

Práctica 4: Gestor de correo *fdimail*

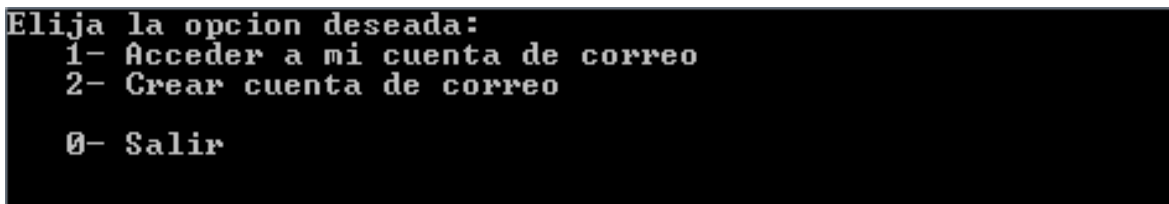
Fecha de entrega: 17 de mayo de 2015

1. Funcionalidad de la aplicación a desarrollar

Prototipo de un gestor de correo local en modo consola, que permite a nuevos usuarios registrarse, y a los usuarios ya registrados enviar, recibir y contestar correos de manera sencilla.

1.1. Registro e inicio de sesión en el gestor de correo

El menú principal del sistema permite realizar dos opciones: acceder a una cuenta de correo ya existente o crear una cuenta de correo nueva. En ambos casos se pide un nombre de usuario y contraseña.



```
Elija la opcion deseada:  
1- Acceder a mi cuenta de correo  
2- Crear cuenta de correo  
  
0- Salir
```

Pantalla del menú principal del gestor de correo

Cuando se elige la opción de **acceder a una cuenta de correo ya existente**, se comprueba que el usuario exista en la base de usuarios del sistema y que la contraseña sea correcta. Si una de estas dos condiciones no se cumple, se muestra un error y el menú principal aparece de nuevo.

Cuando se elige la opción de **crear una nueva cuenta**, se comprueba si el usuario que se quiere registrar ya existe en la lista de usuarios del sistema. Si el nuevo usuario ya existe se mostrará un error y se volverá al menú principal.

Si no ha habido error, y el usuario ha podido acceder a su cuenta o crear una cuenta nueva, se inicia una sesión, y se muestra la pantalla del menú principal de la sesión.

1.2. Pantalla principal de una sesión del gestor de correo

El menú principal de una sesión muestra la bandeja de entrada correspondiente al usuario de la sesión y las opciones asociadas.

```

Correo de ginebra@fdimail.com
-----Bandeja de entrada-----
L N      EMISOR              ASUNTO                      FECHA
-----
* 1 - lancelet@fdimail.com   Besos y abrazos           2015/3/23
* 2 - arturo@fdimail.com     ¡Mesa en oferta!          2015/3/23
  3 - arturo@fdimail.com     Petición a Lancelot        2015/3/23
-----
Elija una opcion:
1- Leer correo
2- Enviar correo
3- Borrar correo
4- Ver bandeja de salida
5- Lectura rapida de correos sin leer
0- Cerrar sesion
-----
Introduzca una opcion:

```

Pantalla del menú principal de una sesión del gestor de correo

La lista de correos recibidos se muestra numerada y en orden inverso de llegada (de más a menos reciente). Los correos no leídos se marcan con un asterisco. Se dispone además las siguientes opciones:

- **Leer correo:** Solicita al usuario el número del correo a mostrar, comprueba dicho número, y si no ha habido error, muestra en una nueva pantalla el correo junto con las opciones que se pueden realizar sobre él (ver sección 1.3).
- **Enviar correo:** Permite al usuario escribir y enviar un nuevo correo. Se solicitarán los datos del nuevo correo en el siguiente orden:
 - El destinatario (sólo uno).
 - El asunto del correo (una línea).
 - El cuerpo del correo (varias líneas).

El resto de datos de un correo (identificador, emisor y fecha) se calcularán automáticamente (ver detalles de implementación).

Si no se ha podido enviar el correo al destinatario (porque no existe o porque sus listas están llenas), se indicará mediante un mensaje de error que el envío a dicho destinatario no se ha podido realizar.

Si un correo se ha enviado con éxito al destinatario, dicho envío quedará reflejado en dos sitios: la bandeja de salida del emisor y la bandeja de entrada del destinatario.

- **Borrar correo:** Solicita un número de correo al usuario, comprueba dicho número, y si no ha habido error, lo borra.

- **Lectura rápida de correos sin leer:** Mostrará todos los correos sin leer de la bandeja correspondiente ordenados en primer lugar por asunto y después por fecha. **OJO:** Para ordenar por asunto debes ignorar los “Re: ” de los correos que sean una respuesta.
- **Ver bandeja de salida:** Cambia la vista para mostrar la bandeja de salida. La bandeja de salida se muestra exactamente igual que la de entrada, con dos excepciones: los correos que muestra son los enviados por el usuario y en lugar de mostrar la opción “Ver bandeja de salida” mostrará la opción “**Ver bandeja de entrada**”, que nos permite volver a la bandeja de entrada.

