Fundamentos de la Programación

Implementación de la Práctica 4

(Basado en la práctica de Victor Lavin Puente y Luis Garmendia Salvador)

Índice

1. Introducción
2. Planificación
3. Array Dinámico en el Módulo Puntuaciones
4. Array de Datos Dinámicos en el Módulo Secuencia
5. Utilización de Memoria Dinámica
6. Ordenación en el Módulo Puntuaciones
7. Parte Opcional

1. Introducción

- ✓ Es importante utilizar la memoria de nuestros ordenadores de forma eficiente y no desperdiciar ni replicar contenido.
- ✓ *Objetivos*: modificar la práctica 4 utilizando punteros y memoria dinámica y, ordenar las puntuaciones.
- ✓ La implementación final será mas eficiente.
- ✓ La práctica se divide en tres partes:
  - Modificación del módulo Puntuaciones para utilizar un **array dinámico** en la lista de puntuaciones.
  - Modificación del módulo Secuencia para utilizar un **array de datos dinámicos** en el mazo de cartas.
  - **Ordenación** en el módulo Puntuaciones.

2. Planificación

- ✓ Para la realización de la parte obligatoria de la práctica se propone la siguiente planificación:
  - **Array Dinámico:** semana del 18/4 (implementar y probar).
  - **Array de Datos Dinámicos:** semana del 25/4 (implementar y probar).
  - **Ordenación:** semanas del 9/5 y 16/5 (implementar y probar).
- ✓ La semana del 16/5 también se dedicará a la parte opcional.
- ✓ La entrega es el día 20/5.

3. Array Dinámico en el Módulo Puntuaciones

- ✓ Se pretende almacenar la puntuación de todos los jugadores, independientemente de los puntos obtenidos.
- ✓ No queremos estimar el número máximo de jugadores.
- ✓ Utiliza un array dinámico que pueda crecer según sea necesario. Considera como tamaño (capacidad) inicial 4 y que se incremente (redimensione) en 4 cuando se necesite.
- ✓ Para esta parte de la práctica realiza los **pasos** indicados a continuación.

- **Primero**, modifica la lista *tPuntuaciones* del módulo introduciendo el array dinámico y su capacidad:

```
typedef tPuntuacionJugador *tPuntuacionJugadorPtr;
typedef struct {
   tPuntuacionJugadorPtr puntuaciones;
   int capacidad;
   int num_jugadores;
} tPuntuaciones;
```

- **Segundo**, añade e implementa los siguientes subprogramas en el módulo:

```
void inicializar(tPuntuaciones &clasificacion) // crea una lista de puntuaciones vacía con el nuevo array dinámico
void redimensionar(tPuntuaciones &clasificacion) // amplia en +4 el tamaño del array dinámico
void liberar(tPuntuaciones &clasificacion) // libera la memoria dinámica de la clasificación
```

- ✓ **Tercero**, modifica el subprograma *actualizarPuntuacion* para poder conocer las puntuaciones de todos los jugadores. Ten en cuenta que el prototipo se mantiene pero la implementación cambia.

```
void actualizarPuntuacion(tPuntuaciones &clasificacion, const string
&nombre, int nuevos) // si el jugador ya estaba, se incrementan sus
puntos en nuevos puntos; si no está en el listado lo inserta con los
nuevos puntos.
```

- ✓ **Cuarto**, modifica el programa principal para inicializar y liberar la lista de puntuaciones cuando sea necesario.
- ✓ **Quinto**, prueba el código implementado viendo que se añaden todos los nuevos jugadores y que el array dinámico se redimensiona correctamente.
- ✓ **Sexto**, mira la sección 5 de esta presentación e introduce los cambios que permiten observar la liberación de toda la memoria dinámica utilizada. Prueba los resultados.

4. Array de Datos Dinámicos en el Módulo Secuencia

- ✓ En esta parte de la práctica se introduce un array de datos dinámicos (array de punteros) en el mazo de cartas para crearlas en memoria dinámica. Para conseguirlo realiza los siguientes **pasos**.
- ✓ **Primero**, modifica el archivo de cabecera con estos tipos:

```
typedef tCarta *tCartaPtr;
typedef tCartaPtr tArrayPtr[MAX_CARTAS_MAZO];
typedef struct {
    tArrayPtr cartas;
    int numCartas;
} tMazo;
```

- ✓ **Segundo**, introduce e implementa este subprograma:

```
void liberar(tMazo &mazo) // libera la memoria dinámica creada para el mazo
```

- ✓ **Tercero**, modifica la implementación del subprograma *insertar* para que se inserten cartas en el mazo usando el array de datos dinámicos. Recuerda que debes utilizar el operador *new*.
- ✓ **Cuarto**, modifica el subprograma *sacar* para que se eliminen cartas del mazo usando el array de datos dinámicos. Recuerda que debes utilizar el operador *delete*.
- ✓ **Quinto**, modifica también el subprograma *crearMazoAleatorio*. Debes utilizar el operador *new* cada vez que se insertan cartas en el array de datos dinámicos.
- ✓ **Sexto**, modifica la función *main* del programa principal para liberar la memoria dinámica utilizada con los mazos.
- ✓ **Séptimo**, prueba la nueva versión del programa.

