

Introducción a los algoritmos

Asignatura: Introducción a la Ingeniería en Computación

Prof. Mónica Saettone

Contenido

¿Qué es un algoritmo?	2
· Finitud:	2
· Definibilidad:	2
· Entrada:	2
· Salida:	2
· Efectividad:	2
Otra definición:	2
Clasificación de algoritmos.....	2
· Algoritmo determinista	2
· Algoritmo no determinista	2
Diagramas de Flujo	3
Algoritmo y Pseudocódigo	4
- En la sección de cabecera.....	4
- En la sección de declaraciones	4
- En el cuerpo.....	4
Pseudocódigo - Tipo de Datos:.....	5
Pseudocódigo - Variables:	5
Pseudocódigo - Constantes:.....	6
Pseudocódigo – Operadores y Expresiones:	7
Pseudocódigo – Asignaciones:	8
Pseudocódigo – Salida:.....	8
Pseudocódigo – Entrada:	9
Pseudocódigo – Alternativa Simple:	10
Pseudocódigo – Estructuras repetitivas: Ciclos, bucles o lazos:	11
Mientras	11
Repetir-hasta.....	11
Desde o Para	11
Ejemplo 1:.....	13

Ejemplo 2:.....	14
Ejercicios:	16

Introducción a los algoritmos

¿Qué es un algoritmo? (<http://www.algoritmia.net/articles.php?id=30>)

De manera informal se define un algoritmo como un conjunto finito de reglas que dan una secuencia de operaciones para resolver todos los problemas de un tipo dado. De forma más sencilla, podemos decir que un algoritmo es un conjunto de pasos que nos permite obtener un dato. Además debe cumplir con las siguientes condiciones:

- **Finitud:** el algoritmo debe acabar tras un número finito de pasos. Es más, es casi fundamental que sea en un número razonable de pasos.
- **Definibilidad:** el algoritmo debe definirse de forma precisa para cada paso, es decir, hay que evitar toda ambigüedad al definir cada paso. Puesto que el lenguaje humano es impreciso, los algoritmos se expresan mediante un lenguaje formal, ya sea matemático o de programación para un computador.
- **Entrada:** el algoritmo tendrá cero o más entradas, es decir, cantidades dadas antes de empezar el algoritmo. Estas cantidades pertenecen además a conjuntos especificados de objetos. Por ejemplo, pueden ser cadenas de caracteres, enteros, naturales, fraccionarios, etc. Se trata siempre de cantidades representativas del mundo real expresadas de tal forma que sean aptas para su interpretación por el computador.
- **Salida:** el algoritmo tiene una o más salidas, en relación con las entradas.
- **Efectividad:** se entiende por esto que una persona sea capaz de realizar el algoritmo de modo exacto y sin ayuda de una máquina en un lapso de tiempo finito.

Otra definición:

(Tomado de http://www.bibliodgsca.unam.mx/tesis/tes9sarg/sec_4.htm)

Por algoritmo se entiende "una lista de instrucciones donde se especifica una sucesión de operaciones necesarias para resolver cualquier problema de un tipo dado". Los algoritmos son modos de resolución de problemas, cabe aclarar que no sólo son aplicables a la actividad intelectual, sino también a todo tipo de problemas relacionados con actividades cotidianas.

