

RECUPERATORIO PARCIAL 1

MATERIA: LABORATORIO 1

DIVISION: 1F

ALUMNO: ALFREDO GONZALEZ

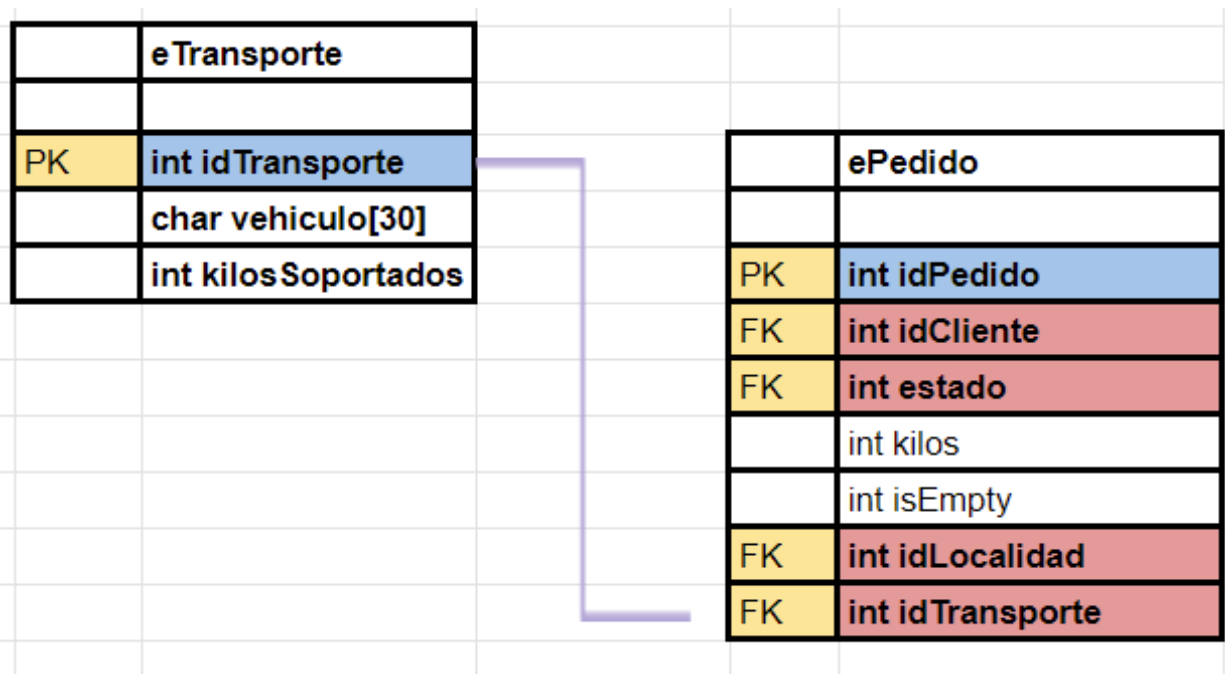
PROFESOR: GERMÁN SCARAFILO

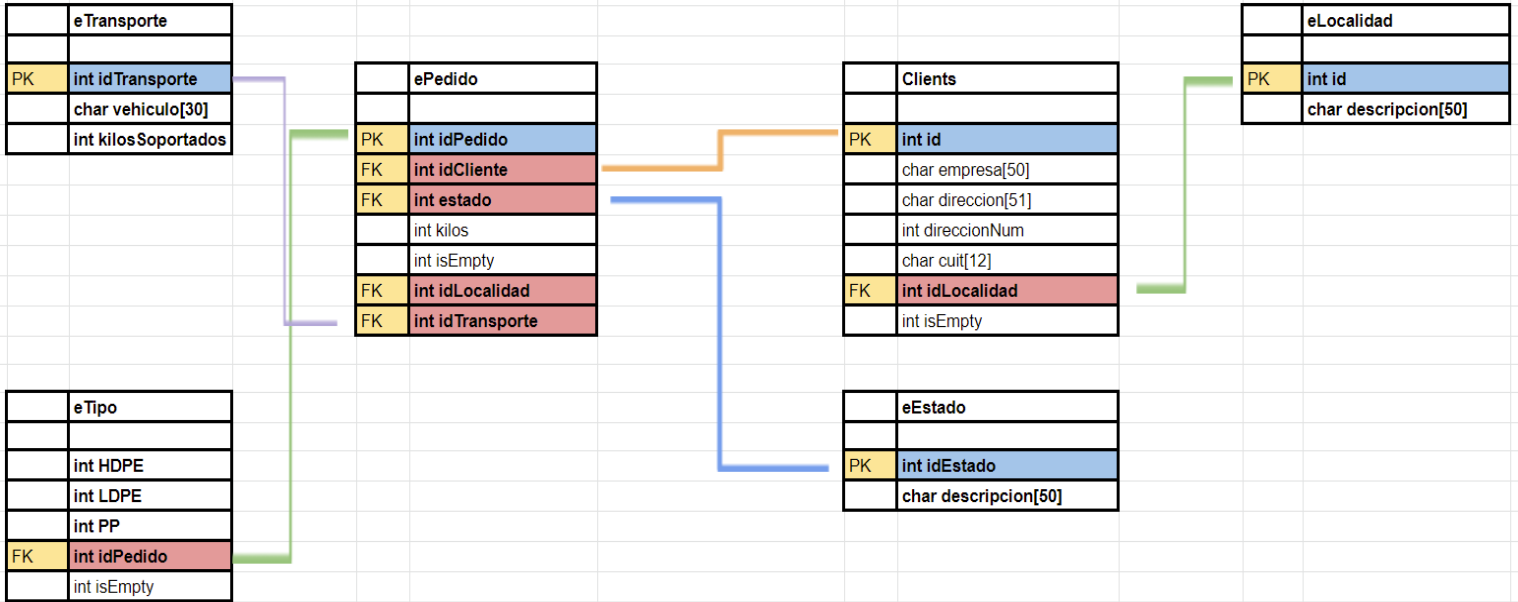
27/11/2021

3.B)

La estructura agregada se trata de eTransporte, al cual va darle al cliente 3 opciones de transporte según la cantidad de kilos del pedido, la misma va a contar con un idTransporte para cada uno de los 3 transportes, la descripción de los mismos que en este caso se llama vehículo y la cantidad de kilos que soportan según el vehículo.

El idTransporte va a ser pedido al usuario al momento de hacer la carga de un pedido, la estructura ePedido va tener su propio idTransporte como una Foreign Key, y esa variable va a ser comparada mediante una función de eTransporte para cargar la descripción de la misma, todo esto se va a ver en el código.





3.C)

ENUNCIADO:

```

459 int primerClienteMoto(ePedido* lista, Clients* list, eTransporte* listaTransporte, int len, int tam, int tamTransporte)
460 {
461     int allOk=0;
462     int flag =1;
463     char primerClienteMoto[50];
464     char cuitPrimerMoto[20];
465
466     if(lista != NULL && list != NULL && listaTransporte != NULL && len > 0 && tam > 0 && tamTransporte > 0)
467     {
468         for(int i=0; i<len;i++)
469         {
470             for(int j=0; j<tam;j++)
471             {
472                 for(int k=0; k<tamTransporte;k++)
473                 {
474                     if(lista[j].idCliente == list[i].id && lista[j].isEmpty == CARGADO && lista[j].estado == 2
475                        && listaTransporte[k].idTransporte == 1 && lista[j].idTransporte == 1 && flag == 1 )
476                     {
477                         flag = 0;
478                         strcpy(primerClienteMoto, list[i].empresa);
479                         strcpy(cuitPrimerMoto, list[i].cuit);
480                         printf("El primer cliente con envio entregado en moto es: \nNOMBRE:%s\nCUIT:%s\n", primerClienteMoto, cuitPrimerMoto);
481                         break;
482                     }
483                 }
484             }
485         }
486         allOk = 1;
487         if(flag == 1)
488         {
489             printf("Error.. no hay pedidos entregados en moto\n");
490         }
491     }
492     return allOk;
493 }
494

