# Notación algorítmica (seudocódigo en castellano) (fecha: 2018-10-18)

## 1. Tipos de datos predefinidos

```
booleano {con los valores verdad y falso; escribimos los comentarios entre llaves}
carácter
natural  {incluímos el 0 en los naturales}
entero
real
cadena   {son secuencias de caracteres de longitud arbitraria (vacías o no)}
```

## 2. Definición de constantes

```
constantes
  letraA = 'A'; maxNum = 100  {usamos el punto y coma para separar}
  hoy = "martes"  {es una cadena no vacía}
  pi = 3.1416
```

## 3. Definición de nuevos tipos de datos

```
tipos
  mes = (ene,feb,mar,abr,may,jun,jul,ago,sep,oct,nov,dic)  {tipo enumerado}
  mesVerano = jul..sep  {tipo subrango de otro tipo discreto}
  día = 1..31
  fecha = registro   {tipo registro (o estructura), agregación de campos}
            elDía: día;
            elMes: mes;
            elAño: natural
          freg
  pluviometría = vector[mes] de real    {tipo vector (array)}
  fiestas = vector[1..maxNum] de fecha
  secFechas = fichero de fecha  {tipo fichero binario de fechas}
  entrada = fichero de texto  {tipo fichero de texto, i.e., secuencia de líneas
                               (una línea es una secuencia de caracteres)}
```

## 4. Declaración de variables

```
variables
  contador: natural:= 0     {podemos inicializar la variable al declararla}
  éxito,error: booleano
  nombre: cadena
  cadenaVacía: cadena:= ""  {la variable se inicializa con la cadena vacía, ""}
  cumpleaños,aniversario: fecha
  festivos,patronos: fiestas
```

## 5. Instrucción de asignación

```
contador:= contador+1
error:= falso; éxito:= verdad;
éxito:= not error and (contador>3) or éxito
cumpleaños.elDía:= 13
nombre:= "Juan"
festivos[2].elMes:= sucesor(ene)  {operaciones sucesor y predecesor para enumerados}
cumpleaños:= aniversario {asignación entre registros: todos los campos se asignan}
festivos:= patronos {asignación entre vectores: todas las componentes se asignan}
```

## 6. Operaciones con datos cadena

```
variables apellido, resto: cadena
          i: natural
          letra: carácter
```

Comparaciones (por orden alfabético, considerando todos los caracteres según su orden en la tabla del código ASCII y sus extensiones):

```
"Costa" = apellido      apellido ≠ "Po3$"     apellido < resto
apellido ≤ resto        apellido > resto      apellido ≥ resto
```

Concatenación:

```
resto:= resto + apellido  {resto toma el valor resultante de concatenar su valor
                           previo con apellido}
apellido:= "de " + apellido  {se antepone a la cadena apellido la cadena "de "}
resto:= "hol" + "a"      {resto toma el valor "hola"}
```

Longitud:

```
i:= long(aux)  {long devuelve la longitud de la cadena, es decir, su número de
                caracteres; la longitud de la cadena vacía, "", es 0}
```

Carácter i-ésimo y subcadenas:

```
letra:= apellido[i]  {es el i-ésimo carácter de la cadena; 1 ≤ i ≤ long(apellido)}

letra:= apellido[1] {primer carácter de la cadena, que debe ser no vacía}
resto:= apellido[2..long(apellido)] {subcadena de apellido desde el carácter 2
                                     hasta el último}
resto:= apellido[i..j] {subcadena desde el carácter i al j; 1 ≤ i ≤ j ≤ long(apellido)}
```

## 7. Instrucciones de entrada/salida

```
escribir("Introduzca su edad: ")  {escribe el mensaje en el dispositivo estándar
                                   de salida (pantalla, por ejemplo)}
leerLínea(edad) {si edad es una variable entera, lee un valor de tipo entero desde
                 el dispositivo estándar de entrada (teclado, por ejemplo), lo asigna
                 a edad y luego lee la marca de fin de línea}
leerLínea(nombre) {si nombre es una variable de tipo cadena, lee todos los caracteres
                   anteriores al fin de línea desde el dispositivo estándar de entrada
                   (teclado), los asigna a nombre y luego lee la marca de fin de línea}

escribirLínea("La edad introducida es ", edad) {escribe el mensaje y el valor de edad
                                                y luego salta de línea}
escribirLínea("Su nombre es:", nombre) {escribe el mensaje entrecomillado, después el
                                        valor de la cadena nombre y luego salta de línea}
escribir("Introduce tres letras: ") {escribe el mensaje}
leer(a,b,c)  {si a,b,c son variables de tipo carácter, lee tres caracteres y los
              asigna a las variables a,b,c}
leerLínea {lee la marca de fin de línea}
```

