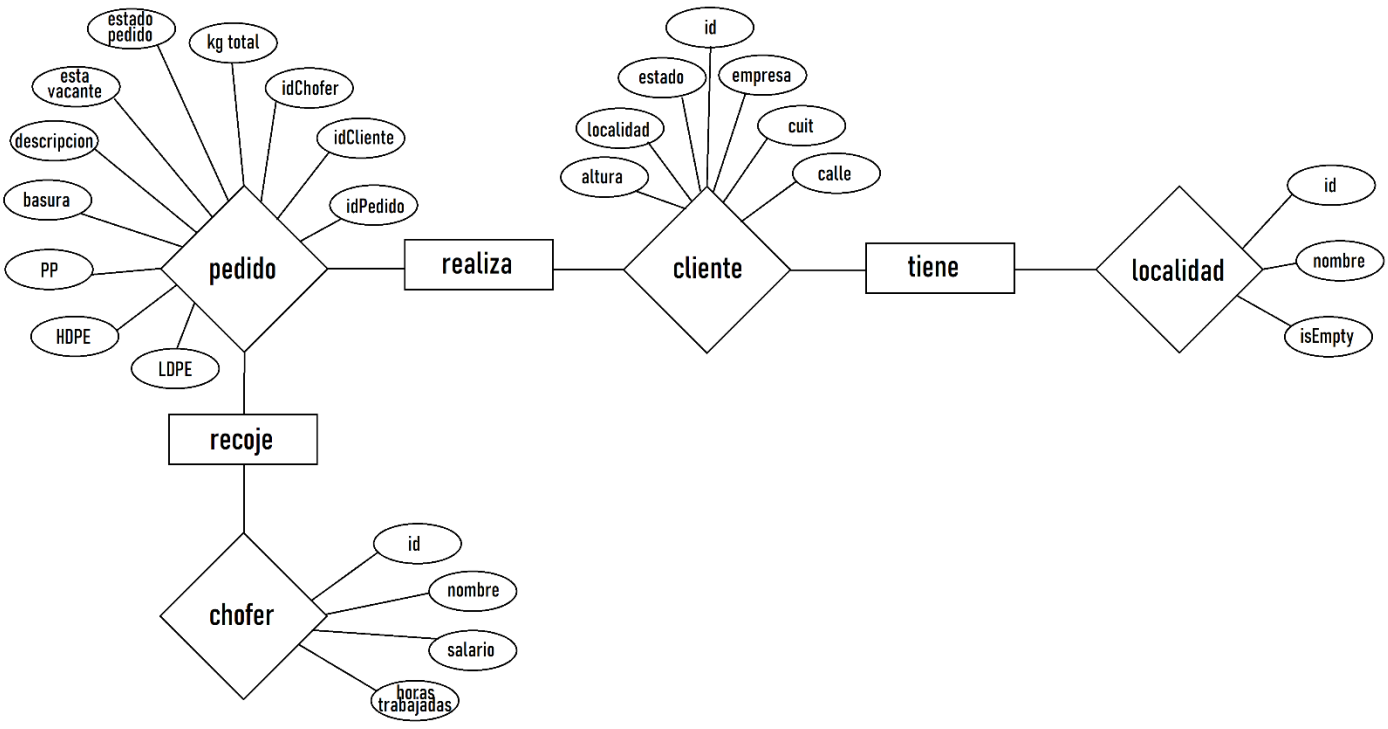
3- **Localidades recorridas por cada chofer**:
En este filtro veremos una relación entre el cliente, el pedido, el chofer y las localidades.
Se muestra un chofer y a continuación se muestra el recorrido de localidades en el orden que las recorrió. Si el chofer hizo dos pedidos de recolección consecutivos en la misma localidad, esta se muestra solo una vez. Cuando se encuentre un chofer activo, con horas trabajadas se imprime un chofer, recorremos la lista de clientes, si encontramos un cliente activo recorremos la lista de pedidos y si encontramos un pedido activo, recorremos la lista de localidades. A continuación, si coinciden el id del pedido con el del cliente, el id del chofer con el del pedido, la localidad del cliente con la localidad y la localidad auxiliar no es la misma que la anterior, se imprime la localidad

donde realizo ese pedido de recolección y se guarda esa última localidad auxiliar para compararla nuevamente con la próxima y que no se repitan. Se recorre toda la lista de choferes.