1.3. Pantalla de lectura de correo

Cuando el usuario elige la opción **Leer correo** se muestra en una nueva pantalla el correo elegido y las opciones disponibles:

```
De: arturo@fdimail.com                2015/3/23 <20:25:32>
Para: lancelet@fdimail.com
Asunto: Comprar mesa de reuniones

Hola,
he pensado que podíamos comprar una mesa redonda para las reuniones...
¿Cómo lo ves?

Un saludo,
Arturo

-----
Elija una opcion:
1- Contestar correo
0- Volver a la bandeja
-----
Introduzca una opcion:
```

Pantalla donde se muestra el correo y las opciones del correo

Las opciones del menú:

- **Contestar correo:** Generará un nuevo correo como contestación del correo original que se está mostrando. Los datos del nuevo correo contestación se rellenarán de la siguiente manera:
 - El emisor será el usuario de la sesión.
 - El receptor será el emisor del correo original. No se permite añadir nuevos destinatarios a la contestación de un correo.
 - El asunto será el mismo, pero se le añadirá al principio la cadena “Re: ”.
 - El cuerpo del correo se pedirá al usuario (varias líneas), y a lo escrito por el usuario se le añadirá al final todo el contenido del correo original (incluyendo los datos de emisor, destinatarios, fecha de envío y asunto).
 - El resto de datos de un correo (identificador y fecha) se calcularán automáticamente (ver detalles de implementación).

Cuando se contesta a un correo, éste se envía al destinatario como un correo normal, y se guarda una copia del mismo en la bandeja de salida del usuario que lo ha contestado. Igual que al enviar un correo nuevo, si no se puede realizar la entrega se mostrará un mensaje avisando al usuario.

- **Volver a la bandeja:** Se volverá a la bandeja de correos (de entrada o salida) en la que se estaba antes de ejecutar la opción de leer un correo.

2. Visión general de la implementación: módulos de la aplicación y relación entre ellos

Hasta ahora hemos hablado de los requisitos de funcionamiento de la aplicación. Ahora vamos a explicar los requisitos de implementación.

La aplicación requiere implementar los tipos de datos `tGestor` y `tSesion` que implementarán todas las funcionalidades de nuestra aplicación.

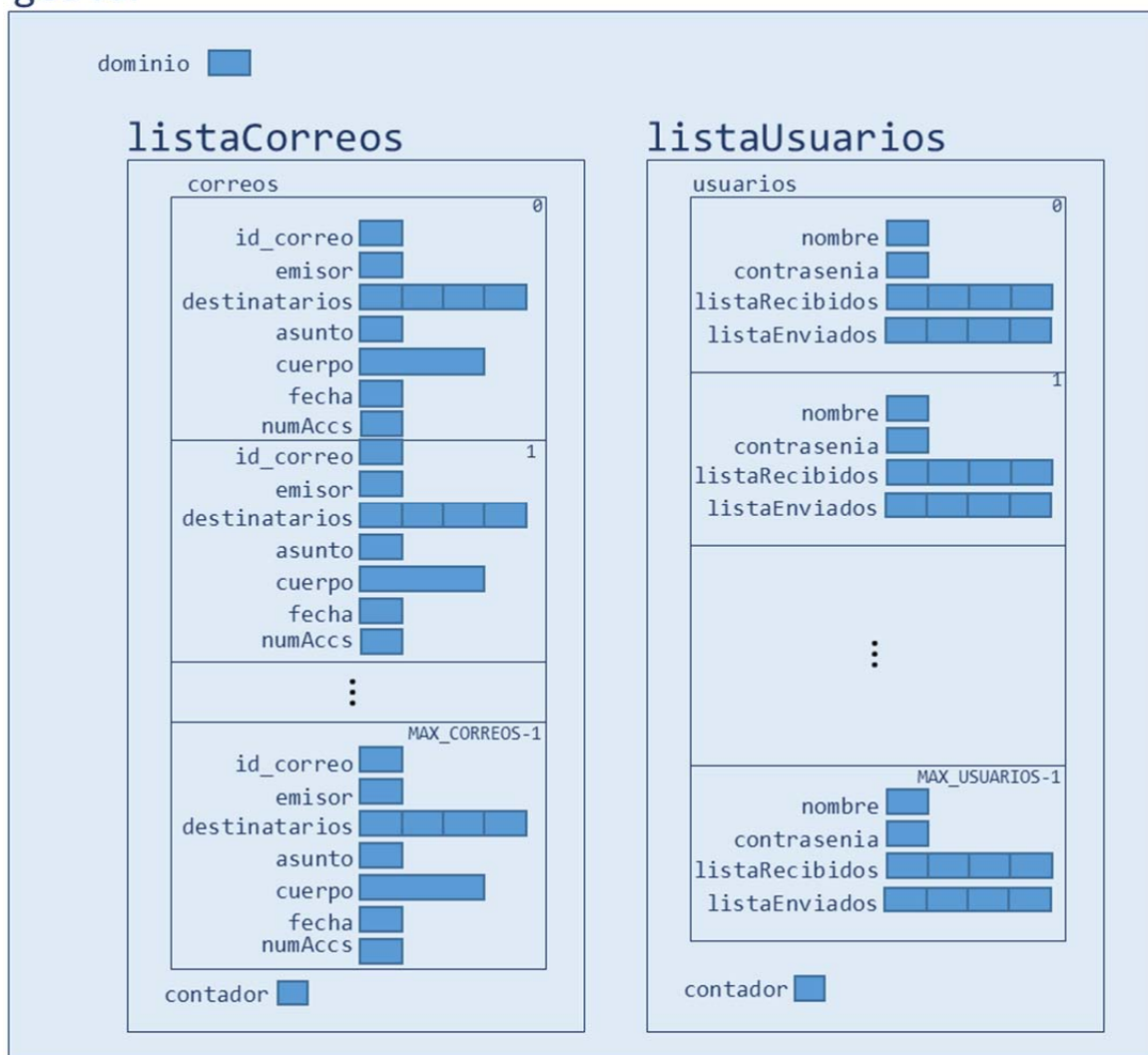
El tipo de datos `tGestor` tendrá, entre otros campos, la lista de correos (tipo `tListaCorreos`) y la lista de usuarios (`tListaUsuarios`). Como es natural, esto supone que tenemos que implementar también los tipos `tCorreo` y `tUsuario`.

