PROGRAMACION SIMBOLICA

ALGOL-60

Carlos Domingo

Caracas, Marzo 1965

Departamento de Cálculo Numérico
U.C.V.

INDICE

1 - Introducción 1

2 - Ejemplo de programa en ALGOL-60 1

3 - Observaciones al programa en ALGOL-60 3

4 - Definición sintáctica con notación de Backus 4

5 - Símbolos básicos 5

6 - Identificadores 6

7 - Números o literales 6

8 - Variables - Declaraciones - Cadenas 7

9 - Estructura de bloques 12

10 - Expresiones aritméticas simples 14

11 - Expresiones lógicas simples 16

12 - Instrucciones condicionales 17

13 - Uso de expresiones condicionales en las expresiones aritméticas y booleanas 18

14 - Instrucción de salto o ramificación 19

15 - Expresión designativa 19

16 - Frases iterativas 19

17 - Variables propias 24

18 - Procedimientos 25

Ejemplo de procedimiento 26

Llamado al procedimiento 26

Especificación de los parámetros formales 28

Llamado por valor 30

Procedimientos recursivos 31

Definición sintáctica del ALGOL-60 32

PROGRAMACION EN ALGOL-60

1 - La diversidad de lenguajes en uso para describir algoritmos ha traído la necesidad de ponerse de acuerdo sobre un lenguaje común para comunicación entre programadores que utilizan diferentes computadoras y diferentes lenguajes simbólicos.

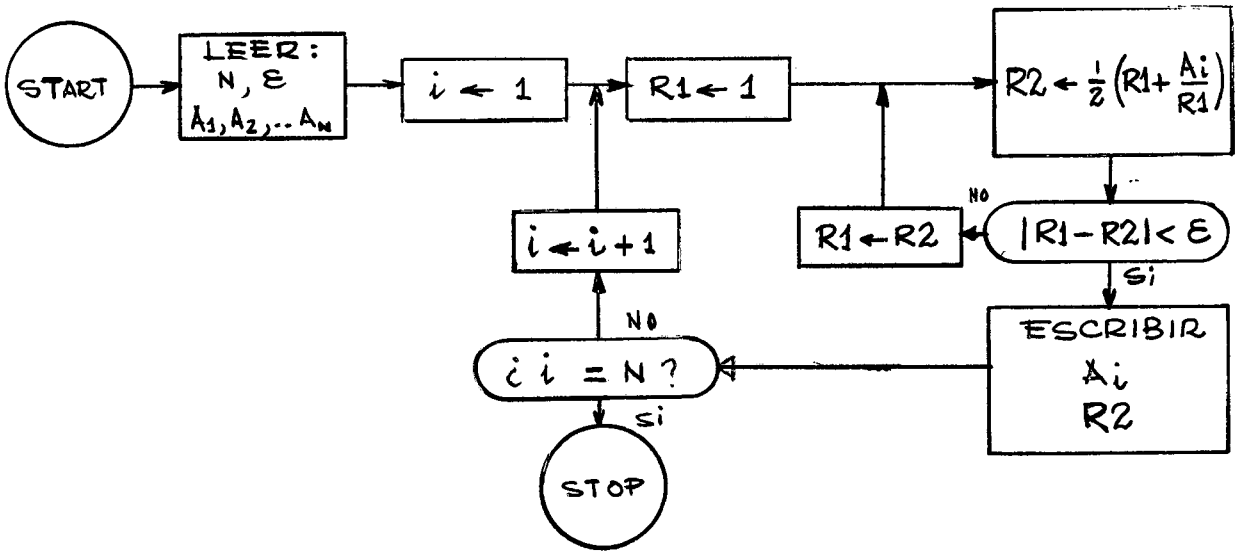
Una serie de trabajos del grupo europeo GAMM y la ACM culminaron en 1960 con la publicación del ALGOL-60, un lenguaje algorítmico que se está imponiendo como universal para la publicación de algoritmos.

Existen traductores de ALGOL-60 o lenguajes muy similares para diversas computadoras. La revista "Communications of the ACM" publica continuamente algoritmos escritos en este lenguaje y certificaciones de dichos algoritmos.

Por último el estudio de ALGOL-60 es útil como introducción a la programación simbólica en general, pues es un lenguaje algorítmico muy poderoso y completo.

2 - Ejemplo de programa en ALGOL-60. Antes de exponer al detalle las convenciones del lenguaje conviene relacionarse con él mediante un ejemplo. Sea el problema de extraer la raíz cuadrada de N números positivos, usando el algoritmo de Newton. Damos primeramente el diagrama de flujo que no requiere explicaciones adicionales y luego la versión del programa en lenguaje común y en ALGOL.

DIAGRAMA DE FLUJO



LENGUAJE COMUN

El arreglo A es de 100 valores reales; ε, R1, R2, son reales;
N es entero
Leer N, ε, A
Repetir las instrucciones siguientes con valores i=1,2,...N

1) Hacer R1=1.0
2) Hacer R2=½(R1+Ai/R1)
   y mientras sea R1-R2 < ε
   hacer R1=R2
3) perforar Ai, R2

LENGUAJE ALGOL

```
comment el siguiente programa calcula
la raíz cuadrada de N números;
begin real array A[1:100];
      real epsi,R1,R2;
      integer N;
      read (N,epsi,A);
      for i=1 step 1 until N do

          begin R2:=1.0;
                for R2:=0.5 *(R1+A[i]/R1)
                    while abs(R1-R2)<epsi do
                    R1:=R2;
                punch(A[i],R2)
          end;

end
```

3 - Observaciones:

1) Las instrucciones sucesivas del lenguaje están separadas por ; Cuando queremos referirnos a un grupo de instrucciones tratada como un solo bloque las ponemos entre los separadores begin end , formando así una instrucción compuesta que pueda contener una o más instrucciones, simples o compuestas.

2) Los comentarios son frases que pueden incluirse en cualquier parte del programa precediéndolas de la palabra comment. El comentario comienza inmediatamente después de este separador, y termina con el primer ; que se encuentre.

3) Las variables usadas deben ser "declaradas" previamente a su uso por "frases declarativas" que indican el modo real o entero (es decir con o sin decimales) y si es variable simple o si es un arreglo, en cuyo caso se especifica entre qué valores varía el o los subíndices.

4) Se han usado dos expresiones algo diferentes para describir los dos procesos iterativos. En una de ellas la repetición de las operaciones se controla por el valor del subíndice i. En la otra por la verdad o falsedad de una variable booleana (en este caso una relación).

5) read y punch son los nombres de los procedimientos que efectúan la lectura de datos y la perforación de resultados.
Sus propiedades sintácticas las veremos al estudiar procedimientos.

4 - Definición sintáctica con notación de Backus.

Además de la explicación del significado de los elementos del ALGOL daremos la definición de la sintaxis de cada elemento. Esto es muy útil en el aprendizaje pues permite definir sin ambigüedad cuando una expresión está bien formada.

Es también importante en la construcción de traductores automáticos pues estos determinan las expresiones y las clasifican apoyándose en estas definiciones.

En la notación de Backus los elementos incluídos entre paréntesis ⟨ ⟩ son nombres de símbolos (variables metalinguísticas). Así ⟨letra⟩ designa un elemento del conjunto de las letras.

Cuando hay varias de estas variables seguidas, esto indica siempre secuencia de los elementos. Así ⟨letra⟩ ⟨dígito⟩ indica una letra seguida de un dígito como por ejemplo A3, B5.

Todo signo no incluído entre ⟨ ⟩ se denota a sí mismo.

