Práctica Evaluable Tema 5.

Introducción a las funciones

Objetivos.

- Repasar los conceptos estudiados hasta ahora.

Consideraciones iniciales.

- Se indica la puntuación de cada ejercicio, incluyendo la puntuación de los subapartados.
- Cada ejercicio tiene que estar en un fichero con el nombre indicado en el enunciado y debe contener una clase con ese mismo nombre, pero sin la extensión ".cs"

Código implementado

Para cada archivo fuente entregado se deberá incluir como comentario en las primeras líneas del archivo el nombre del autor, la indicación de la práctica evaluable y el número del ejercicio.

Además se incluirá un listado de todos los apartados, indicando si han sido implementados totalmente, parcialmente o no ha sido realizado.

Por ejemplo:

```
/*
Perez Gomez, Andres
Practica Evaluable Tema 5
Ejercicio 1
Apartado 1.1  si / no / parcialmente
Apartado 1.2  si / no / parcialmente
Apartado 1.3  si / no / parcialmente
Apartado [...]
Apartado [...]
*/
```

Entrega.

Se debe entregar un archivo comprimido ZIP con los archivos fuente (extensión .cs) de los ejercicios propuestos. En esta entrega no se está pidiendo la entrega de ningún proyecto.

- Nombre del archivo: **Apellidos_Nombre_PracT5.zip**

  Por ejemplo, si te llamas Andrés Pérez Gómez el archivo debe llamarse *Perez_Gomez_Andres_PracT4.zip*

Desarrollo.

Ejercicio único.

Nombre del fichero: "**AlumnosFunciones.cs**"

Puntuación máxima: 10 puntos

Tomando como base el ejercicio "**Alumnos**" entregado por el alumno en la práctica del tema 4, se debe **diseñar un nuevo programa** que aplique los conceptos del diseño modular.

En concreto, se propone que se realice un programa modular siguiendo los requisitos indicados en el enunciado de la práctica del tema 4 y, además, desarrolle como mínimo las funciones indicadas a continuación y cumpla los nuevos requisitos. En caso de contradicción de los nuevos requisitos con los de la práctica anterior, siempre prevalecerán los de esta práctica.

1.1 (0,4 puntos) Requisitos ampliados:

- La nueva capacidad máxima del array de alumnos será de 200.
- De los datos de cada alumno, además se desea almacenar el DNI o NIE, la dirección de residencia y el teléfono.
- El menú de opciones que se mostrará al usuario será el mismo que en la práctica anterior, con una opción más llamada "Mostrar gráfica", que se explica más adelante. El funcionamiento de cada opción también será el mismo que en la práctica anterior, salvo la opción de "Buscar alumno", que cambia como se explica más adelante.
- En el programa principal, se debe sustituir todo el código del "case" correspondiente a cada opción del menú, por las correspondientes llamadas a la funciones que se encargarán de todo el trabajo. Podéis basaros en el ejemplo 05_05c de la página 14 de los apuntes del Tema 5, para ver cómo estructurar el código en funciones y pasarle los parámetros que necesiten desde el programa principal.

Especificación de las **funciones mínimas** a desarrollar:

1.2 (0,4 puntos) La función **Opciones** no recibirá ningún parámetro, devolverá la opción escogida e internamente se encargará de:

- Mostrar por pantalla todas las opciones del menú principal.
- Solicitar al usuario la opción solicitada.

1.3 (1 punto) La función **AñadirAlumno** recibirá como parámetro el array de alumnos, el contador de alumnos (pasado por referencia) y no devolverá nada. Internamente, pedirá al usuario los datos del nuevo alumno, validará que sean correctos, añadirá el alumno al array e incrementará el contador de alumnos. También, deberá comprobar que el alumno a añadir no tiene el mismo DNI o NIE que otro alumno ya almacenado en el array. En el caso de que algún dato sea incorrecto, se pedirá continuamente hasta que sea correcto.