```

El filtro complejo elegido es el de la función `primerClienteMoto`, que lo que hace es por medio de las listas de pedidos, listas de clientes y lista de transportes, recorrer el array de pedidos primero, luego el array de clientes y por el último el de transportes, buscando la primera ocurrencia de cliente con un pedido **COMPLETADO** y el id del transporte sea el id de **MOTO**, cuando el programa encuentra esto, va a setear una flag en 0, lo cual va a hacer que no vuelva a entrar al if nuevamente y pinte el nombre y el cuit de la persona que tuvo el primer pedido completado en moto. Así mismo, si no hay pedidos completados, la flag estado se ocuparía de que se imprima el error correspondiente en pantalla.

La función retorna 0 en caso de error y 1 en caso de que las listas sean distinto de null y el tamaño sea menor al de las listas.

```

17 int vehiculoMasKilosEnviados(ePedido* lista, int len)
18 {
19     int allOk= 0;
20     int acumMoto=0;
21     int acumAuto=0;
22     int acumCamion=0;
23     if(lista != NULL && len >0)
24     {
25         for(int i=0; i<len; i++)
26         {
27             if(lista[i].isEmpty == CARGADO && lista[i].estado == COMPLETADO)
28             {
29                 switch(lista[i].idTransporte)
30                 {
31                     case 1:
32                         acumMoto = lista[i].kilos + acumMoto;
33                         break;
34                     case 2:
35                         acumAuto = lista[i].kilos +acumAuto;
36                         break;
37                     case 3:
38                         acumCamion = lista[i].kilos + acumCamion;
39                         break;
40                 }
41             }
42         }
43     }
44     allOk = 1;
45     printf("-----Vehiculo con mas kilos enviados-----\n");
46     if(acumMoto > acumAuto && acumMoto > acumCamion)
47     {
48         printf("El vehiculo con mas kilos enviados fue moto con un total de: [%d]\n", acumMoto);
49     }
50     else
51     {
52         if(acumAuto > acumMoto && acumAuto > acumCamion)
53         {
54             printf("El vehiculo con mas kilos enviados fue auto con un total de: [%d]\n", acumAuto);
55         }
56         else
57         {
58             if(acumCamion > acumAuto && acumCamion > acumMoto)
59             {
60                 printf("El vehiculo con mas kilos enviados fue camion con un total de: [%d]\n", acumCamion);
61             }
62             else
63             {
64                 printf("No se pudo encontrar el vehiculo con mas envios cargados\n");
65             }
66         }
67     }
68 }
69
70 return allOk;
71
72 }

```

Esta función se trataría del segundo filtro, que solo recibe como parámetro la lista de pedidos con su tamaño, se encarga de por medio de acumuladores, ir llenando variables con el valor de kilos enviados por pedido según el vehículo, para luego mostrar en pantalla el vehículo que más kilos envió. A sí mismo cuenta con validación en caso de que no haya un vehículo con más cantidad de envíos y también en el main, tiene la validación de que tiene que haber pedidos enviados.

retorna 0 si hubo error en la listaPedidos o en el tamaño de la lista, y retorna 1 si pudo acceder a la lista correctamente

3.D)