4- **Choferes con más viajes:**
En esta función se muestra el chofer o los choferes que tengan más viajes realizados. Para ello primero recorro las listas con la misma condición que los anteriores puntos, guardando en un acumulador la cantidad de viajes que realizo cada chofer y filtrándolo comparándolo con un máximo. Cuando obtengo ese máximo, recorro otro bucle con las mismas condiciones y guardando nuevamente un acumulador. Cuando ese acumulador es igual al máximo imprimo al chofer con sus datos y la cantidad de viajes.

5- **Bono Por Viajes:**
Bonifico a los choferes que tengan más de 3 viajes realizados. Es un número bajo que se acota a la cantidad de pedidos que se definen por extensión. Sigo un recorrido similar a los anteriores y en caso que se encuentren 3 viajes o más hechos por cada chofer, a estos se los bonifica aumentándoles el sueldo 1 vez con un 15%. Para esto use el campo de la función eChoferes al que llame .descuento. Cuando se realiza la bonificación estos ya no podrán tener otra.

Diagrama de entidad- Relación



El examen ha sido modificado y testeado. Todo funciona de manera correcta.

Documentación de todas las funciones del programa

(También disponible en los archivos de eclipse)

CONTROLLER.H

```
//funciones creadas en esta etapa del desarrollo del programa para facilitar la lectura. También facilita la ejecución de otras funciones
// Si bien en el main no se utiliza su retorno, estas lo tienen en caso de reutilizar el código en otro programa más grande y que si sean necesarios

/*
 * Controlador de la función de alta de cliente
 * Valida que las listas no sean NULL
 * Parámetros: ListaDeClientes, array de clientes, tam_Clientes, tamaño de la lista de clientes, ListaDeLocalidades, array de localidades, tam_Localidades, tamaño de la lista de localidades
 * Id_Automatico, id que se le asignara al nuevo cliente.
 *retorna -1 si la función alta de cliente no pudo dar de alta un cliente o no se cumplieron las condiciones. Retorna 0 si pudo
 */
int Controller_AltaCliente(eCliente* ListaDeClientes, int tam_Clientes,eLocalidad* ListaDeLocalidades, int tam_Localidades, int* Id_Automatico);

/*
 * Controlador de la función de modificación de cliente
 * Valida que las listas no sean NULL y que al menos haya un cliente cargado
 * Parámetros: ListaDeClientes, array de clientes, tam_Clientes, tamaño de la lista de clientes, ListaDeLocalidades, array de localidades, tam_Localidades, tamaño de la lista de localidades
 *
 *retorna -1 si la función modificación de cliente no pudo modificar un cliente o no se cumplieron las condiciones. Retorna 0 si pudo
 */
int Controller_ModificacionCliente(eCliente* ListaDeClientes, int tam_Clientes,eLocalidad* ListaDeLocalidades, int tam_Localidades);

/*
 * Controlador de la función de baja de cliente
 * Valida que las listas no sean NULL y que al menos haya un cliente cargado en la lista
 * Parámetros: ListaDeClientes, array de clientes, tam_Clientes, tamaño de la lista de clientes, ListaDeLocalidades, array de localidades, tam_Localidades, tamaño de la lista de localidades
 *
 *retorna -1 si la función alta de cliente no pudo dar de baja un cliente o no se cumplieron las condiciones. Retorna 0 si pudo
 */
int Controller_BajaDeCliente(eCliente* ListaDeClientes, int tam_Clientes, eLocalidad* ListaDeLocalidades, int tam_localidades);

/*
 * Controlador de la función crear pedido
 * Valida que las listas no sean NULL y que exista al menos un cliente para asignarle un pedido
 * Parámetros: ListaDePedidos, array de pedidos, tamanio_Pedidos, tamaño del array pedidos, ListaDeClientes, array de clientes, tam_Clientes,tamaño de la lista de clientes,
 *                      ListaDeLocalidades, array de localidades, tam_Localidades,tamaño de la lista de localidades, ListaDeChoferes array de choferes,tam_Choferes, tamaño del array de choferes
 *                       idPedido, unico id asignado automáticamente
 * retorna -1 si la función no pudo cargar el pedido o no se cumplieron las condiciones. Retorna 0 si pudo
```

```
 */
int Controller_CrearPedido(ePedido* ListaDePedidos, int tamanio_Pedidos, eCliente*
ListaDeClientes, int tam_Clientes,eLocalidad* ListaDeLocalidades,
             int tam_localidades,eChofer* ListaDeChoferes, int tam_Choferes, int* idPedido );

/*
 * Controlador de la funcion procesar residuos
 * Valida que las listas no sean NULL, que exista al menos un cliente y que exista al menos un
pedido
 * Parámetros: listaDePedidos, array de pedidos, tam_Pedidos, tamaño del array de pedidos,
listaDeClientes, array de clinetes, tam_Clientes, tamaño del array de clientes
 *
 * retorna -1 si la funcion no pudo procesar un pedido o no cumplio con las condiciones.
Retorna 0 si se procesó un pedido
 */
int Controller_ProcesarResiduos(ePedido* listaDePedidos, int tam_Pedidos,eCliente*
listaDeClientes, int tam_Clientes);

/*
 * Controlador de la funcion imprimir los clientes con pendientes
 * Valida que las listas no sean NULL y que exista al menos un cliente y al menos un pedido
pendiente
 * Parámetros: ListaDeClientes, array de clientes, tam_Clientes, tamaño del array de clintes,
ListaDePedidos, array de pedidos ,
 *                        tam_Pedidos, tamaño del array de pedidos, ListaDeLocalidades, array
de localidades, tam_Localidades tamaño del array de localidades
