Problemas para la asignatura "Fundamentos de Informática".

José Otero Rodríguez

1

Introducción

No hay ejercicios de este tema.

2

Estructuras básicas de algoritmos y de datos.

1. Escribir un programa que calcule y muestre por pantalla las raíces de una ecuación de segundo grado. Considere todos los casos posibles en función del valor del discriminante.

2. Escribir un programa que pida dos números por el teclado y devuelva el menor de ellos.

3. Escribir un programa que pida un número por el teclado y devuelva su valor absoluto, sin usar la librería matemática.

4. Escribir un programa que calcule y muestre por pantalla la suma de los n primeros números naturales, n se pide por teclado.

5. Escribir un programa que calcule y muestre por pantalla el producto de los n primeros números naturales, n se pide por teclado.

6. Escribir un programa que calcule y muestre por pantalla la suma de los n primeros cubos perfectos menores de 10000, devolver también n.

7. Escribir un programa que calcule y muestre por pantalla la media de n números introducidos por el teclado, n también se pide por el teclado.

8. Escribir un programa que calcule y muestre por pantalla la media de n números naturales introducidos por el teclado, el proceso termina cuando se introduce un número menor que cero.

9. Escribir un programa que calcule e de forma aproximada, mediante la sucesión 1+1/1!+1/2!+1/3!+1/4!... . Escribir tres programas, utilizando distintas condiciones de paro:

   a) Número fijo de iteraciones.

   b) Precisión máxima.

   c) Lo que suceda antes.

10. Aproximar e^x por la sucesión $1+x/1!+x^2/2!+x^3/3!+x^4/4!...$. Escribir tres programas utilizando distintas condiciones de paro:

    a) Número fijo de iteraciones.

    b) Precisión máxima.

    c) Lo que suceda antes.

11. La raíz n-ésima de un número puede aproximarse mediante la sucesión definida por la formula recurrente: $x_i = \frac{1}{n}\left((n-1)x_{i-1} + \frac{A}{(x_{i-1})^{n-1}}\right)$ en donde A es el número del cual quiere hallarse su raíz n-ésima, n es el orden de la raíz. Particularizando para el caso n=2 (raíces cuadradas), escribir tres programas que aproximen la raíz cuadrada de un número mediante esta sucesión. Escribir tres programas, utilizando distintas condiciones de paro:

    a) Número fijo de iteraciones.

    b) Precisión máxima.

    c) Lo que suceda antes

12. Abundando en este tema, escribir un programa que calcule una tabla de raíces n-ésimas para varios números. Por ejemplo n= 2, 3, 4, 5 y A= 4, 5, 6, 7, 8, 9, 10. El programa puede terminar por cualquiera de las tres condiciones del ejercicio anterior.

13. Escribir un programa que calcule y muestre por pantalla la suma de las cifras de un número natural expresado en base 10.

14. Escribir un programa que calcule y muestre por pantalla un número natural expresado en base 10 con las cifras en orden inverso.

15. Convertir un número natural de binario a decimal (el número binario se introduce como entero).

16. Convertir un número natural de decimal a binario. Mostrar el resultado en el orden correcto.

17. Escribir un programa que calcule y muestre por pantalla los números menores de 100000 tales que aparezcan al final de su cuadrado, es decir: $xy^2 = zxy$.

18. Escribir un programa que calcule y muestre por pantalla los números menores 100000 tales que sean iguales a la suma de los cubos de sus dígitos, es decir: $xyz = x^3 + y^3 + z^3$

19. Escribir un programa que calcule y muestre por pantalla los números menores de 100000 que son divisibles entre la suma de sus cifras.

20. Escribir un programa que calcule y muestre por pantalla la descomposición de un número en factores primos.

21. Escribir un programa que calcule y muestre por pantalla los 20 primeros números de la sucesión de Fibonacci. La sucesión de Fibonacci se define como:

$$\begin{cases} f_0 &= 1 \\ f_1 &= 1 \\ f_i &= f_{i-1} + f_{i-2} \end{cases}$$

22. Escribir un programa que calcule y muestre por pantalla los números de la sucesión de Fibonacci que en total suman menos de 10000.

