Informe

Para dar con la solución al proyecto 2 se implementaron 3 clases listas enlazadas mediante datos punteros (o apuntadores) cada una con sus respectivos nodos, donde en cada clase lista se organizaron los métodos propios de la clase para cada sección correspondiente a la entrada.

Para la sección "**Usuarios**" se tendrá :

- Una clase **nodoUsers** donde donde albergara los 4 datos datos personales del usuario como : nombre, email, contraseña y el campo nodoSiguiente característico de la implementación de listas enlazadas y con su constructor que inicializa los datos cada vez que se añade un elemento nuevo a la lista respectivamente.

- Una clase **ListaUsers** que tendrá un objeto nodoUsers quien sera el apuntador al primer elemento de la clase. La listaUsers tiene sus métodos definidos los cuales se mostraran sus prototipos y una explicación breve de lo que hacen :

1. *bool VaciaUsers() : Función que verifica si la lista esta vacía o no, retornando un valor booleano que lo indica , verdadero si lo esta, en caso contrario falso .

2. void InsertarUsers(string name,string mail,string name_user,string pass) : Acción que crea un nodo y en el inserta los valores leídos del archivo los cuales son nombre, email, nombre de usuario y contraseña.

3. bool ValidarRegistro(string email, string nom_usua) : Función que verifica si un usuario estuvo registrado anteriormente, solamente se verifica el email y el nombre de usuario ya que estos serian los datos relevantes que no se pueden repetir en el registro respectivamente.

4. bool ValidarUsuario(string& usua, string pass) : Función que verifica si un usuario ya esta registrado y por ende pueda loguearse, aquí básicamente se verifica el email o el nombre de usuario , y la contraseña. Como se puede notar el valor "usua" se pasa por referencia puesto que un usuario puede registrare con el email o nombre de usuario entonces se verifican cualquiera de los dos y si se logueó con el email o nombre de usuario igual lo modificamos a que sea nombre de usuario ya que en main() nos conviene este tipo de cosas.

5. void BuscarSeguir(string user,string following[],string& op) : Acción que llena un arreglo listando a todos los nombres de usuarios que están registrados actualmente excluyendo el usuario que se logueo puesto que se utilizara para cuando el usuario indique si quiere seguir a otro usuario .

6. void GuardarInfoUsuarios() : Acción que guarda toda la información de listaUsers cuando el usuario lo requiera, se copian todos los datos de los usuarios que fueron leídos por el archivo además de los nuevos que se registraron.

Éstas serían todas las funciones que listaUsers requiere para realizar todas las tareas para llegar a la solución, cada una realizando en específico su funcionalidad antes descrita.

-* Cada clase de lista tendrá una función Vacia(), solamente describiré su funcionalidad aquí ya que las demás son análogas.

Para la sección "**Siguiendos**" se tendrá :

- Una clase **nodoSig** donde estarán los campos siguiendo1 y siguiendo2 los cuales representa la relación siguiendo1 "sigue a" siguiendo2 , y siguiendo2 "tiene como seguidor" siguiendo1 respectivamente. Con su apuntador al siguiente nodo y el constructor quien inicializa cada elemento de la lista.

- Una clase **listaSig** que tendrá un objeto de la clase nodoSig para que este sea el apuntador al primer nodo de la lista, listaSig posee unos métodos miembro que realizan una tarea en especifico , los cuales se listan a continuación:

1. bool VaciaSig() :

2. void InsertarSiguiendos(string a, string b) : Acción que crea un nuevo nodo e inserta los valores del siguiendo1 y siguiendo2 de acuerdo a la relación A --> B (A "sigue a" B). Los inserta en orden ascendente con respecto a siguiendo2 , osea ordena en orden alfabético los siguiendos de un usuario.

3. void InsertarSeguidores(string a, string b) : Análoga a la acción anterior con la diferencia que ordena con respecto a siguiendo1, en efecto ordena en orden alfabético los seguidores de un usuario.

4. void VerSeguidores(string user) : Acción que muestra los seguidores de un usuario dado.

5. void VerSiguiendos(string user) : Acción que muestra los siguiendos de un usuario.

6. void DejarDeSeguir(string user) : Acción que permite a un usuario dejar de seguir a un usuario que el halla seguido anteriormente, en caso contrario se le notificara que el no sigue a ningún otro usuario.

7. bool ComprobarSiSeguir(string user,string user2) : Función que comprueba si un usuario sigue a otro , es básicamente para no repetir siguiendo en la lista. Esto lo hace recorriendo las listaSig y verifica si ya un usuario sigue a otro retornando true , en caso contrario retorna false.

8. void BuscarTweetSig(string nom_usua,string Arr2[],int cont,int& k) : Acción que buscara los siguiendos de un usuario y listarlos en un arreglo Arr2[] para luego realizar la acción verMuro que se describirá mas adelante.