 * retorna -1 si la funcion no pudo imprimir los clientes con pendientes o si no cumplió con
las condiciones. Retorna 0 si pudo
 */

int Controller_ImprimirLosClientesConPendientes(eCliente* ListaDeClientes, int
tam_Clientes,ePedido* ListaDePedidos, int tam_Pedidos, eLocalidad* ListaDeLocalidades, int
tam_Localidades);

/*
 * Controlador de la funcion imprimir pedidos pendientes
 * Valida que las listas no sean null y que haya pedidos pendientes
 * Parámetros: listaDePedidos, array de pedidos, tam_Pedidos, tamaño del array de pedidos,
listaDeClientes, array de clientes, tam_Clientes tamaño del array de clientes
 *
 * retorna -1 si la funcion no pudo imprimir los pedidos pendientes o si no cumplió con las
condiciones. Retorna 0 si imprimió la lista
 */
int Controller_ImprimirPendientes(ePedido* listaDePedidos, int tam_Pedidos, eCliente*
listaDeClientes, int tam_Clientes);

/*
 * Controlador de la función imprimir pedidos procesados
 * Valida que las listas no sean null y que haya pedidos procesados
 * Parámetros: listaDePedidos, array de pedidos, tam_Pedidos, tamaño del array de pedidos,
listaDeClientes, array de clientes, tam_Clientes tamaño del array de clientes
 *
 * retorna -1 si la funcion no pudo imprimir los pedidos procesados o si no cumplió con las
condiciones. Retorna 0 si imprimió la lista
 */
int Controller_ImprimirProcesados(ePedido* listaDePedidos, int tam_Pedidos, eCliente*
listaDeClientes, int tam_Clientes);
```

```c
/*
 * Controlador de la función imprimir pendientes por localidad
 * Valida que las listas no sean NULL, que haya pedidos pendientes y que exista al menos un cliente
 * Parámetros: listaDePedidos, array de pedidos, tam_Pedidos, tamaño del array de pedidos, listaDeClientes, array de clientes, tam_Clientes tamaño del array de clientes
 *                      ListaDeLocalidades, array de localidades, tam_Localidades tamaño del array de localidades
 * retorna -1 si no encontró pedidos para dicha localidad o no cumplió con las condiciones. Retorna 0 si encontró al menos un pedido para dicha localidad
 */
int Controller_ImprimirPedidosPendientesPorLocalidad(ePedido* listaDePedidos, int tam_Pedidos, eCliente* ListaDeClientes, int tam_Clientes, eLocalidad* ListaDeLocalidades, int tam_Localidades);

/*
 * Controlador creado para ejecutar más de un promedio. Para el examen se pidió solo uno
 * Valida que las listas no sean null y que las listas estén cargadas
 * Parámetros: ListaDeClientes, array de clientes, tam_Clientes, tamaño de la lista de clientes, listaDePedidos, array de pedidos,tam_Pedidos tamaño de la lista de pedidos
 * retorna -1 si no encontro kg de polipropileno en los pedidos o si no cumplió con las condiciones. Retorna 0 en caso de encontrar kilos de polipropileno
 */
int Controller_Promedios(eCliente* ListaDeClientes, int tam_Clientes, ePedido* listaDePedidos, int tam_Pedidos);

/*
 *controlador de la funcion mas pedidos pendientes
 * Valida que la lista este cargada y que existan pedidos pendientes
 *  Parámetros: listaDePedidos array de pedidos, tam_Pedidos tamaño de la lista de pedidos, ListaDeClientes, array de clientes, tam_Clientes, tamaño de la lista de clientes
 *                  ListaDeLocalidades, array de localidades, tam_Localidades, tamaño de la lista de localidades
 * retorna -1 si no cumplió con las condiciones. Retorna 0 si pudo ejecutar la funcion
 */
int Controller_MasPedidosPendientes(ePedido* listaDePedidos, int tam_Pedidos, eCliente* ListaDeClientes, int tam_Clientes, eLocalidad* ListaDeLocalidades, int tam_Localidades);

/*
 *controlador de la funcion más pedidos procesados
 * Valida que la lista este cargada y que existan pedidos procesados
 *  Parámetros: listaDePedidos array de pedidos, tam_Pedidos tamaño de la lista de pedidos, ListaDeClientes, array de clientes, tam_Clientes, tamaño de la lista de clientes
 *                  ListaDeLocalidades, array de localidades, tam_Localidades, tamaño de la lista de localidades
 * retorna -1 si no cumplió con las condiciones. Retorna 0 si pudo ejecutar la funcion
 */
int Controller_MasPedidosProcesados(ePedido* listaDePedidos, int tam_Pedidos, eCliente* ListaDeClientes, int tam_Clientes, eLocalidad* ListaDeLocalidades, int tam_Localidades);

/*
 * Controlador de la funcion manejo de choferes
 * verifica que la lista de choferes no sea NULL
 * Parámetros: listaDeChoferes, array de choferes, tam_Choferes, tamaño del array de choferes, *idChofer, id único del chofer
 *
 * retorna -1 si no cumplió con las condiciones, retorna 0 si pudo ejecutar la funcion
 */
int Controller_ManejoDeChoferes(eChofer* listaDeChoferes, int tam_Choferes, int* idChofer);
```

```
/*
 *controlador de la funcion listados
 * verifica que las listas no sean null y esten todas cargadas
 * Parámetros: pedido, array de pedidos, tam_pedido, tamaño del array de pedidos, localidad,
array de localidad, tam_localidad, tamaño del array de localidad,
 *                        Cliente, array de cliente, tam_cliente tamaño del array de clintes,
chofer array de choferes, tam_choferes, tamaño del array de choferes
 *retorna -1 si no cumplió con las condiciones, retorna 0 si pudo ejecutar la funcion
 */
int Controller_Listados(ePedido* pedido,int tam_Pedido, eLocalidad* localidad, int
tam_Localidad, eCliente* cliente, int tam_CLiente, eChofer* chofer, int tam_Choferes);
```