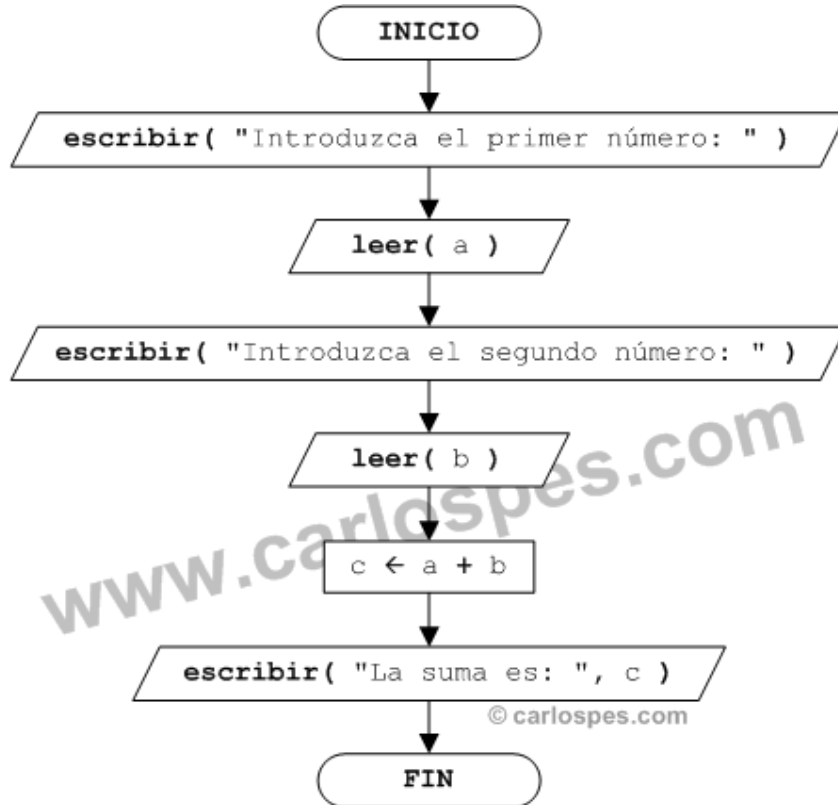
Clasificación de algoritmos

- **Algoritmo determinista:** en cada paso del algoritmo se determina de forma única el siguiente paso.
- **Algoritmo no determinista:** deben decidir en cada paso de la ejecución entre varias alternativas y agotarlas todas antes de encontrar la solución.

Representación de un Algoritmo

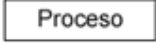
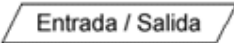


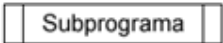
Diagramas de Flujo

Los algoritmos también se pueden representar, gráficamente, por medio de diagramas de flujo. Dicho de otra forma, un diagrama de flujo representa, de manera gráfica, el orden de los pasos o acciones de un algoritmo. Por ejemplo, el algoritmo escrito en pseudocódigo también se puede representar mediante el siguiente diagrama:



© carlospes.com

Símbolos gráficos más utilizados para dibujar algoritmos por medio de diagramas de flujo (ordinogramas):

Símbolo	Descripción (significado):
 Proceso	Instrucción de asignación
 Entrada / Salida	Instrucción de entrada o de salida
 Terminal	Inicio o Fin del algoritmo
 Decisión	Instrucción de control
 Subprograma	Llamada a un subprograma
↓	Indica el orden de las acciones del algoritmo
○	Conector de reagrupamiento de una instrucción de control

Algoritmo y Pseudocódigo

(Tomado de: http://www.carlospes.com/curso_de_algoritmos/01_01_introduccion.php)

Un algoritmo establece, de manera genérica e informal, la secuencia de pasos o acciones que resuelve un determinado problema. Los algoritmos constituyen la documentación principal que se necesita para poder iniciar la fase de codificación y, para representarlos, se utiliza, fundamentalmente, dos tipos de notación: pseudocódigo y diagramas de flujo. El diseño de un algoritmo es independiente del lenguaje que después se vaya a utilizar para codificarlo.

Un algoritmo escrito en pseudocódigo siempre se suele organizar en tres secciones: cabecera, declaraciones y cuerpo.

- **En la sección de cabecera** se escribe el nombre del algoritmo.
- **En la sección de declaraciones** se declaran algunos objetos (variables, constantes,...) que va a utilizar el programa.
- **En el cuerpo** están descritas todas las acciones que se tienen que llevar a cabo en el programa, y siempre se escriben entre las palabras inicio y fin.

