**Fundamentos de Programación**

**(Grados en Ingeniería Mecánica, Electrónica Industrial y Química Industrial)**

**11 de septiembre de 2012**

***Ejercicios 1, 2 y 3 (2 p/ejercicio):*** implementar en C solamente lo que se indica en el enunciado (no hay que construir el programa completo).

***Ejercicio 4 (4.0 p):*** construir un programa en C lo más modular posible (atendiendo a los criterios de modularidad) y documentar el diseño preliminar con el diagrama de módulos (estructura del programa), la definición en C de las nuevas tipologías de datos y los prototipos de las funciones en C, y el diseño detallado con las definiciones de las respectivas funciones en C.

**Ejercicio 1:** Construir en C una función que permita obtener, mediante aproximaciones sucesivas, una raíz real de una función ***f(x)*** cualquiera dada como argumento (esto es, una solución aproximada de ***f(x)=0***), utilizando la técnica de Newton-Raphson que emplea la siguiente relación:

**xi**: valor estimado actual

**xi+1**: siguiente valor calculado

**f(xi)**: función evaluada en **xi**

**f'(xi)**: primera derivada evaluada en **xi**

**f(xi)**

**xi+1 = xi -**

**f'(xi)**

Para muchas funciones, esta secuencia de aproximaciones **x0**, **x1**, **x2**, **x3**,**...** converge a la solución, teniendo como única condición que la primera aproximación (**x0**) esté lo suficientemente cerca de la solución. La función requiere como entrada una estimación inicial **x0**, y deberá concluirse cuando **|xi+1-xi|<E**, o el número de iteraciones exceda algún límite superior dado o **f'(xi)=0**. Los valores **x0**, **E** y el número máximo de iteraciones serán pasados a la función como argumentos de entrada, devolviendo la función la solución aproximada encontrada (o el último valor calculado) más un carácter que indique si ésta se ha conseguido o no y la causa (‘**e**’: encontrada solución, ‘**m**’: excedido nº máximo de iteraciones, ‘**d**’: encontrada una derivada nula). *Nota*: fórmula de 5 puntos para el cálculo del valor aproximado de la derivada de una función **f(x)** cualquiera en un punto **x0** y con una precisión (**Δx**) dadas:

f(x0-2\*Δx)-8\*f(x0-Δx)+ 8\*f(x0+Δx)- f(x0+2\*Δx)

Lim

Δx🡪0

df(x)

12\*Δx

=

x=x0

dx

**Ejercicio 2:** una ecuación química representa una reacción química en la que intervienen una serie de sustancias o fórmulas moleculares, que se subdividen en reactivos (parte izquierda de la ecuación) y productos (parte derecha), las cuales van precedidas por unos números (coeficientes estequiométricos) que indican cuantas moléculas de cada sustancia intervienen en la reacción. Cada fórmula molecular indica los elementos presentes en una sustancia (mediante su símbolo atómico) así como la cantidad exacta de átomos de cada elemento en una molécula de la misma. Ejemplos:

2 C8H18 + 25 O2 🡪 16 CO2 + 18 H20

2 H2O 🡪 2 H2 + O2

Considerar las siguientes estructuras de datos implementadas en C para representar una ecuación química en la que intervienen un máximo de 5 reactivos y 5 productos, y donde cada fórmula molecular tiene un máximo de 5 elementos atómicos.