El tipo `tSesion` tendrá un enlace al gestor de correo y otro al usuario de la sesión. Las siguientes figuras muestran una variable de tipo `tGestor` y de tipo `tSesion` para hacernos una idea.

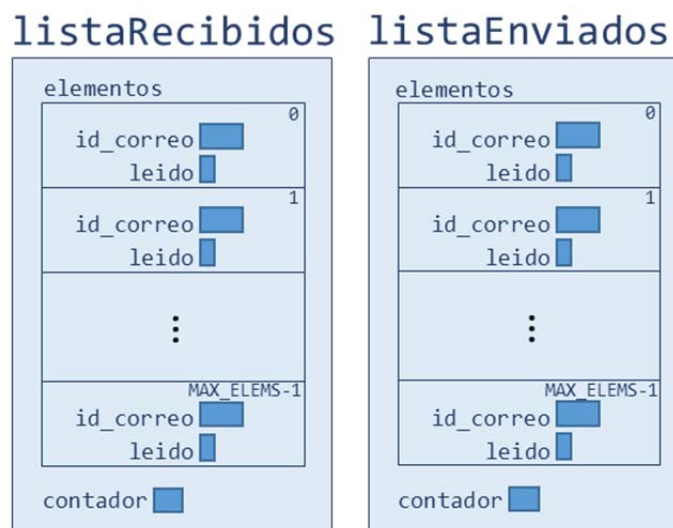


Variable de tipo `tSesion`

gestor

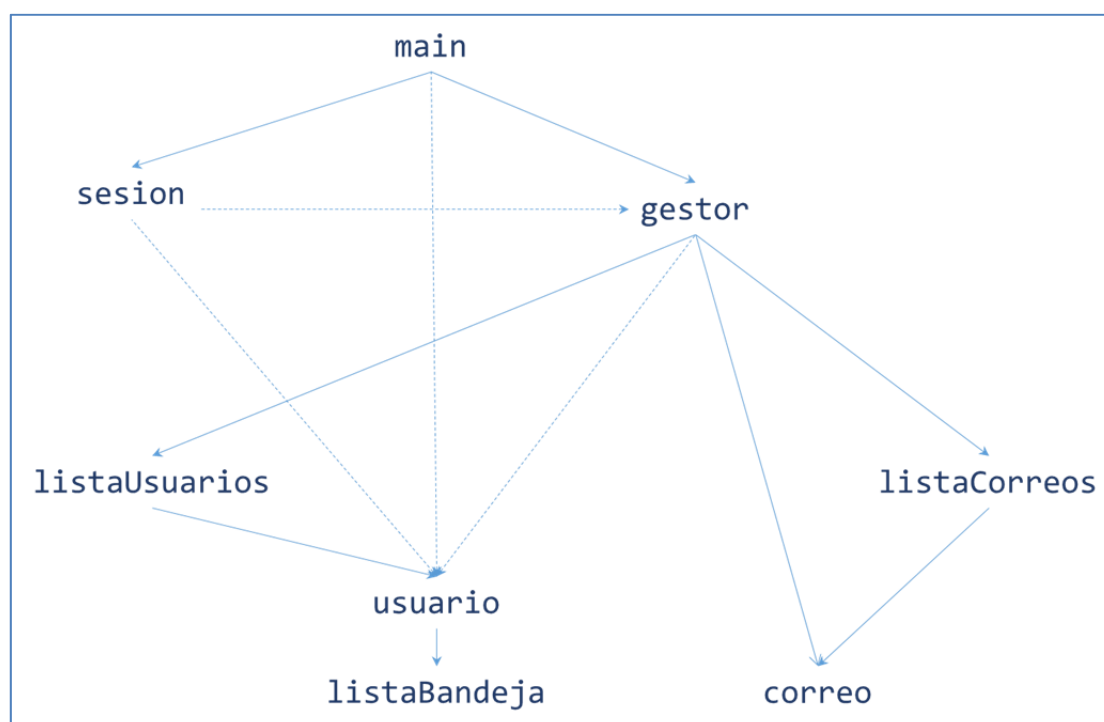
Variable de tipo `tGestor`

Es importante tener en cuenta que **EL GESTOR SÓLO ALMACENA UNA COPIA DE CADA CORREO**, más concretamente, la almacena en la lista de correos. Por su parte, un usuario no almacena sus correos recibidos y enviados, sino que tanto su bandeja de entrada como su bandeja de salida serán una lista de registros que contendrán, entre otras cosas, un identificador que permitirá localizar el correo en la lista de correos. Las siguientes figuras muestran una representación de los campos `listaRecibidos` y `listaEnviados` que tendrá el usuario, ambos campos serán de tipo `tListaBandeja` y requerirán definir un tipo de datos `tElemBandeja`.

Variables `listaRecibidos` y `listaEnviados` de tipo `tListabandeja`

2.1. Organización en módulos

La práctica deberá organizarse en módulos, con un módulo independiente para cada tipo principal. Se deben definir al menos los siguientes módulos:



Grafo de dependencias entre módulos

Módulo Correo

Declara el tipo de datos `tCorreo` para guardar la información de un correo:

- Un identificador **único** (`string`). Para que el identificador sea único, se compone de la concatenación del emisor y la fecha (p.e. *pepe@fdimail.com_143456443*).
- La fecha (`tFecha`). Ver más adelante.
- El emisor y el destinatario (`string`). Corresponderán con identificadores de usuarios en el sistema.
- El asunto y el cuerpo del correo (`string`).
- Número de usuarios que tienen acceso al correo: número de destinatarios + 1 (el emisor).

Ofrece al menos los siguientes subprogramas:

- `void correoNuevo(tCorreo & correo, const string & emisor)`: Recibe un identificador de emisor y devuelve un correo con todos sus datos rellenos según se ha explicado en la sección 1.3.
- `void correoContestacion(const tCorreo & correoOriginal, tCorreo & correo, const string & emisor)`: Recibe un identificador de emisor y el correo original que se va a contestar, y devuelve un correo con todos sus datos rellenos según se ha explicado en la sección 1.3.
- `void guardar(const tCorreo & correo, ofstream & archivo)`: Dado un flujo de archivo de salida (ya abierto), escribe en el flujo los datos que corresponden al correo (ver ejemplo de la sección 3.2).
