Práctica 5
Gestor de partidas online 2.0

Fecha límite de entrega: 5 de junio

Se ha decidido realizar un cambio en el diseño de la aplicación. En lugar de generar, al arrancar el gestor, la lista de acceso a las partidas en curso de todos los usuarios, se va a generar únicamente para el usuario de la sesión. Y es en el gestor donde se va a mantener la lista de acceso a las partidas en curso del usuario de la sesión, en lugar de en cada usuario.

Por otro lado, el uso de punteros y memoria dinámica nos permite mejorar la implementación. Las listas de partidas y usuarios se van a modificar para utilizar memoria dinámica. Y la lista de partidas en curso, en lugar de guardar la posición de la partida en la lista de partidas, guardará el puntero a la partida.

1. Cambio en el diseño

- Ahora la lista de acceso a las partidas en curso no forma parte del usuario. Por tanto ya no requiere el módulo ListaAccesoPartidas. Además de eliminar el correspondiente campo del tipo de datos tUsuario, hay que quitar el segundo parámetro a las funciones nuevaPartida() y aplicarFinPartida(). Compila el módulo usuarios eliminando lo referente a la lista de partidas en curso.
- Añade al gestor un campo del tipo tListaAccesoPartidas, para mantener la lista de accesos a las partidas en curso del usuario de la sesión. Ahora, al arrancar no se generan las listas de acceso a las partidas, es al iniciarse una nueva sesión cuando hay que hacerlo (en el caso de nuevo usuario la lista quedará vacía). Además hay que adaptar las funciones nuevaPartida(), jugarPartida() y abandonarPartida() porque afectan al nuevo campo del gestor. Compila el módulo Gestor, tendrás que corregir algunos errores sencillos.
- Compila mainP5, seguramente tendrás que corregir un par de errores sencillos.

Comprueba que la práctica funciona correctamente antes de continuar.

2. Cambios de implementación

- La **lista de usuarios** será un array de tamaño dinámico, de forma que cuando se llene el array será reemplazado por otro de más capacidad.
  Añade la función **redimensionar**, que aumenta el número de elementos del array siguiendo la formula nuevaCapacidad = (viejaCapacidad * 3) / 2 + 1. Y la función **apagar(lista, nombArch)**, que además de guardar los usuarios en el archivo, elimina la memoria utilizada por el array. Al **carga**r habrá que crear el array dinámico para poder

cargar todos los usuarios del archivo. Al **insertar** un usuario, si la lista está llena, se redimensionará para poder insertarlo.

- La **lista de partidas** será un array de datos dinámicos. Añade la función **apagar(lista, nombArch)** que además de guardar las partidas en el archivo, elimina la memoria utilizada por las partidas. Al **cargar** habrá que crear los datos dinámicos.
- La **lista de acceso a las partidas en curso** en lugar de contener el índice o posición de cada partida en la lista de partidas del gestor, guardará la dirección de memoria de las partidas (un puntero). Añade las funciones para **ordenar** la lista por fecha y turno (con un parámetro para el identificador del usuario del turno). Y modifica la función **bool buscar(**tListaAccesoPartidas const& lista, string const& **id**, **int &**/*in/out*/ **pos)** para hacerla **recursiva** (el parámetro **pos** es de entrada/salida, la función empieza a buscar en **pos** y termina en la posición de **id** en caso de encontrarlo).
- En el **gestor**, el campo para la partida seleccionada en lugar de una posición es un puntero a la partida.
- **Opcional (1 pto. de la práctica):** Haz que la lista de usuarios sea además de datos dinámicos, así, al redimensionar solo se copiarán los punteros. Haz dinámico el array de la lista de partidas, y cuando se llene se redimensiona con la misma política que para los usuarios.

