FERRERO VALENTIN

DNI: 43261392

DIVISION “B”

LABORATORIO I

ENLACE A DRIVE DEL VIDEO:
https://drive.google.com/drive/folders/1MGeDWvQdOVR1WZ3O0wuHYlOCpDerM5Zz?usp=sharing

PROTOTIPO DE LAS FUNCIONES:

ABMclientes.h

```
/// @fn void inicializarClientes(eCliente[], int)
/// @brief inicializa el array de clientes, poniendo en "LIBRE"
el isEmpty
/// @param datosCliente estructura de datos de los clientes
/// @param tam  tamaño del array de clientes
void InicializarClientes(eCliente datosCliente[],int tam);

/// @fn int cargarDatosCliente(eCliente[], int, int*)
/// @brief da de alta un cliente
/// @param datosCliente estructura de datos de los clientes
/// @param tam  tamaño del array de clientes
/// @param pIdCliente devuelve mediante un puntero al id de
cliente generado
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int CargarDatosCliente(eCliente datosCliente[], eLocalidad
localidad[], int tam, int ultimoId, int* pIdCliente);

/// @fn int buscarIdCliente(eCliente[], int, int, int*)
/// @brief busca el id del cliente
/// @param datosCliente estructura de datos de los clientes
/// @param tam  tamaño del array de clientes
/// @param reintentos cantidad de oportunidades que se le dan al
usuario para ingresar los datos
/// @param pIndice devuelve la posicion del array en donde se
encuentra el idcliente
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int BuscarIdCliente(eCliente datosCliente[], int tam, int
reintentos, int* pIndice);

/// @fn int modificarCliente(eCliente[], int, int)
/// @brief modifica datos del cliente
/// @param datosCliente estructura de datos de los clientes
/// @param tam  tamaño del array de clientes
/// @param reintentos cantidad de oportunidades que se le dan al
usuario para ingresar los datos
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int ModificarCliente(eCliente datosCliente[], eLocalidad
localidad[], int tam, int reintentos);
```

```
/// @fn int bajaCliente(eCliente[], int, int)
/// @brief da de baja un cliente (isEmpty en LIBRE)
/// @param datosCliente estructura de datos de los clientes
/// @param tam  tamaño del array de clientes
/// @param reintentos cantidad de oportunidades que se le dan al usuario para ingresar los datos
/// @returndevuelve 0 si se pudo cargar el array, -1 si no se pudo
int BajaCliente(eCliente datosCliente[], int tam, int reintentos);
```

ABMpedidos.h

```
/// @fn void inicializarPedidos(ePedido[], int)
/// @brief inicializa el array de pedidos, estableciendo isEmpty en LIBRE
/// @param datosPedido estructura de datos de pedidos
/// @param tam tamaño del array de pedidos
void InicializarPedidos(ePedido datosPedido[], int tam);

/// @fn int cargarPedido(ePedido[], int, eCliente[], int, int*, int)
/// @brief da de alta un pedido
/// @param datosPedido estructura de datos de pedidos
/// @param tamPedidos tamaño del array de pedidos
/// @param datosCliente estructura de datos de clientes
/// @param tamCliente tamaño del array de clientes
/// @param pIdPedido devuelve el id de pedido generado
/// @param reintentos cantidad de oportunidades que se le dan al usuario para ingresar los datos
/// @return devuelve 0 si se pudo cargar el array, -1 si no se pudo
int CargarPedido( ePedido datosPedido[], int tamPedidos, eCliente datosCliente[], int tamCliente, int* pIdPedido, int reintentos);

/// @fn int verListaPedidosPendientes(eCliente[], int, ePedido[], int)
/// @brief  muestra todos los pedidos pendientes
/// @param datosCliente
/// @param tamCliente tamaño del array de clientes
/// @param datosPedido estructura de datos de pedidos
/// @param tamPedido tamaño del array de pedidos
/// @return devuelve 0 si se pudo cargar el array, -1 si no se pudo
int VerListaPedidosPendientes(eCliente datosCliente[], int tamCliente, ePedido datosPedido[], int tamPedido);

/// @fn int verUnPedidoPendiente(ePedido[], int, int)
```

```
/// @brief  muestra un pedido pendiente en especifico
/// @param datosPedido estructura de datos de pedidos
/// @param tamPedido tamaño del array de pedidos
/// @param idCliente recibe el idCliente
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int VerUnPedidoPendiente(ePedido datosPedido[], int tamPedido,
int idCliente);

/// @fn int buscarIdPedido(ePedido[], int, int, int*)
/// @brief busca un id de pedido
/// @param datosPedido estructura de datos de pedidos
/// @param tam tamaño del array de pedido
/// @param reintentos cantidad de oportunidades que se le dan al
usuario para ingresar los datos
/// @param pIndice devuelve la posicion del array en donde se
encuentra el idPedido
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int BuscarIdPedido(ePedido datosPedido[], int tam, int*
pIndice);

/// @fn int procesarResiduos(ePedido[], int, int)
/// @brief procesa los residuos
/// @param datosPedido estructura de datos de pedidos
/// @param tamPedido tamaño del array de pedidos
/// @param reintentos cantidad de oportunidades que se le dan al
usuario para ingresar los datos
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int ProcesarResiduos(ePedido datosPedido[], int tamPedido, int
reintentos);
```