Por ejemplo, el algoritmo de un programa que va a calcular la suma de dos números enteros cualesquiera introducidos por el usuario y, después, va a mostrar por pantalla el resultado obtenido, puede ser el siguiente:

```

algoritmo                               Sumar
variables
  entero                                a,      b,      c
inicio

```

```

escribir( "Introduzca el primer número (entero): " )
leer( a )
escribir( "Introduzca el segundo número (entero): " )
leer( b )
c ← a + b
escribir( "La suma es: ", c )
fin

```

Pseudocódigo - Tipo de Datos:

Los datos que utilizan los programas se pueden clasificar en base a diferentes criterios. Uno de los más significativos es aquel que dice que todos los datos que utilizan los programas son simples o compuestos.

Un dato simple es indivisible (atómico), es decir, no se puede descomponer.

Ejemplo 1: Un año es un dato simple. Año...: 2006

Un año se expresa con un número entero, el cual no se puede descomponer. Sin embargo, un dato compuesto está formado por otros datos.

Ejemplo 2: Una fecha es un dato compuesto por tres datos simples (día, mes, año).

Fecha: Día...: 30 Mes...: 11 Año...: 2006

Ejemplo 3: Otro ejemplo de dato simple es una letra. Letra...: t

Una letra se representa con un carácter del alfabeto. Pero, cuando varias letras se agrupan, entonces se obtiene un **dato compuesto** por varios caracteres.

Ejemplo 4: Para formar un nombre de persona se utilizan varios caracteres.

Nombre...: Ana (dato compuesto por tres caracteres)

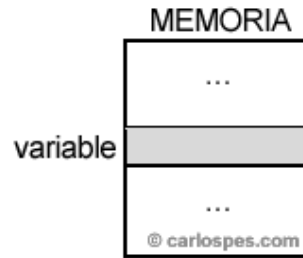
Tipos de datos:

- Entero
- Real
- Lógico
- Carácter
- Cadena

De ellos, tan solo el tipo cadena es compuesto. Los demás son los tipos de datos simples considerados estándares.

Pseudocódigo - Variables:

En programación, una variable representa a un espacio de memoria en el cual se puede almacenar un dato. Gráficamente, se puede representar como:



El programador, cuando desarrolla un programa (o diseña un algoritmo), debe decidir:

- Cuantas son las variables que el programa necesita para realizar las tareas que se le han encomendado.
- El tipo de dato que puede almacenar cada una de ellas.

Durante la ejecución de un programa, el valor que tome el dato almacenado en una variable puede cambiar tantas veces como sea necesario, pero, siempre, tomando valores pertenecientes al tipo de dato que el programador ha decidido que puede almacenar dicha variable, ya que, el tipo de dato de una variable no puede ser cambiado durante la ejecución de un programa.

Pseudocódigo - Constantes:

Una constante representa a un valor (dato almacenado en memoria) que no puede cambiar durante la ejecución de un programa.

En C, una constante puede ser de tipo entero, real, carácter, cadena o enumerado. Las constantes de tipo enumerado se van a estudiar en el apartado 4.1 Datos de tipos enumerados. En cuanto a las demás, se pueden expresar de dos formas diferentes:

- Por su valor.
- Con un nombre (identificador).

Ejemplo 1: Las siguientes constantes de tipo entero están expresadas por su valor: -5, 10

Para expresar una constante con un nombre, la constante debe ser declarada previamente. Todas las constantes que se declaran en un programa son definidas de la misma forma, indicando de cada una de ellas:

1. Su nombre (mediante un identificador).
2. El valor que simboliza (mediante una expresión).

En pseudocódigo, para declarar una constante, vamos a utilizar la sintaxis:

<nombre_de_la_constante> = <expresión>

Y para declarar más de una constante en una misma línea, las separaremos por medio de comas (,).

De modo que, si se quieren declarar las constantes de tipo entero del ejemplo 1, asignándoles un identificador, se puede escribir, por ejemplo:

TEMPERATURA = -5

MES = 10

Pseudocódigo – Operadores y Expresiones:

En un programa, el tipo de un dato determina las operaciones que se pueden realizar con él. Por ejemplo, con los datos de tipo entero se pueden realizar operaciones aritméticas, tales como la suma, la resta o la multiplicación.