23. Escribir un programa que calcule y muestre por pantalla los números de la sucesión de Fibonacci múltiplos de 3 menores de 100.

24. Escribir un programa que calcule y muestre por pantalla los números de la sucesión de Fibonacci menores de 10000 tales que la suma de sus cifras es 7.

25. Escribir un programa que calcule y muestre por pantalla un número de la sucesión de Fibonacci que cumple que la suma de las potencias de sus factores primos es 4.

26. Escribir un programa que calcule y muestre por pantalla la suma de los elementos de un vector.

27. Escribir un programa que calcule y muestre por pantalla el producto de los elementos de un vector.

28. Escribir un programa que calcule y muestre por pantalla la suma de los elementos de una matriz bidimensional.

29. Calcular la distancia euclídea (raíz cuadrada de la suma de los cuadrados de las diferencias de las componentes) entre dos vectores de dimensión "n".

30. Escribir un programa que calcule y muestre por pantalla el máximo de un vector.

31. Escribir un programa que calcule y muestre por pantalla el mínimo de un vector.

32. Escribir un programa que calcule y muestre por pantalla la distancia de Hamming entre dos vectores de booleanos de igual longitud. La distancia de Hamming entre dos vectores de booleanos se define como el número de elementos que, ocupando la misma posición, son distintos.

33. Calcular la traspuesta de una matriz, sin almacenarla en otra matriz.

34. Rotar una matriz 90 en sentido horario, almacenándola en otra matriz. Modificar el programa para girar la matriz en sentido antihorario.

35. Rotar una matriz 180 verticalmente, almacenándola en otra matriz. Modificar el programa para girar la matriz horizontalmente.

36. Escribir un programa que calcule y muestre por pantalla la suma de dos matrices.

37. Escribir un programa que calcule y muestre por pantalla una matriz tipificada: cada elemento es igual a la media de los elementos de su fila.

38. Escribir un programa que calcule y muestre por pantalla la descomposición de una matriz en suma de simétrica y antisimétrica.

39. Escribir un programa que calcule y muestre por pantalla el punto de silla de una matriz, definido como aquel que es mínimo en su fila y máximo en su columna.

40. Escribir un programa que calcule y muestre por pantalla el número de veces que se repite cada elemento de un vector de enteros.

41. Calcular el histograma de un vector de reales, suponga “n” clases distintas que no se solapan. Los números menores que ”mínimo”se considerará que pertenecen a la primera clase. Análogamente, los números mayores que ”máximo”se considerarán pertenecientes a la última clase.

42. Escribir un programa que calcule y muestre por pantalla el producto de dos matrices.

43. Encontrar todos los i,j tales que el elemento correspondiente de una matriz de enteros está rodeado por elementos iguales a 0.

44. Encontrar todos los i,j tales que el elemento correspondiente de una matriz está rodeado por elementos iguales a cero, excepto uno y sólo uno.

45. Encontrar todos los i,j tales que el elemento correspondiente de una matriz está rodeado por elementos distintos de cero.

3

Definición de acciones no primitivas.

1. Escribir una función que dados dos números devuelva el mínimo de ellos.

2. Escribir una función que dado un número devuelva su valor absoluto, sin usar la librería matemática.

3. Escribir una función que sume de los n primeros números naturales, n se pasa a la función como argumento.

4. Escribir una función que calcule el producto de los n primeros números naturales, n se pasa a la función como argumento.

5. Aproximar e por la sucesión $1 + 1/1! + 1/2! + 1/3! + 1/4!...$ escribir tres funciones, en cada caso pasar los argumentos adecuados a la función correspondiente:

   a) Número fijo de iteraciones.

   b) Precisión máxima.

   c) Lo que suceda antes.

6. Aproximar e^x por la sucesión $1 + x/1! + x^2/2! + x^3/3! + x^4/4!....$ Escribir tres funciones, en cada caso pasar los argumentos adecuados a la función correspondiente:

   a) Número fijo de iteraciones.

b) Precisión máxima.

c) Lo que suceda antes.