- `bool cargar(tCorreo & correo, ifstream & archivo)`: Dado un flujo de archivo de entrada (ya abierto), lee los datos que corresponden a un correo y lo devuelve. Devuelve `false` sólo si el correo cargado no es válido.
- `int disminuirNumAccs(tCorreo & correo)`: Resta uno al número de usuarios con acceso al correo y devuelve el contador actualizado.
- `string to_string(const tCorreo & correo)`: Devuelve un `string` con el contenido completo del correo en el formato para mostrarlo en la consola (ver pantalla de la sección 1.3).
- `string cabecera(const tCorreo & correo)`: Devuelve un `string` que contiene la información que se mostrará en la bandeja de entrada/salida: emisor, asunto y fecha sin hora (ver pantalla de la sección 1.2).

Módulo ListaCorreos

Declara el tipo de datos `tListaCorreos` para gestionar una lista de correos. Es **IMPORTANTE** que tengas en cuenta que esta lista se encuentra **ordenada por el identificador de correo**. Debes implementarla como una lista de tamaño variable.

Ofrece al menos los siguientes subprogramas:

- `inline void iniciar(tListaCorreos & correos){ correos.contador= 0; };`
- `inline bool llena(const tListaCorreos & correos)`
`{ return correos.contador == MAX_CORREOS; };`
- `inline int longitud(const tListaCorreo & correos)`
`{ return correos.contador; };`
- `void guardar(const tListaCorreos & correos, const string & nombre):` Guarda la lista de correos en el fichero de correos nombre (ver ejemplo de la sección 3.2).
- `bool cargar(tListaCorreos & correos, const string & nombre):` Carga de la lista de correos desde el fichero de correos nombre.
- `bool insertar(tListaCorreos & correos, const tCorreo & correo):` Dado un correo, si hay espacio en la lista, lo coloca en la posición de la lista que le corresponda de acuerdo con su identificador y devuelve `true`. Si no lo ha podido insertar devuelve `false`.
- `bool buscar(const tListaCorreos & correos, const string & idMail, int & pos):` Dado un identificador de correo y la lista, devuelve, si dicho identificador existe en la lista, su posición y el valor `true`, y si no existe en la lista, la posición que le correspondería y el valor `false`.
- `bool eliminar(tListaCorreos & correos, const string & idMail):` Dado un identificador de correo y la lista, elimina de la lista el correo correspondiente al identificador (¡sin dejar huecos!). Devuelve `false` si el correo no existía, en otro caso devuelve `true`.