Ejemplo 1: Algunos ejemplos son:

- $111 + 6$ (operación suma)
- $19 - 72$ (operación resta)
- $24 * 3$ (operación multiplicación)

Todas las operaciones del ejemplo constan de dos operandos (constantes enteras) y un operador. La mayoría de las veces es así, pero, también es posible realizar operaciones con distinto número de operadores y/u operandos.

Ejemplo 2:

- $111 + 6 - 8$ (tres operandos y dos operadores)
- $-((+19) + 72)$ (dos operandos y tres operadores)
- $-(-72)$ (un operando y dos operadores)

En las operaciones del ejemplo se puede observar que los caracteres más (+) y menos (-) tienen dos usos:

- Operadores suma y resta.
- Signos de un número (también son operadores).

Los operadores de signo más (+) y menos (-) son operadores monarios, también llamados unarios, ya que, actúan, solamente, sobre un operando.

Los caracteres abrir paréntesis "(" y cerrar paréntesis ")" se utilizan para establecer la prioridad de los operadores, es decir, para establecer el orden en el que los operadores actúan sobre los operandos.

Un operador indica el tipo de operación a realizar sobre los operandos (datos) que actúa. Los operandos pueden ser:

- Constantes (expresadas por su valor o con un nombre (identificador)).
- Variables.
- Llamadas a funciones.
- Elementos de formaciones (arrays).

Ejemplo 3: Declaraciones de constantes y variables:

- $PI = 3.141592$
- $entero\ numero = 2$
- $real\ radio_circulo = 3.2$

Algunos ejemplos de expresiones son:

- $2 * PI * radio_circulo$
- $(PI * PI)$
- $numero * 5$

De sus evaluaciones se obtienen los valores:

- 20.106189 (valor real) ($2 * 3.141592 * 3.2$)
- 9.869600 (valor real) ($3.141592 * 3.141592$)
- 10 (valor entero) ($2 * 5$)

Un operador siempre forma parte de una expresión, en la cual, el operador siempre actúa sobre al menos un operando. Por el contrario, un operando sí puede aparecer solo en una expresión.

En programación, de la evaluación de una expresión siempre se obtiene un valor. Dicho valor puede ser de tipo: entero, real, lógico, carácter o cadena. Por consiguiente, una expresión puede ser:

- Aritmética (devuelve un número entero o real).
- Lógica (devuelve un valor lógico: verdadero o falso)
- De carácter (devuelve un carácter representable por el ordenador).
- De cadena (devuelve una cadena).

Pseudocódigo – Asignaciones:

Una instrucción de asignación (o simplemente asignación) consiste en asignar el resultado de la evaluación de una expresión a una variable.

En pseudocódigo, la sintaxis para escribir una asignación es:

<nombre_de_la_variable> ← <expresión>

El valor (dato) que se obtiene al evaluar la <expresión> es almacenado en la variable que se indique.

Ejemplo 1: Dadas las declaraciones

- $PI = 3.141592$
- real area, longitud, radio = 5.78

algunas instrucciones de asignación son:

- $area \leftarrow PI * radio * 2$
- $longitud \leftarrow 2 * PI * radio$

Por consiguiente, las variables area y longitud almacenarán los valores:

- 57.046290 (se obtiene de $3.141592 * 5.78 * 2$)
- 36.316804 (se obtiene de $2 * 3.141592 * 5.78$)

Pseudocódigo – Salida:

Una instrucción de salida (o simplemente salida) consiste en llevar hacia el exterior los valores (datos) obtenidos de la evaluación de una lista de expresiones. Normalmente, los datos son enviados a la salida estándar (la pantalla), pero, también existen otros dispositivos de salida (la impresora, el plotter,...).