Así el + en: + ⟨dígito⟩ ⟨dígito⟩ define expresiones como +11 , +42 , +00 , etc.

El signo | designa el "o" exclusivo.

Damos como ejemplo la definición de número entero (positivo o negativo).

⟨entero⟩ :=⟨dígito⟩|⟨entero⟩⟨dígito⟩|+⟨entero⟩|-⟨entero⟩

⟨entero⟩ := se traduce por entero "es" o "significa". Se ve que hay 4 diferentes estructuras de enteros: damos ejemplos respectivos: 3 , 36 , +4000 , -20

Es claro que si tenemos como en este caso una sola frase definitoria, una al menos de las alternativas no debe contener el término definido.

Esta sirve para proceder a la construcción de las otras. Si 3 es un "entero" (por la primera definición) también lo es 36, que es un entero seguido de un dígito (por la segunda) y los son +36 y -36 (por la tercera y cuarta).

Previamente debe haberse definido dígito:

⟨dígito⟩ := 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

5 - Símbolos básicos. Son las letras, los dígitos, los valores lógicos y los delimitadores.

Las letras son las 26 del abecedario (no existe Ñ) mayúsculas y minúsculas.

Los dígitos son 0123456789

Los valores lógicos son true | false

Los delimitadores son:

Los operadores aritméticos: + - * / ÷ (división entera) y ↑ (exponenciación)

Los operadores lógicos
- ≡ equivalencia
- ⊃ implicación
- ∨ unión
- ∧ intersección
- ¬ negación

Los operadores de relación < | ≤ | = | ≥ | ≠ | >

Los operadores secuenciales goto if then else for do

Los separadores	; : ; := ␣ step until while comment
Los paréntesis	( ) [ ] $\langle$ $\rangle$ begin end
Los declaradores	own Boolean integer real array switch procedure
Los especificadores	string label value

6 - Los identificadores son conjuntos de letras y números (la primera debe ser una letra) que se usarán para designar las variables simples, los arreglos, los títulos o marcas de las frases, las llaves y los procedimientos (subrutinas). En un mismo programa o bloque un identificador dado se refiere siempre a la misma variable. Veremos luego como esto depende de la subordinación de bloques.

Ejemplos de identificadores: a3Ab , X , V17a , radio

Contra-ejemplos: 14 , 3a , 1AAZ , A3.b

7 - Números o literales. Los números tienen su significado convencional en aritmética. Se los suele llamar "literales" porque se asignan su propio valor en vez de ser nombres de variables como los identificadores.

Se los representa como en la aritmética común.

Los que contienen únicamente dígitos y los signos + - son enteros y como tales serán interpretados en el cálculo.

Los demás son reales. Se pueden escribir números reales con factor $10^{\pm N}$.

En estos casos el 10 se escribe como subíndice.

Ejemplos literales +0 -036.2 000 0.03 $452_{10}-11$ $+_{10}-573$ 3333

Contra-ejemplos 5,2

8 - Variables. Declaraciones. Cadenas.

Las variables pueden tomar valores numéricos o lógicos, o también conjuntos ordenados de estos valores.

En ALGOL hay varias clases de variables: las simples, los arreglos, los rótulos, las llaves y los procedimientos.

La clase de variable debe ser definida mediante una declaración como veremos en cada caso.

a. Las variables simples se designan por un identificador y pueden tomar un valor entero, real o lógico. Son, pues, de 3 tipos.

Siguen ejemplos de declaraciones de estos tipos.

integer m, entero, K31

real integral, ALFA1

Boolean SI, SINO, n

a.1 Las variables enteras pueden tomar valores enteros positivos y negativos. Las operaciones aritméticas entre ellas serán las operaciones comunes entre enteros y tendrán las propiedades formales conocidas, salvo en los casos de "overflow" y redondeo que causan pérdida de cifras significativas.

a.2 Las variables reales pueden tomar valores fraccionarios o más exactamente racionales con número finito de cifras significativas. Las operaciones entre ellas son las que en las computadoras se llaman usualmente operaciones con punto decimal flotante. Algunos autores las han llamado pseudo-operaciones para distinguirlas de las operaciones aritméticas comunes. No cumplen todas las leyes formales de las operaciones comunes lo cual debe ser tenido en cuenta para la estimación de errores.

Ejemplo: (3.2*0.11)*0.80=0.28 ; (3.2*0.8)*0.11=0.29 (2 cifras sig. con redondeo).

a.3 Las variables lógicas o booleanas pueden tomar los valores true y false.
El tipo declarado para una variable dada permanece constante en toda una sección del programa que llamaremos bloque. La estructura y subordinación de bloques se explicará más adelante.

b Los arreglos de cualquier orden son variables que pueden tomar un conjunto ordenado de valores. Se escriben con un nombre del arreglo que es un identificador seguido de una lista de subíndices entre signos.
Los subíndices son expresiones aritméticas cualesquiera.
Si son reales se entiende que se toma la parte entera.
Ejemplos:

MATRIZ [I, J]

VECTOR [A+B*COS (PI+X)]

INDIVIDUO [GENERO, ESPECIE, VARIEDAD]

DECISION LOGICA [M, N]

Obsérvese que los subíndices son siempre valores numéricos, mientras que los valores de la matriz pueden ser también variables booleanas.
Nótese que en la definición de subíndice se utiliza "expresión aritmética". Como la definición de "expresión aritmética" utilizará la definición de variable es necesario utilizar la definición recursiva. Véase Apéndice 1.
Las declaraciones de arreglo consisten en el nombre del arreglo y una lista de pares de límites.
Límite inferior:Límite superior
Los límites son expresiones aritméticas y los separadores de los pares de límites son comas.

Ejemplos:

array ALFA1[-1:X+Y,0:1] , VECTOR[Z:Z+26]

integer array INDIVIDUO[-3:+3,1:N,2:16]

boolean array DECISION LOGICA [1:10,-10:0]

Es claro que el valor del límite inferior debe ser siempre menor o igual que el del superior. Si algún límite es una expresión aritmética las variables de ésta deben haber sido definidas previamente a la declaración.

Los rótulos o marcas son designadores de instrucciones.

Debe recordarse que en el programa almacenado en una computadora no hay diferencia de naturaleza entre datos e instrucciones. Siempre nos referimos a ellos por el lugar de la memoria en que se encuentran. En el lenguaje simbólico se hace referencia a las variables (numéricas o booleanas) mediante identificadores. De modo análogo se hace referencia a las instrucciones por un identificador o un entero sin signo que constituye el rótulo o marca.

El rótulo se separa de la instrucción a la cual designa mediante el signo :

Ejemplo:

```
LAZO: a:=b+c;
      go to 36
  XX: IF m<n then go to LAZO;
```

LAZO, 36, XX son marcas o rótulos. Se separan de la instrucción que indican mediante un signo:

Los rótulos no tienen por qué ser declarados explícitamente en una cierta instrucción. Sin embargo, si en una sección del programa se hace referencia a un rótulo que figura en otro lugar muy alejado o que simplemente no figura en la sección del programa exhibido conviene declararlo como rótulo.

Ejemplo:

```
label  ERROR
.............
.............
if a[r,r]=0 then go to ERROR;
.............
```

Se supone que ERROR es un rótulo de una instrucción exterior a la sección de programa que se publica.

Hay expresiones para calcular rótulos (expresiones designativas) que veremos más adelante.

Las llaves son arreglos cuyos elementos son rótulos (o en general expresiones designativas). Los valores que toma una llave son siempre rótulos.

