Fundamentos de Programación (18 de junio de 2018)
(Grados en Ingeniería Mecánica, Eléctrica, Electrónica Industrial y Química Industrial)

Ejercicio 1 – Programación Estructurada (2 puntos): construir un programa en ***C*** que lea por teclado los coeficientes de una ecuación cúbica $A*x^3+B*x^2+C*x+D=0$, con $A\neq0$, y que calcule y presente en pantalla las soluciones reales de dicha ecuación.

Nota: Partiendo de la ecuación general de tercer grado, se convierte a la forma normal dividiendo por ***A***:

$$x^3+a*x^2+b*x+c=0$$

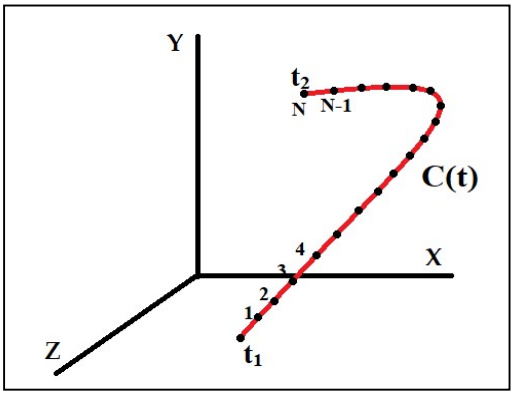
Con el cambio de variable: $x = z - \frac{a}{3}$ se elimina de la forma normal el término cuadrático y se obtiene la forma reducida: $z^3+p*z+q=0$ en la que $p = b - \frac{a^2}{3}$ y $q = \frac{2*a^3}{27} - \frac{a*b}{3} + c$

La forma reducida se resuelve por el método de Cardano:

$p=0$	Una solución real (triple)		$z_0 = z_1 = z_2 = \sqrt[3]{-q}$
$p\neq0$ Discriminante: $\Delta = q^2 + \frac{4*p^3}{27}$	$\Delta > 0$	Una solución real (y dos complejas)	$z_0 = \sqrt[3]{\frac{-q+\sqrt{\Delta}}{2}} - \sqrt[3]{\frac{q+\sqrt{\Delta}}{2}}$
	$\Delta = 0$	Dos soluciones reales (simple y doble)	$z_0 = \frac{3*q}{p}$ $\quad z_1 = z_2 = -\frac{3*q}{2*p}$
	$\Delta < 0$	Tres soluciones reales (***k=0,1,2***)	$z_k = 2 * \sqrt{\frac{-p}{3}} * \cos\left(\frac{1}{3} * \arccos\left(\frac{-q}{2} * \sqrt{\frac{27}{-p^3}}\right) + \frac{2*k*\pi}{3}\right)$

Ejercicio 2 – Diseño Modular (2 puntos): 2.1) construir una función en ***C*** que calcule y devuelva la longitud del tramo de una curva alabeada paramétrica cualquiera $\vec{c}(t) = (x(t), y(t), z(t))$ comprendido entre los valores del parámetro $t=t_1$ y $t=t_2$.
Considerar el siguiente prototipo para dicha función:

```
/* Prototipo de función a construir en ejercicio */
double longCurva(void (*c)(double t, double *x,
      double *y, double *z),double t1, double t2);
```



Nota: la longitud del tramo de la curva se determina mediante la siguiente integral de línea, la cual se puede calcular mediante la división del mismo en ***N*** tramos más pequeños ($\Delta t = \frac{t_2-t_1}{N}$) que se aproximan mediante segmentos de línea recta. Para el diseño de esta función considerar un valor de ***N=10000***.

$$L = \int_{t_1}^{t_2} \|\vec{c}'(t)\| * dt = \int_{t_1}^{t_2} \sqrt{\left(\frac{dx(t)}{dt}\right)^2 + \left(\frac{dy(t)}{dt}\right)^2 + \left(\frac{dz(t)}{dt}\right)^2} * dt \cong \sum_{i=1}^{N} \sqrt{(\Delta x_i)^2 + (\Delta y_i)^2 + (\Delta z_i)^2}$$

$$= \sum_{i=1}^{N} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2}$$

2.2) Construir un programa que, haciendo uso de la función anterior, calcule e imprima en pantalla la longitud del tramo de la hélice cilíndrica ***c(t)=(10*cos(t),-10*seno(t),0.5*t)*** comprendido entre ***t=0*** y ***t=20*π***.