Módulo ListaBandeja

Declara los tipos de datos `tElemBandeja` y `tListaBandeja` para gestionar las bandejas de entrada/salida de los usuarios. El tipo `tElemBandeja` debe tener los siguientes campos:

- Un identificador **único** (string). Para que el identificador sea único, se compone de la concatenación del emisor y la fecha (p.e. *pepe@fdimail.com_143456443*). Este identificador coincide con el identificador de `tCorreo`.
- Un booleano que indica si el correo ha sido leído o no (por el propietario del registro).

El tipo `tElemBandeja` es tan sencillo que no tiene operaciones asociadas.

La lista `tListaBandeja` será una lista de tamaño variable de registros. Es **IMPORTANTE** que tengas en cuenta que esta lista no tiene ningún orden especial, sino que se encuentra **ordenada según “el orden de llegada” de los registros**.

Ofrece al menos los siguientes subprogramas:

- `inline void inicializar(tListaBandeja & listElems)`
 `{ listElems.contador = 0; };`
- `inline bool llena(const tListaBandeja & listElems)`
 `{ return listElems.contador == MAX_ELEMS; };`
- `inline int longitud(const tListaBandeja & listElems)`
 `{ return listElems.contador; };`
- `void guardar(const tListaBandeja & listElems, ofstream & archivo):` Dado un flujo de archivo de salida (ya abierto), guarda los datos de los elementos de la lista (ver ejemplo de la sección 1.3).
- `void cargar(tListaBandeja & listElems, ifstream & archivo):` Dado un flujo de archivo de entrada (ya abierto), lee los datos que corresponden a una lista y la devuelve.
- `bool insertar(tListaBandeja & listElems, const tElemBandeja & elem):` Dado un elemento lo inserta al final de la lista. Si la lista está llena devuelve `false`, en otro caso devuelve `true`. Este subprograma se ejecutará cuando un usuario envíe un correo, ya que se insertará un nuevo elemento en su bandeja de salida, y también en las listas de las bandejas de entrada de cada uno de los destinatarios del correo.
- `int buscar(const tListaBandeja & listElems, const string & idMail):` Dado un identificador de correo y la lista, devuelve, si dicho identificador existe en la lista, su posición, y si no existe devuelve -1.
- `bool eliminar(tListaBandeja & listElems, const string & idMail):` Dado un identificador de correo, lo busca en la lista y si lo encuentra lo elimina de la lista (¡sin dejar huecos!). Si no lo encuentra, devuelve `false`, en otro caso devuelve `true`. Este subprograma representa la acción de borrar un correo de una de sus bandejas del usuario. **OJO:** esta operación sólo supone que el elemento se elimina de la bandeja, pero el correo puede seguir existiendo en la lista de correos del gestor.
- `bool correoLeido(tListaBandeja & listElems, const string & idMail):` Dado un identificador de correo, lo busca en la lista y pone el indicador de leído a `true`. La operación devuelve un booleano indicando si se encontró o no el identificador.

Módulo Usuario

Declara el tipo de datos `tUsuario` para guardar la información de un usuario:

- El nombre o identificador del usuario (`string`). Debe ser único en el gestor, es decir, en la lista de usuarios del gestor.
- La contraseña del usuario (`string`).
- La lista de registros de mensajes recibidos que será de tipo `tListaBandeja` y que nos permite implementar la bandeja de entrada del usuario.
- La lista de registros de mensajes enviados que será de tipo `tListaBandeja` y que nos permite implementar la bandeja de salida del usuario.
- La bandeja de correo activa (entrada o salida).

Ofrece al menos los siguientes subprogramas:

- `void iniciar(tUsuario & usuario, const string & idUser, const string & contrasenia)`: Recibe un identificador de usuario y una contraseña e inicia el usuario. La bandeja activa inicial es la de entrada.
- `void guardar(const tUsuario & usuario, ofstream & archivo)`: Dado un flujo de archivo (ya abierto), guarda el usuario en fichero (ver ejemplo de la sección 1.3).
- `bool cargar(tUsuario & usuario, ifstream & archivo)`: Dado un flujo de archivo (ya abierto), carga el usuario del fichero.
- `bool validarContrasenia(const tUsuario & usuario, const string & contrasenia)`: Recibe una contraseña y un usuario y devuelve si la contraseña es correcta o no.
- `void cambiarBandeja(tUsuario & usuario)`: cambia la bandeja activa del usuario.

