



UNIVERSIDAD DE GUADALAJARA

Red Universitaria e Institución Benemérita de Jalisco

Rodriguez Rentería Jesus Alejandro

Código:215510307

Materia: Seminario de solución de problemas de
estructuras de datos II

Profesor: Gómez Anaya David Alejandro

Sección: D16

Sede: Centro Universitario de Ciencias Exactas e
Ingenierías (CUCEI)

Fecha:09 de octubre del 2020

1. Planteamiento del problema.

Almacenar todos los registros los registros de una lista dinámica, instanciados en memoria principal y recopilarlos en un archivo texto (memoria secundaria) asegurando la persistencia de los registros

2. Objetivos: Lograr la persistencia de los registros:

1. Almacenar los registros de una lista en memoria secundaria al cerrar el programa.
2. Al ejecutar el programa, cargar los registros que se encuentren en memoria secundaria, a memoria principal.

3. Marco teórico.

En el desarrollo de la actividad se hace uso de archivos de texto y delimitadores.

Los archivos o ficheros son la forma en la que C++ permite el acceso al disco.

Todos los procesos tienen abiertos, por defecto, los archivos 0(entrada), 1(salida) y 2(salida de errores), de manera que en C++ se corresponden con los objetos *cin*, *cout* y *cerr*.

Apéndice 1 imagen.

un **delimitador** es un separador que define piezas individuales de datos en un archivo, protocolo de comunicaciones, u otro flujo de datos. El separador puede ser tan simple como un único carácter, como una coma, o tan complejo como una secuencia definida de caracteres de control no imprimibles.

4. Desarrollo.

El código cuenta con 3 clases las cuales son: Clase Nodo, clase Lista y Clase Usuario.

La clase Nodo cuenta con 3 atributos: *Nodo* Siguiente*, *Nodo *anterior* y *Usuario * Registro*

La clase es responsable de crear los lazos entre los elementos de lista, cuenta con 8 métodos los cuales con

1. Constructor y constructor con parámetros: El constructor sin parámetro inicializa todos los punteros a NULL mientras que el constructor con parámetros (apuntador a Usuario y apuntador a siguiente) y hace las asignaciones a los atributos.
2. Set usuario: tiene como parámetro un apuntador a un objeto de tipo usuario que lo asigna a Registro.
3. Set siguiente: ingresa el apuntador al siguiente nodo
4. Get siguiente: avanza a el siguiente Nodo
5. Get anterior: obtiene el nodo que se encuentra atrás
6. Get usuario: obtiene la dirección de memoria del objeto de tipo usuario
7. Set anterior: ingresa el apuntador a el nodo anterior

Clase Lista, cuenta solo con un atributo privado Nodo *primero (se encarga de el inicio de la lista) los métodos con los que cuenta son:

1. constructor
2. ingresa(Usuario *):Ingresa el nodo al principio
3. ingresaf(Usuario *); evalúa si la lista esta vacía y llama a ingresa() de lo contrario ingresa el nodo al final (lo hice de esta manera ya que al hacer el recorrido conozco el nodo anterior y se hace la asignación)
4. validacion_correo(string): Ingresa una cadena de caracteres y revisa si cuenta con un @ y que el correo no se encuentre ya en uso (regresa un valor booleano)
5. validacion_usuario(string): Ingresa una cadena de caracteres y revisa que el usuario no se encuentre en uso (regresa un valor booleano)
6. eliminar (Nodo *): pide la contraseña del nodo a eliminar, de ser esta correcta, se realiza modificaciones a los Nodos vecinos en sus apuntadores para que el registro a eliminar quede aislado y posteriormente eliminado con la palabra “delete”.
7. imprimir_todo(): Recorre todos los elementos de la lista y los manda a imprimir
8. busqueda_impre(Nodo*): manda e imprimir al nodo que se pasa por referencia
9. cambio_de_nombre(Nodo *,string): utiliza el nodo por parámetros y llama a la función modificar de Usuario
10. cambio_de_contrasena(Nodo *,string); utiliza el nodo por parámetros y llama a la función modificar de Usuario
11. cambio_de_usuario(Nodo *,string); utiliza el nodo por parámetros y llama a la función modificar de Usuario
12. cambio_de_correo(Nodo *,string); utiliza el nodo por parámetros y llama a la función modificar de Usuario
13. *verficar_correo_usuario(string): verifica que la cadena de caracteres tenga un @, si es positivo revisa en la lista si ese correo esta en uso, si no cuenta con un @ busca en la lista pero en el apartado de usuarios, al final el método regresa la dirección de memoria del objeto. Apéndice 1
14. *inicio_seccion():Solicita el correo o usuario, llama a la función verficar_correo_usuario(string) si obtiene la dirección de memoria, solicita la contraseña y compara que sea correcta.

Creación de archivos en clase lista

15. Void Lista :: toFile() Encargado de crear o abrir el archivo de texto con nombre Accounts.txt en modo trunc(esto para que pueda actualizar los pocos registros sin necesidad de tener duplicados en el archivo).

Se realiza un recorrido a la lista y se mandan todos los objetos de tipo usuario al tofile() de la clase usuario que concatena todo en una cadena de caracteres, una vez terminado los registros se cierra el archivo.

Consulta apéndice 2.

16. Void Lista :: showall(): Función que se encarga de leer el archivo de texto account.txt, revisa los registro del archivo y manda llamar al el constructor Usuario(string), esto pasa los registros de memoria secundaria a memoria principal y los agrega a la lista.

