**Universidad Tecnológica Nacional Facultad**

**Regional Avellaneda**

Técnico Universitario en Programación

**Recuperatorio Laboratorio Primer parcial**

Materia: LABORATORIO I

Nombre: Carolina

Apellido: Alvarez

Fecha: 26/11/2021

Docente: Germán, Cami, Gastón

División: 1E

Nota:

Legajo: 37009994

Instancia RPP

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).

**¿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.

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.

1. Enlace a github: <https://github.com/caro271092/PrimerParcialLabo>
2. Drive: <https://drive.google.com/drive/folders/1GGTIuFsoTeeflDl8UZCLcKTwOsTaQCX3?usp=sharing>

(\*)

1. Prototipos con documentación:

/\*Clientes.h \*/

**#ifndef** CLIENTES\_H\_

**#define** CLIENTES\_H\_

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#include** <ctype.h>

**#include** <limits.h>

**#include** "Localidades.h"

**#include** "InputOutput.h"

**#define** EMPTY 1

**#define** FULL 0

**typedef** **struct**{

**int** idCliente;

**char** nombreEmpresa[51];

**int** cuit;

**char** direccion[51];

**int** idLocalidad;

**int** isEmpty;

}eCliente;

/// \fn void IniciarEstructura(eCliente[], int)

/// \brief le otorga el valor vacío al campo isEmpty, para identificarlo como espacio libre en el array de clientes

/// \param lista array de clientes

/// \param tam tamaño del array

**void** **IniciarEstructura**(eCliente lista[], **int** tam);

/// \fn int BuscarPrimerEspacioLibre(eCliente[], int)

/// \brief encuentra el primer lugar que se marcó empty

/// \param lista array de clientes

/// \param tam tamaño del array

/// \return retorna el índice de la posición del primer lugar libre en el array

**int** **BuscarPrimerEspacioLibre**(eCliente lista[], **int** tam);

/// \fn int BuscarPrimerEspacioOcupado(eCliente[], int)

/// \brief encuentra el primer lugar que se marcó full

/// \param lista array de clientes

/// \param tam tamaño del array

/// \return retorna el índice de la posición del primer lugar ocupado en el array

**int** **BuscarPrimerEspacioOcupado**(eCliente lista[], **int** tam);

/// \fn int AgregarCliente(eCliente[], int, int, char[], int, char[], int)

/// \brief guarda la información, que recibe por parámetro, en el primer lugar libre del array de clientes

/// \param lista array de clientes

/// \param tam tamaño del array

/// \param idCliente campo identificador del cliente

/// \param nombreEmpresa campo nombre de la empresa

/// \param cuit campo de CUIT

/// \param direccion campo direccion del cliente

/// \param localidad campo localidad del cliente

/// \return devuelve -1 si no logró cargar el array y 0 si pudo hacerlo

**int** **AgregarCliente**(eCliente lista[], **int** tam, **int** idCliente, **char** nombreEmpresa[], **int** cuit, **char** direccion[], **int** localidad);

/// \fn int PedirCliente(eCliente[], int, int\*)

/// \brief solicita los datos para cargar un cliente en el array de clientes

/// \param lista array de clientes

/// \param tam tamaño del array

/// \param idCliente identificador del cliente que se pasa por puntero al main para su incrementación para el próximo ingreso

/// \return devuelve -1 si no logró cargar el array y 0 si pudo hacerlo

**int** **PedirCliente**(eCliente lista[], **int** tam, **int** \*idCliente);

/// \fn int EncontrarClientePorId(eCliente[], int, int)

/// \brief busca un cliente determinado en el array de clientes

/// \param lista array de clientes

/// \param tam tamaño del array

/// \param idIngresado id del cliente a encontrar

/// \return retorna el índice de la posición del cliente en el array, si éste no existe retorna -1

**int** **EncontrarClientePorId**(eCliente lista[], **int** tam, **int** idIngresado);

/// \fn int ModificarDireccion(eCliente[], int, int)

/// \brief modifica el campo dirección de un cliente determinado dentro del array de clientes

/// \param lista array de clientes

/// \param tam tamaño del array

/// \param id identificador del cliente a modificar, de existir

/// \return devuelve -1 si no logró modificar y 0 si pudo hacerlo

**int** **ModificarDireccion**(eCliente lista[], **int** tam, **int** id);

/// \fn int ImprimirListadoClientes(eCliente[], int)