localidad.h

```
/// @fn void InicializarLocalidades(eLocalidad[], int)
/// @brief coloca a cada isEmpty de la entidad en LIBRE
/// @param localidades array de la estructura localidad
/// @param tam tamaño del array localidad
void InicializarLocalidades(eLocalidad localidades[], int tam);

/// @fn int BuscarLocalidad(int*, char[], eLocalidad[], int)
/// @brief entregandole una string compara en la entidad
localidad y si lo encuentra devuelve el id asignado
/// @param idLocalidad obtiene un puntero del idLocalidad
/// @param localidad string a comparar
/// @param localidades array de la estructura localidad
/// @param tamLocalidades tamaño del array localidad
/// @return devuelve 0 si se encontro la localidad
```

```
int BuscarLocalidad(int* idLocalidad, char localidad[],
eLocalidad localidades[], int tamLocalidades);

/// @fn int NuevaLocalidad(int*, eLocalidad[], int)
/// @brief solicita una localidad, verifica si ya fue ingresada
y le asigna un id de ser necesario
/// @param idLocalidad devuelve como puntero el id generado
/// @param localidad array de la estructura localidad
/// @param tamLocalidades tamaño del array localidad
/// @return devuelve 0 si se pudo cargar la localidad
int NuevaLocalidad(int* idLocalidad, eLocalidad localidad[], int
tamLocalidades);

/// @fn void VerLocalidadPorId(char[], eLocalidad[], int, int)
/// @brief Obtiene el dato localidad buscando un id coincidente
en la estructura
/// @param localidad Dato a obtener
/// @param localidades array de la estructura eLocalidad
/// @param tamLocalidad tamaño del array localidad
/// @param idLocalidad id obtenido como parametro para comparar
int VerLocalidadPorId(char localidad[], eLocalidad
localidades[], int tamLocalidad, int idLocalidad);

/// @fn int VerUnaLocalidad(eLocalidad[], int, int)
/// @brief Muestra una localidad
/// @param localidad array de la estructura eLocalidad
/// @param tamLocalidades tamaño del array localidad
/// @param idLocalidad id obtenido como parametro para comparar
/// @return Retorna (-1) Si no encontro la localidad coincidente
///                 (0) Si encontro el dato y imprimio la
localidad
int VerUnaLocalidad(eLocalidad localidad[], int tamLocalidades,
int idLocalidad);

/// @fn int VerListadoDeLocalidades(eLocalidad[], int)
/// @brief Muestra un listado de todas las localidades
/// @param localidad array de la estructura eLocalidad
/// @param tamLocalidad tamaño del array localidad
/// @return Retorna (-1) Si no imprimio datos
///                 (0) Si encontro al menos una localidad y
la imprimio
int VerListadoDeLocalidades(eLocalidad localidad[], int
tamLocalidad);
```

