Convenciones en Notación Algorítmica (Pseudocódigo)

1. Introducción

Estas convenciones de programación son reglas para hacer la documentación interna del programa.
Estas convenciones no son las más agradables a primera vista. Sin embargo, después de un tiempo de usarlas, nuestro cerebro se acostumbra a ellas, y las acepta. Al tiempo, programas que no siguen las convenciones se ven mal.

2. Estructura de un Algoritmo

```
Algoritmo <nombre>
Lexico
  <declaración de constantes, variables, tipos, acciones y funciones>
Inicio
  <concatenación de acciones>
Fin
```

Un algoritmo comienza siempre con la palabra **Algoritmo**, y se divide luego en dos partes, el **Léxico** y el **Cuerpo**. Cada palabra que identifica una parte del algoritmo debe estar subrayada.

3. Declaración Constantes, Variables, Tipos y Acciones

Todos sabemos que las constantes, variables, tipos, funciones y acciones usados en el algoritmo deben ser expresivos, claros, definitivamente relevantes a la solución del problema, y legibles. Sus nombres deben ser significativos o mnemónicos. Aunque es posible escribir sus nombres de cualquier forma, es conveniente utilizar alguna convención.

Ejemplo	Identificador definido, y su uso
cont	Declaración de Variable. Deben comenzar siempre con una letra minúscula. Este estilo es el que usaremos.
variableLarga	Declaración de Variable. Este estilo es el que usaremos cuando usamos más de una palabra.
variable_larga	Declaración de Variable. "_" separa las palabras del identificador. No lo usaremos.
Blanco	Declaración de Constante. Deben comenzar siempre con una letra mayúscula.
Tpersona	Declaración de un tipo. La declaración de tipos comienza siempre con la letra T.
Ppersona	Declaración de un puntero. La declaración de punteros comienza siempre con la letra P.
Sumar(a,b)	Declaración de una acción. Una acción comienza en mayúscula.
PotenciaDos(x)	Declaración de una función. Una función comienza en mayúscula.

Hay que destacar que cuando se usan varias palabras para formar un identificador, éstas deben ser muy legibles: **estoSiSeVeBien**, **peroestocasiquenoseentiende**. ¿O si?
Los únicos identificadores que comienzan con una minúscula son las variables; todos los demás identificadores que no pueden cambiar comienzan con una mayúscula: constantes, tipos, acciones, funciones, etc.
No se debe restringir el tamaño de un identificador, sino más bien debe escogerlo para

que sea significativo. Sin embargo, ante la posibilidad de escoger entre uno largo y otro corto, deberá escogerse el corto si tiene la misma expresividad del largo. Si queremos denotar la altura de una persona, es mejor usar el identificador **alt** que **alturaPersona**, siempre y cuando **alt** sea suficientemente claro en el contexto de programación. En general, cuando se trata de varias palabras, es mejor pegar las palabras es el estilo que se está imponiendo como estilo general cuando se trata de dos o tres palabras.
Es importante utilizar constantes para evitar el uso de números en las expresiones, por ejemplo: pago:= total * 1.1 * 0.25; pago:= total * ImpVentas * propina;
La expresión de la derecha es más clara y más simple de modificar.

En resumen: para declarar **variables** utilizaremos la notación **camelCase** y para **constantes**, **tipos**, **funciones** y **acciones** usaremos notación **PascalCase**.

4. Declaraciones en el léxico: orden y comentarios

En el léxico del algoritmo se declaran primero los identificadores que representan datos de entrada, luego los auxiliares y finalmente los que representan resultados.
Además, es conveniente aclarar qué es lo que almacena cada identificador, para un mejor entendimiento del algoritmo. Esto se hace mediante el uso de comentarios, los cuales se escriben en el mismo renglón que el identificador declarado y con // a la izquierda. También es conveniente escribir todos los comentarios a la misma altura, para mayor legibilidad. Eventualmente se puede colocar comentarios en una línea, antes de las acciones a que hacen referencia

Ejemplo: Calcular el promedio de dos notas.

```
Lexico
  nota1, nota2 ∈ R     //almacenan las notas a promediar
  sum ∈ R         //almacena la suma de las dos notas
  prom ∈ R        //almacena el resultado de dividir sum por 2
```