5. Utilización de Memoria Dinámica

- ✓ Al terminar la ejecución del programa debes mostrar información sobre la basura (memoria dinámica no liberada) que haya podido dejar. Para ello realiza los siguientes **pasos**.
- ✓ **Primero**, crea un archivo de cabecera *checkML.h* con las siguientes directivas de VS y agrégalo al programa

```
#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
```

- ✓ **Segundo**, incluye el archivo *checkML.h* en todos los archivos fuente (archivos .cpp) del proyecto.
- ✓ **Tercero**, añade al inicio de la función *main* del programa principal el siguiente comando:

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

- ✓ **Cuarto**, ejecuta el programa observando la salida en la consola de Visual Studio. Corrige los posibles errores liberando toda la memoria dinámica.

6. Ordenación en el Módulo Puntuaciones

- ✓ Para esta última parte obligatoria de la práctica queremos mostrar las puntuaciones ordenadas por ranking y por orden alfabético de los usuarios. Por ello es necesario volver al módulo Puntuaciones y realizar más cambios.
- ✓ **Primero**, introduce el siguiente subprograma para comparar elementos *tPuntuacionJugador* de tal manera que podamos ordenar por ranking (puntuación) y a igual puntuación por orden alfabético de nombre.

```
bool operator<(const tPuntuacionJugador &p1, const tPuntuacionJugador
&p2)
```

- **Segundo**, modifica la función *cargarPuntuaciones* para que se realicen inserciones de forma ordenada. Recuerda que para insertar en lista ordenada hay que buscar la posición donde insertar. Además, utilizando el array dinámico redimensionable hay que comprobar cuando es necesario aumentar el tamaño de dicho array.
- Ten en cuenta que el fichero *puntuaciones.txt* debe mantenerse siempre ordenado por el ranking.
- **Tercero**, implementa el siguiente subprograma para ordenar una lista de puntuaciones por orden alfabético. Para ello utiliza alguno de los algoritmos de ordenación vistos en clase como, por el ejemplo, el de la burbuja.

```
void ordenaJugador(tPuntuaciones &clasificacion)
```

- **Cuarto**, implementa la siguiente función para localizar a un jugador. Utilízala donde sea necesario.

```
bool localizarJugador(const tPuntuaciones &puntos, const string
&nombre, int &posicion)
```

- **Quinto**, modifica el siguiente subprograma teniendo en cuenta que cada nuevo jugador debe insertarse en la lista de puntuaciones de forma ordenada (en la posición adecuada) y que cuando el jugador ya existe debe reubicarse después de actualizar su puntuación.

```
bool actualizarPuntuacion(tPuntuaciones &puntos, const string &nombre,
int nuevos)
```

- ✓ **Sexto**, modifica el subprograma *mostrarPuntuaciones* para abrir el siguiente submenú cuando se elige la opción 2 del menú del programa principal.

```
2.1. Puntuaciones ordenadas por ranking
2.2. Puntuaciones ordenadas por orden alfabético
```

- ✓ Utiliza alguna de las funciones previamente implementadas para la ejecución de la opción 2.2.

7. Parte Opcional

- ✓ En la parte opcional se pide utilizar un **array dinámico de punteros** en el módulo puntuaciones. Este array dinámico debe contener punteros a la estructura que almacena la información de cada jugador.
- ✓ Es necesario volver a modificar el archivo de cabecera con los nuevos tipos y adaptar el código de los subprogramas a las nuevas definiciones.
- ✓ **Primero**, introduce los siguientes tipos en el módulo:

```
typedef tPuntuacionJugador *tPtrPuntuacionJugador;
typedef tPtrPuntuacionJugador *tArrayDinamico;
typedef struct {
   int capacidad;
   int num_jugadores;
   tArrayDinamico array_clasificacion;
} tPuntuaciones;
```

- ✓ No es necesario que modifiques los prototipos del módulo.
- ✓ **Segundo**, modifica el subprograma *inicializar* para crear un array dinámico de punteros utilizando el tipo base *tPtrPuntuacionJugador*.
- ✓ **Tercero**, modifica *cargarPuntuaciones* teniendo en cuenta que cuando se inserta un jugador hay que crear primero el puntero con el operador *new*.

```
clasificacion.array_clasificacion[posicion_insercion] = new
tPuntuacionJugador;
*(clasificacion.array_clasificacion[posicion_insercion]) = jugador;
```

- ✓ **Cuarto**, modifica el subprograma *liberar* introduciendo un bucle *for* que elimine cada uno de los punteros del nuevo array dinámico.

- ✓ **Quinto**, modifica *ordenaJugador* para que se intercambien punteros en lugar de estructuras *tPuntuacionJugador* y para que se lea el campo *nombre* a través de puntero (usando el operador ->).
- ✓ **Sexto**, modifica también *guardarPuntuaciones* usando el operador –>. Haz lo mismo en *mostrarPuntuaciones* y *localizarJugador*.
- ✓ **Séptimo**, en el subprograma *redimensionar* hay que utilizar el nuevo tipo *tPtrPuntuacionJugador* en lugar de *tPuntuacionJugador*. Lo mismo sucede en *actualizarPuntuacion*. Y siempre que se accede a alguno de los campos *nombre* o *puntuacion* hay que utilizar el operador ->.