ARRAYEMPLOYEES.H

```
/*
 * inicializa la lista de pedidos poniendo el campo estado en 0 para dar lugar a la carga de
clientes
 * Parámetros: cliente, array de clientes, tamaniLista tamaño del array de clientes
 *
 *
 */
void InicializarLista(eCliente* cliente, int tamanioLista);

/*
 * Verifica que exista al menos un campo de estado que este en 1 o ACTIVO
 * Parámetros: cliente, array de clientes, tamaniLista tamaño del array de clientes
 *
 * retorna -1 si todos los campos están en 0 o 0 si encontró al menos un cliente ACTIVO
 */
int ValidarListaCargada(eCliente* lista, int tamanioLista);

/*
 * Da de alta un cliente y le asigna una localidad especificada por el usuario y un id
automático
 * Parámetros: Listacliente, array de clientes, tamanioLista, tamaño del array de clientes,
localidad, array de localidades, tamLocalidad, tamaño del array de localidades,*idAutomatico,
id único que se le asigna al cliente
 *
 * retorna: -1 si no encontró un espacio en la lista para dar de alta al cliente o 0 si pudo
hacerlo
 */
int AltaDeCliente(eCliente* cliente, int tamanioLista, eLocalidad* localidad, int tamLocalidad,
int *idAutomatico);

/*
 * Parámetros: Listacliente, array de clientes, tamanioLista,tamaño del array de clientes,
localidad, array de localidades, tamLocalidad, tamaño del array de localidades,
 *           Parámetro, para mostrar un tipo de cabecera
 *
 */
void ImprimirListaDeClientes(eCliente* ListaCliente, int tamanioLista, eLocalidad*
ListaLocalidades, int tam_localidades, int parametro);

/*
 * imprime un cliente en la posición especificada del array
 * Parámetros: cliente, cliente en posición especificada, localidad, localidad en posición
especificada
 *
 */
void ImprimirCliente(eCliente cliente, eLocalidad localidad);
```

```c
/*
 * busca un cliente por el id que ingrese el usuario
 * Parámetros: cliente, array de clientes, tamanioLista tamaño de la lista de clientes,
localidad, array de localidad, tam_localidad, tamaño del array de localidades, id, id que se
busca
 *
 * Esta función retorna -1 si no encontró el id que se especifica o 0 si lo encontró
 */
int BuscarYMostrarClientePorId(eCliente* cliente, int tamanioLista, eLocalidad* localidad, int
tam_localidad, int id);

/*
 * Da de baja un cliente permanentemente
 * utiliza la función buscar y mostrar cliente por id, de encontrarlo ejecuta la funcion borrar
cliente
 * Parámetros: cliente, array de clientes, tamanioLista tamaño de la lista de clientes,
localidad, array de localidad, tam_localidad, tamaño del array de localidades
 * retorna -1 si no pudo encontrar el usuario que se especificó borrar o si se canceló la
operación. Retorna 0 si pudo dar de baja al usuario
 *
 */
int BajaDeCliente(eCliente* cliente, int tamanioLista, eLocalidad* localidad, int
tam_localidad);

/*
 * Elimina el usuario poniendo en 0 el campo de estado
 * Parámetros cliente array de clientes, tamanioLista tamaño del array de clientes, id, id a
borrar
 *
 */
void BorrarCliente(eCliente* cliente,int tamanioLista, int id);

/*
 * modifica un cliente permanentemente
 * utiliza la función buscar y mostrar cliente por id, de encontrarlo ejecuta la funcion
modificar cliente
 * Parámetros: cliente, array de clientes, tamanioLista tamaño de la lista de clientes,
localidad, array de localidad, tamLocalidad, tamaño del array de localidades
 * retorna -1 si no se encontró el usuario o si no se realizaron modificaciones, retorna 0 en
caso de realizar la operación
 *
 */
int ModificarCliente(eCliente* cliente,int tamanioLista,eLocalidad* localidad, int
tamLocalidad);

/*
 * permite modificar los campos al usuario permitidos por el desarrollador, mostrando un
submenú
 * Parámetros: cliente, array de clientes, tamanioLista tamaño del array de clientes, localidad
array de localidades, tamanioLocalidad, tamaño del array de localidades, idaModificar, id del
cliente que se le realizaron modificaciones
 * retorna -1 si no se realizan modificaciones en el cliente o 0 si se realizado alguna
modificacion
 *
 */
int CambiarDatosDeCliente(eCliente* cliente,int tamanioLista,eLocalidad* localidad, int
tamLocalidad,int idAModificar);

/*
 * muestra dos tipos de cabeceras parecidas, salvo que en una imprime también los pedidos
pendientes
 * Parámetro: caso, elije el tipo de cabecera que se quiere mostrar
 *
 */
void MostrarCabezalClientes(int caso);
```