Cientes

```

12 typedef struct
13 {
14     int id;
15     char empresa[51];
16     char direccion[51];
17     int direccionNum;
18     char cuit[12];
19     int idLocalidad;
20     int isEmpty;
21 }Clients;

23 /// @fn int IniciarClientes(Clients*, int)
24 /// @brief recorre el array de lista de clientes dejando todos los espacios vacios
25 /// @param list
26 /// @param len
27 /// @return 0 despues de recorrer el array
28 int IniciarClientes(Clients* list, int len);
29 /// @fn int BuscarLibre(Clients*, int)
30 /// @brief recorre el array de lista de clientes hasta encontrar el primer indice vacio y lo guarda.
31 /// @param list
32 /// @param len
33 /// @return el indice vacio encontrar
34 int BuscarLibre(Clients* list, int len);
35 /// @fn int AgregarCliente(Clients*, int*, int)
36 /// @brief llama a BuscarLibre y guarda el indice vacio encontrado, en ese indice pide y valida los datos personales del usuario
37 /// @param list
38 /// @param pid
39 /// @param len
40 /// @return retorna 1 si pudo acceder a la lista y 0 si hubo algun error
41 int AgregarCliente(Clients *list, eLocalidad* localidades, int *pid,int len);
42 /// @fn int EncontrarClientePorID(Clients*, int, int)
43 /// @brief pasa un id por parametro y lo compara con todos los id de la lista de clientes, al encontrar el id, guarda el indice.
44 /// @param list
45 /// @param len
46 /// @param id
47 /// @return retorna el indice del id buscado, o -1 si no pudo encontrarlo
48 int EncontrarClientePorID(Clients* list, int len,int id);
49 /// @fn int ModificarCliente(Clients*, int)
50 /// @brief pide un id al usuario, luego con el id llama a la funcion EncontrarClientePorID para encontrar el indice del cliente, una vez
51 /// accede a los datos da al cliente la opcion de cambiar la direccion o la localidad.
52 /// @param list
53 /// @param len
54 /// @return retorna 0 si tuvo algun error y 1 si pudo acceder correctamente.
55 int ModificarCliente(Clients* list, int len, eLocalidad* localidades);
56 /// @fn int BajaCliente(Clients*, int, int)
57 /// @brief recibe un id como parametro, llama a la funcion EncontrarClientePorID y lo guarda en un indice, al acceder al indice da opcion al usuario
58 /// de eliminarlo o cancelar la baja. Si la baja se realiza, deja la direccion de ese cliente en 0
59 /// @param list
60 /// @param len
61 /// @param id
62 /// @return retorna -1 si hubo algun error y 0 si pudo acceder correctamente
63 int BajaCliente(Clients* list, int len, int id);

```

```

64@/// @fn int ContadorClientes(Clients*, int*, int)
65 /// @brief recibe un puntero de int como parametro, recorre el array de clientes y por cada uno que este cargado suma 1 al contador
66 /// @param list
67 /// @param contadorClientes
68 /// @param len
69 /// @return -1 si hubo algun error y 0 si pudo acceder correctamente
70 int ContadorClientes(Clients* list, int *contadorClientes, int len);
71@/// @fn void MostrarCliente(Clients, eLocalidad*, int)
72 /// @brief recibe como parametro los clientes, la lista de localidades y el tamaño de la lista de localidades.
73 /// Llama a CargarDescripcionLocalidad y si no tiene ningun error muestra el cliente junto con la localidad
74 /// @param x
75 /// @param localidades
76 /// @param tamLoc
77 void MostrarCliente(Clients x, eLocalidad* localidades, int tamLoc);
78@/// @fn int ImprimirClientes(Clients*, eLocalidad*, int, int)
79 /// @brief recibe como parametro los clientes, la lista de localidades y de clientes. recorre el array de clientes y muestra todos los que esten cargados
80 /// @param list
81 /// @param localidades
82 /// @param tamLoc
83 /// @param len
84 /// @return -1 si hubo error y 0 si pudo acceder correctamente
85 int ImprimirClientes(Clients* list, eLocalidad* localidades, int tamLoc, int len);

```

Pedidos

```

14 typedef struct
15 {
16     int idCliente;
17     int idPedido;
18     int estado;
19     int kilos;
20     int isEmpty;
21     int idLocalidad;
22     int idTransporte;
23 }ePedido;
24

