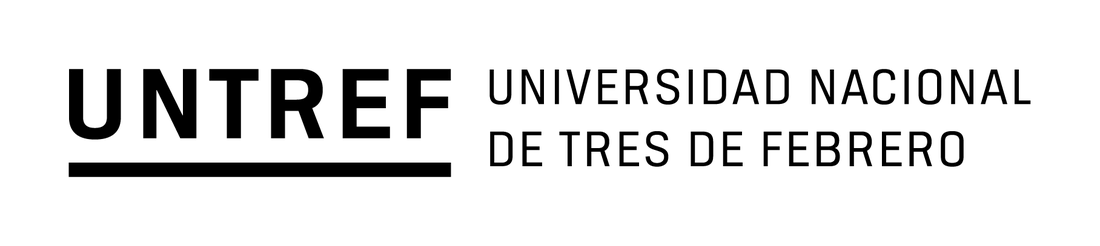
**INFORME - TRABAJO PRÁCTICO FINAL**

**ALGORITMOS Y PROGRAMACIÓN III**

**Sistema de créditos en lenguaje C**



Alumnos:

Nahuel Ahumada, Oscar Frasson, Carolina Monte, Emiliano Pellegrino

Julio 2019

**HEADER FILES**

- ListaClientes.h: Se define lista enlazada de los clientes con la definición de typedef struct nodoCliente, a través de la cual se tiene cada cliente y un puntero al siguiente en la lista.

- ListaCreditos.h: Análogamente a ListaClientes se define lista enlazada de los créditos con la definición de typedef struct nodoCredito, a través de la cual se tiene cada crédito y un puntero al siguiente en la lista. Se agrega un ID por crédito.

- Cliente.h: Se define el cliente con sus características: su nombre, apellido, id, edad, dirección, un puntero al referido y un puntero a su lista de créditos.

- Credito.h: Se define el crédito con sus características: montoTotal, montoDeLaCuota, cantidadDeCuotasRestantes, cantidadDeCuotas, nombre y un puntero a cliente.

**DECISIONES DE DISEÑO**

Se decidió utilizar un modelo de estructura de trabajo práctico, basado en el uso de listas simples, debido a la facilidad y orden que presentan las mismas para el manejo de información. Algo a tener en cuenta es que, debido al uso de distintas funciones de búsqueda y la longitud que pueden alcanzar las listas, sería acertado cambiar este diseño en versiones futuras con el fin de mejorar dichas funciones y no recurrir a la búsqueda secuencial.

El programa cuenta con un menú por consola, el cual contiene las siguientes opciones:

1. Agregar cliente

2. Imprimir clientes

3. Buscar cliente por nombre

4. Buscar cliente por rango de edad

5. Buscar cliente por ID

6. Cargar archivo

7. Salir

Continuemos con los elementos del programa.

**Cliente:**

Para el total de créditos que puede obtener un mismo cliente, se decidió que sea un total de tres, para mantener dentro de lo simple el esquema del tp. Esta cantidad se limita uso mediante de una variable int y un if.

La cantidad de clientes es ilimitada, usando una id autoincrementable a medida que se agrega un nuevo cliente en la lista. En cuanto a los otros datos del cliente(nombre, apellido, edad y dirección), estos se pasan por pantalla a través del método iniciar\_cliente(), el cual a su vez llama el método nuevo\_cliente() que recibe por parámetro dichos valores que se pasaron previamente. Este último método es el que se encarga de reservar memoria y guardar los datos del cliente. La función de estos dos métodos podrían haberse resuelto en uno solo, pero creemos que de esta forma el código resulta más comprensible.

En cuanto a la información créditos del cliente, este puede solicitar el valor total a pagar con el método “getMontoTotalAPagar()” sumando el monto de todos los créditos que posea (En caso de que no tenga ninguno devuelve 0).

Por último, en este archivo hicimos un sub-menú que permite las siguientes opciones:

1. Agregar cliente recomendado
2. Dar de alta Crédito del cliente
3. Listar Creditos
4. Pagar cuota
5. Cancelar Total de Créditos
6. Salir

El menú se despliega a través del método void menuCliente() el cual recibe como parámetro un struct Cliente.

Opción 1: Añade un cliente referenciado, tal que al cliente que se pasó por parámetro en menuCliente() : cliente->referenciado=iniciar\_cliente(), pasando por pantalla los datos del nuevo cliente.

Opción 2: Declara un nuevo crédito e intenta añadirlo a la lista que de créditos que tiene el cliente. Si ya tiene 3 elementos en la lista, no realiza cambios

Opción 3: Si la lista de créditos asociada a el cliente no está vacía, imprime por pantalla todos los créditos que posea: Nombre, saldo, cantidad de cuotas, monto de las cuotas y cantidad de cuotas restantes a pagar.

