Herramienta para la descripción de algoritmos: PSEUDOCÓDIGO

Para describir algoritmos/programas existen varias técnicas : ordinogramas, diagramas de Chapin , pseudocódigo, nosotros usaremos esta última por ser, una técnica más compacta y porque la traducción de este a un lenguaje de programación (en nuestro caso el C) es mucho más rápida que con las otras técnicas de descripción de algoritmos.

Antes que nada , indicar que el pseudocódigo no es un programa para introducir en el ordenador, sino una técnica para indicar las acciones e ideas que debe realizar un programa para conseguir resolver la tarea encomendada, es decir en forma de etapas a la hora de realizar un programa la secuencia que se sigue es la siguiente

1. Planteamiento del problema (gestión, científico, de calculo etc.)
2. Diseño , ideas gráficos , desarrollo en texto
3. Descripción del algoritmo (PSEUDOCODIGO)
4. Codificación (paso de PSEUDOCODIGO a lenguaje concreto Java,Pascal C++ etc.)

Elementos usados en el pseudocódigo

- **Constantes** : para indicarlas usaremos la notación de números para cte. numéricas y el punto para decimales así como el signo - caso de usar números negativos

Ejemplo: -27.66 , 5212 .

- **Variables** : son los elementos de un programa que permiten almacenar valores bien mediante asignaciones, cálculos ó lecturas desde el exterior del programa pueden ser de diferente tipo según el contenido que posean, aunque esta clasificación es genérica para cualquier lenguaje, hay que estudiarse el manual concreto de cada uno para ver las precisiones y rangos de cada variable en general podemos clasificarlas como:

  - Numéricas (contienen datos numéricos)
    - Enteras datos sin decimales
    - Reales datos con decimales y como flotante

  - Alfanuméricas : contiene caracteres o tiras de caracteres, es decir pueden almacenar letras y frases compuestas por cualquier carácter accesible desde el teclado.

El nombre de las variables es OBLIGATORIO que empiece por letras pudiendo completarse su nombre con más letras, números y guiones bajos, el numero de caracteres que componen su nombre depende de cada compilador concreto pero suele ser suficientemente largo para marcar bien el nombre.
Es útil que los nombres de variables, hagan referencia a su contenido para guiarnos mejor sobre el uso de la misma.

- **Operadores**

- Aritméticos: son los usados para indicar operaciones de cálculo entre variables y/o constantes numéricas, estos son: **DIV, MOD, ^, / , *, +, -** , están colocados en orden de prioridad de mayor a menor

- Relacionales nos sirven para indicar en el algoritmo que queremos efectuar operaciones de comparación estos son :

- **>, <, >=, <=, =, <>**, son todos de igual prioridad salvo paréntesis

  Su uso es entre datos del mismo tipo es decir podemos comparar variables numéricas con numéricas pero no se puede texto con numéricas

- Lógicos: sirven para concatenar resultados de comparaciones mediante las tablas de verdad de la lógica. Existen tres: **Y, O, NO**

Y enlaza dos resultados de tipo verdadero/falso siendo el resultado verdadero si y solo si ambos son verdaderos

Ejemplo : `(z>=5) Y (c=5)` (se utilizan con acciones de bifurcación), el resultado de esta operación será verdadero si `z` es mayor o igual que 5 Y `c` es igual a 5 siendo falso si cualquiera de las dos o las dos fallasen, la tabla de verdad del Y es la siguiente:

Operando primero	Operando segundo	Resultado de op Y
V	V	V
V	F	F
F	V	F
F	F	F

O enlaza dos resultados verdad/falso siendo el resultado de la operación verdadero cuando uno de los dos o los dos lo son su tabla de verdad es la siguiente:

Operando primero	Operando segundo	Resultado de op O
V	V	V
V	F	V
F	V	V
F	F	F

NO, es un operador lógico unario no binario es decir no liga dos expresiones lógicas (V/F) sino solo una:
Su tabla se define cambiando el sentido del operador al que se aplica

Operando	Resultado de op NO
V	F
F	V

- Paréntesis (), los paréntesis pueden aplicarse a cualquier operación de cálculo ó comparación, e indican prioridad en la ejecución es decir al igual que en expresiones matemáticas todo los que se encierra en paréntesis se calculara primero. Ejemplo:

El resultado de `(5+8)*3` es `39`.

Pero sin paréntesis seria `29`, puesto que la prioridades del operador `*` está por delante de la suma.

➢ Acciones o instrucciones

Lectura: Indica la acción de tomar datos desde el exterior del programa, es decir introducidos por el usuario, necesario para que el programa empiece a trabajar, su sintaxis es:

Leer `v1,v2,v3...vn`

Es decir leemos las variable `v1` hasta `vn`, es decir se pueden leer varias variables en una sola acción, cuestión que se hace para compactar más el algoritmo.