Módulo ListaUsuarios

Declara el tipo de datos `tListaUsuarios` para gestionar la lista de usuarios. Es **IMPORTANTE** que tengas en cuenta que esta lista se encuentra **ordenada por el identificador del usuario**. Debes implementarla como una lista de tamaño variable.

Ofrece al menos los siguientes subprogramas:

- `inline void iniciar(tListaUsuarios & usuarios)`
 `{ usuarios.contador == 0; };`
- `inline bool llena(const tListaUsuarios & usuarios)`
 `{ return usuarios.contador == MAX_USUARIOS; };`
- `inline int longitud(const tListaUsuarios & usuarios)`
 `{ return usuarios.contador; };`

- `bool cargar(tListaUsuarios & usuarios, const string & nombre):` Carga la lista de usuarios desde el fichero de usuarios nombre (ver ejemplo de la sección 1.3).
- `void guardar(const tListaUsuarios & usuarios, const string & nombre):` Guarda de la lista de usuarios en el fichero de usuarios nombre.
- `bool insertar(tListaUsuarios & usuarios, const tUsuario & usuario):` Añade un usuario en la posición de la lista que le corresponde, si hay sitio para ello. Además devuelve un booleano indicando si la operación tuvo éxito o no.
- `bool buscar(const tListaUsuarios & usuarios, const string & idUser, int & posicion):` Dado un identificador de usuario y la lista, devuelve, si dicho identificador existe en la lista, su posición y el valor `true`, y si no existe en la lista, la posición que le correspondería y el valor `false`.

Módulo Gestor

Declara el tipo de datos `tGestor` para guardar la información de un gestor:

- El dominio del gestor de correo (`string`). Se trata de un identificador como por ejemplo “`fdimail.com`”. Nos indica la lista de correos y de usuarios que debemos cargar.
- La lista de correos del sistema que será de tipo `tListaCorreos` y que guarda la única copia de los correos que existirá en la aplicación.
- La lista de usuarios del sistema que será de tipo `tListaUsuarios`.

Ofrece al menos los siguientes subprogramas:

- `bool arrancar(tGestor & gestor, const string & dominio):` Inicializa el gestor e intenta arrancarlo cargando la información del dominio que se le pasa como parámetro. Para ello inicializará y cargará la lista de usuarios y de correos de dicho dominio. Si tiene éxito en todas las operaciones devuelve `true` y si alguna falla devuelve `false`. El nombre de los archivos de donde cargar las listas es: `<NombreDominio>_correos.txt` `<NombreDominio>_usuarios.txt`.
- `void apagar(const tGestor & gestor):` Esta operación apaga el gestor y guarda para ello las listas de usuarios y de correos del dominio activo. Los nombres de los archivos son los mismos que al arrancar.
- `tUsuario* registraUsuario(tGestor & gestor):` Lee los datos de usuario necesarios para validar la cuenta (id y contraseña) y comprueba si el usuario existe y la contraseña coincide. Devuelve la dirección de memoria del usuario si la operación tuvo éxito, y `nullptr` en otro caso.

- **tUsuario* crearCuenta(tGestor & gestor):** Lee los datos de usuario necesarios para crear una cuenta (id y contraseña) y si el id de usuario no existe y hay espacio en la lista de usuarios, crea la cuenta del usuario. Devuelve la dirección de memoria del usuario si la operación tuvo éxito, y `nullptr` en otro caso.
- **void enviarCorreo(tGestor & gestor, tUsuario & usuario, const tCorreo & correo):** Inserta el correo recibido como parámetro en la lista de correo. Si tiene éxito, se inserta el elemento correspondiente en la lista de la bandeja de elementos enviados del emisor, y se inserta igualmente un elemento en la lista de la bandeja de elementos recibidos del destinatario del correo. Si el destinatario no existe o si su bandeja está llena, entonces se mostrará un mensaje de error.
- **void eliminarCorreo(tGestor & gestor, tUsuario & usuario, const string & idMail):** Elimina de la bandeja activa del usuario el elemento correspondiente al identificado de correo. Además, disminuye el número de usuarios con acceso al correo y en caso de que quede a cero lo elimina de la lista de correos del gestor. (**OJO:** el correo sólo se elimina de la lista de correos del gestor en caso de que el número de usuarios con acceso al mismo sea cero).

Módulo Sesión

Declara un tipo de datos **tSesion** para guardar la información de una sesión:

- Enlace al gestor de correo (**tGestor ***) de la sesión.
- Enlace al usuario (**tUsuario ***) de la sesión.

Ofrece al menos los siguientes subprogramas:

- **bool iniciarSesion(tSesion & sesión, tGestor* gestor, tUsuario* usuario):** Inicia la sesión y muestra la pantalla del menú principal de la sesión.

Además, para cada una de las opciones del menú:

- **void leerCorreo(tSesion & sesión):** Solicita el correo que el usuario quiere leer (será el número con el que el correo es mostrado por pantalla en la bandeja correspondiente), valida que existe y si es así, lo marca como correo leído. A continuación, busca el correo en la lista de correos y si lo encuentra muestra en la pantalla el menú de lectura del correo (aparece el correo y las opciones: contestar,...).
- **void enviarCorreo(tSesion & sesión):** Solicita los datos del nuevo correo (destinatario, asunto, cuerpo, ...) y lo envía a través del gestor.