1.4 La función **BorrarAlumno** hará lo siguiente:

- **(0,5 puntos)** Recibirá como parámetro el array de alumnos, el contador de alumnos (pasado por referencia) y no devolverá nada. Internamente, comprobará que haya algo que borrar, mostrará el listado de alumnos y pedirá al usuario que elija la posición a borrar. Después de eso, llamará a *BorrarAuxAlumnos* para realizar el borrado, y en función de lo que devuelva, mostrará un mensaje de "Alumno borrado" u otro de "Error al borrar el alumno".

- **(0,5 puntos)** La función **BorrarAuxAlumno** recibirá como parámetro el array de alumnos, el contador de alumnos (pasado por referencia) y el índice del alumnos a borrar. Internamente, deberá verificar que el índice sea válido, realizará el borrado y el desplazamiento de los alumnos a la derecha del borrado. Esta función devolverá un booleano indicando si se ha podido borrar el alumno o no.

1.5 (1,4 puntos) La función **OrdenarAlumno** recibirá como parámetro el array de alumnos, el contador de alumnos (pasado por valor) y el criterio de ordenación. En esta ocasión, la función podrá utilizarse para ordenar los alumnos por DNI/NIE, por apellidos (en ambos casos, en orden ascendente) o por la ciudad (en orden descendente). Desde el menú principal se la llamará para ordenar por apellidos, pero desde otras partes del programa se la podrá llamar también para ordenar por los otros dos criterios.

- Se valorará de forma positiva la implementación de una única función auxiliar para realizar cualquier tipo de ordenación, según el criterio establecido (DNI/NIE, apellidos o ciudad).
- Se podrá emplear el algoritmo de ordenación que se prefiera de los vistos en el tema 4 (burbuja, selección directa, etc.)
- La función NO debe mostrar nada por pantalla, solo dejar el array ordenado por el criterio indicado.

1.6 (0,2 puntos) La función **CalcularPorcentajes** recibirá como parámetro el array de alumnos, el contador de alumnos (pasado por valor) y no devolverá nada. Internamente, realizará la tarea especificada en la práctica anterior.

1.7 (3 puntos) La función **BuscarAlumnos** hará lo siguiente:

- **(0,5 puntos)** Recibirá como parámetros el array de alumnos, el contador de alumnos (pasado por valor) y no devolverá nada. Internamente, solicitará al usuario que escoja el criterio para buscar la información: por DNI/NIE o por apellidos. A continuación, pedirá el texto a buscar.
- **(0,5 puntos)** Se deberá ordenar, antes de empezar a buscar, el array por el campo indicado (DNI/NIE o apellidos) en orden ascendente. Podéis utilizar para ello la función *Ordenar* comentada anteriormente, llamándola con los parámetros adecuados.
- **(1,5 puntos)** Una vez ordenado el array, se deberá realizar una búsqueda binaria y recursiva sobre el array, buscando el alumno que coincida exactamente con el DNI/NIE o apellidos indicados. Para hacer una búsqueda binaria, el array debe estar previamente ordenado por el campo a buscar, por ello es importante el paso anterior de ordenar el array. Para hacer la búsqueda recursiva, debemos valernos de otra función auxiliar, que podemos llamar ***BusquedaRecursiva****,* que recibirá como parámetros el array previamente ordenado, el texto a buscar, y el campo donde buscar (DNI/NIE o apellidos), además de los parámetros adicionales que podáis necesitar. La función deberá devolver la posición en el array donde se encuentra el alumno buscado, o -1 si no se encuentra.
    - Se valorará de forma positiva la implementación de una única función auxiliar para realizar cualquier tipo de búsqueda, según el criterio establecido: por DNI/NIE o por apellidos.
- **(0,5 puntos**) Una vez finalizada la llamada a la función recursiva, se deberá mostrar por pantalla los datos del alumno encontrado, o el mensaje "No se ha encontrado ningún alumno coincidente", si la función devolvió -1.

1.8 (0,8 puntos) La función **MostrarGrafica** es para la nueva opción del menú que se pide que añadáis. Recibirá como parámetro el array de alumnos, el contador de alumnos (pasado por valor) y no devolverá nada. Realizará la representación de una gráfica de barras vertical del porcentaje de alumnos aprobados y suspensos considerando solo la nota final.