ARRAYLOCALIDADES.H

```
/* Imprime la lista de localidades
 * Parámetros: localidad, estructura de localidades, tam localidad, tamaño de la estructura de localidades
 *  retorna 0 en caso de no contar ninguna localidad  o retorna la cantidad de localidades que encontró activas
 *
 */
int ImprimirListaDeLocalidades(eLocalidad* localidad, int tamLocaldiad);

/*
 * toma una localidad ingresada por el usuario como un entero
 * Parámetros: localidad, estructura de localidades, tam localidad, tamaño de la estructura de localidades
 * retorna -1 por error o el numero ingresado como localidad
 */
int TomarLocalidad(eLocalidad* localidad, int tamLocaldiad);

/*
 * Carga un archivo .csv desde la carpeta del programa
 * Parámetros: localidad, estructura de localidades, tam localidad, tamaño de la estructura de localidades
 * retorna: -1 si no pudo abrir el archivo y 0 si imprimió aunque sea una localidad
 */
int cargaDeCiudades(char* path, eLocalidad* Localidades, int tam_localidades);
```

ARRAYPEDIDOS.H

```
//funciones de PEDIDOS
/*
 * Inicializa la lista de pedidos como si todos estuvieran vacíos poniendo el estado de estaVacante en 0
 * Parámetros: pedidos, estructura de pedidos, tam_pedidos tamaño del array de pedidos
 *
 *
 */
void InicializarListaPedidos(ePedido* pedidos, int tamanio_Pedidos);

/*
 * Se fija si al menos un pedido está procesado
 * Parámetros: pedidos, estructura de pedidos, tam_pedidos tamaño del array de pedidos
 * retorna: -1 si todos los pedidos están pendientes o 0 si algún pedido está procesado

 */
int validarPedidosProcesados(eCliente* cliente, int tam_Clientes,ePedido* pedido, int tam_Pedidos);

/*
 * Se fija si al menos un pedido está pendiente
 * Parámetros: pedidos, estructura de pedidos, tam_pedidos tamaño del array de pedidos
 * retorna: -1 si todos los pedidos están procesados o 0 si algún pedido está pendiente

 */
int validarPedidosPendientes(eCliente* cliente, int tam_Clientes,ePedido* pedido, int tam_Pedidos);
```

```
/*
 * Se fija si al menos un pedido fue dado de alta
 * Parámetros: pedidos, estructura de pedidos, tam_pedidos tamaño del array de pedidos
 * retorna: -1 si todos los pedidos están vacantes o 0 si algún pedido fue dado de alta
 *
 */
int ValidarListaPedidos(eCliente* cliente, int tamanio_Cliente,ePedido* pedido, int
tam_Pedido);

/*
 * Crea un pedido de recolección el cual se encontrara como pendiente en la lista de pedidos y
lo enlaza con un cliente y un chofer
 * Parámetros: pedido, estructura de pedidos, tamanio_pedidos tamaño del array de pedidos,
cliente, estructura de cliente  y tamanio_clientes con el tamaño del array de clientes
 *                        lista de localidades, array de localidades, tam_localidades tamaño
del array de localidades, choferes array de choferes,
 *                         tam_choferes tamaño del array de choferes, idPedido, id del pedido
 *
 * retorna -1 si alta de pedido no pudo crear un pedido o 0 si pudo cargarlo correctamente
 */
int CrearPedidoDeRecoleccion(ePedido* pedido,int tamanio_Pedidos, eCliente* listaDeClientes,
int tamanio_Cliente,
             eLocalidad* listaDeLocalidades, int tam_localidades,eChofer* listaDeChoferes, int
tam_choferes, int* idPedido);

/*
 *     da de alta un pedido y carga los datos a los arrays
 *      Parámetros: pedido, estructura de pedidos, tamanio_pedidos tamaño del array de pedidos,
      cliente, estructura de cliente  y tamanio_clientes con el tamaño del array de clientes
 *                        lista de localidades, array de localidades, tam_localidades tamaño
del array de localidades, choferes array de choferes, tam_choferes tamaño del array de choferes
 *
 *retorna -1 si no encontro un espacio en el array para cargar un pedido o 0 si cargo el pedido
correctamente
 */
int AltaPedido(ePedido* pedido, int tam_Pedidos, eCliente* listaDeClientes,int
tam_Clientes,eLocalidad* localidad, int tam_localidad,eChofer* choferes, int tam_choferes, int
idPedido);

/*
 * procesa los residuos, convierte el total de pedidos en la cantidad especificada por el
usuario de los tipos de plásticos dados por el parcial
 * Parámetros: listaDepedidos, estructura de pedidos, tam_pedidos tamaño del array de pedidos,
listaDeClientes, array de clientes, tam_clientes tamaño del array de clientes
 *
 * retorna -1 si no pudo procesar el pedido dado que no se encontro el id del pedido
especificado o 0 si se proceso correctamente
 */
int ProcesarResiduos(ePedido*listaDePedidos,int tam_Pedidos,eCliente* listaDeClientes, int
tam_Clientes);

/*
 * muestra una lista con todos los pedios pendientes
 * Parámetros: listaDepedidos, estructura de pedidos, tam_pedidos tamaño del array de pedidos,
listaDeClientes, array de clientes, tam_clientes tamaño del array de clientes
 *
 * retorna -1 si no pudo imprimir ningún pedido pendiente o 0 si imprimió al menos un pedido
 */
int MostrarListaDePedidosPendientes(ePedido* pedidos,int tam_Pedidos, eCliente* clientes,int