```

```

27 /// @brief inicializa la lista de pedidos y las deja todas en vacio
28 /// @param list
29 /// @param len
30 /// @return -1 si hubo error y 0 si pudo funcionar correctamente
31 int IniciarPedidos(ePedido *list, int len);
32 @fn int BuscarLibrePedidos(ePedido*, int)
33 /// @brief recorre el array y guarda la primera direccion vacio en un indice
34 /// @param list
35 /// @param len
36 /// @return -1 si hubo error y el indice si pudo acceder correctamente
37 int BuscarLibrePedidos(ePedido *list, int len);
38 @fn int AgregarPedido(ePedido*, eEstado*, int, int, int, int*)
39 /// @brief busca el indice libre, pide y valida los datos personales del usuario, por ultimo cambia el estado del pedido a pendiente
40 /// @param list
41 /// @param estados
42 /// @param tamEstado
43 /// @param id
44 /// @param len
45 /// @param idPedido
46 /// @return -1 si hubo error y 0 si funciona correctamente
47 int AgregarPedido(ePedido* list, eEstado* estados, Clients* lista, eTransporte* listaTransporte, int tamTransporte, int tamEstado, int id, int len, int *idPedido);
48 @fn void MostrarPedido(ePedido, eEstado*, int)
49 /// @brief Muestra los pedidos tanto en pendiente como en completado junto con los datos de un cliente
50 /// @param x
51 /// @param estados
52 /// @param tamEstado
53 void MostrarPedido(ePedido x, eEstado* estados, eTransporte* listaTransporte, int tamEstado, int tamTransporte);
54 @fn int ImprimirPedidos(ePedido*, eEstado*, int, int)
55 /// @brief llama a mostrar pedidos e imprime todos los pedidos junto con el id de los clientes
56 /// @param list
57 /// @param estados
58 /// @param tamEstado
59 /// @param len
60 /// @return -1 si hubo error y 0 si pudo funcionar correctamente
61 int ImprimirPedidos(ePedido* list, eEstado* estados, eTransporte* listaTransporte, int tamEstado, int tamTransporte, int len);
62 int BuscarPedido(ePedido* list, int len, int id);
63 @fn int BuscarPedidoPorID(ePedido*, int, int)
64 /// @brief recibe un id como parametro y recorre el array en busca del pedido de ese id, cuando lo encuentra guarad el indice
65 /// @param list
66 /// @param len
67 /// @param id
68 /// @return -1 si hubo error y el indice si funciono correctamente
69 int BuscarPedidoPorID(ePedido* list, int len, int id);

```



```

70@/// @fn int CantidadPedidosPorID(ePedido* list, int len, int id)
71 /// @brief busca por un id en la lista de pedidos el indice, al matchear agrega 1 al contador de pedidos
72 /// @param list
73 /// @param len
74 /// @param id
75 /// @return -1 si hubo error y los pedidos encontrados si funciona correctamente
76 int CantidadPedidosPorID(ePedido* list, int len, int id);
77@/// @fn int ContadorPedidos(ePedido* list, int* contadorPedidos, int len, int id);
78 /// @brief recorre el array de pedidos, al encontrar matchs con los ids y cuando las listas estan cargadas y pendientes, agrega 1 al contador de pedidos
79 /// @param list
80 /// @param contadorPedidos
81 /// @param len
82 /// @param id
83 /// @return -1 si hubo error y 0 si pudo funcionar
84 int ContadorPedidos(ePedido* list, int* contadorPedidos, int len, int id);
85@/// @fn int PedidosPorLocalidad(ePedido* lista, eLocalidad* list, int tam, int len, int id);
86 /// @brief compara el id que pasa por parametro junto con el id de localidad y el estado del pedidos, al hacer match carga 1 al contador de pedidos
87 /// @param lista
88 /// @param list
89 /// @param tam
90 /// @param len
91 /// @param id
92 /// @return -1 si hubo error y el contador de pedidos si funciona correctamente
93 int PedidosPorLocalidad(ePedido* lista, eLocalidad* list, int tam, int len, int id);
94@/// @fn int ImprimirClientePendiente(ePedido* list, Clients* list, int len, int tam);
95 /// @brief recorre la listas de pedidos y la lista de clientes, compara los ids de la lista y valida que el estado del pedido sea en pendiente, luego imprime los datos del cliente
96 /// @param lista
97 /// @param list
98 /// @param len
99 /// @return -1 si hubo error y 0 si pudo funcionar correctamente
100 int ImprimirClientePendiente(ePedido* lista, Clients* list, int len, int tam);
101@/// @fn void MostrarPedidoPendiente(Clients* list, ePedido* lista);
102 /// @brief compara los id del pedido junto con lo del cliente y muestra los datos del cliente al hacer match
103 /// @param list
104 /// @param lista
105 void MostrarPedidoPendiente(Clients* list, ePedido* lista);
106@/// @fn int ImprimirPedidosPendientes(ePedido* list, Clients* list, int len, int tam);
107 /// @brief recorre los array de cliente y de pedidos, verifica que el estado del pedido sea pendiente y por ultimo imprime todos los pedidos en pendiente junto con info del cliente
108 /// @param lista
109 /// @param list
110 /// @param len
111 /// @return -1 si da error y 0 si funciona correctamente
112 int ImprimirPedidosPendientes(ePedido* lista, Clients* list, int len, int tam);
113
114
115
116
117
118
119@/// @fn int vehiculoMasKilosEnviados(ePedido* list, int len);
120 /// @brief Recorre el array en busca del vehiculo que mas cantidad de kilos envio en el transcurso del programa y lo printea
121 /// @param lista
122 /// @param len
123 /// @return 0 si da error y 1 si funciona correctamente
124 int vehiculoMasKilosEnviados(ePedido* list, int len);
125@/// @fn int primerClienteMoto(ePedido* list, Clients* list, eTransporte* listaTransporte, int len, int tam, int tamTransporte);
126 /// @brief recorre los array de las 3 listas en busca del primer envio completado que haya sido realizado en moto y printea el nombre y el cuil de cliente
127 /// @param lista
128 /// @param list
129 /// @param listaTransporte
130 /// @param len
131 /// @param tam
132 /// @param tamTransporte
133 /// @return 0 si da error y 1 si funciona correctamente
134 int primerClienteMoto(ePedido* list, Clients* list, eTransporte* listaTransporte, int len, int tam, int tamTransporte);
135#endif /* PEDIDOS_H_ */
136
137
138
139
140
141
142
143
144
145
146

