Práctica 5

La memoria

Material: PC y Visual Studio 2013

Duración: 2 horas

Lugar: Laboratorios de prácticas (Laboratorio de Redes-Hardware)

La herramienta que vamos a utilizar para el desarrollo de las prácticas de C será Microsoft Visual Studio 2013 que permite la creación, compilación y ejecución de programas escritos en lenguaje C/C++ y C#. Se recuerda al alumnado que, para la mejor comprensión y asimilación de los conocimientos y habilidades desarrollados durante la práctica, deberá haberse estudiado previamente el material docente disponible. Durante la práctica se realizarán diferentes actividades que serán objeto de evaluación.

Desarrollo de la práctica:

1. Manejo de la memoria dinámica mediante C/C++.

En esta primera parte de la práctica procederemos a realizar y resolver los ejercicios planeados mediante un proyecto de consola y escritos en el lenguaje C/C++, de la misma forma en la que fue resuelta la práctica 1.

Ejercicios:

1) Escribir un programa en C para descomponer un entero corto sin signo en dos bytes utilizando exclusivamente operaciones con punteros.

2) Escribir un programa en C que imprima en pantalla los elementos de un vector de 10 enteros, una vez hayan sido inicializadas sus posiciones, así como la suma de todos sus elementos; accediendo a ellos únicamente mediante aritmética de punteros. El vector deberá ser creado mediante memoria dinámica.

3) Escribir un programa en C que pida al usuario, durante la ejecución del mismo, el tamaño de un vector de enteros. Posteriormente, deberán almacenarse en él números enteros aleatorios. Para recorrer el vector se deben utilizar punteros en lugar de índices.

- **Nota 1**. Es obligatorio crear el vector en tiempo de ejecución. El usuario elegirá el tamaño del vector.
- **Nota 2**. La función de C *rand()* devuelve un número entero seudoaleatorio. Su prototipo es "*int rand(void)*", y se encuentra en la librería *stdlib.h*.

2. Manejo de la memoria dinámica mediante C#.

En esta segunda parte de la práctica comentaremos, en primer lugar, cómo se declaran vectores y matrices (arrays) en C#. Finalmente, procederemos a realizar y resolver los ejercicios planeados en la primera parte de la práctica mediante un proyecto en Visual C#. No veremos cómo se declaran punteros en C#, debido a que su declaración y manejo implica la gestión de los proyectos realizados de una forma particular; ya que implica código considerado como inseguro. Con todo, podéis consultar información al respecto en http://msdn.microsoft.com/en-us/library/y31yhkeb.aspx

Vectores y matrices en C#

Muchas veces necesitamos tener vectores y matrices de los cuales no conocemos qué tamaño deberán tener hasta que el programa que hace uso de ellos no se ejecuta. El siguiente es un ejemplo de una matriz dinámica. Los valores y el número de ellos se basan en la entrada del usuario:

```
namespace WpfApplication8
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        int[]  intVector;
        int[,] intMatriz;
        int[,] intMatriz_2;

         public MainWindow()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e){
            //Ejemplo de generación dinámica
```

```
            //vector de cinco elementos
            //matriz de 4x3
            intVector = new int[5];
            intMatriz = new int[4,3];

            //los valores podrían capturarse en tiempo real
            //desde un componente visual, por ejemplo, desde
            //un TextBox
            intMatriz_2 = new int[4, int.Parse(textBox1.Text)];
        }
    }
}
```

Nota. Recuerda que, al añadir el componente de tipo texto debes darle el nombre textBox1, para que pueda establecerse la relación entre el componente visual y las instrucciones de nuestro programa.