/// \brief muestra los clientes cargados en el array de clientes

/// \param lista array de clientes

/// \param tam tamaño del array

/// \return devuelve -1 si no logró imprimir y 0 si pudo hacerlo

**int** **ImprimirListadoClientes**(eCliente lista[], **int** tam);

/// \fn int BajaCliente(eCliente[], int, int)

/// \brief cambia el estado del campo isEmpty a empty, para dar de baja a un cliente determinado del array de clientes

/// \param lista array de clientes

/// \param tam tamaño del array

/// \param idBaja identificador del cliente a dar de baja

/// \return devuelve -1 si no logró cambiar el campo y 0 si pudo hacerlo

**int** **BajaCliente**(eCliente lista[], **int** tam, **int** idBaja);

/\*Pedidos.h \*/

**#ifndef** PEDIDOS\_H\_

**#define** PEDIDOS\_H\_

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#include** <ctype.h>

**#include** <limits.h>

**#include** "Clientes.h"

**#include** "InputOutput.h"

**#define** EMPTY 1

**#define** FULL 0

**typedef** **struct**{

**int** idPedido;

**int** idCliente;

**float** cantidadKilosARecolectar;

**char** estado[30];

**float** kilosHDPE;

**float** kilosLDPE;

**float** kilosPP;

**int** isEmpty;

}ePedido;

/// \fn void IniciarEstructuraPedido(ePedido[], int)

/// \brief le otorga el valor vacío al campo isEmpty, para identificarlo como espacio libre en el array de pedidos

/// \param lista array de pedidos

/// \param tam tamaño del array

**void** **IniciarEstructuraPedido**(ePedido lista[], **int** tam);

/// \fn int BuscarPrimerEspacioLibrePedidos(ePedido[], int)

/// \brief encuentra el primer lugar que se marcó empty

/// \param lista array de pedidos

/// \param tam tamaño del array

/// \return retorna el índice de la posición del primer lugar libre en el array

**int** **BuscarPrimerEspacioLibrePedidos**(ePedido lista[], **int** tam);

/// \fn int AgregarPedido(ePedido[], int, int, float, float, float, float, int)

/// \brief guarda la información, que recibe por parámetro, en el primer lugar libre del array de pedidos

/// \param lista array de pedidos

/// \param tam tamaño del array

/// \param idPedido campo de identificador de pedido

/// \param cantidadKilosARecolectar campo de cantidad de kilos a recolectar

/// \param kilosHDPE campo de kilos de HDPE "a reciclar"

/// \param kilosLDPE campo de kilos de LDPE "a reciclar"

/// \param kilosPP campo de kilos de PP "a reciclar"

/// \param idClienteIngresado campo del identificador del cliente al cual pertenece el pedido

/// \return devuelve -1 si no logró cargar el array y 0 si pudo hacerlo

**int** **AgregarPedido**(ePedido lista[], **int** tam, **int** idPedido, **float** cantidadKilosARecolectar, **float** kilosHDPE, **float** kilosLDPE, **float** kilosPP, **int** idClienteIngresado);

/// \fn int IngresarDatosDePedido(ePedido[], int, int\*, int)

/// \brief carga un pedido de recoleccion al cliente ingresado

/// \param lista array de pedidos

/// \param tam tamaño del array

/// \param idPedido identificador del pedido

/// \param idClienteIngresado identificador del cliente al cual se le cargará el pedido

/// \return devuelve -1 si no logró cargar el array y 0 si pudo hacerlo

**int** **IngresarDatosDePedido**(ePedido lista[], **int** tam, **int** \*idPedido, **int** idClienteIngresado);

/// \fn int ImprimirListadoPedidos(ePedido[], int)

/// \brief muestra los pedidos cargados en el array de pedidos

/// \param lista array de pedidos

/// \param tam tamaño del array

/// \return devuelve -1 si no logró imprimir y 0 si pudo hacerlo

**int** **ImprimirListadoPedidos**(ePedido lista[], **int** tam);

/// \fn int BuscarPrimerEspacioOcupadoPedidos(ePedido[], int)