```

Localidad:

```

11 typedef struct
12 {
13     int id;
14     char descripcion[50];
15 }eLocalidad;
16
17 /// @fn int mostrarLocalidades(eLocalidad*, int)
18 /// @brief recibe como parametro las localidades y el tamaño de la lista. Recorre la lista de localidades y muestra el id de las localidades junto con la descripcion
19 /// @param localidades
20 /// @param tam
21 /// @return 0 si hubo error y 1 si pudo acceder correctamente
22 int mostrarLocalidades(eLocalidad* localidades, int tam);
23 /// @fn int cargarDescripcionLocalidad(eLocalidad*, int, int, char[])
24 /// @brief recibe el idLocalidad como parametro y recorre el array de localidades mientras compara el id ingresado con el de la estructura.
25 /// Al matchear copia la descripcion de la estructura con la descripcion recibida por parametro
26 /// @param localidades
27 /// @param tam
28 /// @param idLocalidad
29 /// @param descripcion
30 /// @return -1 si hubo error y 0 si pudo acceder correctamente
31 int cargarDescripcionLocalidad(eLocalidad* localidades, int tam,int idLocalidad, char descripcion[]);
32
33 #endif /* LOCALIDAD_H_ */

```

Transporte:

```

13 typedef struct
14 {
15     int idTransporte;
16     char vehiculo[30];
17     int kilosSoportados;
18 }eTransporte;
19
20 #define TRANSPORTE_H_
21
22 /// @fn int mostrarTransportes(eTransporte*, int)
23 /// @brief Muestra la lista de transportes y los kilos correspondientes que soporta cada uno
24 /// @param listaTransporte
25 /// @param tamTransporte
26 /// @return 0 si da error y 1 si funciona correctamente
27 int mostrarTransportes(eTransporte* listaTransporte, int tamTransporte);
28 /// @fn int cargarDescripcionTransporte(eTransporte*, int, int, char[])
29 /// @brief recibe el id de un vehiculo como parametro y busca matchear con el id de la estructura. Al matchear, copia la descripcion de dicho vehiculo en una variable
30 /// @param listaTransporte
31 /// @param tamTransporte
32 /// @param idVehiculo
33 /// @param descripcion
34 /// @return 0 si da error y 1 si funciona correctamente
35 int cargarDescripcionTransporte(eTransporte* listaTransporte, int tamTransporte,int idVehiculo, char descripcion[]);
36
37 #endif /* TRANSPORTE_H_ */