Ejemplo:

A continuación, se muestra otro ejemplo de uso de vectores y matrices en C#, con el fin de mostrar que todas las posibilidades que teníamos en C/C++ están disponibles en C#.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //array de dos dimensiones e inicialización
            int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 },
                                          { 7, 8 } };
            //otro array en el que hemos especificado sus dimensiones e
            //inicialización
            int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6},
```

```
                                                { 7, 8 } };
            //un array que almacena Strings e inicialización
            string[,] array2Db = new string[3, 2] { { "one", "two" }, {
                                  "three", "four" },{ "five", "six" } };

            //array de tres dimensiones
            int[, ,] array3D = new int[,,] { { { 1, 2, 3 }, { 4, 5, 6 } },
                                { { 7, 8, 9 }, { 10, 11, 12 } } };
            //una array de tres dimensiones con especificación de sus
            //dimensiones e inicialización
            int[, ,] array3Da = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5,
                                6 } }, { { 7, 8, 9 }, { 10, 11, 12 } } };

            //Mostramos por un TextBox algunos de sus elementos
            textBox1.AppendText(array2D[0, 0].ToString() +"\r\n");
            textBox1.AppendText(array2D[0, 1].ToString() + "\r\n");
            textBox1.AppendText(array2D[1, 0].ToString() + "\r\n");
            textBox1.AppendText(array2D[1, 1].ToString() + "\r\n");
            textBox1.AppendText(array2D[3, 0].ToString() + "\r\n");
            textBox1.AppendText(array2Db[1, 0].ToString() + "\r\n");
            textBox1.AppendText(array3Da[1, 0, 1].ToString() + "\r\n");
            textBox1.AppendText(array3D[1, 1, 2].ToString() + "\r\n");