En pseudocódigo, la sintaxis de una instrucción de salida es:

escribir(<expresión_1>, ..., <expresión_n>)

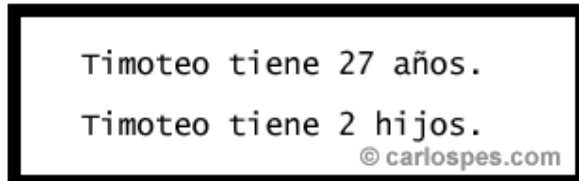
También se puede escribir como: escribir(<lista_de_expresiones>)

Ejemplo: Partiendo de las variables

- cadena nombre = "Timoteo"
- entero edad = 27, hijos = 2

escribir(nombre, " tiene ", edad, " años.")

escribir(nombre, " tiene ", hijos, " hijos.")



```
Timoteo tiene 27 años.
Timoteo tiene 2 hijos.
© carlospes.com
```

Pseudocódigo – Entrada:

Una instrucción de entrada (o simplemente entrada) consiste en asignar a una o más variables, uno o más valores (datos) recibidos desde el exterior. Normalmente, los datos son recogidos desde la entrada estándar (el teclado), pero, también existen otros dispositivos de entrada (el ratón, el escáner,...).

En pseudocódigo, la sintaxis de una instrucción de entrada es:

leer(<nombre_de_la_variable_1>, <nombre_de_la_variable_2>, ..., <nombre_de_la_variable_n>)

También se puede escribir como:

leer(<lista_de_variables>)

Ejemplo 1: Partiendo de las variables

- cadena nombre, apellidos
- entero edad

para cada una de ellas se puede recoger un valor (dato) desde el teclado, escribiendo:

- leer(nombre)
- leer(apellidos)
- leer(edad)

Otra posibilidad es

- leer(nombre, apellidos, edad)

Ejemplo 2: Si se han declarado

- cadena nombre
- real numero

al escribir

```

escribir( "Introduzca su nombre: " )
leer( nombre )
escribir( "Introduzca un número real: " )
leer( numero )
escribir( nombre, ", el doble de ", numero, " es: ", numero * 2 )

```

Estructuras de control

Pseudocódigo – Alternativa Simple:

Una instrucción alternativa simple (o simplemente alternativa simple) es una variante (más sencilla) de una instrucción alternativa doble. En pseudocódigo, para escribir una alternativa simple se utiliza la sintaxis:

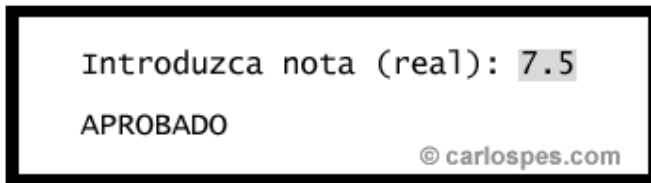
```

si ( <expresión_lógica> )
    <bloque_de_instrucciones>
fin_si

```

Ejemplo: Se quiere diseñar el algoritmo de un programa que:

- Pida por teclado la nota (dato real) de una asignatura.
 - Muestre por pantalla:
- * "APROBADO", en el caso de que la nota sea mayor o igual que 5.



algoritmo Calificacion_segun_nota

variables

real nota

inicio

escribir("Introduzca nota (real): ")

leer(nota)

si (nota >= 5)

escribir("APROBADO")

```
fin_si
```

```
fin
```

Pseudocódigo – Estructuras repetitivas: Ciclos, bucles o lazos:

Las estructuras repetitivas son aquellas que repiten una secuencia de instrucciones un número determinado de veces se denominan bucles y se denomina iteración al hecho de repetir la ejecución de una secuencia de acciones. Un bucle es una estructura que repite una o varias instrucciones dependiendo de una condición lógica.:

Existen diversos tipos de estructuras repetitivas:

- Mientras
- Repetir - hasta
- Desde/Para

Mientras

La estructura repetitiva Mientras (en inglés While) es aquella en que el cuerpo del bucle se repite mientras se cumple una determinada condición

```
Mientras condición hacer
```

```
    acción 1
```

```
    acción 2
```

```
    -
```

```
    -
```

```
    acción n
```

```
Fin Mientras
```

Repetir-hasta

La estructura repetitiva Repetir Hasta que (en inglés Repeat Until), se ejecuta hasta que se cumpla una condición determinada que se comprueba al final del bucle.