- Las barras se dibujarán con “asteriscos” y un asterisco representará 1/5 del porcentaje. Por ejemplo, si hay un 32% de suspensos, se presentará con 32 * 1/5 = 6 “asteriscos”.
- Dado que no se pueden utilizar métodos y funciones de posicionamiento en consola, primero se “dibujará” la gráfica en horizontal en un array bidimensional con 20 columnas y dos filas, una para el porcentaje de aprobados y la otra para los suspensos. Una vez completada la operación anterior, se deberá mostrar el array bidimensional en pantalla “girándolo” para “pintar” la gráfica de barras en vertical.

Por ejemplo, si hay un 68% de aprobados y un 32% de suspensos.

En el array bidimensional se almacenaría así:

```
APTO  **************
SUSP  ******
```

Y se mostraría por pantalla así:

```
 %
70|    *
65|    *
60|    *
55|    *
50|    *
45|    *
40|    *
35|    *
30|    *      *
25|    *      *
20|    *      *
15|    *      *
10|    *      *
5 |    *      *
   ————————————
      APTO   SUSP
```

1.9 (0,3 puntos) Una función **MostrarAlumnos** que se empleará desde todas las opciones que requieran listar los alumnos por pantalla. Por ejemplo, desde la opción de borrado, o la de ordenación.

1.10 (1,5 puntos) Además de la implementación del programa siguiendo los pasos indicados, se deberá tener en cuenta:

- La usabilidad, es decir, que el usuario que lo emplee sepa en todo momento lo que tiene que introducir, y se le muestre la información adecuada.
- La utilización adecuada de funciones auxiliares.
- La no repetición de código innecesario.
- Para mostrar datos por pantalla o pedir datos al usuario solo estará permitido utilizar las siguientes funciones de Console: ReadLine, Read, WriteLine, Write, Clear.
- En todas las peticiones de datos por consola, en el caso de que algún dato sea incorrecto, se pedirá continuamente hasta que sea correcto.
- Cuando se realiza la petición de números por consola y se quiere evitar que se interrumpa el programa en caso de introducir un dato incorrecto, se podrá encerrar dentro de un bloque *try-catch-finally* una conversión de cadena a número (por ejemplo, con Convert.ToInt32), o utilizar la función alternativa *TryParse,* como se indica en el apartado 5.7.4 del tema.

Observaciones generales de la práctica

- Además de las funciones a implementar que se indican en cada ejercicio, podéis implementar las funciones adicionales que consideréis que pueden ser útiles para resolver el problema propuesto.
- A partir de esta práctica se valorará negativamente la suciedad de código, en lo referente a lo visto en el Tema 9 del bloque 1 del módulo de Entornos de Desarrollo. Fundamentalmente, evaluaremos negativamente las siguientes malas prácticas:
  - Nombres poco apropiados de variables o funciones (ya se venía penalizando con anterioridad).
  - Espaciado y alineación vertical (separación entre funciones y entre bloques de código con propósito diferente).
  - Espaciado y alineación horizontal (incluyendo que las líneas de código no excedan del ancho recomendado por la línea vertical del editor).
- En las funciones que devuelvan algún tipo de dato (*return*), se valorará negativamente que haya más de un punto de salida. Las buenas prácticas de programación indican que las funciones deben tener un único punto de salida (una única instrucción *return*), por lo que, si se emplea más de una, se penalizará.

  Así, por ejemplo, si una función devuelve un booleano que depende de un bucle.

  En lugar de hacer esto:

```
public static bool MiFuncion()
{
   while (condicion1)
   {
      if (condicion2)
         return true;
   }
   return false;
}
```

  Es preferible hacer esto:

```
public static bool MiFuncion()
{
   bool resultado = false;

   while (condicion1 && !resultado)
   {
      if (condicion2)
         resultado = true;
   }

   return resultado;
}
```

- También se valorará negativamente la utilización incorrecta de las estructuras de control, estructuras repetitivas, las instrucciones,... Algunos casos típicos que :
  - Utilización incorrecta de la estructura de control “switch-case”.
  - La instrucción “goto” no debe utilizarse salvo en “switch-case”.
  - Los bucles “for” solo deberían utilizarse cuando se conoce el principio y fin del bucle.