Se las declara usando el declarador switch

Ejemplo:

Supongamos que se declare:

```
switch LLAVEMULT:LAZO,36,XX,ERROR
```

LLAVEMULT es entonces un arreglo de 4 rótulos

Entonces

```
J:=2;
go to LLAVEMULT(j)
```

Indicará ramificar a la instrucción de rótulo 36.

```
go to LLAVEMULT(4)
```

es idéntica a

```
go to ERROR
```

Los procedimientos, que son las subrutinas o subprogramas, se verán en detalle más adelante. Basta decir que se designan mediante un identificador y que, en los casos en que el procedimiento define el valor de una función, dicho valor se asigna al identificador.

Se reservan ciertos nombres para las funciones más usadas. Siendo E una expresión aritmética estas funciones son

abs(E)	valor absoluto de E
sign(E)	signo de E (=-1 si E<0;+1 si E>0;0 si E=0)
sgrt(E)	raíz cuadrada positiva de E
sin(E)	seno de E
cos(E)	coseno de E
arctan(E)	valor principal de arco tangente de E
ln(E)	logaritmo natural de E
exp(E)	exponencial de E en base e
entier(E)	parte entera de E

Las cadenas son sucesiones de símbolos básicos separadas por símbolos ' '. El ALGOL no posee instrucciones especiales para manejar cadenas. Si se las quiere usar deben definirse los procedimientos correspondientes.

Las cadenas pueden contener cadenas.

Ejemplo:

'AbCde;Z'mn1'K3'bC'

9 - Estructura de bloques. Cuando no hay declaraciones, lo comprendido entre begin y end forma una instrucción compuesta.

Un bloque es una sección del programa comprendida entre los delimitadores especiales: begin end y que contiene, en su comienzo declaraciones de variables, que son válidas en ese bloque. Estas variables se llaman locales.

Ejemplo:

```
begin real X,Y,Z; integer t; a:Y:=X+Z ;
      X:=Y+Z*t ; Y:=Z ; t:=2 end
```

Las frases comprendidas entre begin y end forman un bloque. Las declaraciones valen dentro del bloque. Fuera de él X,Y,Z,t no están definidas o pueden estar definidas de otra manera mediante otras declaraciones.

Puede haber bloques dentro de otros bloques. Una variable definida en el bloque exterior puede usarse en el más interno conservando su valor.

Estas variables se llaman globales. Claro que si en el bloque interior se define alguna variable de igual nombre que una anteriormente definida en un bloque exterior, en el bloque interior pasará a valer la última definición que se hizo.

Ejemplo:

El programa siguiente:

```
begin  integer a,b,c,d;
       a:=2;
       b:=3;
       c:=4;
       begin real a,b;
             b:=d:=c;
             a:=2.8
             c:=5
             begin integer b,c;
                   c:=a ; b:=1;
                   print (a,b,c,d)      2.8  1  3  4
             end
             print (a,b,c,d)            2.8  4.0  5  4
       end;
       d:=c;
       print (a,b,c,d)                  2  3  4  4
end
```

imprime los valores:

2.8 1 3 4

2.8 4.0 5 4

2 3 4 4

Obsérvese que al transformar de real a entero hay truncación con redondeo.

La estructura de bloque es muy útil cuando intervienen varios programadores

en un solo programa. Cada uno usa las variables locales propias, sin preocuparse de los nombres de las variables locales de los demás.

Las variables globales que deben conservarse a través de los bloques se declaran en un bloque que abarca todos los bloques programados independientemente.

Los rótulos son locales al bloque en que se declaran (recuérdese que los rótulos se declaran simplemente al usarlos). No tienen validez fuera del bloque y por lo tanto es imposible hacer un salto desde afuera del bloque a una instrucción que está en su interior. También se entiende que pueden usarse rótulos iguales siempre que sea en bloques diferentes.

10 - Expresiones aritméticas simples. Son reglas para computar un valor numérico Son análogas a las expresiones del álgebra común. El signo ↑ se usa para indicar elevación a potencia. El signo ÷ indica división entera de números enteros.

Ejemplo:

Si tenemos

```
integer w,a,b; a:=3; b:=4; real z,k
z=a/b ; w:=a÷b ; k:=b↑a
```

resultaría

```
z=0.75  w=0  k=1.0
```

En ALGOL se considera que el orden en que se realizan las operaciones es el indicado por la expresión de acuerdo con los paréntesis y las reglas de precedencia.

Estas se expresan en tres niveles de precedencia

1) ↑

2) * / + -(unaria)

3) + -

En casos de igual precedencia se opera de izquierda a derecha.

Ejemplo:

La expresión

A/B-(M-K*(N+(COS(ALFA+V[1,U]))↑(m-n))+7.5$_{10}$2)*5

Se realiza de la siguiente forma:

1 A/B → T1

2 ALFA+V[1,U] → T2

3 COS(T2) → T2

4 m-n → T3

5 T2↑T3 → T2

6 N+T2 → T2

7 K*T2 → T2

8 M-T2 → T2

T2+7.5 10^2 → T2

T2*5 → T2

T1-T2 → T1

En T1 quedará el valor de la expresión.

La escritura de la expresión da al programador la posibilidad de controlar el orden de las operaciones, orden que influye sobre la acumulación de errores de redondeo.

Hay expresiones que no quedan definidas sin paréntesis.

Por ejemplo:

$2 \uparrow n \uparrow K$ significa $(2^n)^k$.

Debe ponerse

$2 \uparrow (n \uparrow K)$ para significar $2^{(n^k)}$

Una instrucción aritmética se forma con una variable y una expresión.

X:=a+b*c

Es posible asignar, en una misma frase, el valor de una expresión a varias variables

X:=Y:=Z:=a+b*c

11 - Expresiones lógicas simples. Son reglas para computar un valor lógico. Los elementos de estas expresiones son variables lógicas, valores lógicos y relaciones. Se escriben como las expresiones lógicas usuales.

Ejemplo: sean X, Y, Z variables lógicas.

a, b, c variables numéricas.

Son expresiones lógicas:

$X \wedge a \geqslant b \supset b < 5 \vee X \wedge Z$

$a \leqslant b \equiv X \wedge (Y \vee \neg Z)$

La precedencia en orden decreciente es:

1) $< \leqslant = \geqslant > \neq$
2) $\neg$ NO
3) $\wedge$ Y
4) $\vee$ Ø (EXCLUYENTE?) no; O INCLUSIVO
5) $\supset$ IMPLICA
6) $\equiv$ EQUIVALENCIA

Una instrucción lógica contiene una variable lógica a la que se asigna el valor de una expresión:

Ejemplo:

X := a<b≡X∧(Y∨¬Z)

X = a misma que b equivale a X y Y o no Z

12 - Instrucciones condicionales. Son las que expresan la realización de uno u otro proceso de acuerdo con el valor de verdad de una expresión lógica. Se construyen con los separadores if then else

Ejemplos:

```
; if  Y<5 then f:= a*y+b else f:= a*5+b;
; if B else Y:=1;
```

La interpretación de la primera instrucción es la siguiente:

Si Y< 5 entonces se realiza la instrucción entre then y else y luego se pasa a la instrucción siguiente (la que sigue al ;)

Si Y≥5 se ejecuta la instrucción entre else y el ; y luego se pasa a la instrucción siguiente.

La interpretación de la segunda instrucción es así:

Si la variable booleana B es verdadera se hace Y:=1 y se pasa a la instrucción siguiente.

Si B es falsa se pasa directamente a la instrucción siguiente.

¿Si?