Sintaxis:

```
Repetir
```

```
<acciones>
```

```
    -
```

```
    -
```

```
Hasta que <condición>
```

Desde o Para

La estructura repetitiva Desde o Para (en inglés For), ejecuta las acciones del cuerpo del bucle un número especificado de veces y de modo automático controla el número de iteraciones o pasos a través del cuerpo del bucle.

Sintaxis:

Desde $v \leftarrow v_i$ **Hasta** v_f [incremento incr] **Hacer**
<acciones>

.

.

Fin Desde

Para $v \leftarrow v_i$ **Hasta** v_f [incremento incr] **Hacer**
<acciones>

.

.

Fin Para

Ejemplo 1:

```

algoritmo Numeros_del_1_al_10
variables
entero contador
inicio
  contador ← 1 /* Inicialización del contador */
  escribir( contador )
  contador ← contador + 1 /* Incremento */
  mientras ( contador <= 10 ) /* Condición */
fin

```

La traza del algoritmo es:

<u>Secuencia:</u>	<u>Acción (instrucción):</u>	<u>Valor de:</u> contador
1	contador ← 1	1
	Inicio de la iteración 1.	
2	escribir (contador)	1
3	contador ← contador + 1	2
	Fin de la iteración 1.	
4	(Comprobar si contador <= 10)	2
	La condición es verdadera . Inicio de la iteración 2.	
5	escribir (contador)	2
6	contador ← contador + 1	3
	Fin de la iteración 2.	
...		
n-3	(Comprobar si contador <= 10)	10
	La condición es verdadera . Inicio de la iteración 10.	
n-2	escribir (contador)	10
n-1	contador ← contador + 1	11
	Fin de la iteración 10.	
n	(Comprobar si contador <= 10)	11
	La condición es falsa . El bucle finaliza después de 10 iteraciones.	

Figura. Traza del ejemplo 1.

Explicación de la traza:

- En primer lugar, se le asigna el valor **1** a `contador` (acción 1).
- A continuación, se ejecuta el bloque de instrucciones del bucle **hacer... mientras**, mostrándose por pantalla el valor de `contador` (acción 2) y, después, se incrementa en **1** el valor de la variable `contador` (acción 3).

- Una vez ejecutado el bloque de instrucciones, se evalúa la condición de salida del bucle (`contador <= 10`) (acción 4) y, puesto que, es **verdadera**, se ejecuta, de nuevo, el bloque de instrucciones.
- Y así sucesivamente, mientras que, la condición sea **verdadera**, o dicho de otro modo, hasta que, la condición sea **falsa**.

En este algoritmo, el bloque de instrucciones del bucle se ejecuta diez veces (iteraciones).

Ejemplo 2:

Se quiere diseñar el algoritmo de un programa que:

1. Pida por teclado un número (dato entero).
2. Pregunte al usuario si desea introducir otro o no.
3. Repita los pasos 1º y 2º, mientras que, el usuario no responda 'n' de (no).
4. Muestre por pantalla la suma de los números introducidos por el usuario.

En pantalla:

```

Introduzca un número entero: 7
¿Desea introducir otro (s/n)?: s
Introduzca un número entero: 16
¿Desea introducir otro (s/n)?: s
Introduzca un número entero: -3
¿Desea introducir otro (s/n)?: n
La suma de los números introducidos es: 20
© carlospes.com

```

Solución:

```

algoritmo Suma de numeros introducidos por el usuario

variables
  caracter seguir
  entero acumulador, numero

inicio
  /* En acumulador se va a guardar la suma
  de los números introducidos por el usuario. */

  acumulador • 0
  hacer
  escribir( "Introduzca un número entero: " )
  leer( numero )
  acumulador • acumulador + numero
  escribir( "¿Desea introducir otro número (s/n)?: " )
  leer( seguir )

```

```
mientras ( seguir <> 'n' )  
/* Mientras que el usuario desee introducir  
   más números, el bucle iterará. */  
  
escribir( "La suma de los números introducidos es: ",  
          acumulador )  
fin
```

Ejercicios:

EJERCICIO 1 - CLASIFICAR DATOS

Clasifique los siguientes datos en simples y compuestos:

- El número de botones de una camisa.
- La altura de una torre.
- Los datos de una cuenta bancaria.
- El número de pasajeros de un avión.
- El resultado de hacer una apuesta (ganar o perder).
- La capital de Canadá.
- La letra 'b' escrita en mayúscula.