5. Indentación

La indentación es la posición en que comienza una línea de código en el renglón. Es un anglicismo de uso común en informática. Por indentación se entiende mover un bloque de texto hacia la derecha insertando espacios o tabuladores para separarlo del texto adyacente. La indentación se utiliza para mejorar la legibilidad del código, teniendo en cuenta que los compiladores o intérpretes raramente los consideran. La indentación es uno de los factores más importantes para lograr mejores algoritmos. Al indentar un algoritmo adecuadamente se logra destacar su estructura de control. Son frecuentes las discusiones sobre cómo o dónde usar la indentación, si es mejor usar espacios en blanco o tabuladores. Para unificar la forma de escribir los algoritmos usaremos un estilo en particular, aunque no es el único.
Debe indentarse **dos (2) espacios** cada vez que se utilice una construcción anidada. Siempre debe indentarse una construcción anidada, y siempre debe indentarse en **dos espacios**. La consistencia en la aplicación de esta regla permite a nuestro cerebro predecir la forma del algoritmo, y hacer la lectura más sencilla.
La **declaración** de variables, constantes, tipos, acciones y funciones en el léxico debe indentarse **dos (2) espacios**.
El **cuerpo principal** del algoritmo debe indentarse **dos (2) espacios**.

Ejemplo:

```
Algoritmo
Lexico
  a, b ∈ R //almacenan los datos a sumar, restar, multiplicar y dividir
  sum, res, mul, di ∈ R    //almacenan los resultados de cada operación
Inicio
  // a continuación ingresar dos números
  Entrada:a b
  sum <-- a+b
  res  <--  a–b
  mul  <--  a*b
  di  <--  a/b
  Salida: sum  res  mul di
Fin
```

5.1. Indentación del SI ENTONCES - SI ENTONCES SINO

La indentación de SI – ENTONCES que se utilizará en los algoritmos será:

```
si <condicion> entonces //cometario: condición del si
  <acciones>
fsi
```

La indentación de SI – ENTONCES – SINO que se utilizará en los algoritmos será:

```
si <condicion> entonces //cometario: condición del si
  <acciones1>
sino //cometario: complemento de la condición del si
  <acciones2>
fsi
```

5.2. Indentación del SEGUN – SEGUN OTROS

La indentación de SEGUN que se utilizará en los algoritmos será:

```
segun
  <condicion1>: <acciones1>
  <condicion2>: <acciones2>
      :
  <condicionn>: <accionesn>
fsegun
```

La indentación de SEGUN – OTROS que se utilizará en los algoritmos será:

```
segun
  <condicion1>: <acciones1>
  <condicion2>: <acciones2>
      :
  <condicionn>: <accionesn>
  otros:     <accionesm>

fsegun
```

5.3. Indentación del MIENTRAS

La indentación de MIENTRAS que se utilizará en los algoritmos será:

```
mientras <condición de continuación> hacer
  <acciones>
fmientras
```

5.4. Indentación del REPETIR

La indentación de REPETIR que se utilizará en los algoritmos será:

```
repetir
  <acciones>
hasta que <condición de terminación>
```

5.5. Indentación del PARA

La indentación de PARA que se utilizará en los algoritmos será:

```
para (valor inicial, condición de continuación, incremento)
  <acciones>
fpara
```

5.6. Indentación de Acciones

Acción <identificador> (<lista de parámetros>)
Lexico local
(si es necesario)
Inicio
{Pre-condición: ...**}**

<acción A1>

...

<acción An>
{Pos-condición: ...**}**
Facción

5.7. Indentación de Funciones

Función <identificador> (<lista de parámetros>) → <tipo de resultado>
Lexico local (si es necesario)
{Definición: ...**}**
Inicio

<accion A1>

...

<accion An>
Ffunción

6. Operadores

6.1. La asignación
Se anotará como sigue:

A <-- 5

A <-- A + 1

6.2 Operadores lógicos

La conjunción y
La disyucción o
La negación no
Ej: (a-1=q) y (no (esta o (a< p))

6.3 Operadores de comparación

Igualdad =
Distinto de <> o ≠
Mayor >
Mayor o igual >=
Menor <
Menor igual <=