tam_Clientes);
```

```c
/*
 * imprime un pedido especificado en una posición del array
 *  Paramentos: pedido,  array del pedido en esa posición, cliente, array de cliente en esa posición, caso, en caso de ser pedido pendiente o procesado (0 y 1)
 *
 */
void ImprimirPedido(ePedido pedido, eCliente cliente, int caso);

/*
 * muestra la cabecera del pedido que se le especifique, procesado o pendiente (0 y 1)
 * Parámetro: cabezal, elije el tipo de cabecera que se le especifique
 *
 *
 */
void MostrarCabezal(int cabezal);

/*
 * busca y muestra un pedido especificado por el usuario
 * Parámetros: pedido array de pedidos, tam_pedidos tamaño del array de pedidos, cliente array de clientes, tam_clientes tamaño del array de clientes, id id especificado por el usuario
 * retorna -1 si no encontró el pedido ingresado o 0 si logro mostrarlo
 *
 */
int BuscarYMostrarPedidoPorId(ePedido* pedido, int tam_Pedidos,eCliente* cliente, int tam_Clientes, int id);

/*
 * Sub función que permite ingresar los kilos de cada tipo de plástico
 * Parámetros: pedido array de pedidos, tam_pedidos tamaño del array de pedidos, idIngresado, id ingresado previamente por el usuario
 *
 * retorna -1 si no pudo cargar los datos o 0 si concluyo l carga de datos de pedidos procesados
 */
int IngresarKilos(ePedido* pedido, int tam_Pedidos,int idIngresado);

/*
 * muestra una lista con todos los pedios procesados
 * Parámetros: pedido, estructura de pedidos, tam_pedidos tamaño del array de pedidos, listaDeClientes, array de clientes, tam_clientes tamaño del array de clientes
 *
 * retorna -1 si no pudo imprimir ningún pedido procesado o 0 si imprimió al menos un pedido
 *
 */
int MostrarListaDePedidosProcesados(ePedido* pedido,int tam_Pedidos, eCliente* cliente, int tam_Clientes);

//funciones mixtas

/*
 * muestra una lista con los pedidos pendientes que tiene cada cliente
 * Parámetros: pedido array de pedidos, tam_pedidos tamaño del array de pedidos, cliente array de clientes, tam_clientes tamaño del array de clientes,
 *                      localidad, array de localidad, tam_localidad tamaño del array de localidad.
 *
 * retorna -1 si no hay nada para mostrar en la lista o 0 si imprimió la lista
 * nota: si los clientes no tienen pedidos pendientes en el campo de pendientes se marca un 0. pero si lo imrpime
 */
int ListaDeClientesConPendinetes(eCliente* cliente, int tam_Clientes, ePedido* pedido,int tam_Pedidos, eLocalidad * localidad, int tam_localidad);
```

```
/*
 * compara la cadena de localidad ingresada por el usuario con la cadena guardada en la descripción de las localidades
 * de encontrar una localidad, muestra los pedidos que hay en dicha localidad con la cantidad que encontró
 * parametros: cliente array de clientes, tam_clientes tamaño del array de clientes, pedido array de pedidos, tam_pedidos tamaño del array de pedidos,
 *                        localidad, array de localidad, tam_localidad tamaño del array de localidad.
 * retorna -1 si no encontró la cadena que compara con las localidades, o 0 si la encontró
 */
int MostrarPedidosPorLocalidad(eCliente* cliente, int tam_Clientes, ePedido* pedido, int tam_Pedidos, eLocalidad * localidades, int tam_localidades);

/*
 * realiza un promedio de la cantidad de pp que se encontro en el array de pedidos con la cantidad total de clientes
 * parametros: cliente array de clientes, tam_clientes tamaño del array de clientes, pedido array de pedidos, tam_pedidos tamaño del array de pedidos
 * retorna -1 si no encontro registros de pp o 0 si pudo realizar el promedio
 */
int CantidadPoliPropilenoPromedioCliente(eCliente* cliente,int tam_Clientes, ePedido* pedido, int tam_Pedidos);

//funciones segunda parte parcial

/*
 *imprime los clientes que tengan la mayor cantidad de pedidos pendientes
 *parametros: cliente array de clientes, tam_clientes tamaño del array de clientes, pedido array de pedidos, tam_pedidos tamaño del array de pedidos,
 *                        Localidad, array de localidad, tam_localidad tamaño del array de localidad.
 *retorna -1 si no encontro ningun cliente con pedidos pendientes o 0 si encontro al menos uno
 */
int ClientesConMasPedidosPendientes(eCliente* cliente, int tam_Clientes, ePedido* pedido, int tam_Pedidos,eLocalidad* localidades, int tam_Localidades);

/*
 *imprime los clientes que tengan la mayor cantidad de pedidos procesados
 *parámetros: cliente array de clientes, tam_clientes tamaño del array de clientes, pedido array de pedidos, tam_pedidos tamaño del array de pedidos,
 *                        Localidad, array de localidad, tam_localidad tamaño del array de localidad.
 *retorna -1 si no encontró ningún cliente con pedidos procesados o 0 si encontró al menos uno
 */
int ClientesConMasPedidosProcesados(eCliente* cliente, int tam_Clientes, ePedido* pedido, int tam_Pedidos,eLocalidad* localidades, int tam_Localidades);
```