/// \brief encuentra el primer lugar que se marcó full

/// \param lista array de pedidos

/// \param tam tamaño del array

/// \return retorna el índice de la posición del primer lugar ocupado en el array

**int** **BuscarPrimerEspacioOcupadoPedidos**(ePedido lista[], **int** tam);

/// \fn int EncontrarPedidoPorId(ePedido[], int, int)

/// \brief busca un pedido determinado en el array de pedidos

/// \param lista array de pedidos

/// \param tam tamaño del array

/// \param idPedidoIngresado id del pedido a encontrar

/// \return retorna el índice de la posición del pedido en el array, si éste no existe retorna -1

**int** **EncontrarPedidoPorId**(ePedido lista[], **int** tam, **int** idPedidoIngresado);

/// \fn int ClasificarPlastico(ePedido[], int, int)

/// \brief completa los campos a reciclar del array de pedidos

/// \param lista array de pedidos

/// \param tam tamaño del array

/// \param id identificador del pedido a cargar kilos reciclados

/// \return devuelve -1 si no logró cargar el array y 0 si pudo hacerlo

**int** **ClasificarPlastico**(ePedido lista[], **int** tam, **int** id);

/// \fn float KilosTotalesDePPObtenidos(ePedido[], int)

/// \brief realiza la sumatoria de los kilos de PP obtenidos en todos los pedidos

/// \param listaPedidos array de pedidos

/// \param tamPedidos tamaño del array

/// \return devuelve la sumatoria de kilos de PP reciclados de todos los pedidos del array

**float** **KilosTotalesDePPObtenidos**(ePedido listaPedidos[], **int** tamPedidos);

/\*InputOutput.h \*/

**#ifndef** INPUTOUTPUT\_H\_

**#define** INPUTOUTPUT\_H\_

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#include** <ctype.h>

**#include** <limits.h>

**#define** EMPTY 1

**#define** FULL 0

/// \fn int IngresarEntero(char[])

/// \brief pide al usuario el ingreso de un entero

/// \param mensaje mensaje para pedir el ingreso

/// \return número ingresado

**int** **IngresarEntero**(**char** mensaje[]);

/// \fn int ValidacionIngresoDeUnFlotantePositivo(char[])

/// \brief valida que el ingreso que recibe por parámetro sea un flotante positivo

/// \param ingreso ingreso a validar

/// \return retorna -1 si el ingreso no es un flotante positivo, 0 si lo es

**int** **ValidacionIngresoDeUnFlotantePositivo**(**char** ingreso[]);

/// \fn int ValidacionIngresoDeUnEnteroPositivo(char[])

/// \brief valida que el ingreso que recibe por parámetro sea un entero positivo

/// \param ingreso ingreso a validar

/// \return retorna -1 si el ingreso no es un entero positivo, 0 si lo es

**int** **ValidacionIngresoDeUnEnteroPositivo**(**char** ingreso[]);

/// \fn int PedirNumeroEntero(int\*, char[], char[], int, int, int)

/// \brief pide al usuario el ingreso de un entero y valida que éste sea un entero positivo y esté dentro de determinado rango

/// dándole la posibilidad de reingresar, de ser necesario

/// \param ingresoValidado si el ingreso es un entero positivo y está dentro del rango lo devuelve a través de este puntero

/// \param mensaje mensaje para pedir el ingreso

/// \param mensajeError mensaje para informar que el ingreso no corresponde

/// \param minimo valor mínimo que puede tener el ingreso

/// \param maximo valor máximo que puede tener el ingreso

/// \param reintentos cantidad de veces que puede reingresar en caso de ser necesario

/// \return -1 si luego de los reintentos el ingreso sigue siendo incorrecto, 0 si se validó correctamente el ingreso

**int** **PedirNumeroEntero**(**int**\* ingresoValidado, **char** mensaje[], **char** mensajeError[], **int** minimo, **int** maximo, **int** reintentos);

/// \fn int PedirNumeroFlotante(float\*, char[], char[], float, float, int)

/// \brief pide al usuario el ingreso de un flotante y valida que éste sea positivo y esté dentro de determinado rango

/// dándole la posibilidad de reingresar, de ser necesario

/// \param ingresoValidado si el ingreso es un flotante positivo y está dentro del rango lo devuelve a través de este puntero