informes.h

```
/// @fn int mostrarUnCliente(eCliente)
/// @brief muestra en pantalla los datos de un cliente
/// @param datosCliente estructura de datos de los clientes
```

```
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int MostrarUnCliente(eCliente datosCliente, eLocalidad
datosLocalidad[], int tam);

/// @fn int verListaClientes(eCliente[], int)
/// @brief muestra todos los datos de todos los clientes
/// @param datosCliente estructura de datos de los clientes
/// @param tam  tamaño del array de clientes
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int VerListaClientes(eCliente datosCliente[], eLocalidad
datosLocalidad[], int tam);

/// @fn int ContadorEstadoPedido(int*, int, ePedido[], int)
/// @brief cuenta cuantos pedidos pendientes hay
/// @param pCantidad devuelve por puntero la cantidad de pedidos
en estado pendiente
/// @param idCliente envia el id de cliente a buscar
/// @param datosPedido estructura de datos de pedidos
/// @param tamPedido tamaño del array de pedidos
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int ContadorEstadoPedido(int* pCantidad, int idCliente, int
estadoPedido, ePedido datosPedido[], int tamPedido);

/// @fn int informarClientes(eCliente[], int, ePedido[], int)
/// @brief informa todos los clientes con su cantidad de pedidos
pendientes
/// @param datosCliente estructura de datos de clientes
/// @param tamCliente tamaño del array de clientes
/// @param datosPedido estructura de datos de pedidos
/// @param tamPedido tamaño del array de pedidos
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int InformarClientes(eLocalidad datosLocalidades[], eCliente
datosCliente[], int tamCliente, ePedido datosPedido[], int
tamPedido);

/// @fn int ClientesPorPedido(int, eCliente[], int)
/// @brief Imprime cuit y direccion de un empleado en especifico
/// @param idClientePedido Id a buscar para imprimir
/// @param datosCliente estructura de datos eCliente
/// @param tamClientes tamaño de la estructura
/// @return Retorna(-1) Si no encontro ningun cliente que
coincida con el id
///                 (0) Si encontro e imprimio al menos un
cliente
```

```
int ClientesPorPedido(int idClientePedido, eCliente
datosCliente[], int tamClientes);

/// @fn int InformarClientesPorEstado(int, eCliente[], int,
ePedido[], int)
/// @brief Imprime un listado de clientes con  los pedidos que
se especifiquen por parametro
/// @param estado Si el listado es de pedidos PENDIENTES o
COMPLETADOS
/// @param datosCliente estructura de datos eCliente
/// @param tamCliente tamaño del array de clientes
/// @param datosPedido estructura de datos de pedidos
/// @param tamPedidotamaño del array de pedidos
/// @return
int InformarClientesPorEstado(int estado, eCliente
datosCliente[], int tamCliente, ePedido datosPedido[], int
tamPedido);

/// @fn int InformarPendientesPorLocalidad(eLocalidad[],
eCliente[], int, ePedido[], int)
/// @brief informa la cantidad de pedidos pendientes que hay
para una localidad ingresada por el usuario
/// @param localidades estructura de datos de localidades
/// @param datosCliente estructura de datos de clientes
/// @param tamCliente tamaño del array de clientes
/// @param datosPedido estructura de datos de pedidos
/// @param tamPedido tamaño del array de pedidos
/// @return devuelve 0 si se encontro la localidad y -1 si no se
enocntro
int InformarPendientesPorLocalidad(eLocalidad localidades[],
eCliente datosCliente[], int tamCliente, ePedido datosPedido[],
int tamPedido);

/// @fn int promedioPolipropileno(ePedido[], int, int, float*)
/// @brief calcula el polipropileno promedio por cliente
/// @param datosPedido estructura de datos de pedidos
/// @param tamPedido tamaño del array de pedidos
/// @param idCliente envia el idCliente a realcionar con los
pedidos
/// @param pPromedio devuelve el promedio de kilos por cliente
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int PromedioPolipropileno(ePedido datosPedido[], int tamPedido,
int idCliente, float* pPromedio);

/// @fn int informarPolipropilenoPromedio(eCliente[], int,
ePedido[], int)
/// @brief informa el polipropileno promedio por cliente
/// @param datosCliente estructura de datos de clientes
```

```
/// @param tamCliente tamaño del array de clientes
/// @param datosPedido tamaño del array de pedidos
/// @param tamPedido tamaño del array de pedidos
/// @return devuelve 0 si se pudo cargar el array, -1 si no se
pudo
int InformarPolipropilenoPromedio(eCliente datosCliente[], int
tamCliente, ePedido datosPedido[], int tamPedido);

/// @fn int BuscarMaxEstadoPedido(int*, int*, int, eCliente[],
int, ePedido[], int)
/// @brief busca el cliente con la mayor cantidad de pedidos en
un estado especifico
/// @param pIndice devuelve como puntero el indice del array de
cliente que mas pedidos tiene
/// @param pCantidad devuelve como puntero la cantidad de
pedidos que resulto ser la mayor
/// @param estadoPedido señala que estado de pedido se busca, si
PENDIENTE o COMPLETADO
/// @param datosCliente estructura de datos de clientes
/// @param tamCliente tamaño del array de clientes
/// @param datosPedido tamaño del array de pedidos
/// @param tamPedidotamaño del array de pedidos
/// @return devuelve 0 si se encontro un maximo y -1 si no se
encontro
int BuscarMaxEstadoPedido(int* pIndice, int* pCantidad, int
estadoPedido,eCliente datosCliente[], int tamCliente, ePedido
datosPedido[], int tamPedido);;

/// @fn int InformarClienteMaxPorEstado(int, eCliente[], int,
ePedido[], int)
/// @brief Iforma el cliente que tiene la mayor cantidad de
pedidos en un estado en especifico
/// @param estado señala que estado de pedido se busca, si
PENDIENTE o COMPLETADO
/// @param datosCliente estructura de datos de clientes
/// @param tamCliente tamaño del array de clientes
/// @param datosPedido tamaño del array de pedidos
/// @param tamPedido tamaño del array de pedidos
/// @return Retorna (-1) Si no encontro un maximo
///                  (0) Si encontro un maximo y lo imprimio
int InformarClienteMaxPorEstado(int estado, eCliente
datosCliente[], int tamCliente, ePedido datosPedido[], int
tamPedido);
```