ARRAYCHOFERES.H

```
/* Inicializa la lista de choferes poniendo su campo is empty en 0.
 *
 *  Parámetros: eChoferes estructura de choferes, tam_ choferes tamaño del array.
 *
 * retorna -1 si la lista es NULL o 0 si la lista se inicializo como corresponde.
 *
*/
int InicializarListaDeChoferes( eChofer* choferes, int tam_Choferes);
```

```c
/* Valida que la lista tenga al menos un dato cargado.
 *
 * Parámetros: eChoferes estructura de choferes, tam_ choferes tamaño del array.
 *
 * retorna -1 si la lista es NULL o si no tiene ningún elemento cargado.
 * retorna  0 si encontró algún campo de is empty en 1.
 */
int ValidarListaChoferes(eChofer* choferes, int tam_Choferes);

/* Muestra un sub menu en pantalla
 *
 * Parámetros: eChoferes estructura de choferes, tam_ choferes tamaño del array, *idChofer para
aumentar el id del chofer en otra función
 *
 */
void SubMenuChoferes(eChofer* choferes, int tam_Choferes,int *idChofer);

/* Da de alta un chofer
 * Parámetros: eChoferes estructura de choferes, tam_ choferes tamaño del array, *idChofer
 *  Puntero al id del menú para aumentar el id del chofer
 *
 * retorna -1  si no pudo cargar un chofer a la lista ya que no hay campos disponibles.
 * retorna  0 si encontró un espacio y cargo un chofer.
 */
int AltaDeChofer(eChofer* choferes, int tam_Choferes, int* idChofer);


/* Da de baja un chofer
 * Parámetros: eChoferes estructura de choferes, tam_ choferes tamaño del array
 *  retorna -1 si no encuentra el id ingresado o si se da de baja la operación
 *  retorna 0 si borra al chofer
 *
 */
int BajaDeChofer(eChofer* choferes, int tam_Choferes);
/*
 * Modifica datos de un chofer
 * Parámetros: eChoferes estructura de choferes, tam_ choferes tamaño del array
 * retorna -1 si no encontró el id ingresado  o si no se modifico el chofer
 * retorna 0 si modifico al chofer
 *
 */
int ModificacionDeChofer(eChofer* choferes, int tam_Choferes);

/* Imprime la lista de choferes
 * Parámetros: eChoferes estructura de choferes, tam_ choferes tamaño del array
 *
 */
void ImprimirListaDeChoferes(eChofer* choferes, int tam_Choferes);
/*Imprime un solo chofer
 * Se utiliza dentro del Bucle de la función anterior
 * Parámetro: eChofer en posición del array indicado
 *
 */
void imprimirUnChofer(eChofer chofer);


/*
 * Imprime cabecera de choferes con los datos que componen a la estructura
 * Para facilitar la lectura
 *
 */
void imprimirCabeceraChoferes(void);
```