Puede haber frases condicionales dentro de frases condicionales en cualquier número.

Ejemplo:

Sea m:=máximo(a,b,c)

El cálculo de m puede expresarse por

if a≥b then begin if a≥c then m:=a else m:=c end

else (if b≥c then m:=b else m:=c)

Aunque no es siempre necesario, en general es conveniente separar con paréntesis las instrucciones subordinadas. Obsérvese que la instrucción que sigue al then debe ser siempre no condicional o debe estar entre los paréntesis begin end es decir, debe ser una frase compuesta.

13 - Uso de expresiones condicionales en las expresiones aritméticas y booleanas.

Una expresión aritmética puede incluir expresiones condicionales.

Ejemplo:

Sea la instrucción aritmética

Z:=if t>12 then t↑2-1 else t↑2-t*r;

Z quedará igual a t^2-1 si t>12; en caso contrario quedará igual a t^2-tr

La segunda parte comprendida entre else y ; puede ser también una expresión condicional.

Pero no es correcto usar expresiones condicionales como operandos a menos que se las ponga entre paréntesis.

14 - Instrucción de salto o de ramificación. Expresa que la realización del programa deja de seguir la secuencia de instrucciones y salta a una instrucción cuyo rótulo se indica.

Ejemplo:

; go to hell ;

El control salta a la frase cuyo rótulo es hell. En vez de un rótulo puede ponerse una expresión designativa.

15 - Expresión designativa. Es una regla para calcular un rótulo. Las llaves, que ya hemos visto (8) son expresiones designativas. También pueden formarse con los elementos if, then, else

Ejemplo:

go to if SINO then SI else NO ;

Si la variable lógica SINO es verdadera la transferencia se hace a la frase de rótulo SI. En caso contrario la transferencia es a NO.

16 - Frases iterativas.

Se utilizan para repetir varias veces una o varias instrucciones de un programa variando alguno de los parámetros.

En la escritura de la frase se indica la variable que se cambiará en las repeticiones sucesivas (variable de control), la lista de los valores que tomará y la condición que da por terminada la repetición.

La forma es

for variable := lista de valores

Tenemos diversos tipos de lista:

a) la lista puede ser una simple sucesión de expresiones aritméticas.

Ejemplo:

```
for Z:=a,b+c,32,sin(t-2) do
    begin r:=Z+1 ;
          if r < m then go to ALFA;
          a:=a+1
    end ; I:
```

El conjunto de instrucciones entre begin y end se ejecuta sucesivamente con Z=a, Z=b+c, Z=sin(t-2). Luego se pasa a la instrucción I.

Como en este caso entre las instrucciones ejecutadas hay un salto condicional puede que la sucesión de ejecuciones quede interrumpida sin haberse agotado la lista.

b) La lista puede incluir elementos del tipo siguiente:

(donde expresión n son expresiones aritméticas)

expresión 1 step expresión 2 until expresión 3

Esta es una forma abreviada de indicar muchos valores de la variable de control.

Se interpreta así:

La primera vez la variable de control toma el valor de la expresión 1.

Cada vez siguiente que se repite el lazo la variable de control toma el valor que tenía en el ciclo anterior más el de la expresión 2.

La repetición se sigue mientras la suma que así se forma sea menor o igual que la expresión 3 si expresión 2 es positiva, o mientras la suma sea mayor o igual que la expresión 3 si expresión 2 es negativa.

Ejemplo: Una suma de n números a_i expresaría por

```
S:=0 ; for i=1 step 1 until n do S:=S+a[i];
```

La transposición de una matriz b[i,j] se puede hacer mediante la frase

```
for i:=1 step 1 until n do
    for j:=i+1 step 1 until n do
        begin T:=b[i,j] ; b[i, j] :=b[j,i] ;
            b[j,i]:=T
        end
```

En este ejemplo hay frases iterativas unas dentro de otras.

c) Por último un elemento de la lista puede ser de la forma:

expresión aritmética while expresión booleana do

La instrucción simple o compuesta que sigue a do se repetirá mientras la expresión booleana sea verdadera. Al principio de cada repetición se hará la variable de control igual a la expresión aritmética.

Ejemplo:

Hallar e^{-x} con una cota de error ε prefijada. El programa podría ser:

```
n:=1 ;
Z:=S:=1.0 ; for Z:=-x*Z/n while Z < epsilon do
                begin S:=S+Z ; n:=n+1 end
```

Hay que hacer notar que una lista puede tener elementos de las tres clases mezclados.

Todas las operaciones que se hacen usando las instrucciones iterativas for se pueden hacer mediante instrucciones condicionales.

La conveniencia de usar unas u otras depende de la comodidad de programación (en general es más cómodo usar instrucciones for) y de la longitud del programa traducido.

Damos a continuación el equivalente (usando instrucciones if) de instrucciones for con los dos últimos elementos de lista explicados.

Estos equivalentes aclaran el orden de las instrucciones de control del lazo iterativo y de ellos se deduce cuanto valdrá la variable de control al salir del for.

Sean A B y C expresiones aritméticas.

La instrucción

```
for Z:=A step B until C do  instrucciones
```

significa

```
Z:=A ;
SIGA:if(Z-C)*sign(B)< 0 then go to FIN ;
 instrucciones ;
Z:=Z+B ;
go to SIGA ;
FIN:
```

donde sign(B) es el signo del valor que toma la expresión B.

Si V es una expresión booleana

```
for Z:=A while V do <instrucciones>
```

significa

```
SIGA:Z:=A ;
     if ¬ V then go to FIN ;
     <instrucciones> ;
      go to SIGA ;
FIN:
```

Observamos que los valores de las variables que figuran en las expresiones de la instrucción for pueden variarse en el programa controlado por la instrucción for.

Cuando se usa un for conteniendo a otro for el rango de este último (parte del programa que se repite) debe estar totalmente contenido en el rango del primero.

Si dentro de un for se debe saltar la parte final del rango es necesario usar una instrucción muda como última instrucción del rango.

Ejemplo:

Sea el siguiente programa para ordenar los elementos del arreglo A[i] en orden creciente:

```
switch S:=L1,L2:
   L1:r:=2 ;
      for i:=1 step 1 until n-1 do
          begin if A[i] < A[i+1] then go to M ;
                   T:=A[i] ;
                   A[i]:=A[i+1] ;
                   A[i+1]:=T  ;
                   r:=1 ;
                M:
          end ;
      go to S(r) ;
   L2 :
```

M es el rótulo de una instrucción muda que es la última de las que se repiten en el lazo.

17 - Variables propias.

Hemos visto que al salir de un bloque las variables locales del mismo quedan no definidas. En el programa en lenguaje de máquina esto significa que sus lugares de memoria son utilizados para otros fines.

Si entramos nuevamente al bloque es necesario asignar nuevamente valores a dichas variables si se quiere utilizarlas.

Sin embargo existe en ALGOL un medio de conservar los valores de una variable local cuando se sale del bloque. Para ello se antepone a la declaración de variable el declarador own.

Ejemplo:

Sea el bloque que comienza

```
L:begin real a
        own real b,c
        own real array alfa[1:n]
        ..................
        a:=6.0 ; b:=5.0 ; c:=0.2
        ..................
  end
```

Si salimos del bloque es claro que a,b y c quedan indefinidas y no pueden utilizarse en otros bloques. Si re-entramos al bloque a seguirá indefinido, pero b y c estarán definidos con el valor de salida 5.0 y 0.2 respectivamente.

En cuanto a los arreglos vale la misma regla pero se presenta una dificultad.