9. void GuardarInfoSiguiendos() : Acción que guarda la información de los siguiendos en el archivo cada vez que un usuario lo notifique.

Para la sección "**Tweets**" se tendrá :

- Una clase **nodoTweets** para albergar el el nombre de usuario y su respectivo tweet, un apuntador al siguiente tweet y el constructor.

- Una clase **ListaTweet** que contendrá un objeto de la clase nodoTweet que sera el apuntador al primer elemento la lista creada y varios métodos miembros que harán una tarea en especifico :

1. bool VacioTweet().

2. void InsertarTweet(string _t,string _t2) : Acción que inserta un nuevo elemento a la listaTweet el cual contendrá el nombre del usuario y su tweet correspondiente.

3. void EliminarTweet(string user,int cont) : Acción que primero busca si ese usuario tiene tweets que pueda eliminar , si ese usuario no ha creado tweets se le notificara que no tiene tweets que eliminar , en caso contrario se le mostrara todos sus tweet que el creo para eliminar el que el desee.

4. void VerMuro(string nom_usua,string tweetsig[],int k) : Acción que muestra los tweets de los siguiendos de un usuario mas los propios que el creo. Esto se hace mediante una búsqueda de los tweets de los usuarios que están guardados en el arreglo tweetsig[] y se van buscando los tweets para cada usuario almacenado en el arreglo, luego se buscan los tweets del usuario que se logueó.

5. void GuardarTweets() : Acción que guarda la información de los tweets cuando un usuario lo notifique.

   Para dar con la utilización de estas estructuras fue necesario crear objetos donde se describirá su funcionalidad :

- 1 objeto de la clase claseUsers , donde de guardaba toda la información de los usuarios

- 2 objetos de la clase claseSig , en el primero se guardaba la información ordenada de forma ascendente de los siguiendos con respecto a siguiendo1 , y en el segundo se guardaba ordenados pero con respecto a siguiendo2.

- 1 objeto de la clase claseTweets , donde se guardara el nombre de usuario y su tweet correspondiente.

Ya sabiendo la estructura a utilizar procedemos a describir paso a paso como se manipularon estas y como se emplearon las validaciones con respecto a la interacción con el usuario. Primero para la entrada del archivo donde se obtuvo la información previa de usuarios con 3 ciclos while en donde de una vez se insertaban los valores leídos del archivo correspondientes a las segmentos “usuarios”, ”siguiendos” y “tweets” , para la validación de las opciones de los menú y submenus se emplearon dentro de un ciclo repetir y estas opciones se repiten hasta que el usuario ingrese una opción válida , en algunas ocasiones se utilizo la instrucción “goto” para validar estas opciones.

Para registrarse : se le pide al usuario que ingrese los datos correspondiente al registro , y se invoca a la función validar registro para que no se repitan usuarios.

Para el login : se le pide a usuario nombre de usuario o email , y la contraseña , estos se validan si el usuario estuvo registrado anteriormente.

Para ver seguidores : una vez el usuario logueado e ingresa la opción ver seguidores , de nuevo se le pedirá que ingrese el nombre del usuario que el desea ver sus seguidores , estos se irán listando en orden alfabético.

Para ver siguiendos : una vez el usuario logueado e ingresa la opción ver siguiendos , de nuevo se le pedirá que ingrese el nombre del usuario que el desea ver sus siguiendos , estos se irán listando en orden alfabético.

Para crear tweet : Se le informa a usuario el tweet a crear, se valida que no se pase de 140 caracteres , si excede el numero de caracteres , se le pide nuevamente que ingrese el tweet a crear.

Para eliminar tweet : Se le irán listando uno a uno en orden cronológico los tweets creados por un usuario que se logueo, si ese usuario no tiene tweets que eliminar se le notifica que no tiene tweets y volvemos al submenú, si el usuario tiene tweets se le iran listando en orden cronológico con opciones de 1 hasta n , y la opción n+1 por si no quiere eliminar ningún tweet.

Para seguir : se le irán listando los usuarios que el puede seguir, si el ingresa la opción de seguir un usuario y ya el lo seguía anteriormente, se le notifica que ya sigue ese usuario y lo invitamos a que lo intente nuevamente, y también se mostrara la opción salir si no quiere seguir a nadie .

Para dejar de seguir : Se le listaran los usuarios que el seguía anteriormente con la opción de 1 hasta n para elegir que usuario eliminar y n+1 para salir .

Para ver muro : primero se hace una búsqueda de los suguiendos de un usuario , luego se listan en orden cronológico los tweet de los usuarios siguiendos y finalmente se listan lo tweets del usuario logueado.

Para guardar archivo: una vez ingresada esta opción se hace invocaciones a las 3 acciones que guardan la información personal de los usuarios , los siguiendos y los tweets.