UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Fundamentos de Programación (2021/2022)
1º GII / GII-M / GII-ADE
Conv. Extraordinaria - 10 de Febrero de 2022

Normas para la realización del examen: **Duración: 2 horas**

- **El único material permitido durante la realización del examen es un lápiz o bolígrafo**
- **Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.**
- **No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.**

◁ Ejercicio 1 ▷ Proceso de Kaprekar [3 puntos]

El matemático D.R. Kaprekar descubrió en 1949 una característica del número **6174** (conocido como *constante de Kaprekar*), la cual está relacionada con el siguiente proceso iterativo. Dado un número estrictamente positivo de cuatro cifras (o menos) que tenga al menos dos dígitos diferentes (los números con menos de cuatro dígitos se completan colocando el dígito 0 al principio, por lo que el número 9 se convierte en 0009, por ejemplo):

1. Colocar sus dígitos en orden descendente y en orden ascendente para formar dos nuevos números.
2. Restar el menor al mayor.
3. Volver al paso 1 considerando ahora el resultado de la resta del paso 2.

La curiosa característica de este proceso es que siempre converge al número **6174** (y como mucho, en **7** iteraciones).

Observe cómo se alcanza la constante de Kaprekar para los números 3524, 25 y 1121 en 7 ó menos iteraciones:

	3524		**25**		**1121**
Iteración 1	5432 - 2345 = 3087		5200 - 0025 = 5175		2111 - 1112 = 0999
Iteración 2	8730 - 0378 = 8352		7551 - 1557 = 5994		9990 - 0999 = 8991
Iteración 3	8532 - 2358 = **6174**		9954 - 4599 = 5355		9981 - 1899 = 8082
Iteración 4			5553 - 3555 = 1998		8820 - 0288 = 8532
Iteración 5			9981 - 1899 = 8082		8532 - 2358 = **6174**
Iteración 6			8820 - 0288 = 8532		
Iteración 7			8532 - 2358 = **6174**		

Los únicos números de cuatro cifras para los que el proceso de Kaprekar no converge a 6174 (no se puede aplicar) son aquellos que tienen todas las cifras iguales (1111, 2222, 3333, ...).

Hay que realizar:

1. Una función que cree un `string` invirtiendo los caracteres de otro `string`.
2. Una función que ordene los caracteres de un `string`.
3. Un programa **main** que lea números menores de 10000 hasta que se introduzca el 0, y muestre para cada uno si se puede aplicar Kaprekar, y en ese caso cuántas iteraciones tarda en alcanzar el número **6174**. Para realizarlo puede asumir que ya se dispone de la implementación de las siguientes funciones:

- `string ToStringInt (int entero, int num_casillas);`

  que devuelve el valor textual (en forma de `string`) del dato `entero`. Para ello emplea un mínimo de `num_casillas` casillas, si dicho es mayor que el que se necesita, se rellena, al principio, con el carácter `0`.

- `bool ValoresDistintos(string cadena);`

  que indica si los caracteres de la cadena son todos distintos o no.

Recuerde que dispone de la función `int stoi(string cadena)` de la biblioteca `string`, para convertir el dato `string cadena` a un valor `int` y devolverlo.

◁ Ejercicio 2 ▷ Mayor palíndromo [2 puntos]

Vamos a trabajar sobre la clase `SecuenciaCaracteres` que permite almacenar y manipular una secuencia de caracteres. Su descripción aparece en la tabla incluida abajo y no es necesario implementar los métodos que se enumeran.

Un **palíndromo** es una secuencia que se lee igual de izquierda a derecha que de derecha a izquierda. Se pide que defina un método que calcule y devuelva el *palíndromo de mayor longitud* contenido en una secuencia. Dicho palíndromo será un objeto de la clase `SecuenciaCaracteres`. Si hay varios con la misma longitud, tiene libertad para elegir cuál de ellos devolver.

Por ejemplo, el mayor palíndromo contenido en `CABBADE` es la secuencia `ABBA`, mientras que el mayor palíndromo contenido en `ABCDEF` es cualquiera de las subsecuencias de tamaño 1, por ejemplo `A`. El mayor palíndromo contenido en `ABBA` es `ABBA`, y el mayor palíndromo en `gAAtySHHSvvABCCBAfh` es `ABCCBA`.

Se valorará especialmente la eficiencia del método. Puede construir todos los métodos auxiliares que considere. Si lo hace, indique su ámbito (público, privado). NO tiene que construir el programa principal

SecuenciaCaracteres
Datos miembros privados:

`static const int TAMANIO = 500`
`char v[TAMANIO]`
`int utilizados`

Métodos públicos disponibles:

`SecuenciaCaracteres()`
`int Utilizados()`
`int Capacidad()`
`void Aniade (char nuevo)`
`char Elemento (int indice)`
`bool EsPalindromo ()`

◁ Ejercicio 3 ▷ Comparando con vecinos [1 punto]

Se dispone de la clase *TableroCuadrado* descrita como se indica a continuación.

TableroCuadrado
Datos miembros privados:

`static const int MAX = 15`
`int celdas[MAX][MAX]`
`int dimension`

Métodos públicos disponibles:

`TableroCuadrado()`
`TableroCuadrado(int dimension)`
`int GetDimension()`
`int Elemento(int fil, int col)`

Definir el método *TableroImportanciaRelativa* que devuelve un nuevo tablero de las mismas dimensiones que las del tablero sobre el que se ejecuta. El valor de cada posición en el nuevo tablero será el numero de veces que es mayor que el de sus vecinos usando un entorno 3 × 3. Por el hecho de usar esta vecindad **no** se consideran los valores de los bordes del tablero, de manera que, en el resultado, los bordes tienen valor 0.

Por ejemplo, dado el tablero 6 × 6 de la izquierda, se creará el tablero 6 × 6 de la derecha:

5	6	3	6	5	4		**0**	**0**	**0**	**0**	**0**	**0**
4	1	2	2	6	3		**0**	0	3	2	6	**0**
4	7	1	1	6	1		**0**	8	0	0	7	**0**
3	4	1	1	3	5	⇒	**0**	3	0	0	3	**0**
4	6	4	4	6	4		**0**	7	4	5	6	**0**
4	7	3	3	7	6		**0**	**0**	**0**	**0**	**0**	**0**