Puede ser que un límite esté definido como variable global y sea susceptible de cambios exteriormente al bloque. Si por ejemplo al entrar al bloque es n=10 y salimos del bloque habiendo dado valores a todo

alfa[i] i=1,2,...10

Y si luego re-entramos al bloque con el valor n=12 los primeros 10 valores de alfa[i] readquirirán los valores de salida y los otros dos quedarán indefinidos. Si hubiéramos re-entrado con n=5 se perderían los 5 últimos valores; mientras que los 5 primeros adquirirían los valores que tenía

alfa[i] i=1,2,3,4,5 en la última salida.

18 - Procedimientos.

Los procedimientos del ALGOL desempeñan el papel de las subrutinas en la codificación.

Recordamos que éstas son secciones de programa a las que se puede entrar desde el programa principal ("llamada" de la subrutina) con determinados valores de ciertos parámetros. Cuando se han efectuado las instrucciones de la subrutina se vuelve al punto adecuado del programa principal. En algunos casos la realización de la subrutina puede consistir en la asignación de un valor a una variable simple (en este caso decimos que la subrutina es una función). En otros casos puede ser otro tipo de transformación.

En ALGOL todo procedimiento debe ser declarado previamente a su uso.

La declaración del procedimiento consta de un encabezamiento que en su forma más simple contiene la palabra procedure seguida del nombre y los parámetros formales que corresponden a variables que utiliza el procedimiento (y que vienen del programa principal), o a variables que definirá el procedimiento (y que irán al programa principal).

Sigue luego un bloque en el que están las instrucciones del procedimiento.

Ejemplo de procedimiento:

Sea el siguiente procedimiento de multiplicar una matriz a de f1 filas y m columnas por otra b de m filas y c2 columnas:

```
procedure Multmat(a,b,c,f1,m,c2) ;
    begin integer i,j,k ;
        for l:=1 step 1 until f1 do
            for j:=1 step 1 until c2 do
                begin c[i,j]:=0 ;
                    for k:=1 step 1 until m do
                    c[i,j]:=c[i,j]+a[i,k]*b[k,j]
                end
    end Multmat ;
```

En un procedimiento se pueden usar los parámetros formales, las variables locales definidas en el bloque del procedimiento y las variables globales definidas en el bloque que incluye el procedimiento.

Todas estas variables pueden cambiarse en la realización.

Llamado al procedimiento. En el programa principal se indica el llamado al procedimiento mediante una instrucción especial que consta de su nombre seguido de los valores actuales de los parámetros que pueden ser expresiones aritméticas.

Ejemplo:

Suponiendo definido el procedimiento Multmat del ejemplo anterior

```
real array  alfa[1:n,4:13] ,beta[1:10,2:5] ; d[1:n,1:1]
.....................
Z:=a+b ;
L:Multmat(alfa, beta, d, J+K, 9, a+1) ;
R:.............................
```

El significado del llamado es el siguiente:

Al llegar a la instrucción L se bifurca al procedimiento Multmat.

Los parámetros formales a,b,c,f1,m,c2 del procedimiento son sustituídos en todo el cuerpo del procedimiento por los parámetros actuales alfa, beta, d, J+K, 9, a+1 respectivamente en el orden indicado.

Si los parámetros actuales son expresiones (como J+K, a+1) se sustituyen en el procedimiento colocados entre paréntesis siempre que ello no produzca expresiones incorrectas.

Las variables locales del procedimiento se cambian de nombre si es que coinciden con alguna variable de los parámetros actuales (tal es el caso de J en nuestro ejemplo).

Hechos estos cambios se ejecuta el procedimiento.

Terminada la ejecución se vuelve a la instrucción siguiente a la llamada, R en nuestro caso.

Si el procedimiento es una función, es decir lo único que debe ser transmitido al programa principal es un valor que se asigna a una variable simple entonces puede utilizarse en el programa principal como un identificador. Su aparición implica una llamada como la que se explicó antes.

Ejemplo:

Sea el procedimiento para sumar términos de un arreglo

```
procedure SUM (VEC,m,n)
    begin integer L;
        SUM:=0.
        for L=m step 1 until n do
        SUM:=SUM+VEC(L)
    end
.....................
```

Una llamada en el programa principal podría ser

```
.....................
P:=r*sqrt(SUM(VEC,1,1+S)+Z) ;
.........
```

Es claro que estos casos en el cuerpo del procedimiento debe haber siempre una instrucción asignativa que tenga por variable el nombre del procedimiento.

Especificación de los parámetros formales.

En los ejemplos anteriores no se declara el tipo de los parámetros formales. Se supone en estos casos que los parámetros formales tienen el mismo tipo que los actuales de la llamada. Esto puede originar dificultades en la interpretación del procedimiento pues éste puede variar con el tipo de los parámetros en las diversas llamadas.

Ejemplo:

Sean B1,B2 variables booleanas y c,d reales en un procedimiento. Supongamos que se entra a un procedimiento con el parámetro actual X de tipo booleano. La instrucción del procedimiento

B1:=if B2 then X else c<d

Se interpreta

B1:=if B2 then X else (c<d)

Donde el paréntesis es una variable booleana.

Si se entrara al procedimiento con X real la interpretación sería:

B1:=(if B2 then X else c)<d

donde el paréntesis es una expresión aritmética.

Estas y otras dificultades se evitan especificando, a continuación del nombre y los parámetros formales el tipo de estos parámetros. La coincidencia de tipo entre los parámetros actuales y los formales se controla antes de realizar el procedimiento.

Los especificadores de parámetros formales tienen los mismos nombres que los declaradores aunque su función sea diferente. Son:

label

switch

string

real, integer, Boolean

procedure, real procedure, integer procedure, Boolean procedure

array, real array, integer array, Boolean array

Ejemplo:

El encabezamiento de un ejemplo anterior sería

```
procedure Multmat (a,b,c,fl,m,c2) ; real array a,b,c ;
integer fl,m,c2 ;
    begin.............
```

Llamado por valor. La sustitución de los parámetros formales por los actuales, tal como lo vimos, cuando estos últimos son expresiones aritméticas puede ser ineficiente y a veces imposible por conducir a expresiones inválidas.
En ALGOL puede hacerse que estas expresiones aritméticas sean calculadas de una vez por todas antes de entrar a ejecutar el procedimiento. Para ello se antepone a las especificaciones el nombre de la variable que queremos sea previamente calculado, precedido del especificador value.

Ejemplo:

En el ejemplo de la multiplicación de matrices sería deseable calcular primero los subíndices, ya que éstos pueden ser expresiones aritméticas. El encabezamiento del procedimiento sería:

```
procedure Multmat (a,b,c,fl,m,c2) ; value fl,m,c2 ;
  real array a,b,c ; integer fl,m,c2 ;
      begin integer i,j,k ; ............
```

Para la llamada Multmat (alfa,beta,d,J+K,9,a+1) el procedimiento equivaldría a un bloque en el cual fl,m,c2 fueran variables locales y que contuviera como subbloque al cuerpo del procedimiento.

El comienzo de aquél sería:

```
begin integer f1,m,c2 ;
      f1:=J+K ;
      m:=9 ;
      c2:=a+1 ;
      begin integer i,j,k.........
```

Ahora los parámetros f2,m,c2 no se cambian en el cuerpo del procedimiento.