Memoria dinámica

Para que al terminar la ejecución del programa se muestre información sobre la basura que ha podido dejar, añade al inicio de la función `main` el comando:

```
_CrtSetDbgFlag ( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );
```

(También se puede mostrar la basura con el comando: `_CrtDumpMemoryLeaks()`)

Además, para que la información muestre el nombre del módulo y la línea de código, añade al proyecto un archivo `"checkML.h"` con las siguientes directivas de VS, e inclúyelo en los archivos de código fuente (archivos `.cpp`) del proyecto.

```
// archivo checkML.h
#ifdef _DEBUG
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#ifndef DBG_NEW
#define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
#define new DBG_NEW
#endif
#endif
```

3. Nueva funcionalidad

- Las partidas con más de un mes (30 días) sin actividad se van a cerrar automáticamente al **apagar**, dando por ganador al jugador que no tiene el turno.
- Las partidas terminadas se van a almacenar en otro archivo (denominado historico.txt). Al **arrancar**, no se cargan las partidas del archivo histórico. Al **apagar**, las partidas terminadas se añaden al archivo de partidas históricas.
- **Opcional (para el punto adicional):** Añade al menú de usuario la opción **Ver estadística por adversarios**, que mostrará para cada uno de los adversarios contra los que ha jugado, el número de partidas ganadas, perdidas y empatadas. Tendrás que añadir un nuevo módulo. Implementa la función de forma recursiva.

Archivos

- Para añadir datos al final de un archivo podemos abrirlo en modo *append*.

```
ofstream flujo;
flujo.open(nombArch, ios::app);
```

  Si el archivo existe lo abre sin eliminar los datos que contenga, dejando el cursor al final del archivo para escritura. Si no existe lo crea.

- Podemos modificar directamente los datos de un archivo, pero solo se debe modificar un dato por otro que ocupe exactamente lo mismo (en otro caso se sobrescribirá el siguiente dato del archivo). Por ejemplo, para poder modificar un número entero, podemos escribirlo utilizando 11 caracteres y así podemos modificarlo por otro entero aunque tenga distinto número de dígitos (nunca más de 11).

  La variable `flujo` la declaramos de forma que podamos abrir el archivo en modo lectura y escritura:

```
fstream flujo;   // declaramos la variable
flujo.open(nombArcho, ios::in | ios::out);  // abrimos en modo in/out
```

  Si el archivo existe lo abre sin eliminar los datos que contenga.

  Ahora podemos mover el cursor del archivo con la función `flujo.seekg()` (para mover el cursor de lectura) y `flujo.seekp()` (para mover el cursor de escritura).

  `flujo.seek*(nBytes, ios::beg)`: mueve el cursor nBytes desde el inicio del flujo.
  `flujo.seek*(nBytes, ios::end)`: mueve el cursor nBytes desde el final del flujo.
  `flujo.seek*(nBytes, ios::cur)`: mueve el cursor nBytes desde el cursor.

  Por ejemplo, sabiendo que el primer dato del archivo es un entero de 11 caracteres:

```
flujo.seekp(0, ios::end); // movemos el cursor al final
flujo << …; // añadimos datos al final del archivo
flujo.seekg(0, ios::beg);  // movemos el cursor al inicio
flujo >> ent;  // leemos el entero
```

```
flujo.seekp(0, ios::beg);  // movemos el cursor al inicio
flujo << setw(11) << ent + 1;  // escribimos ocupando 11 caracteres
flujo.close();   // cerramos el archivo
```

Además podemos preguntar por la posición del cursor con la función `flujo.tellg()` (para el cursor de lectura) y `flujo.tellp()` (para el cursor de escritura).

```
if (flujo.tellp() == streampos(0)) flujo << setw(11) << 0 << '\n';
```

5. Entrega de la práctica

La práctica se entregará a través del Campus Virtual, en la tarea **Entrega de la Práctica 5** donde debes subir el archivo **practica5.zip** con todos los archivos `.h` y `.cpp` del proyecto.

Asegúrate de poner el **nombre** de los miembros del grupo en un comentario al principio de cada archivo.

Recuerda:

- La aplicación no debe dejar basura.
- No se permite el uso de variables globales, ni de instrucciones de salto, salvo un `return` como última instrucción de las funciones y `break` en los casos del `switch`.

Fin del plazo de entrega: **5 de junio a las 23:55.**