## 8. Instrucciones condicionales

```
si <condición> entonces
  <secuencia de acciones>
fsi

si <condición> entonces
  <secuencia de acciones>
sino
  <secuencia de acciones>
fsi

si <condición> entonces
  <secuencia de acciones>
sino_si <condición> entonces
  <secuencia de acciones>
```

```
sino_si <condición> entonces
  <secuencia de acciones>
...
sino
  <secuencia de acciones>
fsi

selección
  <condición 1>: <secuencia de instrucciones>;
  <condición 2>: <secuencia de instrucciones>;
  ...
  <condición n>: <secuencia de instrucciones>;
  [ otrosCasos: <secuencia de instrucciones> ]
fselección
{se requiere que las condiciones sean excluyentes entre sí;
 una vez ejecutada la secuencia de instrucciones de la primera
 condición verdadera se termina la instrucción de selección}
```

## 9. Instrucciones iterativas

```
para <variable_de_tipo_discreto>:=<valor_inicial> hasta <valor_final> hacer
  <secuencia de acciones>
fpara

para <variable_de_tipo_discreto>:=<valor_inicial> descendiendo hasta <valor_final>
hacer
  <secuencia de acciones>
fpara

mientrasQue <condición> hacer
  <secuencia de acciones>
fmq

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

## 10. Procedimientos y funciones

```
procedimiento <nombre>(ent <parámetros_1>:<tipo_1>;
                       sal <parámetros_2>:<tipo_2>;
                       e/s <parámetros_3>:<tipo_3> ... )
{ent,sal,e/s significa parámetro de entrada, salida, entrada y salida, respec.}
<declaraciones locales de constantes, tipos, variables, proced., funciones...>
principio
  <secuencia de acciones>
fin
{un procedimiento es una acción o instrucción virtual;
 el programa principal es un procedimiento sin parámetros}

función <nombre>(<parám_1>:<tipo_1>; <parám_2>:<tipo_2> ...) devuelve <tipo_fun>
<declaraciones locales de constantes, tipos, variables, proced., funciones...>
principio
  <secuencia de acciones>
  devuelve <valor_de_tipo_fun>   {tras devolver el valor la función termina}
fin
{una función es un valor virtual, es decir, se evalúa dentro de una expresión
 y el resultado es un valor}
```

## 11. Módulos

```
módulo tablas
importa <lista de módulos que necesita usar>
exporta
  {parte pública: constantes, nombres de tipos, encabezamientos de proced. y func...}
```

```
  ...
implementación
  {parte privada: se incluyen las def. de los tipos cuyos nombres aparecen en la parte
   pública, otros tipos privados, el código de procedimientos y funciones...}
  ...
fin
```

## 12. Módulos genéricos

```
módulo genérico listasGenéricas
parámetros
  tipo elemento
  con función "<"(e1,e2:elemento) devuelve booleano
exporta
  tipo lista
  procedimiento crear(sal l: lista)
  procedimiento añadirÚltimo(e/s l: lista; ent e:elemento)
  ...
implementación
  ...
fin
```

## 13. Uso de módulos genéricos

### a. Para definir otro tipo también genérico:

```
módulo genérico pilasGenéricas
importa listasGenéricas
. . .
parámetro tipo elemento
exporta
  tipo pila
   . . .
implementación
  tipo pila = listasGenéricas.lista {el tipo exportado en listasGenéricas}

  procedimiento crear(sal p:pila)
  principio
    listasGenéricas.crear(p)
  fin
   . . .

fin {del módulo}
```

### b. Concretar un modulo genérico y utilizarlo:

```
. . .
importa pilasGenéricas;
modulo pila_naturales = pilasGenéricas(natural); {que ofrece el tipo: pila}
{admitimos también esta otra sintaxis aleternativa:}
modulo pila_naturales concreta pilasGenéricas(natural)
. . .
{para declarar variables o parámetros:}
p: pila;   {o bien: p:pila_naturales.pila}
{para utilizar sus operaciones:}
crear(p);  {o bien: pila_naturales.crear(p)}
. . .
```

## 14. Datos puntero

```
tipo pila = ↑unDato {tipo puntero (o referencia) a unDato}
     unDato = registro
                dato:natural;
                siguente:pila
              freg
variables p,q: pila
```

Valor especial (constante) de cualquier tipo puntero: `nil` (ninguna dirección)

```
p:=nil
```

Instrucciones con punteros:

```
nuevoDato(p);
p↑.dato:=3;
q:=p;
disponer(p);

si p=q entonces ...
```

**15. Instrucciones de creación y uso de ficheros**

A continuación se muestra un esquema básico del trabajo con ficheros de texto o binarios, junto con las instrucciones disponibles en pseudocódigo para trabajar con ellos.

- **Ficheros de texto**

```
variables
   f: fichero de texto
   d: carácter
   nombre: cadena
...
principio
   {Asociar la variable f con el fichero externo de nombre "Leeme.txt":}
   asociar(f, "Leeme.txt");
   {Inicializar el fichero para escritura: el fichero externo es creado vacío}
   iniciarEscritura(f);
   {Para escribir en un fichero de texto se considerarán también disponibles las
   mismas instrucciones que para escribir en pantalla o salida estándar, con el mismo
   significado y la misma sintaxis salvo porque se les añade un primer parámetro que
   es la variable fichero. Ejemplo:}
   escribir(f,d); {escribe en f el caracter d al final del fichero}
   escribir(f,nombre); {escribe en f la cadena nombre al final del fichero}
   . . .
   {Inicializar para lectura el fichero asociado con f, la posición de lectura para el
   fichero f se situa para que el primer dato que se lea sea el primero del fichero:}
   iniciarLectura(f);

   {Para saber si ya no quedan más datos que leer en el fichero (se ha leído ya el
   último dato), está disponible la función: finFichero(f) (devuelve booleano
   indicándolo)}
   mientrasQue not finFichero(f) hacer
      {Para leer de fichero de texto se considerarán también disponibles las mismas
      instrucciones que para leer de teclado o entrada estándar, con el mismo
      significado y la misma sintaxis salvo porque se les añade un primer parámetro
      que es la variable fichero. Ejemplo:}
      leer(f,d); {lee el siguiente dato del fichero f y lo deja en d (lee carácter), y
      deja disponible para ser leído el siguiente dato en el fichero}
         . . .
   fmq;
   {Eliminar la asociación entre f y el fichero externo:}
   disociar(f);
   ...
fin
```

- **Ficheros binarios**

Similares a los ficheros de texto, pero en ellos la unidad mínima de información para la lectura o escritura es un dato del tipo especificado en la creación de la variable fichero (y por supuesto no están estructurados en líneas).

```
variables
   ff: fichero de fecha  {tipo fichero binario de fechas. En el se leerán o escribirán
   datos de tipo fecha}
   dia: fecha
```

```
...
principio
   {asociar la variable ff con el fichero externo de nombre “misDatos.dat”:}
   asociar(ff, “misDatos.dat”);
   {Inicializar para escritura: el fichero externo es creado vacío y la posición de
   escritura en el fichero se mantendrá siempre al final del mismo}
   iniciarEscritura(ff);
   escribir(ff,dia); {escribe en ff el dato dia, siempre al final del fichero}
   . . .
   iniciarLectura(ff); {Inicializar para lectura el contenido del fichero asociado con
   ff, la posición de lectura para el fichero ff queda justo por delante del primer
   dato del fichero}

   {Para saber si ya no quedan más datos que leer en el fichero (se ha leído ya el
   último dato), está disponible la función: finFichero(ff) (devuelve booleano
   indicándolo)}
   mientrasQue not finFichero(ff) hacer
      {Para leer un dato de un fichero binario:}
      leer(ff,dia); {lee la siguiente fecha en el fichero ff y la deja en dia, y deja
      disponible para ser leído el siguiente dato del fichero}
   fmq;
   {eliminar la asociación entre ff y el fichero externo:}
   disociar(ff);
   ...

fin
```