|  |  |
| --- | --- |
| #define MAX\_E 5 /\* Máximo de elementos por fórmula \*/  #define MAX\_R 5 /\* Máximo de reactivos por ecuación \*/  #define MAX\_P 5 /\* Máximo de productos por ecuación \*/  #define MAX 108 /\* Nº elementos de tabla periódica \*/ | typedef char cadena2[3];  typedef char cadena15[16];  typedef char cadena30[31]; |
| typedef struct{ /\* Elemento atómico \*/  int z; /\* Numero atómico \*/  float a; /\* Masa atómica \*/  cadena2 sim; /\* Símbolo del elemento\*/  cadena15 nom; /\* Nombre del elemento \*/  }tipo\_elemento;  typedef tipo\_elemento tipo\_t\_p[MAX]; /\*Tabla periódica\*/ | typedef struct{ /\* Atomos-elemento \*/  cadena2 sim; /\* Símbolo del elemento \*/  int n; /\* Nº de átomos en la fórmula \*/  }tipo\_atomos;  typedef tipo\_atomos tipo\_v\_ae[MAX\_E];  /\* Vector de átomos-elemento \*/ |
| typedef struct{ /\* fórmula molecular \*/  cadena30 nom; /\* Nombre de la formula química \*/  int n; /\* Numero de elementos diferentes \*/  tipo\_v\_ae v; /\* Vector de átomos-elemento \*/  }tipo\_formula; | typedef struct{ /\* término (producto/reactivo) \*/  int ce; /\* Coeficiente estequiométrico \*/  tipo\_formula f; /\* fórmula de la molécula \*/  }tipo\_termino; |
| typedef tipo\_termino tipo\_v\_r[MAX\_R];  /\* Vector de reactivos \*/  typedef tipo\_termino tipo\_v\_p[MAX\_P];  /\* Vector de productos \*/ | typedef struct{ /\* Ecuación Química \*/  int nr; /\* Nº de reactivos \*/  int np; /\* Nº de productos \*/  tipo\_v\_r vr; /\* Vector de reactivos \*/  tipo\_v\_p vp; /\* Vector de productos \*/  }tipo\_ecuacion; |

Diseñar e implementar una función en C para validar una ecuación química que acepte como parámetro de entrada una ecuación química (***tipo\_ecuacion***), y que devuelva a través de su identificador un código entero con valor 1 (ecuación correcta) o 0 (ecuación no válida). Dicha función tan solo deberá comprobar si coinciden o no los números de átomos de cada elemento atómico a ambos lados de la ecuación.

**Ejercicio 3:** Diseñar e implementar una función en C para que dados como argumentos una ecuación química validada (***tipo\_ecuacion***) y la tabla periódica de elementos (***tipo\_t\_p***) clasificada en orden alfabético por símbolo del elemento, calcule e imprima en pantalla las masas en gramos de cada reactivo de la misma necesarias para producir una determinada cantidad en gramos del primer producto de dicha reacción dada también como argumento. Ejemplo:

*Nota*: para calcular la masa de cada reactivo, basta multiplicar la masa dada del producto por la relación de masas entre ambas sustancias que se deriva de la ecuación ajustada:

m1=m\*223.2/319.2 m2=m\*96.0/319.2

Ecuación ajustada: 4 Fe + 3 O2 🡪 2 Fe2O3

Moles: 4 3 2

Masas moleculares (u): 4\*55.8 3\*32 2\*159.6

223.2 96.0 319.2

Masas (g): m1? m2? m

**Ejercicio 4:** Construir un programa que calcule y grabe la trayectoria que debe seguir el robot “*Curiosity*” en su exploración del suelo de Marte para que partiendo de su posición inicial **(x0,y0)** alcance una posición final **(xfin,yfin)** sorteando los diferentes obstáculos del camino. Para ello se dispone de una imagen digital del terreno de tamaño 1000x1000 píxeles, almacenada en un archivo de texto con el siguiente formato: ***mapa.txt***={0 0 1 0 1…..} donde cada valor representa un píxel de la imagen (0: libre de obstáculos, 1: con obstáculo), estando almacenados todos los píxeles por columnas de oeste a este y de norte a sur. El tamaño de cada píxel coincide con las dimensiones del robot, y éste solo puede efectuar movimientos horizontales ó verticales (en ambas sentidos) con respecto a la orientación del mapa, sin salirse de los límites del mapa disponible. El programa solicitará por teclado las posiciones inicial y final en píxeles (según coordenadas del mapa), calculará una trayectoria posible y la imprimirá en pantalla (o bien imprimirá un mensaje indicando que dicha posición final es inalcanzable). Finalmente, grabará en un archivo binario las instrucciones de movimiento del robot. Ejemplo (con un mapa de 5x5 píxeles de tamaño):

***mapa.txt***={1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0}

**Y**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 1 |  |  |  |  |  |
| 0 |  |  |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 |

**X**

**(x0,y0)**=(0,0)

**(xfin,yfin)**=(4,4)

Trayectoria1={(0,0),(1,0),(2,0),(3,0),(3,1),(3,2),(2,2),(2,3),(2,4),(3,4),(4,4)}

***robot.bin***={E 3, N 2, W 1, N 2, E 2}

Otra solución posible podría ser:

Trayectoria2={(0,0),(0,1),(0,2),(0,3),(1,3),(1,4),(2,4),(3,4),(4,4)}

***robot.bin***={N 3, E 1, N 1, E 3}