Ejercicio 3 – Estructuras de Datos (2 puntos): construir una función en ***C*** que devuelva el momento (ó torque) ***M*** con respecto a un punto del espacio ***O*** producido por un sistema de fuerzas no concurrentes dado como argumento. Considerar las siguientes estructuras de datos y prototipo de función:

```
/* Estructuras de datos a considerar en el ejercicio */
#define MAX 100
typedef struct{
      double x;
      double y;
      double z;
}tipo_coord;
```

```
typedef struct{
      tipo_coord f;      // Componentes espaciales de la fuerza
      tipo_coord p;      // Punto de aplicación de la fuerza
}tipo_fuerza;
typedef tipo_fuerza tipo_array[MAX];
typedef struct{
      int n;              // N° de fuerzas registradas en las
      tipo_array v;       // primeras posiciones del array
}tipo_sistema;

/* Prototipo de función a construir en el ejercicio */
void momentoSF(tipo_sistema *s, tipo_coord o, tipo_coord *m);
```

<u>*Notas*</u>: momento de una fuerza $\vec{F}$ aplicada en un punto P con respecto a un punto O: $\vec{M}_o = \overrightarrow{OP}\ x\ \vec{F} = \vec{r}\ x\ \vec{F}$

Momento de un sistema de fuerzas: $\vec{M}_o = \sum_i \vec{r}_i\ x\ \vec{F}_i$

Producto vectorial: $\vec{a}\ x\ \vec{b} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix} = (a_y * b_z - a_z * b_y,\ a_z * b_x - a_x * b_z,\ a_x * b_y - a_y * b_x)$

Ejercicio 4 – Archivos (2 puntos): construir un programa que evalúe la exactitud de una máquina clasificadora automática de productos hortícolas, la cual dispone de una cinta de entrada de productos y hasta ***10*** cintas de salida para cada uno de los diferentes tipos de productos a clasificar. Para ello, se ha realizado una prueba de la máquina introduciendo en la cinta transportadora una secuencia conocida de ***N*** hortalizas y se han analizado los resultados obtenidos. Toda esa información se ha registrado en dos archivos de texto con organización secuencial que contienen exactamente el mismo nº de datos ***N***. Ejemplo:

"Referencia.txt"= {1 3 5 2 0 4 2 6 3 2 1 0 1 1 4 4 5 6 0 3 1 3 6}

"Clasificado.txt"={1 3 0 2 0 4 2 5 3 0 1 0 1 1 4 4 5 6 0 2 1 3 6}

El sistema de codificación (clases temáticas) empleado es el siguiente:

1: Berenjena, 2: Calabacín, 3: Pepino, 4: Pimiento amarillo, 5: Pimiento rojo, 6: Pimiento verde, 0: Otros

El programa deberá construir a partir de dichos archivos la matriz de error correspondiente, calcular los parámetros más significativos de la misma (exactitud total, errores de comisión y omisión de cada clase temática), y presentar en pantalla los resultados. Los errores de cada clase temática se presentarán clasificados en orden descendente del error de comisión. <u>*Notas*</u>:

Exactitud total: porcentaje de datos clasificados correctamente.

Error de comisión de la clase i: porcentaje de los datos clasificados en la clase i que no son realmente de dicha clase.

Error de omisión de clase i: porcentaje de los datos de referencia de la clase i que no han sido clasificados correctamente.

.

Matriz de error (tabla de contingencia):

		Datos de referencia						
		0	**1**	**2**	**3**	**4**	**5**	**6**
Dat. clasificados	**0**	3	0	1	0	0	1	0
	1	0	5	0	0	0	0	0
	2	0	0	2	1	0	0	0
	3	0	0	0	3	0	0	0
	4	0	0	0	0	3	0	0
	5	0	0	0	0	0	1	1
	6	0	0	0	0	0	0	2

$\mathbf{a_{ij}}$: nº de productos clasificados por el algoritmo de la clase i pero que realmente son de la clase j

Exactitud total: $\sum_{i=0}^{6} a_{ii} / N$

Error comisión clase i: $C_i = 1 - \frac{a_{ii}}{\sum_{j=0}^{6} a_{ij}}$

Error omisión clase i: $O_i = 1 - \frac{a_{ii}}{\sum_{j=0}^{6} a_{ji}}$

Exactitud total: 0.826

Lista de errores por clase temática:

Producto	**Error Comisión (%)**	**Error Omisión (%)**
Pimiento rojo	50.000	50.000
Otros	40.000	0.000
Calabacín	33.333	33.333
Berenjena	0.000	0.000
Pepino	0.000	0.000
Pimiento amarillo	0.000	0.000
Pimiento verde	0.000	33.333