7. La raíz n-ésima de un número puede aproximarse mediante la sucesión definida por la formula recurrente: $x_i = \frac{1}{n}\left((n-1)x_{i-1} + \frac{A}{(x_{i-1})^{n-1}}\right)$ en donde A es el número del cual quiere hallarse su raíz n-ésima. Particularizando para el caso n=2 (raíces cuadradas), escribir tres funciones que aproximen la raíz cuadrada de un número mediante esta sucesión, en cada caso pasar los argumentos adecuados a la función correspondiente:

   a) Número fijo de iteraciones.

   b) Precisión máxima.

   c) Lo que suceda antes

8. Abundando en este tema, escribir una función que calcule la raíz n-ésima de un número. La función tendrá como argumentos el radicando, el orden de la raíz, la tolerancia y el máximo número de iteraciones a realizar. Modifique la función anterior de modo que desde `main` se tenga el conocimiento de por cual de las dos razones (iteraciones máximas o tolerancia) ha terminado la función.

9. Escribir una función que devuelva la suma de las cifras de un número expresado en base 10.

10. Escribir una función que devuelva un número expresado en base 10 del revés. Usando esta escriba una función booleana que devuelva `true` o `false` en función de si el número es o no es capicúa. Esta segunda función debe llamar a la primera.

11. Escribir una función que convierta un número de binario a decimal (el número binario se introduce como entero).

12. Escribir una función que calcule el término "n"de la sucesión de Fibonacci, "n"se pasa como argumento a la función. Una vez realizada esta función, escríbala en forma recursiva, explique porque esta implementación realiza mas operaciones que la primera. La sucesión de Fibonacci se define como: $\begin{cases} f_0 &= 1 \\ f_1 &= 1 \\ f_i &= f_{i-1} + f_{i-2} \end{cases}$

13. Escribir una función que calcule el factorial de un número. Utilizando esta función escriba otra que calcule el número combinatorio "n sobre m". Una vez resuelto este problema de esta forma, escriba una función diseñada específicamente para que calcule dicho número combinatorio, de modo que se realice el menor número posible de operaciones.

14. Escribir una función que elimine los elementos duplicados en un vector desordenado. Por ejemplo, si el vector inicial contiene los elementos 3,1,1,1,2,2,3,1,9,1, la función devolverá un vector de tamaño 4, que contendrá los elementos 3,1,2,9.

15. Repetir el ejercicio anterior, en este caso como función genérica.

16. Escribir una función que busque dentro de una matriz de enteros la submatriz que posee más términos iguales a una dada. La función devolverá los índices en donde esté situada la submatriz dentro de la matriz.
Ejemplo:

$$M = \begin{pmatrix} 1 & 7 & 3 & 5 \\ 0 & 1 & 4 & 7 \\ -1 & 45 & -10 & 8 \end{pmatrix} \quad m = \begin{pmatrix} 0 & 4 \\ 45 & -10 \end{pmatrix}$$

Devolvería los valores 1, 1.

17. Escribir una función que reciba un vector de enteros y devuelva la suma máxima de los segmentos del vector. Un segmento es un subconjunto de componentes correlativas de un vector. Por ejemplo:
{43 -52 78 15 -67 39 93 -99 -32 48}
La suma del segmento comprendido entre las componentes segunda y cuarta es -52+78+15=41. La suma máxima de los segmentos de este vector es 158, que corresponde al segmento comprendido entre las componentes tercera y séptima.

18. Dados dos vectores de enteros ordenados de menor a mayor, escribir una función booleana que devuelva `true` si existen dos elementos pertenecientes a cada uno de los vectores, tales que el valor absoluto de la diferencia sea menor que 9. La función devolverá `false` en caso contrario.

19. Escribir una función que reciba como argumento un vector de enteros y calcule la menor diferencia entre los elementos del vector. Por ejemplo:
{23 5 2 31 7 13 54 19 44}
La menor diferencia se da entre 5 y 7, la función debe devolver 2.

20. Escribir una función que devuelva un `vector<int>` a partir de un `vector<vector<int> >`, en el cual cada una de sus filas es un vector de enteros ordenados de mayor (columna 0) a menor (última columna). En el `vector<int>` que devuelve la función, deben de estar todos los elementos del `vector<vector<int> >`, mezclados en el mismo orden.