Procedimientos recursivos. En ALGOL se permite el llamado de un procedimiento en el cuerpo del mismo procedimiento. Para aclarar la interpretación tenemos el ejemplo de un procedimiento que calcule el factorial de un entero n.
Esto puede hacerse como un procedimiento iterativo que para este caso es lo más económico:

```
integer procedure fac(n) ; value n ; integer n ;
        begin fac:=1 ; if n:=1 then fac:=1 else
              fac:=n*fac(n-1)
        end
```

Es claro que así el cálculo inplica n entradas al procedimiento. Si por ejemplo n=4 al llegar a 4*fac(4-1) la operación no puede realizarse y el 4 se guarda. Se entra al procedimiento con n=3 y al llegar a 3*fac(3-1) debe hacerse una nueva entrada con n=2; al llegar a 2*fac(2-1) se hace una nueva entrada con n=1, la cual da fac=1 y permite hacer la multiplicación de la entrada enterior, etc.

DEFINICION SINTACTICA DEL ALGOL-60

Letras

⟨letra⟩::=a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

Dígitos

⟨dígito⟩::=0|1|2|3|4|5|6|7|8|9

Valores lógicos

⟨valor lógico⟩::=true|false

Delimitadores

⟨delimitador⟩::=⟨operador⟩|⟨separador⟩|⟨paréntesis⟩|⟨declarador⟩|⟨especificador⟩

⟨operador⟩::=⟨operador aritmético⟩|⟨operador de relación⟩|⟨operador lógico⟩|
⟨operador de secuencia⟩

⟨operador aritmético⟩::=+|-|*|/|÷|↑

⟨operador de relación⟩::=<|≤|=|≥|>|≠

⟨operador lógico⟩::=≡|⊃|∨|∧|¬

⟨operador de secuencia⟩::=go to|if|then|else|for|do

⟨separador⟩::=,|.|$_{10}$|:|;|:=|␣|step|until|while|comment

⟨paréntesis⟩::=(|)|[|]|‘|’|begin|end

⟨declarador⟩::=own|Boolean|integer|real|array|switch|procedure

⟨especificador⟩::=string|label|value

NOTA: La secuencia de los símbolos básicos: equivale a

;comment⟨cualquier secuencia que no contiene ;⟩; ;

begin comment⟨cualquier secuencia que no contiene ;⟩; begin

end⟨cualquier secuencia que no contiene end o ; o else⟩ end

Identificadores

⟨identificador⟩::=⟨letra⟩|⟨identificador⟩⟨letra⟩|⟨identificador⟩⟨dígito⟩

Números

⟨entero sin signo⟩::=⟨dígito⟩|⟨entero sin signo⟩⟨dígito⟩

⟨entero⟩::=⟨entero sin signo⟩|+⟨entero sin signo⟩|-⟨entero sin signo⟩

⟨fracción decimal⟩::=.⟨entero sin signo⟩

⟨parte exponencial⟩::=$_{10}$⟨entero⟩

⟨número decimal⟩::=⟨entero sin signo⟩|⟨fracción decimal⟩|
⟨entero sin signo⟩⟨fracción decimal⟩

⟨número sin signo⟩::=⟨número decimal⟩|⟨parte exponencial⟩
⟨número decimal⟩⟨parte exponencial⟩

⟨número⟩::=⟨número sin signo⟩|+⟨número sin signo⟩|-⟨número sin signo⟩

Cadenas

⟨cadena propia⟩::=⟨cualquier secuencia de símbolos básicos que no contenga
'o'⟩|⟨vacío

⟨cadena abierta⟩::=⟨cadena propia⟩|'⟨cadena abierta⟩'|⟨cadena abierta⟩⟨cadena
abierta⟩

⟨cadena⟩::='⟨cadena abierta⟩'

Variables

⟨identificador de variable⟩::=⟨identificador⟩

⟨variable simple⟩::=⟨identificador de variable⟩

⟨expresión subíndice⟩::=⟨expresión aritmética⟩

⟨lista de subíndices⟩::=⟨expresión subíndice⟩|⟨lista de subíndices⟩,⟨expresión subíndice⟩

⟨identificador de arreglo⟩::=⟨identificador⟩

⟨variable con subíndice⟩::=⟨identificador de arreglo⟩[⟨lista de subíndices⟩]

⟨variable⟩::=⟨variable simple⟩|⟨variable con subíndice⟩

Designadores de función

⟨identificador de procedimiento⟩::=⟨identificador⟩

⟨parámetro actual⟩::=⟨cadena⟩|⟨expresión⟩|⟨identificador de arreglo⟩|⟨identificador de llave⟩|⟨identificador de procedimiento⟩

⟨cadena de letras⟩::=⟨letra⟩|⟨cadena de letras⟩⟨letra⟩

⟨delimitador de parámetros⟩::=,|)⟨cadena de letras⟩:(

⟨lista de parámetros actuales⟩::=⟨parámetro actual⟩|⟨lista de parámetros actuales⟩⟨delimitador de parámetro⟩⟨parámetro actual⟩

⟨especificación de parámetros actuales⟩::=⟨vacío⟩|(⟨lista de parámetros actuales⟩)|

⟨designador de función⟩::=⟨identificador de procedimiento⟩⟨especificación de parámetros actuales⟩

Expresiones aritméticas

⟨operador aditivo⟩::=+|-

⟨operador multiplicativo⟩::=*|/|÷

⟨primario⟩::=⟨número sin signo⟩|⟨variable⟩|⟨designador de función⟩

(⟨expresión aritmética⟩)

⟨factor⟩::=⟨primario⟩|⟨factor⟩|⟨primario⟩

⟨término⟩::=⟨factor⟩|⟨término⟩⟨operador multiplicativo⟩⟨factor⟩

⟨expresión aritmética simple⟩::=⟨término⟩⟨operador aditivo⟩

⟨término⟩|⟨expresión aritmética simple⟩⟨operador aditivo⟩⟨término⟩

⟨expresión condicional⟩::=if⟨expresión booleana⟩then

⟨expresión aritmética⟩::=⟨expresión aritmética simple⟩

⟨expresión condicional⟩⟨expresión aritmética simple⟩

else⟨expresión aritmética⟩

Expresiones booleanas

⟨operador de relación⟩::= < | ≤ | = | ≥ | > | ≠

⟨relación⟩::=⟨expresión aritmética⟩⟨operador de relación⟩⟨expresión aritmética⟩

⟨primario booleano⟩::=⟨valor lógico⟩|⟨variable⟩|⟨designador de función⟩|

⟨relación⟩|(⟨expresión booleana⟩)

⟨secundario booleano⟩::=⟨primario booleano⟩¬⟨primario booleano⟩

⟨factor booleano⟩::=⟨secundario booleano⟩|⟨factor booleano⟩∧⟨secundario booleano⟩

⟨término booleano⟩::=⟨factor booleano⟩|⟨término booleano⟩∨⟨factor booleano⟩

⟨implicación⟩::=⟨término booleano⟩|⟨implicación⟩⊃⟨término booleano⟩

⟨booleano simple⟩::=⟨implicación⟩|⟨booleano simple⟩≡⟨implicación⟩

⟨expresión booleana⟩::=⟨booleano simple⟩|

⟨expresión condicional⟩⟨booleano simple⟩else⟨expresión

booleana⟩

Expresión designativa

⟨rótulo⟩::=⟨identificador⟩|⟨entero sin signo⟩

⟨identificador de llave⟩::=⟨identificador⟩

⟨designador de llave⟩::=⟨identificador de llave⟩[⟨expresión subíndice⟩]

⟨expresión designativa simple⟩::=⟨rótulo⟩|⟨designador de llave⟩
(⟨expresión designativa⟩)