EJERCICIO 2 - TIPOS DE DATOS SIMPLES

De la lista siguiente:

- dato booleano
- dato cadena
- dato enumerado
- dato estructurado
- dato numérico
- dato ordinal
- dato real
- dato verdadero

¿Cuáles son tipos de datos simples?

EJERCICIO 3 - EVALUACIÓN DE EXPRESIONES

Dadas las siguientes declaraciones:

- TRES = 3
- entero a = 5, b = 4
- real x = 5e-2, y = 2.

¿qué valores se obtienen de evaluar las siguientes expresiones?

1. $a = b \bmod \text{TRES}$
2. $6 \text{ div TRES} < \text{TRES mod } 6$
3. $\text{TRES} + b - 1 <> a \text{ o } b \geq -b * a \text{ y } a * 2 \leq 10$
4. $x * y * 10 = 10.E-1$
5. $b \bmod a \text{ div TRES}$
6. $\text{no } (x * a > y / b)$

EJERCICIO 4 - ASIGNACIÓN, SALIDA Y ENTRADA

Dadas las siguientes declaraciones:

- A = 'a'
- ONCE = 11

- entero $r = 50$, $s = 6$, t
- caracter vocal

después de las asignaciones:

1. $t \leftarrow r \text{ div } s$
2. $s \leftarrow r \text{ mod } s$
3. $r \leftarrow \text{ONCE} - t ** s$
4. $\text{vocal} \leftarrow A$

¿qué valores se habrán almacenado en la memoria del ordenador para los objetos declarados?

EJERCICIO 5 - SALIDA POR PANTALLA

A partir de las declaraciones:

- $C = \text{"xyz"}$
- cadena letras
- entero $a = 12$, $b = 3$
- real k

al escribir las siguientes instrucciones:

1. $k \leftarrow a / (b * 8)$
2. $\text{letras} \leftarrow \text{"A"} + \text{"B"} + C$
3. $\text{escribir}(\text{"El valor de k es:"}, k)$
4. $\text{escribir}(\text{letras})$
5. $a \leftarrow b ** b \text{ div } a$
6. $\text{escribir}(a / k > k * a)$

¿qué se mostrará por pantalla?

EJERCICIO 6 - ÁREA DE UN TRIÁNGULO (ORDINOGRAMAS Y PSEUDOCÓDIGO)

Diseñe el algoritmo (ORDINOGRAMAS Y PSEUDOCÓDIGO) de un programa que:

1. Pida por teclado la base (dato real) de un triángulo.
2. Pida por teclado la altura (dato real) de un triángulo.
3. Calcule el área del triángulo.
4. Muestre por pantalla el resultado (dato real).

EJERCICIO 7 - CUBOS DE NÚMEROS PARES (REPETITIVA PARA - PSEUDOCÓDIGO)

Diseñe el algoritmo (en pseudocódigo) de un programa que muestre por pantalla los cinco primeros números naturales pares elevados al cubo.

Nota 1: Cubo de un número = número³

Nota 2: Utilice un bucle para.

EJERCICIO 8 - NÚMEROS MÚLTIPLOS DE 3 DEL -15 AL -3 (REPETITIVA PARA - PSEUDOCÓDIGO)

Diseñe el algoritmo (en pseudocódigo) de un programa que muestre por pantalla todos los números múltiplos de 3 que hay entre el -15 y el -3, ambos inclusive.

Nota: Utilice un bucle para.

Para mas ejercicios consulte: http://www.carlospes.com/ejercicios_de_algoritmos/