21. Escribir una función que reciba un `vector<int>` y devuelva dos, uno conteniendo los elementos de valor par y otro conteniendo los elementos de valor impar.

22. Sea un vector de booleanos, se dice que el vector es par si hay un número par de `true` e impar en caso contrario. Por ejemplo:
{`true false false false true false`} es par, ya que existen 2 `true`.
{`false true true false true true false true false`} es impar, ya que existen 5 `true`.
Escribir una función que devuelva `true` si el vector es par, `false` en caso contrario.

    a) De forma iterativa.

    b) De forma recursiva.

23. Calcular el determinante de una matriz desarrollándolo por adjuntos.

4

Definición de nuevos tipos de datos.

1. Implementar el TAD `resistencia`.
   Atributos:
   `valor` tipo real, codifica la resistencia en Ohmios.
   `pmax` tipo real, codifica la potencia máxima disipable en Watios.
   `tol` tipo real, tolerancia o error máximo en % sobre el valor de tipo real.
   Constructores:
   Constructor con un sólo parámetro, el valor de la resistencia, `pmax` se inicializa con el valor 1.0 W y la `tol` con el 5.0 %.
   Constructor con tres parámetros, uno para cada atributo.
   Constructor de copia.
   Operadores:
   Operador de asignación.
   Operador `+`, representa la asociación en serie. La expresión `R1+R2` devuelve un valor de tipo `resistencia` con los siguientes atributos:
   `valor=R1+R2`.
   `pmax=resistencia3*menor(pmax1/resistencia1, pmax2/resistencia2)`
   `tol=(R1, R2)`
   Operador `*`, representa la asociación en paralelo. La expresión `R1*R2` devuelve un valor de tipo `resistencia` con los siguientes atributos:
   `valor=R1*R2/(R1+R2)`
   `pmax=menor(pmax1*resistencia1, pmax2*resistencia2)/resistencia3`
   `tol=mayor(R1, R2)`

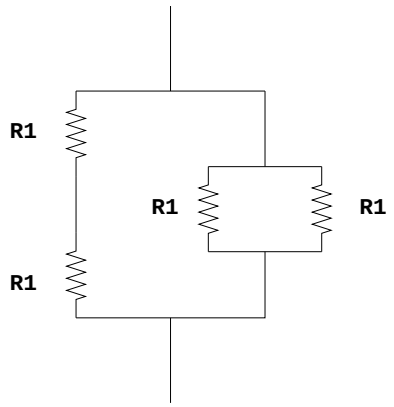

Figura 4.1: Circuito de ejemplo.

a) Implementar la clase `resistencia`

b) Escribir un programa que use la clase anterior para calcular la resistencia equivalente del circuito que se muestra en la figura. Los valores de las resistencias son:

resistencia	valor	potencia máxima	tolerancia
R1	7	2	2
R2	3	1	2
R3	40	1	5
R4	10	2	5

2. Implementar el TAD rectángulo en R^2, considere sólo los rectángulos de lados paralelos a los ejes cartesianos.
Atributos:
Cuatro reales que codifican las coordenadas de los vértices superior izquierdo e inferior derecho.
Constructores:
Constructor por defecto.
Constructor a partir de cuatro reales.
Funciones:
`area` devuelve el área de un rectángulo.
Operadores:

`*` devuelve el rectángulo resultado de realizar la intersección de otros dos.
`<` devuelve `true` si el primer rectángulo está contenido en el segundo, `false` en caso contrario.

3. Implementar el TAD `punto2d`.
   Atributos:
   Dos reales, codificando sus coordenadas.
   Constructores:
   Constructor por defecto.
   Constructor a partir de dos reales.
   Funciones:
   `distancia` devuelve la distancia de un punto a otro.

4. Implementar el TAD `circa` (circunferencia).
   Atributos:
   Tres atributos de tipo real codificando las coordenadas del centro y el radio.
   Constructores:
   Constructor por defecto.
   Constructor a partir de cuatro reales.
   Funciones:
   `tangente` devuelve la esfera de centro conocido tangente a una dada. Si hay dos soluciones, escoja la que prefiera.
   Operadores:
   `<` devuelve `true` si la primera circunferencia está contenida en la segunda, `false` en caso contrario.