/// \param mensaje mensaje para pedir el ingreso

/// \param mensajeError mensaje para informar que el ingreso no corresponde

/// \param minimo valor mínimo que puede tener el ingreso

/// \param maximo valor máximo que puede tener el ingreso

/// \param reintentos cantidad de veces que puede reingresar en caso de ser necesario

/// \return -1 si luego de los reintentos el ingreso sigue siendo incorrecto, 0 si se validó correctamente el ingreso

**int** **PedirNumeroFlotante**(**float** \*ingresoValidado, **char** mensaje[], **char** mensajeError[], **float** minimo, **float** maximo, **int** reintentos);

/// \fn int GetString(char[], char[], int)

/// \brief solicita el ingreso de una cadena de caracteres, de determinado máximo de caracteres

/// \param string ingreso validado

/// \param message mensaje para solicitar el ingreso

/// \param len tamaño máximo del string

/// \return -1 si no se pudo guardar el string, 0 si se pudo

**int** **GetString**(**char** string[], **char** message[], **int** len);

/// \fn char ToUpperFirstLetter\*(char[])

/// \brief recibe una cadena de caracteres y coloca en mayúscula el caracter de la posición 0

/// \param cadena cadena a modificar

/// \return cadena con el caracter en la posición 0 en mayúscula

**char**\* **ToUpperFirstLetter**(**char** cadena[]);

/// \fn int ValidacionEnteroDentroDeUnRango(int, int, int)

/// \brief valida que el entero ingresado esté dentro de un rango determinado

/// \param ingreso ingreso a validar

/// \param minimo valor mínimo que puede tomar el ingreso

/// \param maximo valor máximo que puede tomar el ingreso

/// \return ingreso validado

**int** **ValidacionEnteroDentroDeUnRango**(**int** ingreso,**int** minimo,**int** maximo);

/\* Informes.h \*/

**#ifndef** INFORMES\_H\_

**#define** INFORMES\_H\_

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#include** <ctype.h>

**#include** <limits.h>

**#include** "Pedidos.h"

**#include** "Clientes.h"

**#include** "InputOutput.h"

**#define** EMPTY 1

**#define** FULL 0

/// \fn void MostrarCliente(eCliente)

/// \brief muestra la información ingresada en los campos de un cliente

/// \param cliente índice del cliente en el array de clientes

**void** **MostrarCliente**(eCliente cliente);

/// \fn void MostrarCuitYDireccionCliente(eCliente[], int, int)

/// \brief imprime cuit y dirección del cliente cuyo id recibe por parámetro

/// \param listaClientes array de clientes

/// \param tamClientes tamaño del array

/// \param idClienteRecibido id a imprimir datos

**void** **MostrarCuitYDireccionCliente**(eCliente listaClientes[], **int** tamClientes, **int** idClienteRecibido);

/// \fn void DetallePedidosPendientes(ePedido[], int, eCliente[], int)

/// \brief imprime Cuit del cliente, dirección del cliente y cantidad de kilos a recolectar de cada pedido que está pendiente

/// \param listaPedidos array de pedidos

/// \param tamPedidos tamaño del array de pedidos

/// \param listaClientes array de clientes

/// \param tamClientes tamaño del array de clientes

**void** **DetallePedidosPendientes**(ePedido listaPedidos[], **int** tamPedidos, eCliente listaClientes[], **int** tamClientes);

/// \fn void DetallePedidosProcesados(ePedido[], int, eCliente[], int)

/// \brief imprime Cuit y dirección del cliente, junto con cantidad de kilos reciclados de cada tipo de plástico, de cada pedido que está completado

/// \param listaPedidos array de pedidos

/// \param tamPedidos tamaño del array de pedidos

/// \param listaClientes array de clientes

/// \param tamClientes tamaño del array de clientes

**void** **DetallePedidosProcesados**(ePedido listaPedidos[], **int** tamPedidos, eCliente listaClientes[], **int** tamClientes);

/// \fn int ContadorEstadoPedidoCliente(ePedido[], int, char[], int)

/// \brief cuenta la cantidad de pedidos que hay en determinado estado en determinado cliente