Opción 4: Primero solicita el nombre de un crédito que esté asignado al cliente. Luego realiza una búsqueda por nombre a la lista de créditos que posee el cliente (El método está especificado en ListaCreditos). En el caso de que lo encuentre, descuenta 1 de la cantidad de cuotas a pagar.

Opción 5: Primero solicita el nombre de un crédito que esté asignado al cliente. Luego realiza una búsqueda por nombre a la lista de créditos que posee el cliente (El método está especificado en ListaCreditos). En el caso de que lo encuentre, la cantidad de cuotas se torna 0, pagando el total de las cuotas.

Opción 6: Fin de la ejecución.

**Lista de clientes:**

Incluye los elementos de ListaClientes.h y de cliente.c, implementando una lista enlazada simple en el cual cada elemento contiene un puntero de struct Cliente y un puntero que referencia al siguiente elemento de la lista.

El primer método, “insertarNodoCliente” consiste en insertar un elemento al final de la lista y en el cual se reserva el espacio en memoria para el struct y se llama al de cliente “iniciar\_cliente()” para guardar el elemento principal. El elemento siguiente se deja nulo, de esa forma al encontrar el último elemento, el nuevo será insertado como sucesor.

En lo que respecta a los métodos de búsqueda, todos comparten una estructura similar a excepción de la búsqueda por id, el cual solo devuelve un elemento recorriendo de forma secuencial. El resto parten del criterio de devolver una lista de varios elementos(o al menos en los casos más realistas). Durante la ejecución del método se utilizan las siguientes variables:

1. ListaCliente lista\_a\_recorrer, que apunta a lista que se pasa por parámetro
2. ListaCliente lista\_resultado, que comienza siendo nulo
3. Lista ultimo\_elemento, último elemento de la lista resultado

Los métodos recorren a la lista “a recorrer”, en un loop(while) que finaliza al encontrarse con el elemento nulo, el cual se alcanza al final de la lista. Dentro del loop se encuentra la condición propia de cada búsqueda cuyo/s elemento/s se pasan por parámetro, como por ejemplo el nombre o el rango de edad(minima y maxima), haciendo una comparación con el/los elemento/os propios del nodo en el que se encuentra la variable lista\_a\_recorrer.

En caso de no encontrar ningún elemento en toda la lista que coincida con la condición, la variable lista\_resultado se devolverá siendo nula, tal y como se inicializó.

En el caso de que la condición se cumpla para un elemento de la lista, pueden darse dos situaciones diferenciadas por un “if/else”:

Si es el primer elemento encontrado: lista\_resultado reserva espacio en memoria con un malloc, guarda lista\_resultado->cliente = lista\_a\_recorrer->cliente y el elemento siguiente como nulo. La variable ultimo\_elemento se iguala a lista\_resultado(lo que significa que ahora apunta a esta ultima).

Caso contrario: Ahora la variable ultimo\_elemento->siguiente es la que se le reserva la memoria, lo que equivale a que en este punto los datos se guardarán al final de la lista. Esto para evitar tener que recorrer la lista para llegar al último elemento. Por lo tanto, seguido de esto, corresponde colocar ultimo\_elemento=ultimo\_elemento->siguiente, para que esta variable siempre esté apuntando al final de la lista. Acto seguido, la variable cliente de último\_elemento se igual al cliente del nodo actual y el elemento siguiente la variable se iguala a nulo.

Tras analizar las condiciones se avanza en la lista tal que lista\_a\_recorrer=lista\_a\_recorrer->siguiente.

El proceso se repite hasta cortar la condición del loop y finaliza retornando lista\_resultado.

**Credito:**

Incluye credito.h los elementos del struct Credito:

1. Monto Total(double)
2. Monto por cuota(double)
3. Cantidad de cuotas(int)
4. Cantidad de cuotas restantes(int)
5. Cliente correspondientes(Cliente)
6. Nombre (cha[100])

Se incorpora el método iniciar\_credito recibe como parámetro un cliente y toma por pantalla los datos del struct. Con ellos llama al método nuevo\_credito() pasandole los datos por parámetro. Es en este último método donde se reserva el espacio en memoria y se asignan los datos a las variables de los struct

//Completar: Si tiene el máximo de créditos

**ListaCredito:**

Incluye el archivo credito.h. Se implementa esta lista con el fin de asignar créditos a los clientes registrar todos los créditos que estén o hayan estado vigentes. Consiste en un struct que contiene el nodo crédito y el struct siguiente.