5. Repita el ejercicio anterior utilizando un objeto de tipo `punto2d`, implementado en un ejercicio anterior, como atributo que codifica el centro. Aproveche la existencia de la función `distancia` para ese tipo de dato.

6. Implementar el TAD `intervaloR`, que representa un intervalo cerrado en R.
   Atributos:
   Dos reales codificando el extremo derecho e izquierdo.

Constructores:
Constructor por defecto.
Constructor a partir de dos reales.
Operadores:
`<` devuelve `true` si el primer intervalo está contenido en el segundo, `false` en caso contrario.
`*` devuelve la intersección de dos intervalos.

7. Implementar el TAD `intervaloR2`, que representa un intervalo cerrado en R^2.
Atributos:
Dos objetos de tipo `intervaloR`.
Constructores:
Constructor por defecto.
Constructor a partir de cuatro reales.
Constructor a partir de dos objetos de tipo `intervaloR`. Utilice inicializadores aquí
Operadores:
`<` devuelve `true` si el primer intervalo está contenido en el segundo, `false` en caso contrario.
`*` devuelve la intersección de dos intervalos.
Funciones:
`area` devuelve el área de un intervalo en R2. Considere la modificación de la clase `intervaloR` para añadirle una función pública que devuelva la longitud de un objeto de ese tipo.

8. Implementar la clase `segmento`, que representa un segmento delimitado por dos puntos en R^2.
Atributos:
Dos objetos de tipo `punto2d`, definido en un ejercicio anterior.
Constructores:
Constructor por defecto, utilice inicializadores.
Constructor a partir de dos objetos de tipo `punto2d`.
Constructor a partir de cuatro números reales, utilice inicializadores.
Funciones:
`longitud`, devuelve la longitud de un segmento.
Operadores:
`<` devuelve `true` si un segmento es menor que otro.

NOTA: escriba la definición del operador `<` de dos formas, utilizando la función `longitud` y sin utilizarla.

9. Implementar el TAD `cuaternio`. Un cuaternio es el análogo a un número complejo en R^3.
Consideremos el espacio vectorial que tiene como base $1, i, j, k$. Un elemento del mismo será $a + bi + cj + dk$, (a, b, c, d son números reales) y se denomina cuaternio. La suma y el producto por un escalar de esos vectores se define de la forma habitual. La multiplicación entre esos vectores se define en función de los elementos de la base, de acuerdo con la siguiente tabla:

	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

El producto de dos vectores $q = a+bi+cj+dk$ y $q' = a'+b'i+c'j+d'k$ se realiza multiplicando cada término del primero por todos los del segundo, de modo que resultará un real multiplicando a un elemento de la base, que se obtiene de la tabla anterior. Finalmente se suman todos los coeficientes que multiplican a cada elemento de la base.
Se define el conjugado de un cuaternio como aquél en el que se cambia el signo de todos los coeficientes a excepción del primero. El conjugado de $q = a + bi + cj + dk$ es $\overline{q} = a - bi - cj - dk$. El producto de un cuaternio por su conjugado se denomina norma y se cumple que $q\overline{q} = a + b^2 + c^2 + d^2$. El inverso de un cuaternio es:

$$q^{-1} = \frac{1}{q\overline{q}}\overline{q}$$

Atributos:
Cuatro números reales.
Constructores:
Constructor por defecto.
Constructor a partir de cuatro reales.
Operadores:
+,-,* interno, * por un real (a la izquierda y a la derecha), /.
Funciones:

`conjugado` devuelve el conjugado de un cuaternio.
`norma` devuelve la norma.

10. Implementar las clases `punto` y `cuadrangulo` (polígono de cuatro lados), de acuerdo con las siguientes consideraciones:

    a) La clase `punto` tiene dos atributos de tipo real, representa un punto en R^2.

    b) La clase `cuadrangulo` tiene un atributo de tipo `vector<punto>`, una vez inicializado el objeto, contiene los cuatro vértices del cuadrángulo, de tipo `punto`.

    c) La clase `cuadrangulo` posee una función miembro que devuelve el área de un objeto de ese tipo.

