Fundamentos de la programación

Grado en Desarrollo de Videojuegos

Examen parcial Abril 2016

Indicaciones generales:

- Se entregará un único archivo `Program.cs` (generado con MonoDevelop) con el programa completo.
- La línea 1 será un comentario de la forma `// Nombre Apellido1 Apellido2`
- **Lee atentamente** el enunciado e implementa el programa tal como se pide, con los métodos, parámetros y requisitos que se especifican.
- El programa **tiene que compilar** y además de ser correcto, debe estar bien estructurado y comentado. Se valorarán la claridad, la concisión y la eficiencia.

En este ejercicio vamos a implementar un *resolutor de sopas de letras*. Dada una sopa de letras (una matriz de letras) y una secuencia de palabras a buscar, el resolutor buscará cada una de esas palabras en la sopa. Para las que encuentre, indicará la localización. Para las que no estén en la sopa indicará que no están presentes. Por ejemplo, supongamos que tenemos la siguiente sopa (numeramos filas y columnas para manejar después coordenadas):

	0	1	2	3	4	5	6	7	8
0	A	B	C	D	B	A	R	C	O
1	E	K	L	M	N	O	P	Q	R
2	H	T	A	V	I	O	N	O	R
3	C	G	R	T	U	I	T	X	B
4	O	R	O	H	F	O	V	A	Z
5	C	M	P	P	M	E	V	A	N

y la secuencia de palabras a buscar: COCHE AVION BARCO MOTO PATINES

Podemos observar que COCHE aparece en las coordenadas (5,0), en dirección norte; MOTO en las coordenadas (5,4) en dirección noreste; mientras que PATINES no aparece en la sopa. Para codificar las 8 posibles direcciones, en vez de norte, noreste, etc, vamos a utilizar *vectores de incremento*, i.e., vectores que sumados reiteradamente a una posición de la sopa proporcionan una secuencia de letras en una dirección en la sopa. Así por ejemplo, localizamos COCHE en las coordenadas (5,0), en la dirección (-1,0): la posición (5,0) contiene la primera letra C y si vamos sumando a esa posición el vector de incremento (-1,0) obtenemos la posición (4,0) que tiene la letra O, (3,0) que tiene la C, etc. La palabra MOTO se localiza en la sopa en la posición (5,4), dirección (-1,1). La codificación de direcciones es la siguiente:

↖ ↑ ↗	(-1,-1) (-1,0)	(-1,1)
← →	(0,-1)	(0,1)
↙ ↓ ↘	(1,-1) (1,0)	(1,1)

Obsérvese que tanto las posiciones como las direcciones se pueden representar mediante pares de enteros (x,y). Para la sopa tendremos las dimensiones *alto* y *ancho*, y un array de strings, para representar las filas de la sopa de letras. Así pues, el programa tendrá el siguiente aspecto:

```
struct Sopa{ // sopa de letras junto con sus dimensiones
  public int alto, ancho;  // dimensiones
  public string [] matriz; // array de strings
}

class MainClass {
  public static void Main (){
    // sopa de letras del ejemplo
    Sopa s;
    s.alto = 6;
    s.ancho = 9;
    s.matriz = new string[] { // sopa de letras del ejemplo
      "ABCDBARCO" , "EKLMNOPQR" , "HTAVIONOR" , "CGRTUITXB" , "OROHFOVAZ" , "CMPPMEVAN" };

    string[] pals = { "COCHE", "AVION", "BARCO", "MOTO", "PATINES" };
    resuelve (s,pals);
  }
  ...
}
```

Observemos que la matriz de letras se representa como un *array de strings* (en vez de utilizar un array bidimensional de *chars*), donde cada string representa una fila de la sopa de letras. Esto facilita la inicialización, la lectura de archivo, etc, y esta representación permite un manejo similar al array bidimensional, ya que el tipo `string` permite el acceso a componentes de manera similar a un array. Por ejemplo, para acceder a la posición (1,2) de la matriz escribiremos `s.matriz[1][2]` (en vez de `s.matriz[1,2]` como se haría en un array bidimensional).

El algoritmo de resolución que codificaremos es como sigue: cada palabra a localizar en la sopa se busca en cada una de las coordenadas de la matriz, en cada una de las direcciones posibles, hasta encontrarla (sí está) o determinar que no está. Si una palabra aparece más de una vez en la sopa, basta con obtener una de las apariciones. Para ello codificar este algoritmo implementaremos los siguientes métodos (se indican los parámetros y su tipo, pero la forma de paso `ref`, `out`,... debe determinarla el alumno):

- [**1pt**] `static Par[] dirs()`: devuelve un array con las 8 direcciones codificadas como pares, según se ha explicado. Pista: estas direcciones son de la forma `(i,j)` con `i` *recorriendo los valores [-1,0,1]* y `j` *recorriendo los valores [-1,0,1]*, a excepción del par `(0,0)`, que no codifica ninguna dirección.

- [**2pt**] `static bool compruebaPosDir(Sopa s, string pal, Par pos, Par dir)`: determina si la palabra `pal` se encuentra en la sopa `s`, en la posición `pos` en la dirección `dir`.

  Por ejemplo, con la sopa del ejemplo, la palabra `"MOTO"`, la posición `(5,4)` y la dirección `(-1,1)` devolvería `true`. Con este ejemplo y otros similares, pueden hacerse algunas pruebas para comprobar el correcto funcionamiento del método.

- [**2pt**] `static bool buscaDir(Sopa s, string pal, Par pos, Par dir)`: utilizando el método anterior, determina si la palabra `pal` se encuentra en la sopa `s`, en la posición `pos`, en alguna de las direcciones posibles, proporcionadas por el método `dirs`. Si es así, devolverá `true` y en `dir` devolverá

Por ejemplo, para la palabra "MOTO" devolverá true, asignará (5,0) al parámetro pos y (-1,1) al parámetro dir.

- [1pt] `static void resuelve(Sopa s, string [] pals)`: dada una sopa `s` y un array de palabras `pals`, para cada una de ellas determina utilizando el método anterior si está o no en la sopa, indicando posición y dirección en caso afirmativo. Para la sopa y las palabras de nuestro ejemplo, la salida sería:

```
Encontrada COCHE en Posicion (5,0)  direccion (-1,0)
Encontrada AVION en Posicion (2,2)  direccion (0,1)
Encontrada BARCO en Posicion (0,4)  direccion (0,1)
Encontrada MOTO en Posicion (5,4)  direccion (-1,1)
No encontrada PATINES
```

- [2pt] `static void leeSopa(Sopa s, string[] pals)`: este método permitirá leer de un archivo `sopa.txt` una sopa de letras y la lista de la palabras a buscar, y las devolverá en `s` y `pals` respectivamente. Modificar el método `Main` para leer de archivo la sopa en vez de inicializarla ad-hoc tal como se ha presentado arriba.

  El archivo `sopa.txt` contendrá en este orden: el alto y el ancho de la sopa, las filas de la sopa (una por línea), el número de palabras a buscar, las palabras en cuestión (una por linea). Así, para nuestra sopa de ejemplo el archivo sería:

```
6
9
ABCDBARCO
EKLMNOPQR
HTAVIONOR
CGRTUITXB
OROHFOVAZ
CMPPMEVAN
5
COCHE
AVION
BARCO
NAVE
MOTO
```

Recordemos que la declaración y creación del flujo de entrada se hace de la forma:

```
StreamReader entrada = new StreamReader ("sopa.txt");
```

Cada línea de texto se leerá con la instrucción:

```
entrada.ReadLine();
```

que devuelve un `string`, igual que la instrucción `Console.ReadLine()`.