Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

12 de septiembre de 2019

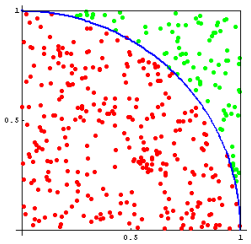
Examen de **Fundamentos de Informática**

Grado en Tecnologías Industriales

Duración total del examen: **3 horas**

NOTA. Responde a cada ejercicio en una hoja diferente.

Ejercicio 1 [3 puntos]



Los métodos de *MonteCarlo* se emplean para realizar cálculos complejos mediante la repetición de experimentos aleatorios. Por ejemplo, podemos aproximar el valor del número π (pi) como se describe a continuación.

Trabajamos con un cuadradro unidad $[0,1] \times [0,1]$, que tiene inscrito un cuarto de círculo de radio 1. Repetimos el siguiente experimento:

- Generamos un punto **aleatorio** de coordenadas (x, y) dentro del cuadrado (ver nota).
- Si el punto está dentro del cuarto de círculo (su distancia al origen es menor o igual que 1), lo contamos como acierto.
- El cociente entre el número de aciertos y el número total de puntos generados es igual a la relación entre las áreas del cuarto de círculo y el cuadrado, que es $\pi/4$, por lo que podemos aproximar π como

  $$\pi = 4(numero\ de\ aciertos/numero\ total\ de\ puntos)$$

  hasta que la diferencia entre dos valores consecutivos de π sea menor que una precisión dada. También se puede fijar un número mínimo de repeticiones.

Se pide escribir en Pascal la función:

```
function pi_approx(minit: integer; precision: real) : real;
```

que calcule una aproximación del número π con este método, realizando un número de iteraciones mayor o igual que `minit` y hasta que la diferencia entre un valor de pi y el anterior sea menor que `precision`.

NOTA: para generar un valor real aleatorio en el rango [0.0,1.0] puedes usar la función predefinida de Pascal llamada '`random`': no tiene parámetros, y cada vez que se la llama devuelve un número distinto con valor completamente aleatorio entre 0.0 y 1.0, ambos incluídos.

Ejercicio 2 [3.5 puntos]

Para la comprobación del tiempo de respuesta de un motor, se dispone en un fichero secuencial de números reales que contiene los datos medidos de revoluciones por segundo de dicho motor a intervalos regulares, de nombre `'revoluciones.dat'`. El número de revoluciones podrá ser positivo si el motor va en sentido dextrógiro o negativo si va en sentido levógiro. Para analizar los datos se necesitan calcular y analizar los **máximos y mínimos locales**.

Se pide escribir un programa que lea la secuencia de datos del fichero `'revoluciones.dat'` y muestre por pantalla primero todos los máximos locales (números mayores que sus dos elementos contiguos) junto con su posición en el fichero, y después todos los mínimos locales (números menores que sus dos elementos contiguos) también junto con su posición en el fichero.

NOTAS:

- Si hay varios elementos contiguos que tienen el mismo valor, entonces ninguno de ellos es máximo ni mínimo local.
- Ni el primer elemento ni el último elemento del fichero podrán ser máximos ni mínimos locales (a pesar de sí que podrían ser máximos y mínimos globales).
- El fichero podría estar vacío.

Ejemplo de ejecución:

Fichero:

10.0	4.0	2.0	3.0	1.0	-1.0	-1.0	-5.0	0.0	-10.0

Salida por pantalla:

```
Máximos:
4 -> 3.0
9 -> 0.0
Mínimos:
3 -> 2.0
8 -> -5.0
```

Ejercicio 3 [3.5 puntos]

El ajedrez es un juego entre dos personas, cada una de las cuales dispone de 16 piezas que se colocan sobre un tablero dividido en 64 casillas, dispuestas en 8 filas y 8 columnas. Los tipos de piezas son peón, alfil, caballo, torre, reina y rey. La Figura 1 (izquierda) ilustra un tablero con las piezas en la posición inicial.

Una forma de representar un tablero de ajedrez en un programa es a través de una matriz 8x8 de datos de tipo **tpCasilla**. El tipo **tpCasilla** es un tipo de dato que almacena si hay una pieza en la casilla, así como el tipo de pieza y color de la pieza si la hubiese. Una torre puede moverse horizontal y verticalmente todas las casillas que se quiera mientras no choque con otra pieza, y puede matar cualquier pieza enemiga con la que choque (y ocupar su lugar). No puede saltar piezas. Teniendo en cuenta esto **se pide**:

1. Crear el tipo de dato **tpCasilla** y el tipo de dato **tpTablero**.
2. Crear un **subprograma** que, dado un `tablero` (con piezas almacenadas en él), y un `color` de pieza, muestre por pantalla la posición y el tipo de todas las piezas que pueden matar las torres de dicho color en el tablero (puede haber 0, 1 o 2 torres de dicho color). Para el tipo, basta con escribir la letra inicial del mismo (y una 'x' en el caso del rey, para distinguirlo de la reina). La cabecera (a completar) será:

```
??? alcanceTorres (??? tablero: tpTablero; ??? color: ???);
```

Así, si se pasasen como parámetros el tablero de la Figura 1 (derecha), y el color negro, el subprograma debería escribir por pantalla:

```
5-1 c
2-8 x
```

Y si se pasasen el mismo tablero y el color blanco, el subprograma debería escribir por pantalla:

```
8-4 a
6-5 p
```

Figura 1

Si no hay piezas al alcance de las torres del color indicado, el subprograma no escribirá nada por pantalla. Se pueden definir otros subprogramas si se considera apropiado.