Clase Usuario cuenta con 4 atributos: string nombre, string usuario, string correo y string password. Los métodos son:

1. Constructor sin parámetro y con parámetro
2. void print(): imprime los atributos del objeto usuario
3. void print2(): Imprime y enumera los atributos del objeto usuario
4. void mod_nombre(string): modifica el nombre por la nueva cadena de caracteres que ingresa por parametros
5. void mod_usuario(string): modifica el usuario por la nueva cadena de caracteres que ingresa por parámetros
6. void mod_correo(string): modifica el correo por la nueva cadena de caracteres que ingresa por parámetros
7. void mod_password(string): modifica el password por la nueva cadena de caracteres que ingresa por parámetros
8. string recupera_correo() devuelve el correo
9. string recupera_usuario(): devuelve el usuario
10. string recupera_password(): devuelve el password

Creación de archivos en clase Usuario

11. Usuario (string cadena): El constructor está diseñado para obtener una cadena de caracteres en la cual se encuentran concatenados todos los atributos para instanciar un objeto de tipo usuario. Se realiza el recorrido del arreglo en busca de los delimitador “|” con el cual se separan los atributos.

12. String Usuario :: toFile(): Función que tiene los atributos de un objeto usuario y lo concatena en una sola cadena de caracteres agregando los delimitadores, esto para tener conocimiento de donde terminar los atributos y por ultimo regresa la cadena

Se inicia con un main el cual contiene un switch se 4 opciones:

1. Crear una cuenta
2. Acceder a una cuenta
3. Mostrar todo
4. Salir

El cual cuenta con 3 funciones que son las que se encargan de llamar a los métodos de las clases según el orden que quiera el usuario.

5. Pruebas y resultados.

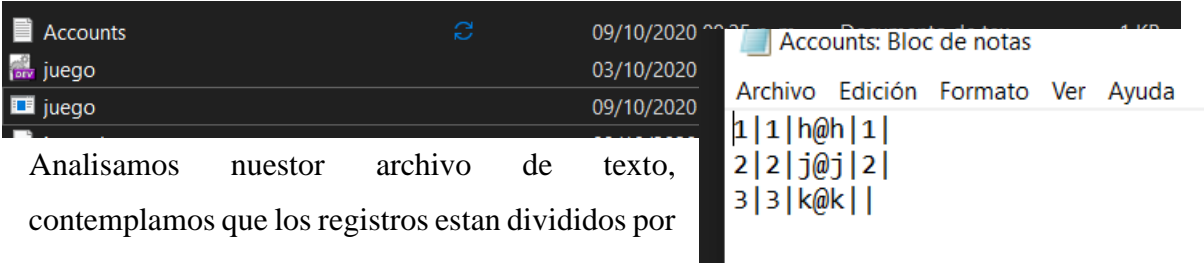
Iniciamos el programa en el menú

```
Menu principal
1. Crear una cuenta
2. Acceder a una cuenta
3. Mostrar todo
4. Salir
```

Creamos varios registros y utilizamos la función de mostrar todo para poder visualizar

```
Mostrar todo
Nombre ----> 1
Usuario ----> 1
correo ----> h@h
password ----> 1
-----
Nombre ----> 2
Usuario ----> 2
correo ----> j@j
password ----> 2
-----
Nombre ----> 3
Usuario ----> 3
correo ----> k@k
password ----> 
-----
Presione una tecla para continuar . . .
```

Cerramos el programa y revisamos la carpeta, verificamos que se creó nuestro archivo de texto



Analizamos nuestro archivo de texto, contemplamos que los registros están divididos por saltos de línea y los campos por delimitadores

Corremos nuevamente el programa y podemos contemplar nuevamente en mostrar todo que se a reconocido a los registros del archido de texto

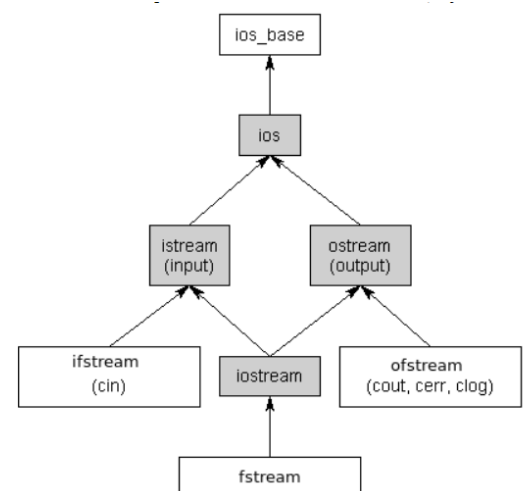
```
Mostrar todo
Nombre ----> 1
Usuario ----> 1
correo ----> h@h
password ----> 1
-----
Nombre ----> 2
Usuario ----> 2
correo ----> j@j
password ----> 2
-----
Nombre ----> 3
Usuario ----> 3
correo ----> k@k
password ---->
-----
Presione una tecla para continuar . . .
```

6. Conclusiones.

La aplicación de delimitadores en el archivo implico que se crearan nuevas funciones muy simples, como lo son constructores, concatenar los atributos de un objeto para su posterior guardado, creación y lectura del archivo. Basta con unas pocas modificaciones al código fuente del programa pasado, para que este cumpliera con los requisitos de esta práctica.

7. Apéndice(s).

Apéndice 1 el primero pertenece a la clase **ifstream**, que a su vez desciende de **istream** (flujo de entrada). Los dos últimos pertenecen a la clase **ofstream**, que desciende de la clase **ostream** (flujo de salida). Una jerarquía aproximada puede verse a continuación.



Apéndice 2

Función que sobrescribe en el archivo se ejecuta al finalizar al programa

```
void Lista::toFile(){
    Nodo *p = primero;
    ofstream fout("Accounts.txt",ios::trunc);//

    while(p!=NULL){
        fout << p->get_usuario() ->toFile() <<endl;
        p=p->get_siguiente();
    }
    fout.close();
}
```