    NOTA: defina también los constructores necesarios. Observe que el orden en el que se tomen los vértices define un cuadrángulo o una figura que no lo es. Para calcular el área utilice el producto vectorial de sus lados.

11. Implemente el tipo de datos paramétrico `conjunto<T>`. Defina los operadores * (intersección), + (unión), == (igualdad de conjuntos), = (asignación) y < (pertenencia de un elemento a un conjunto), junto con los constructores oportunos. A continuación escriba un programa en el que se definan dos variables A, B del tipo `conjunto<conjunto<char> >` y otras dos, X e Y del tipo `conjunto<char>`, y realice sobre ellas las siguientes operaciones:

    a) Declarar las variables.

    b) Asignar a A el conjunto vacío.

    c) Asignar a X el conjunto formado por el carácter 'x'.

    d) Asignar a Y el conjunto formado por el carácter 'y'.

    e) Asignar a B el conjunto cuyos elementos son X e Y.

    f) A=A*B

    g) B=A+B

    h) si X<A escribir "X pertenece a A"

i) si X<B escribir "X pertenece a B"

NOTA: puede utilizar un vector para almacenar los elementos de cada conjunto.

12. Implemente el tipo de datos paramétrico `pila<T>`. Una pila es una agregación de elementos del mismo tipo. La introducción o extracción de elementos de ese agregado induce un orden en los mismos, de modo que el último de los que se ha introducido es el primero que se puede extraer. Utilizando un vector como almacenamiento de estos elementos y un entero para señalar el elemento superior de la pila, implemente las siguientes operaciones:
`primero`, devuelve una copia del elemento que está en la parte superior de la pila.
`desapila`, elimina el elemento que está en la cima de la pila.
`apila`, añade un elemento a la pila, por la parte de "arriba".
`vacia`, devuelve `true` si la pila está vacía, `false` en caso contrario.
NOTA: gestione como desee cosas como el número máximo reservado inicialmente para almacenar elementos y el comportamiento de la pila si se añaden más elementos que el máximo inicial o si se intenta utilizar `primero` o `desapila` si la pila está vacía.

5

Estructuras de datos.

1. Escribir en la pantalla el contenido de una pila, iterativa y recursivamente.

2. Escribir en la pantalla el contenido de una cola, iterativa y recursivamente.

3. Escribir en la pantalla el contenido de una pila al revés, iterativa y recursivamente.

4. Sumar iterativa y recursivamente los elementos de una pila de enteros.

5. Escribir una función booleana que devuelva `true` si un elemento pertenece a una pila, `false` en caso contrario.

6. Escribir una función booleana que devuelva `true` si un elemento pertenece a una cola, `false` en caso contrario.

7. Escribir una función que devuelva una pila de enteros conteniendo los elementos de valor par de una pila de enteros, en el mismo orden.

8. Escribir una función que devuelva una cola de enteros conteniendo los elementos de valor par de otra cola de enteros, en el mismo orden.

9. Concatenar dos pilas a y b, de modo que los elementos de b queden por encima de los de a. El orden relativo de los elementos de a y b dentro de la nueva pila debe permanecer inalterado.

10. Concatenar dos colas a y b, de modo que los elementos de b queden por encima de los de a. El orden relativo de los elementos de a y b dentro de la nueva cola debe permanecer inalterado.

11. Mezclar dos colas ordenadas en otra cola ordenada de igual forma.

12. Mezclar dos pilas ordenadas en otra pila ordenada de igual forma.

13. Escribir un programa que analice si una expresión, que se introduce por el teclado, está bien parentizada o no. La expresión contiene '()', '[]' y ''. Está bien parentizada si por cada uno de los elementos anteriores 'abierto' existe otro 'cerrado' de modo que las parejas de caracteres abiertos y cerrados o son disjuntas, o contienen a otras parejas o están contenidas en otras parejas.

14. Escribir un programa que genere y resuelva un laberinto de forma aleatoria, representado por un `vector<vector<char> >`. Utilice un carácter para representar una pared sólida y un espacio para representar un hueco.