Observación realizada para posible modificación de la función de contadores:

Luego de re-pensar el código encontré otra forma de no escribir tantas líneas de código para las funciones: **ClientesConMasPedidosProcesados, ClientesConMasPedidosPendientes y MostrarChoferesConMasViajes**

Y es la siguiente:

```
for(int s = 0; s < 2; s++) {

	if(s == 1 && retorno == 0) {

		printf("ID	CHOFER	VIAJES TOTALES	HORAS TRABAJADAS	SALARIO \n");
	}

	for(int c = 0; c < tam_Choferes; c++) {

		if(choferes[c].isEmpty == ACTIVO && choferes[c].horasTrabajadas > 0) {

			for(int i = 0; i < tam_CLiente; i++) {

				if(cliente[i].estado == ACTIVO) {

					for(int j = 0; j < tam_Pedido; j++) {

						if(pedido[j].estaVacante == ACTIVO && cliente[i].id ==
						pedido[j].idCliente && pedido[j].iDdelChofer == choferes[c].id) {

								contador ++;
						}
					}
				}
			}

			if(s == 0 && (flag || contador > maximo)) {

				maximo = contador;
				flag = 0;
				retorno = 0;
			}
			if(s == 1 && maximo == contador && retorno == 0) {

				printf("%4d %20s %15d %20s %6d %30s %5d\n",cliente[i].id, cliente[i].empresa,
				cliente[i].cuit,cliente[i].calle,cliente[i].altura, localidades[l].descripcion,
				contadorProcesados);
			}

			contador = 0;

		}
	}
}
```

Y aunque esta función ahorre líneas, me siento más cómodo con la anterior, me resulta más fácil de leer y a nivel iteraciones son las mismas. Claro está que el desarrollo de esta función es más práctico pero menos visible. Voy a tenerlo en cuenta para próximos desarrollos.

FIN DE LA EXPOSICIÓN DEL RECUPERATORIO