- `void borrarCorreo(tSesion & sesión)`: Solicita el correo que el usuario quiere borrar (será el número con el que el correo es mostrado por pantalla en la bandeja correspondiente), valida que existe y si es así, indica al gestor que lo borre.
- `void lecturaRapida(tSesion & sesión)`: Este subprograma implementa la lectura rápida de correos sin leer. El resultado es que muestra en una pantalla todos los correos sin leer (de la bandeja activa) ordenados por asunto (ignorando todos los "Re: ") y por fecha. Al finalizar su ejecución los correos sin leer quedarán marcados como leídos.

Módulo Principal (main)

Habrás además un fichero .cpp adicional que contendrá el `main` de la aplicación y que incluirá todos los módulos que necesite para su correcto funcionamiento.

3. Detalles de implementación

A la hora de completar la práctica, deberás seguir las siguientes indicaciones.

3.1. Tipos de datos especiales

Fecha

Las fechas se representarán en el formato UNIX, es decir, como un entero con el número de segundos transcurridos desde el 1 de enero de 1970. Debes por tanto declarar el tipo `tFecha` mediante:

`typedef time_t tFecha;` → será necesario incluir la librería `ctime`

En la función de creación de un correo nuevo se obtendrá la fecha actual del sistema: `correo.fecha = time(0);` → será necesario incluir la librería `ctime`

La lectura (escritura) de la fecha de (en) fichero se realiza usando el operador habitual de extracción (inserción) de los enteros.

Deberás definir además la siguiente función para mostrar la fecha en formato Año/Mes/Día, Hora/Mins/Segs:

```
string mostrarFecha(tFecha fecha){
    ostringstream resultado;
    tm ltm;
    localtime_s(&ltm, &fecha);
    resultado << 1900 + ltm.tm_year << '/' << 1 + ltm.tm_mon << '/' << ltm.tm_mday;
    resultado << ' (' << ltm.tm_hour << ':' << ltm.tm_min << ':' << ltm.tm_sec << ')';
    return resultado.str();
}
```

Para mostrar sólo el día de la fecha en formato Año/Mes/Día (sin hora):

```

string mostrarSoloDia(tFecha fecha){
    ostringstream resultado;
    tm ltm;
    localtime_s(&ltm, &fecha);
    resultado << 1900 + ltm.tm_year << '/' << 1 + ltm.tm_mon << '/' << ltm.tm_mday;
    return resultado.str();
}

```

stringstream

En C++ es posible operar con un string como lo hacemos con los flujos de E/S. En esta práctica puede resultarnos útil generar strings como si estuviéramos mandando información a un flujo de salida ostream. En lugar de dar una explicación usaremos un ejemplo:

```

#include <sstream> // Es necesario incluir la biblioteca sstream
...
string resultado;
string nombre= "Pepe";
ostringstream flujo; // Flujo de salida
flujo << "Hola " << nombre << endl; // Pasamos datos al flujo
resultado=flujo.str(); // string del flujo, contendrá la cadena "Hola Pepe\n"

```

3.2. Ficheros de datos

La siguiente figura muestra el contenido de un fichero de ejemplo de nombre genérico <NombreDominio>_usuarios.txt que contiene la lista de usuarios del sistema ordenada por orden alfabético del identificador de usuario y que acaba con el centinela XXX.

```

arturo@fdimail.com 61N3br4 1
lancelot@fdimail.com 1426613678 0
2 contador(int)
arturo@fdimail.com 14266143811 1 tRegistro
arturo@fdimail.com 14266144581 1 tRegistro
ginebra@fdimail.com
14Nc310t_XX
3
lancelot@fdimail.com 1426613678 1
arturo@fdimail.com 14266143810 0
arturo@fdimail.com 14266144580 0
0
lancelot@fdimail.com
61n3BR4_XX
1
arturo@fdimail.com 14266144580 1
1
lancelot@fdimail.com 1426613678 1
XXX

```

El diagrama muestra el contenido de un archivo de texto con una lista de usuarios. Se muestran tres usuarios: arturo@fdimail.com, lancelot@fdimail.com y ginebra@fdimail.com. Cada usuario tiene una contraseña y un contador. Los registros se almacenan en una bandeja de salida (tListaRegistros).

Ejemplo de fichero con una lista de usuarios

Cada usuario tiene sus campos: nombre, contraseña, bandeja de entrada y bandeja de salida. Ambas bandejas son del tipo `tListaRegistros`. Cada lista de registros se compondrá en primer lugar del valor del contador y a continuación de tantos datos de tipo `tRegistro` como indique el contador (si el contador vale 0, no habrá ningún registro). El `tRegistro` se compone de su identificador de correo y del booleano que indica si el mensaje está leído o no ambos en la misma línea.