Implementa el método insertarNodoCredito, el cual recibe como parámetro la lista de créditos totales del tipo ListaCredito y el cliente de tipo Cliente, al cual debe estar asociado. Si el cliente tiene 3 elemento, devuelva la lista tal y como estaba. De lo contrario busca el final de la lista para ingresar el dato. Al encontrarlo, reserva el espacio en memoria y luego llama al método iniciar\_credito(), para guardar el resultado en el dato Crédito del nodo, y que quede asignado al cliente. Para finalizar, el siguiente elemento de la lista se guarda como nulo.

El siguiente método es BuscarCreditoByNombre() el cual recibe como parámetro una lista de créditos y un nombre a buscar. Retorna una lista de créditos cuyos nodos posean el mismo nombre que se pasó. Realiza una búsqueda secuencial e inserta los elementos encontrados en una “lista\_resultado”. En este método, siempre se busca insertar los elementos en la última posición. Para esto se guarda una variable ”ultimo\_agregado” que apunte siempre al final, con el fin de evitar recorrer la lista resultado cada vez que se inserte un nuevo elemento.

**Main:**

Menú:

1. Agregar cliente
2. Imprimir clientes
3. Buscar cliente por nombre
4. Buscar cliente por rango de edad
5. Buscar cliente por ID
6. Sub-menu de cliente
7. Cargar clientes
8. Salir

Implementa este menú con las siguientes variables:

1. Lista de creditos
2. Lista de clientes
3. Archivo (FILE)
4. Clientes

El menú se implementa con un do/while, analizando como condición de corte que la variable opción no sea igual a 8. Esta última variable varía con el método de c “scanf()”, asignándole por consola un valor entero.

Opción 1: Inserta un nuevo cliente con el método de ListaCliente: insertarNodoCliente, asignando el resultado en la variable Lista de créditos.

Opción 2: Imprime lista de clientes usando el método de ListaCliente: imprimirLista

Opción 3: Busca en la variable Lista de Clientes usando el correspondiente método de búsqueda a la condición. En este caso, utiliza el método BuscarByNombre() especificado en ListaCliente, pasando el nombre a buscar por consola. Al devolver la lista por nombre, imprime esta última y luego libera el espacio en memoria con el método free() de C.

Opción 4:Busca en la variable Lista de Clientes usando el correspondiente método de búsqueda a la condición. En este caso, utiliza el método BuscarByRangoDeEdad(), pasando por consola los parámetros de edad mínima y máxima. Al terminar la búsqueda, imprime el resultado y luego libera la memoria con el método free() de C.

Opción 5:Busca en la variable Lista de Clientes usando el correspondiente método de búsqueda a la condición. En este caso utiliza el método BuscarByID() especificado en ListaCliente, pasando la ID a buscar mediante la consola.Al terminar la búsqueda, imprime el resultado y luego libera la memoria con el método free() de C.

Opción 6: Solicita la Id a buscar, la cual debe ser ingresada por consola. Busca el elemento en la lista de clientes y, si lo encuentra, utiliza dicho cliente para ejecutar el método menuCliente() especificado en Cliente.

Opción 7: Abre el archivo registro.csv, en caso de que haya algún error lo imprime, cierra el archivo y terminar la ejecución. En el caso contrario, por cada empleado imprime por fila su apellido, nombre y id, siendo que cada elemento se ingresa en otra celda, uno al lado del otro.

Opción 8: Fin de la ejecución.

**Conclusiones:**

El trabajo final resultó desafiante, aunque no injusto. Se presentaron algunas dificultades debido a los “includes” simultáneos que planteamos, como por ejemplo en crédito y cliente. En ambos structs, poseen un elemento del struct contrario, generando dudas debido a la sobreescritura de los “includes” y las posibles falla (como que los métodos figuren como no definidos).

En nuestra opinión, hicimos un correcto uso de los punteros, aunque en la etapa final notamos que el código se podía mejorar agregando complejidad, como por ejemplo un header para las cuotas para determinar si están o no pagas. Por cuestiones de tiempo decidimos omitir esto último. Otro ejemplo es que al denotar el gran uso de algoritmos de búsqueda, hubiese sido óptimo implementar otra estructura que no sean lista simples(no eliminandolas del todo, pero si reemplazandola en elementos más extensos).

Aun con los defectos encontramos, creemos que esta versión más simple satisface los puntos que debíamos cubrir para el trabajo, pese a que en sí el código es muy mejorable. El menú, si bien al principio se tenga que hacer algunas aclaraciones a la hora de interactuar con un cliente en específico, es sencillo y fácil de usar y se complementa muy bien con el resto de los elementos. Por lo tanto, en términos del uso de la interfaz y del cumplimiento de las consignas consideramos el trabajo resultó satisfactorio.