⟨expresión designativa⟩::=⟨expresión designativa simple⟩|
⟨expresión condicional⟩⟨expresión designativa simple⟩else⟨expresión designativa⟩

Expresiones compuestas y bloques

⟨instrucción básica sin rótulo⟩::=⟨instrucción asignativa⟩|⟨instrucción go to⟩|
⟨instrucción muda⟩|⟨instrucción de procedimiento⟩

⟨instrucción básica⟩::=⟨instrucción básica sin rótulo⟩|⟨rótulo⟩:⟨instrucción básica⟩

⟨instrucción no condicional⟩::=⟨instrucción básica⟩|⟨instrucción for⟩|
⟨instrucción compuesta⟩|⟨bloque⟩

⟨instrucción⟩::=⟨instrucción no condicional⟩|⟨instrucción condicional⟩

⟨final compuesto⟩::=⟨instrucción⟩end|⟨instrucción⟩;⟨final compuesto⟩

⟨encabezamiento de bloque⟩::=begin⟨declaración⟩|⟨encabezamiento de bloque⟩;
⟨declaración⟩

⟨compuesto sin rótulo⟩::=begin⟨final compuesto⟩

⟨bloque sin rótulo⟩::=⟨encabezamiento de bloque⟩;⟨final compuesto⟩

⟨instrucción compuesta⟩::=⟨compuesto sin rótulo⟩|⟨rótulo⟩:⟨instrucción compuesta⟩

⟨bloque⟩::=⟨bloque sin rótulo⟩|⟨rótulo⟩:⟨bloque⟩

Instrucciones asignativas

⟨parte izquierda⟩::=⟨variable⟩:=

⟨lista de parte izquierda⟩::=⟨parte izquierda⟩⟨parte izquierda⟩

⟨instrucción asignativa⟩::=⟨lista de parte izquierda⟩⟨expresión aritmética⟩|
⟨lista de parte izquierda⟩⟨expresión booleana⟩

Instrucciones condicionales

⟨expresión condicional⟩::=if⟨expresión booleana⟩then

⟨instrucción no condicional⟩::=⟨instrucción básica⟩|⟨instrucción for⟩|
⟨instrucción compuesta⟩|⟨bloque⟩

⟨instrucción if⟩::=⟨expresión condicional⟩⟨instrucción no condicional⟩|
⟨rótulo⟩:⟨instrucción if⟩

⟨instrucción condicional⟩::=⟨instrucción if⟩|⟨instrucción if⟩else⟨instrucción⟩

Instrucción for

⟨elemento de lista for⟩::=⟨expresión aritmética⟩
⟨expresión aritmética⟩step⟨expresión aritmética⟩until
⟨expresión aritmética⟩|
⟨expresión aritmética⟩while⟨expresión booleana⟩

⟨lista for⟩::=⟨elemento de lista for⟩|⟨lista for⟩,⟨elemento de lista for⟩

⟨expresión for⟩::=for⟨variable⟩:=⟨lista for⟩do

⟨instrucción for⟩::=⟨expresión for⟩⟨instrucción⟩
⟨rótulo⟩:⟨instrucción for⟩

Instrucción de procedimiento (llamado al procedimiento)

⟨parámetro actual⟩::=⟨cadena⟩|⟨expresión⟩|⟨identificador de arreglo⟩|
⟨identificador de llave⟩|⟨identificador de procedimiento⟩

⟨cadena de letras⟩::=⟨letra⟩|⟨cadena de letras⟩⟨letra⟩

⟨delimitador de parámetro⟩::=,|)⟨cadena de letras⟩:(

⟨lista de parámetros actuales⟩::=⟨parámetro actual⟩|
⟨lista de parámetros actuales⟩⟨delimitador de parámetros⟩
⟨parámetro actual⟩

⟨especificación de parámetros actuales⟩::=⟨vacío⟩|(⟨lista de parámetros actuales⟩)

⟨instrucción de procedimiento⟩::=⟨identificador de procedimiento⟩⟨especificación
de parámetros actuales⟩

Declaraciones

⟨declaración⟩::=⟨declaración de tipo⟩|⟨declaración de arreglo⟩
⟨declaración de llave⟩|⟨declaración de procedimiento⟩

Declaraciones de tipos

⟨lista de tipos⟩::=⟨variable simple⟩|⟨variable simple⟩,⟨lista de tipos⟩

⟨tipo⟩::=real|integer|Boolean

⟨tipo local o own⟩::=⟨tipo⟩|own⟨tipo⟩

⟨declaración de tipos⟩::=⟨tipo local o own⟩⟨lista de tipos⟩

Declaraciones de arreglo

⟨límite inferior⟩::=⟨expresión aritmética⟩

⟨límite superior⟩::=⟨expresión aritmética⟩

⟨par de límites⟩::=⟨límite inferior⟩:⟨límite superior⟩

⟨lista de pares de límites⟩::=⟨par de límites⟩|⟨lista de pares de límites⟩,
⟨par de límites⟩

⟨segmento de arreglo⟩::=⟨identificador de arreglo⟩[⟨lista de pares de límites⟩]|
⟨identificador de arreglo⟩,⟨segmento de arreglo⟩

⟨lista de arreglos⟩::=⟨segmento de arreglo⟩|⟨lista de arreglos⟩,⟨segmento de arreglo⟩

⟨declaración de arreglo⟩::=array⟨lista de arreglo⟩|
⟨tipo local o own⟩array⟨lista de arreglos⟩

Declaraciones de llave

⟨lista de llaves⟩::=⟨expresión designativa⟩|
⟨lista de llaves⟩,⟨expresión designativa⟩

⟨declaración de llave⟩::=switch⟨identificador de llaves⟩:=⟨lista de llaves⟩

Declaraciones de procedimiento

⟨parámetro formal⟩::=⟨identificador⟩

⟨lista de parámetros formales⟩::=⟨parámetro formal⟩
⟨lista de parámetros formales⟩⟨delimitador de parámetros⟩⟨parámetro formal⟩

⟨especificación de parámetros formales⟩::=⟨vacío⟩|(⟨lista de parámetros formales⟩)

⟨lista de identificadores⟩::=⟨identificador⟩|⟨lista de identificadores⟩,
⟨identificador⟩

⟨especificación de valores⟩::=value⟨lista de identificadores⟩;|⟨vacío⟩

⟨especificador⟩::=string|⟨tipo⟩|array|⟨tipo⟩array|label|switch|
procedure|tipo⟨procedure⟩

⟨declaración de especificaciones⟩::=⟨vacío⟩|⟨especificador⟩⟨lista de identificadores⟩;|⟨declaración de especificaciones⟩⟨especificador⟩⟨lista de identificadores⟩;

⟨encabezamiento⟩ ::= ⟨identificador de procedimiento⟩⟨lista de parámetros formales⟩;⟨especificación de valores⟩⟨declaración de especificaciones⟩

⟨cuerpo de procedimiento⟩ ::= ⟨instrucción⟩ | ⟨código⟩

⟨declaración de procedimiento⟩ ::= procedure ⟨encabezamiento⟩⟨cuerpo de procedimiento⟩ | ⟨tipo⟩ procedure ⟨encabezamiento⟩⟨cuerpo de procedimiento⟩

EJERCICIOS

1 - Dar 4 razones que justifiquen el estudio del lenguaje algorítmico ALGOL.

2.3 - Observar el ejemplo del párrafo 2.