/// \param listaPedidos array de pedidos

/// \param tam tamaño del array de pedidos

/// \param estadoRequerido estado a chequear

/// \param idAchequear id del cliente que se quiere consultar

/// \return cantidad de pedidos del cliente en el estado solicitado

**int** **ContadorEstadoPedidoCliente**(ePedido listaPedidos[], **int** tam, **char** estadoRequerido[], **int** idAchequear);

/// \fn void MostrarCantidadDePedidosPendientesPorCliente(ePedido[], int, eCliente[], int)

/// \brief muestra el listado de clientes con la cantidad de pedidos en estado pendiente de cada uno

/// \param listaPedidos array de pedidos

/// \param tamPedidos tamaño del array de pedidos

/// \param listaClientes array de clientes

/// \param tamClientes tamaño del array de clientes

**void** **MostrarCantidadDePedidosPendientesPorCliente**(ePedido listaPedidos[], **int** tamPedidos, eCliente listaClientes[], **int** tamClientes);

/// \fn int CantidadPedidosPendientesPorLocalidad(ePedido[], int, eCliente[], int)

/// \brief calcula la cantidad de pedidos pendientes en localidad ingresada

/// \param listaPedidos array de pedidos

/// \param tamPedidos tamaño del array de pedidos

/// \param listaClientes array de clientes

/// \param tamClientes tamaño del array de clientes

/// \param idLocalidadIngresada identificador de la localidad devuelto por puntero

/// \return cantidad de pedidos pendientes en la localidad determinada

**int** **CantidadPedidosPendientesPorLocalidad**(ePedido listaPedidos[], **int** tamPedidos, eCliente listaClientes[], **int** tamClientes, **int**\* idLocalidadIngresada);

/// \fn int ContadorPedidosPorCliente(ePedido[], int, int)

/// \brief cuenta la cantidad de pedidos que tiene determinado cliente del cual se recibe el id por parámetro

/// \param listaPedidos array de pedidos

/// \param tam tamaño del array de pedidos

/// \param idAchequear id del cliente del cual se calculará cantidad de pedidos

/// \return cantidad de pedidos del cliente indicado

**int** **ContadorPedidosPorCliente**(ePedido listaPedidos[], **int** tam, **int** idAchequear);

/// \fn int ClienteConMasPedidos(ePedido[], int)

/// \brief encuentra el id del cliente con mayor cantidad de pedidos de recolección

/// \param listaPedidos array de pedidos

/// \param tamPedidos tamaño del array de pedidos

/// \return id del cliente que tiene más pedidos cargados

**int** **ClienteConMasPedidos**(ePedido listaPedidos[], **int** tamPedidos);

/// \fn int ClienteConMasPedidosEstado(ePedido[], int, char[])

/// \brief encuentra el id del cliente con mayor cantidad de pedidos de recolección de determinado estado

/// \param listaPedidos array de pedidos

/// \param tamPedidos tamaño del array de pedidos

/// \param estadoDeseado estado de pedidos que se desea consultar

/// \return id del cliente que tiene más pedidos cargados con el estado deseado

**int** **ClienteConMasPedidosEstado**(ePedido listaPedidos[], **int** tamPedidos, **char** estadoDeseado[]);

/\* Localidades.h \*/

**#ifndef** LOCALIDADES\_H\_

**#define** LOCALIDADES\_H\_

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#include** <ctype.h>

**#include** <limits.h>

**#include** "InputOutput.h"

**#define** EMPTY 1

**#define** FULL 0

**typedef** **struct**{

**int** idLocalidad;

**char** descripcion[30];

**int** isEmpty;

}eLocalidad;

/// \fn void IniciarEstructuraLocalidades(eLocalidad[], int)

/// \brief otorga el valor vacío al campo isEmpty para luego identificarlo y poder completarlo (en este caso está hardcodeado)

/// \param lista array de localidades

/// \param tam tamaño del array

**void** **IniciarEstructuraLocalidades**(eLocalidad lista[], **int** tam);

(\*)

Enlaces a web del programa que usé para realizar los videos (por las dudas):

<https://www.loom.com/share/93cd5ff931714e90b1460651f0f3828f>

<https://www.loom.com/share/f46a0aa091694cf2b4f4f039ed7fcaee>

<https://www.loom.com/share/d1aada186b4c4d28bac05780b24333ec>