```

Estado:

```

11 typedef struct
12 {
13     int idEstado;
14     char descripcion[50];
15 }eEstado;
16
17 /// @fn int MostrarEstados(eEstado*, int)
18 /// @brief recibe como parametro la lista de estado junto con el tamaño de la lista, recorre la lista y muestra el id del estado junto con la descripcion
19 /// @param estados
20 /// @param tamEstado
21 /// @return 0 si hubo error y 1 si pudo funcionar correctamente
22 int MostrarEstados(eEstado* estados, int tamEstado);
23 /// @fn int CargarDescripcionEstado(eEstado*, int, int, char[])
24 /// @brief recibe como parametro la lista de estado junto con el tamaño de la lista,
25 /// recorre la lista y compara el idEstado que recibe como parametro con el idEstado de la estructura, al matchear copia la descripcion de ese id en una variable
26 /// @param estados
27 /// @param tamEstado
28 /// @param idEstado
29 /// @param descripcion
30 /// @return -1 si hubo error y 0 si pudo funcionar correctamente
31 int CargarDescripcionEstado(eEstado* estados, int tamEstado, int idEstado, char descripcion[]);
32
33

```

Tipos

```

11 #include "Pedidos.h"
12 #include "ArrayClients.h"
13
14 typedef struct
15 {
16     int HDPE;
17     int LDPE;
18     int PP;
19     int idPedido;
20     int isEmpty;
21 }eTipo;
22

```

```

23@/// @fn int IniciarTipos(eTipo*, int)
24 /// @brief recorre el array de tipo dejando todos los espacios vacios
25 /// @param lista
26 /// @param len
27 /// @return -1 si error y 0 si pudo funcionar correctamente
28 int IniciarTipos(eTipo *lista, int len);
29@/// @fn int BuscarLibreTipo(eTipo*, int)
30 /// @brief recorre el array de tipo hasta encontrar indice vacio y lo guarda
31 /// @param lista
32 /// @param len
33 /// @return el indice encontrado previamente
34 int BuscarLibreTipo(eTipo* lista, int len);
35@/// @fn int PedirTipos(ePedido*, int, int, eTipo*, int*, int*)
36 /// @brief recibe como parametro la lista de pedidos y 2 punteros de int, uno de acumulador y otro de contador de clientes.
37 /// Busca el indice deseado por medio de un ID, al encontrar el indice lo compara con el del estado del pedido y
38 /// si hay match pide al usuario los tipos de plasticos para su pedido
39 /// @param list
40 /// @param len
41 /// @param id
42 /// @param lista
43 /// @param pAcum
44 /// @param pClientes
45 /// @return 0 si hubo error y 1 si pudo funcionar correctamente
46 int PedirTipos(ePedido* list, int len, int id, eTipo* lista, int *pAcum, int *pClientes);
47@/// @fn int ImprimirPedidosCompletados(ePedido*, Clients*, int, eTipo*)
48 /// @brief recorre el array de pedidos y de tipos. Compara el estado del pedido, los ids del pedido y verifica que ambas listas esten cargadas,
49 /// por ultimo imprime los pedidos completados con los datos del cliente junto con los tipos de plasticos del pedido
50 /// @param lista
51 /// @param list
52 /// @param len
53 /// @param listaT
54 /// @return -1 si hubo error y 0 si pudo funcionar correctamente
55 int ImprimirPedidosCompletados(ePedido* lista, Clients* list, int len, eTipo* listaT);
56@/// @fn void calcularPromedio(ePedido*, int, int*, int*)
57 /// @brief calcula el promedio de PP que fue entregado en el transcurso del programa y lo printea
58 /// @param lista
59 /// @param len
60 /// @param pAcum
61 /// @param pContador
62 void calcularPromedio(ePedido* lista, int len, int *pAcum, int* pContador);
63

```

VIDEO DEFENSA:

OPCION 1:

https://drive.google.com/file/d/1cd3UFCi8pbj6g__kR_-6PRyWz97T1-_L/view?usp=sharing

OPCION 2(RESPALDO): <https://youtu.be/kDU8EeqrCQQ>