Escritura: indica la acción de emitir valores o mensajes por el dispositivo estándar de salida, (generalmente la pantalla), la sintaxis es:

Imprimir lista de variables resultado y/o mensajes entre comillas

Ejemplo: Imprimir `“el resultado de la suma es:” suma`

Asignación de valor: sirve para cargar valores en variables bien mediante asignación simple:
Ejemplo: `sueldo =125000` o bien `nombre="PEPE"`
O también mediante cálculos:
Ejemplo:

```
Media =(a1+a2+a3)/3
```

En general la sintaxis será **`variable = variable/cte/expresión de calculo`**

Acciones compuestas.

Se denominan así porque su construcción engloba generalmente a otras instrucciones simples o compuestas, son las siguientes:

Toma de decisión simple:

Nos indica por donde debe "tirar" o bifurcarse el programa en función de un cierto valor Verdad/falso (comparaciones)

Su sintaxis es:

```
SI <CONDICION LOGICA (VERDADERO/FALSO)> ENTONCES
        S1
        S2
.
.
.
        SN
FIN_SI
```

Donde `S1...  Sn` son acciones englobadas en la toma de decisión que serán cualquiera de las admitidas en el pseudocodigo, el margen se DEBE indicar para mostrar cual es el campo de acción del `SI..FIN_SI`.
La ejecución es como sigue: se evalúa la condición lógica si es cierta se ejecuta el bloque de sentencia `S1..Sn` si no el programa lo evita y continúa hacia abajo a continuación del `FIN_SI`

Toma de decisión doble

```
SI <CONDICION LOGICA (VER/FALSO)> ENTONCES
        S1
        S2
.
.
.
        SN
SINO
        R1
        R2




        RN
FIN_SI
```

Siempre ejecuta uno de los dos bloques. Si la condición es verdadera se ejecuta el `S1..Sn`, si no, se ejecuta el `R1--Rn`.

Acciones de repetición

Bucle MIENTRAS---FIN_MIENTRAS

Su sintaxis es la siguiente

```
MIENTRAS <CONDICION LOGICA> HACER
        S1
        S2
        S3
        .
        .
        .
        SN
FIN_MIENTRAS
```

Su ejecución es la siguiente:

1. Se evalúa la condición lógica
2. Si es cierta se ejecuta el bloque `S1...Sn` y se vuelve a la cabecera a comprobar la condición nuevamente
3. Si es falsa, se salta el bloque y se continúa por el final del `FIN_MIENTRAS`.

Con esta repetición se hará el bucle, mientras la condición sea cierta.

Esta acción es muy potente, pues no se repite un cierto número de veces, sino que se hace en función de una condición que puede ser un contador de intentos del bloque ó no.
Obviamente lo que el programador debe considerar es que hay un momento en el que la condición se "rompa", es decir, no se verifique y podamos continuar con el flujo del programa, cuestión que si no ocurre produce un bucle "infinito" nada deseable en un programa que se quiera considerar eficiente.

Bucle REPETIR ---MIENTRAS<CONDICION>

Su sintaxis es:

```
REPETIR
        S1
        S2
        S3
        .
        .
        sn
MIENTRAS SEA CIERTO<CONDICION LÓGICA>
```

Es decir primero se ejecuta el bloque (al menos una vez)
Si la condición es CIERTA vuelve a ejecutarse y así sucesivamente hasta que sea FALSA.

La diferencia con el bucle MIENTRAS es que aquí el bloque al menos se ejecuta una vez, es decir, primero bloque después comprobación, y en el bucle anterior las cosas eran al revés.

Bucle PARA---FIN_PARA

Este bucle a diferencia de los anteriores es un bucle con contador, que es una variable del programa que además de gobernar el numero de repeticiones (que no tiene que ser entero) puede usarse como variable para operaciones dentro del bucle si se desea
Su sintaxis es:

```
PARA <VAR_CON>=VALOR_INI HASTA <VALOR FINAL> PASO <INC>
        S1
        S2
        S3
        .
        .
        SN
FIN_PARA
```

Su semántica es la siguiente: VAR-CON que es una variable numérica toma el valor inicial, mira el incremento y si el valor es mayor(menor) que el valor final acaba, y sigue después del FIN_PARA, si no se ejecuta el bloque S1---SN y se vuelve a la cabecera incrementando VAR-CON mas el INC indicado, volviendo a ejecutar el bloque si VAR-CON está entre los límites indicados.

Ejemplo:

```
PARA X=1 HASTA 5 PASO 1
...
...
..
FIN_PARA
```

X toma valor 1. Como el salto es de 1 en 1 y no hemos llegado a 5, entra en S1..SN , después X valdría 2, 3, 4, 5 después tomaría valor 6 y, como se sale del valor final y los pasos son positivos, el bucle acaba y se habrá ejecutado 5 veces.
Si jugamos con incrementos negativos podremos hacer bucle en reversa, y si se desea se puede hacer un bucle con valores y saltos fraccionarios, es decir, 0.5 1.75 etc.