            //obtención del número de elementos de un array
            var allLength = array3D.Length;
            textBox1.AppendText("El número de elementos es igual a "+
                                allLength.ToString());
            //desplazamos el cursor para mostrar la última línea impresa
            textBox1.ScrollToEnd();
        }
    }
}
```

Nota. Recuerda que, puedes hacer que el componente TexBox muestre un desplazador vertical con la propiedad *HorizontalScrollBarVisibility*, que se encuentra dentro de la pestaña *Diseño* en las propiedades avanzadas, hay que expandir el menú. Además, si queremos que baje automáticamente el cursor a la última línea escrita, debemos emplear la instrucción *textBox1.ScrollToEnd();*, última instrucción que se muestra en el código anterior.

Ejercicios:

4) Escribir un programa en Visual C# que pida al usuario, durante la ejecución del mismo, el tamaño de un vector de enteros. Posteriormente, deberán almacenarse en él números enteros aleatorios y mostrarlos adecuadamente en dicho proyecto.

- **Nota 1**. Es obligatorio crear el vector en tiempo de ejecución. El usuario elegirá el tamaño del vector.
- **Nota 2**. Repasa los ejercicios del Tema 3 para ver cómo se generan números aleatorios en C#.

5) Escribe un programa en Visual C que permita reservar asientos de una sala de cine (4 filas x 4 columnas) mediante una matriz. El programa visual nos permitirá:

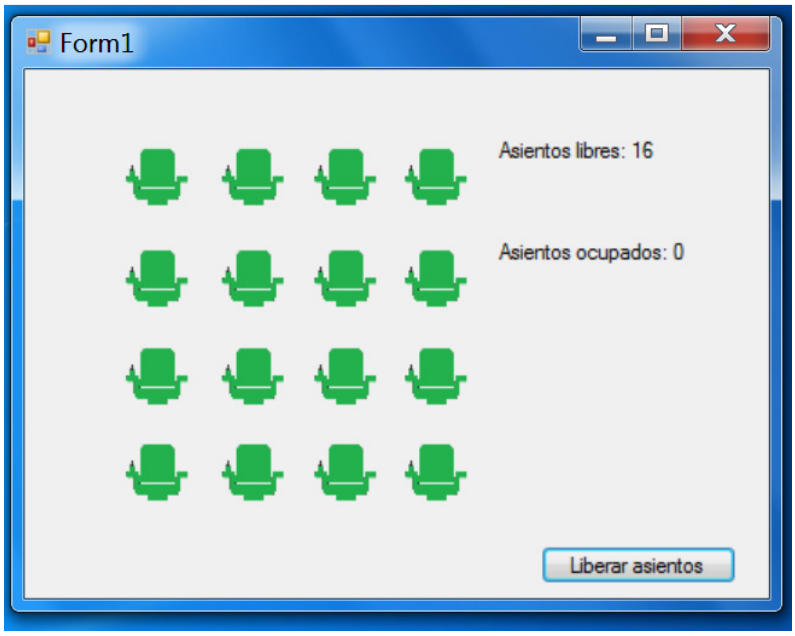
1. Liberar todos los asientos mediante un botón.
2. Reservar un asiento en concreto mediante un click en el mismo y liberarlo de la misma forma.
3. Visualizar la sala de cine.
4. Mostrar, con componentes de tipo etiqueta, el número de asientos libres y el número de asientos ocupados.

El tamaño de la matriz será definido en el constructor del programa.

Nota. Descarga, del espacio web de la asignatura, las imágenes png para el asiento libre y el asiento ocupado.

Nota 2. Para añadir el evento de click en un componente de tipo PictureBox, simplemente debes hacer doble click sobre dicho componente, antes de añadir una imagen en él.

El aspecto visual deberá ser similar al siguiente:



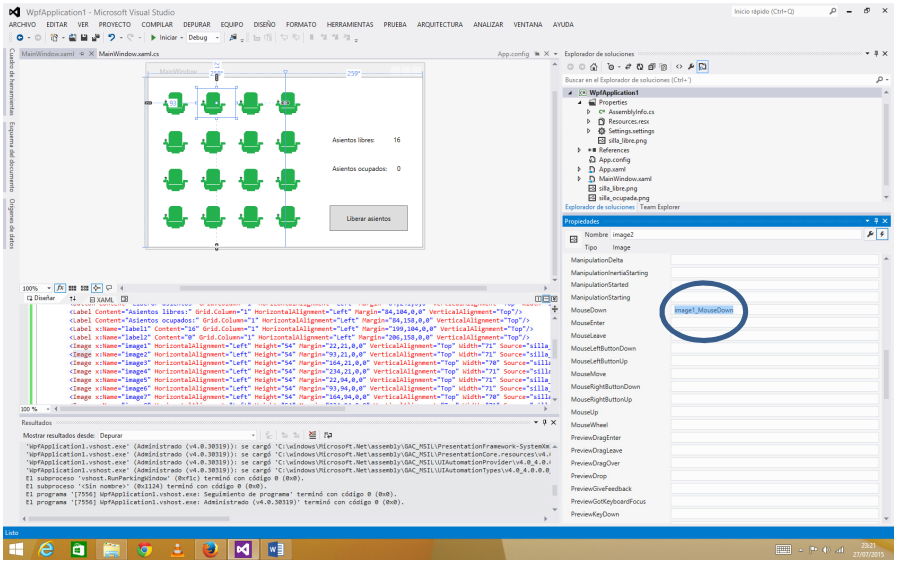
Nota. Para realizar el siguiente ejercicio, es necesario realizar previamente el anterior. Además, aparecerá mucho código repetido (bastante código en realidad), es la finalidad del ejercicio, en el siguiente veremos cómo podemos simplificar dicho código.

6) Si queremos repetir el ejercicio 5 con una sala de 15 filas y 20 butacas por fila, ¿qué problema de programación nos encontramos? ¿Qué código tendrías que repetir? De hecho, ¿qué código repetiste en el ejercicio anterior? Cambia el código realizado en el ejercicio anterior siguiendo las siguientes instrucciones, con el fin de aprender cómo simplificar la realización de programas en Visual C#, cuando trabajamos con código que opera sobre varios componentes del mismo tipo, es decir, cuando repetimos el mismo código para más de un componente:

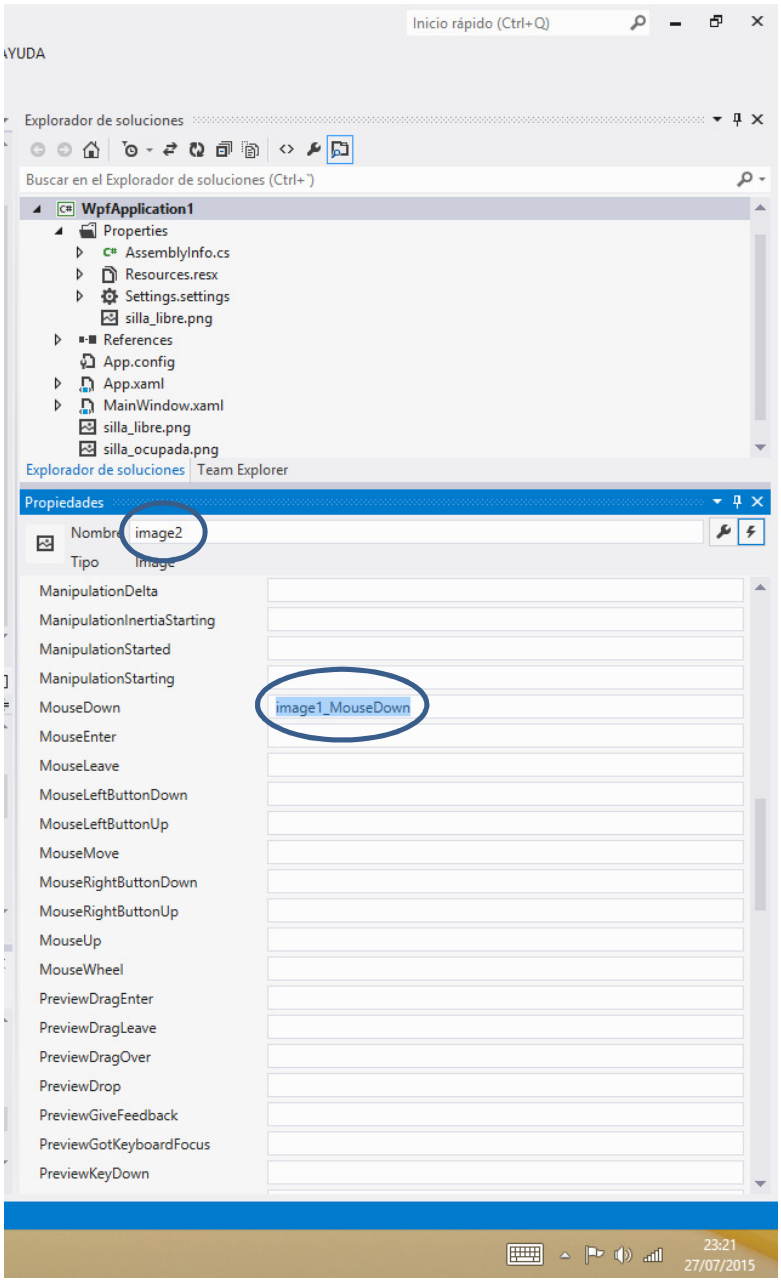
1. Al pulsar sobre la imagen de un asiento debemos cambiar el color de butaca, cambiando la imagen que se muestra en el componente *Image* (consulta la práctica 3). Para ello, activábamos el evento MouseDown en dicho componente. Si tenemos 16 imágenes, cada una de ella representando una butaca, has debido añadir al código 16 funciones de la forma:

```
private void image1_MouseDown(object sender, MouseButtonEventArgs e){
}
```

   Una función para cada imagen, capturando el click sobre la misma.
2. Sin embargo, C# nos permite hacer esto de una forma mucho más sencilla, con una sola función y analizando los parámetros que recibe la misma. Lo primero que haremos será asociar la misma función a todos los componentes imagen que hemos definido, concretamente emplearemos la función que se encarga del cambio de la imagen para la butaca 1. En principio, el efecto que veremos es que se ejecute siempre la función *image1_MouseDown*, se pulse la imagen de la butaca que se pulse.



Nota. También podéis cambiarlo si, en el editor XAML que aparece justo debajo de la vista de diseño, se cambia la propiedad *MouseDown* de todos los botones a *MouseDown="image1_MouseDown"*.



3. En el siguiente paso, modificaremos lo comentado en el punto dos para conseguir el efecto deseado, es decir, que se cambie la imagen que corresponda. Para ello, trabajaremos analizando la información de uno de los parámetros de dicha función, concretamente el parámetro *sender* de tipo *object*. Dicho parámetro contiene toda la información del componente que provoca la ejecución de la función (el evento) y para

poder acceder a él debemos castearlo o formatearlo al tipo de componente que estamos manejando, concretamente un componente de tipo *Image*. Por ello, añadiremos, como primera instrucción al entrar en la función la instrucción que mostramos más adelanta, y que nos realiza una conversión explícita realmente; de la misma forma que podíamos convertir un entero a un real mediante casteo explícito, solamente que ahora emplearemos la sintaxis que se muestra a continuación:

```
private void image1_MouseDown(object sender, MouseButtonEventArgs e){
     Image image = sender as Image;
}
```

4. Una vez hecho lo detallado en los puntos anteriores, dentro de la función *image1_MouseDown* ya podemos utilizar, para acceder a la imagen, la variable *image*, en lugar de la variable *imagen1* que tenía esta función y que era el identificador del componente imagen (ver práctica 3). Si ejecutamos el código veremos que cambia correctamente el valor de la butaca pulsada, porque el evento del click para todas las imágenes habrá sido cambiado según lo indicado en el punto 3.

5. Ahora tenemos que ajustar correctamente los valores de la matriz de datos 4x4 y que representa la reserva de butacas, para actualizar la matriz cada vez que se ejecuta la función. ¿Cómo podemos identificar la butaca (o el componente Image que la representa) que se ha pulsado? La instrucción *image.Name* contiene un valor de tipo String, el cual deberá ser procesado para actualizar correctamente la interfaz de este ejercicio.

   Nota. Emplea el depurador para escribir la instrucción indicada y comprueba su valor cada vez que se pulsa sobre una imagen. También puedes declarar una variable de tipo *String* y almacenar su valor:

```
private void image1_MouseDown(object sender, MouseButtonEventArgs e){
     Image image = sender as Image;
     String temp = image.Name;
}
```

6. Finalmente, haciendo uso de instrucciones de C# para el tratamiento de cadenas podemos saber qué imagen se ha pulsado:

```
private void image1_MouseDown(object sender, MouseButtonEventArgs e){
    Image image = sender as Image;
    String temp = image.Name;
    temp = temp.Replace("image", "");
    int butaca = int.Parse(temp);
}
```

   La variable butaca contiene el valor numérico que identifica la imagen pulsada. Depura el código anterior y comprueba los valores obtenidos.

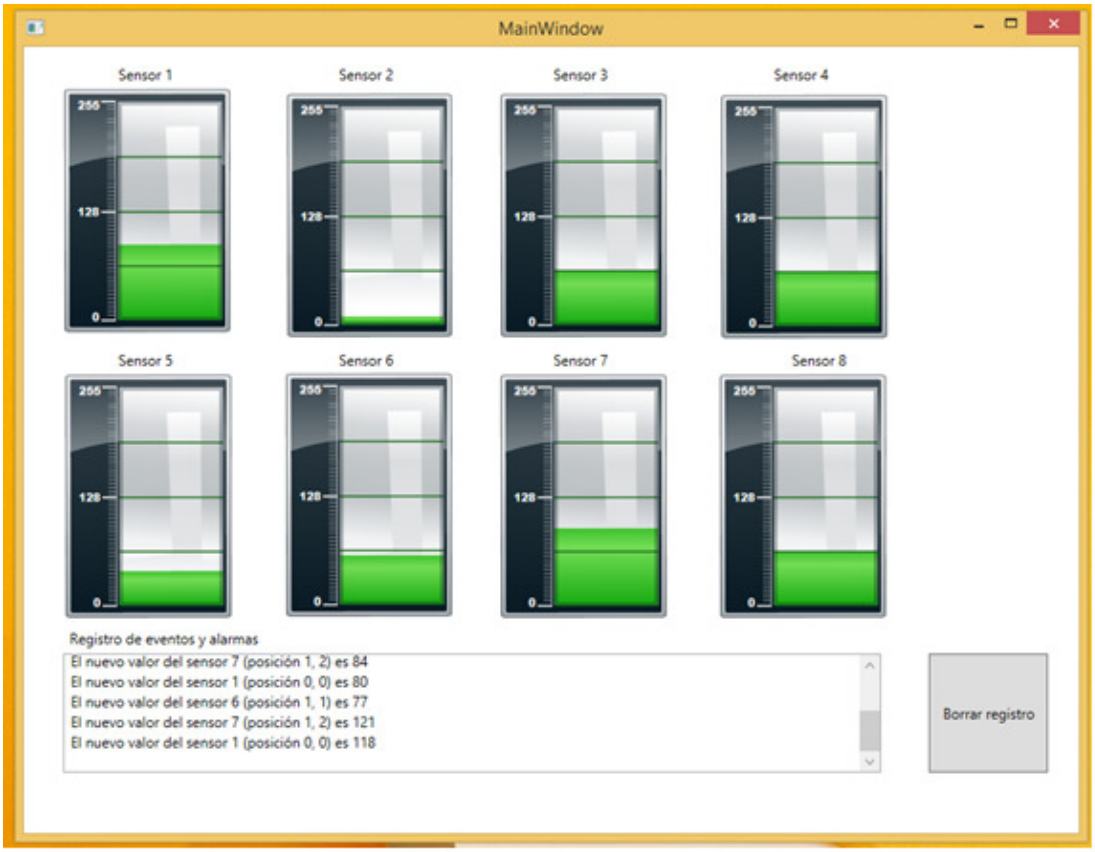
7) Completa el ejercicio anterior para que la función *image1_MouseDown()* actualice la matriz de datos correctamente, así como el resto de variables que sea necesario. **Pista**. Emplea las operaciones de división y resto enteras sobre la variable butaca para obtener los valores de la fila y la columna de la matriz correspondiente.

8) Escribe un programa en Visual C# que permita visualizar 8 sensores de temperatura (expresada en grados centígrados), cuyo valor será modificado aleatoriamente mediante un *timer*, según las instrucciones que se indican. El programa visual nos permitirá:

1. Cada 2 segundos el *timer* generará un valor aleatorio comprendido entre 0 y 100 y actualizará el valor de uno de los sensores de temperatura elegido al azar. De esta forma, simulamos el comportamiento de un sistema real.
2. Se declarará una matriz para representar internamente el estado de los sensores de temperatura.
3. Podremos poner todos los sensores a cero mediante un botón.
4. Incluir el componente de parada de emergencia del sistema. Si lo pulsamos, se detendrá tanto la generación de valores aleatorios como el refresco de las temperaturas en la pantalla.
5. Visualizar en la interfaz a diseñar todos los sensores, como se muestra más adelante, numerados del uno al seis.
6. Mostrar a través del registro de eventos y alarmas, los sensores que tienen una temperatura:
    a. Mayor o igual que 50 grados y menor que 70 grados.
    b. Mayor o igual que 70 grados y menor que 90 grados.

c. Mayor o igual que 90 grados.

El aspecto visual deberá ser similar al siguiente:



Nota. La imagen del proceso, los sensores de temperatura, será representada internamente mediante una matriz de enteros de tamaño 2x3.

Nota 2. Consulta la práctica 3 para repasar cómo se introducen los componentes visuales en el programa a realizar.

9) Busca cómo cambiar el color de la línea impresa en el registro de eventos y alarmas para que, cada vez que una línea se imprima salga con un color concreto, atendiendo a:

- Negro, si la temperatura no excede los 50 grados.
- Verde, si la temperatura es igual a 50 grados y menor que 70.
- Amarillo, si la temperatura es igual a 70 y menor que 90 grados y,
- Rojo, si la temperatura es igual o mayor a 90 grados.

10) Añade un botón para reiniciar el sistema y fijar los valores de todos los sensores de temperatura a 0 grados antes de comenzar.

11) Personaliza la interfaz y añade diferentes tipos de controles.