Por otro lado, existe un fichero `<NombreDominio>_listaCorreo.txt` que guarda todos los correos gestionados por el sistema. En la siguiente figura se muestra un ejemplo de dicho fichero. Como se puede ver se compone de correos ordenados por el identificador de correo y acaba con el centinela XXX.

```

correo {
  arturo@fdimail.com 1426614381
  1426614381
  arturo@fdimail.com
  ginebra@fdimail.com
  Mensaje de prueba
  Hola,
  Parece que ya tenemos correo en Camelot!
  Arturo
  X
}

correo {
  arturo@fdimail.com_1426614458
  1426614458
  arturo@fdimail.com
  lancelot@fdimail.com
  Comprar mesa de reuniones
  Hola,
  he pensado que podíamos comprar una mesa de forma redonda
  para las reuniones. ¿Os parece buena idea?
  Arturo
  X
}

correo {
  lancelot@fdimail.com_1426613678
  1426613678
  lancelot@fdimail.com
  arturo@fdimail.com
  Cazamos mañana?
  Hola a todos!

  Os parece buena idea salir a cazar mañana.
  Lancelot
  X
}

XXX

```

Ejemplo de fichero con la lista de correos

Cada correo se compone de varias líneas: identificador, fecha, emisor, receptor, asunto y cuerpo del mensaje. El cuerpo de un mensaje está compuesto de varias líneas. Esto hará que necesites crear unos procedimientos para leer el cuerpo de teclado y para cargarlo.

Para indicar el fin de un correo usaremos un centinela X. Esto nos permite saber también cuándo acaba el cuerpo del mensaje.

3.3. Cargar y guardar

Para realizar la carga y el guardado de los ficheros es **MUY IMPORTANTE** que tengas en cuenta lo siguiente: la carga y guardado en ficheros NO se realiza en un único módulo. En su lugar, cada módulo se encarga de cargar/guardar los datos que le corresponden.

Para que esto sea posible se tienen que dar dos condiciones:

- La apertura y cierre de un fichero se realiza en un único sitio: el subprograma que inicia la operación de cargar/guardar.
- El subprograma coordina la operación invocando al resto de subprogramas de cargar/guardar los cuales reciben el fichero (ifstream / ofstream) abierto **como parámetro por referencia** para cargar/guardar lo que corresponda.

Vamos a poner un ejemplo con la carga de la lista de usuarios desde fichero. La operación de carga de la lista de usuarios se inicia con la invocación al subprograma cargar del módulo de la lista de usuarios:

```
bool cargar(tListaUsuarios & usuarios, const string & nombre)
```

Este subprograma intentará abrir el fichero `nombre` y si lo consigue invocará para cada usuario al subprograma que gestiona la carga de un usuario que estará en el módulo usuario:

```
bool cargar(tUsuario & usuario, ifstream & archivo)
```

Como podemos ver, a este subprograma habrá que pasarle como parámetro de E/S el flujo de entrada que hemos obtenido al abrir el fichero. Este subprograma leerá los campos básicos del usuario en orden (identificador y contraseña), sin embargo, cuando llegue el turno de leer la bandeja de entrada y la bandeja de salida (que son de tipo `tListaBandeja`) tendrá que realizar dos invocaciones consecutivas al subprograma cargar del módulo de la lista de bandejas:

```
void cargar(tListaBandeja & listaElems, ifstream & archivo)
```

De nuevo a este subprograma habrá que pasarle como parámetro de E/S el flujo de entrada que recibió el cargar del módulo usuario.

3.4. Indicaciones importantes

- Las listas de tamaño variable deberán implementarse con la estructura de array + contador vista en el Tema 6.
- Siempre que se pueda, las operaciones de búsqueda serán binarias.
- Todos los módulos (excepto el principal) se implementarán con archivos `.h` y `.cpp`. En el archivo `.h` se incluirán los tipos de datos del módulo y los

prototipos de las funciones públicas con comentarios describiendo qué hace cada función, qué datos recibe y qué datos devuelve.

- Los módulos `ListaUsuarios`, `ListaCorreos` y `ListaBandeja` únicamente realizan operaciones con listas, y nunca deben escribir nada en pantalla. Pueden devolver valores al programa que los llame para que éste escriba en pantalla lo que sea necesario.

4. Partes opcionales

Se puede hacer más realista la aplicación permitiendo:

- Múltiples destinatarios para un correo. Para ello se puede definir una lista de tamaño variable de `string` para los destinatarios de un correo.
- Reenviar un correo. Añade al menú de lectura de un correo la opción de reenviarlo.

5. Entrega de la práctica

La práctica se entregará a través del Campus Virtual. Se habilitará la tarea **Entrega de la Práctica 4** que permitirá subir un único archivo. Hay que subir un archivo comprimido `Practica4.zip` que contenga los archivos `.h` y `.cpp` del proyecto.

Fin del plazo de entrega: **17 de mayo a las 23:55**.