- Las instrucciones sucesivas se separan por ;
- Un grupo de instrucciones puede encerrarse entre los paréntesis begin end
- Los comentarios van precedidos por el separador comment
- Toda variable que se utiliza debe ser declarada previamente.
- En la declaración debe indicarse el modo y si la variable es simple o un arreglo .
- Los subíndices se indican entre corchetes .
- Después de la expresión

  for i=1 step 1 until N do

  hay una instrucción declarativa .

4.5.6.

- La definición de identificador en notación de Backus es

< letra >|<identificador>< letra >|<identificador>dígito >

- Con la definición anterior probar que

AB4C7 es un identificador:

A es una letra, luego es un identificador

AB es un identificador seguido de una letra luego es un ident.

AB4 es un identificador seguido de un dígito luego es un identificador

AB4C " " " " una letra " " "

AB4C7 " " " " un dígito " " "

- Suponiendo definido término de una expresión aritmética la definición de expresión aritmética es:

<término>|-< >|+< >|< >-< >
|< >+< >

7 - Qué literales son incorrectos

1	3.10↑4	127.	$3_{10}-4$	6
-4.8	6,2832	127.0	3.0	$_{10}-43$	0.

8 - En las instrucciones siguientes se dan 18 ejemplos de variables. Se supone que no hay cambios de modo ni truncaciones al ejecutar las instrucciones.

```
S:r:=2 ; m:=2.07 ;
UCV[im,r,x,q]=1 ;
go to S ;
L1:a:=false ; go to Z[j] ;
L2[K]:=a ;
real array P[1:m1 , Kj:t+4]
```

Hacer las declaraciones correspondientes a las variables anteriores.

```
LABEL S, L1, Z
INTEGER ARRAY UCV[im, r, x, q]
BOOLEAN a, L2
REAL ARRAY P[1:m1, Kj:t+4]
```

- Declarar las siguientes variables:

M es un arreglo de orden 2 cuyos suíndices varían entre -10 y +K y entre -I y K+1 respectivamente. (K y I se suponen definidos previamente).

S es una llave que puede valer los rótulos L1, L2, L3.

Z1 es una variable lógica.

R0 es real y T2 entera.

```
ARRAY M[-10:K, -I:K+1]
SWITCH S:=L1, L2, L3
BOOLEAN Z1
REAL R0
INTEGER T2
```

9 - Qué diferencia hay entre instrucción compuesta y bloque?

En el bloque hay instrucciones declarativas

- Escribir los valores actuales de las variables en las etapas del programa indicadas a la derecha.

Obsérvese el ejemplo. IND indica que la variable no está definida. Poner el punto en los valores reales.

```
begin integer x,y,z ; real t ;
      x:=5.6 ; y:=4 ; t:=y ;                 | x=6 y=4 z=IND t=4.
      z:=2 ;
      begin real x,a ; integer z ;
            L:x:=y ; a:=1.0 ; t:=0           | x=4.0   z=2
      end ;                                  | a=1.0   y=4   z=2
      x:=x+1 ; t:=t+1 ; go to L ;            | x=5     t=1.0 L=IND
end                                          | x=IND   y=4   t=0.0
```

10 - Escribir en ALGOL las expresiones siguientes:

$\frac{a+b^2}{c+d}$ (a+b↑2)/(c+d)

$\cos^2 \cos\ 3.5^m$ cos↑2 cos 3.5↑m

$\frac{a}{c}.d$ a/c*d

$\frac{a-b}{a} - b$ (a-b)/a-b

$\frac{a}{c.d}$ a/(c*d)

parte entera de $\frac{z}{a-z}$ z÷(a-z)

- Supongamos variables con los valores siguientes:

s=5.88 $y=132_{10}-4$ z=0.009

a=1 b=2 c=3

d[1]=4 d[2]=3 d[3]=9 d[4]=3

real S ; integer m,n

Calcular el valor con que quedan las variables que aparecen en las siguientes instrucciones.

Usar 4 cifras significativas en las operaciones con números reales. En cada operación se trunca el resultado a tres cifras significativas sin redondear.

m:=(a÷c)+(a/c)+1	entero	m= 1
S:=(a÷c)+(a/c)	real	S= 0.333
n:=d[d[b]]	entero	n= 9
d[m] :=m:=4		m= 4 d[4] = 4
S:=x+y+z		S= 5.902
S:=z+y+x		S= 5.902
S:=x*y*z		S= 0.069
S:=z*y*x		S= 0.000
S:=x/y/z		S=
S:=x/(y*z)		S=

588
132,5-4
0.009

- random(X) es una subrutina que calcula un número aleatorio entre 0 y X

¿Son equivalentes las expresiones siguientes?

random(z)*random(z)

random(z)↑2

No. En el primer caso la subrutina calcula dos números aleatorios entre 0 y X, en el segundo uno

11 - Suponiendo p=true q=false hallar los valores de las siguientes expresiones lógicas:

p true

$\neg q$ true

$p \wedge q$ false

$p \vee \neg q$ true

$p \vee q$ true

$\neg p \vee q$ false

$p \equiv q$ false

$\neg (p \vee q) \equiv \neg p \vee \neg q$ _____

$\neg (p \wedge q) \equiv \neg p \vee \neg q$ _____

$p \equiv q \wedge p$ _____

$p \supset q \equiv \neg p \wedge q$ _____

$p \equiv p$ _____

$p \supset q \equiv p \wedge q$ _____

$p \equiv \neg q$ _____

$p \supset q$ _____

$q \supset p$ _____

$\neg p \vee q \supset 3=3$ _____

false $\supset p$ _____

$\neg q \supset \neg$ false _____

$p \vee (q \wedge p)$ _____

Marcar con I las equivalencias que son identidades.

- q es verdadero si lo es p y además x no está entre 0 y 25 o si, aunque p no sea verdadero, x es igual a 10. En los otros casos q es falso. Escribir la expresión de q

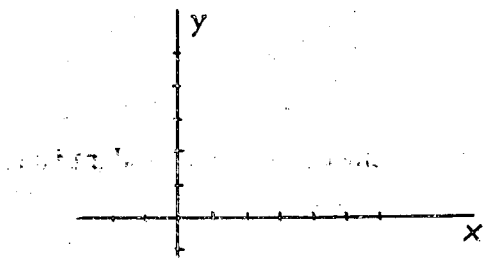
q:=

- Se desean sumar por lo menos L términos A_i de una serie, pero, pasados los L términos hay que suspender la suma si se han sumado más de U términos, o si la suma excede el valor M.
  Escribir una expresión lógica que sea verdadera mientras haya que proseguir la suma de la serie y falsa en caso contrario.

- Cuando x es mayor que 50, y es positivo.
  Escribir una expresión lógica que sea falsa sólamente cuando la condición anterior no se cumple.

- Indicar la región del plano cartesiano para la cual es verdadero

$x < 3 \equiv y < 2$



12.13.14.15.

- Completar el siguiente programa para hallar el mayor de n números a[i].

Se compara el primero con el segundo. El mayor de ellos con el tercero, etc.

n se supone previamente definido.

```
begin real array a[1:n] ; integer i,j ;
      j:=1 ;
   L1:i:=j ;
   L2:j:=j+1 ;
     if j⩽n then begin if          then go to
                       else        ;
     print(a[i])
end
```

- abc son variables booleanas, r s t son reales, beta es un arreglo. Decir qué expresiones son correctas.

```
r:=if r⩽s then r else t                    _____
r:=r+if s⩽t then s else t                  _____
a:=if s=t then a else s<t                  _____
t:=beta[if a then r else s,3]              _____
r:=s+(if s⩾t then s else t)                _____
```

- Modificar el programa de hallar el mayor de n números usando expresiones designativas.

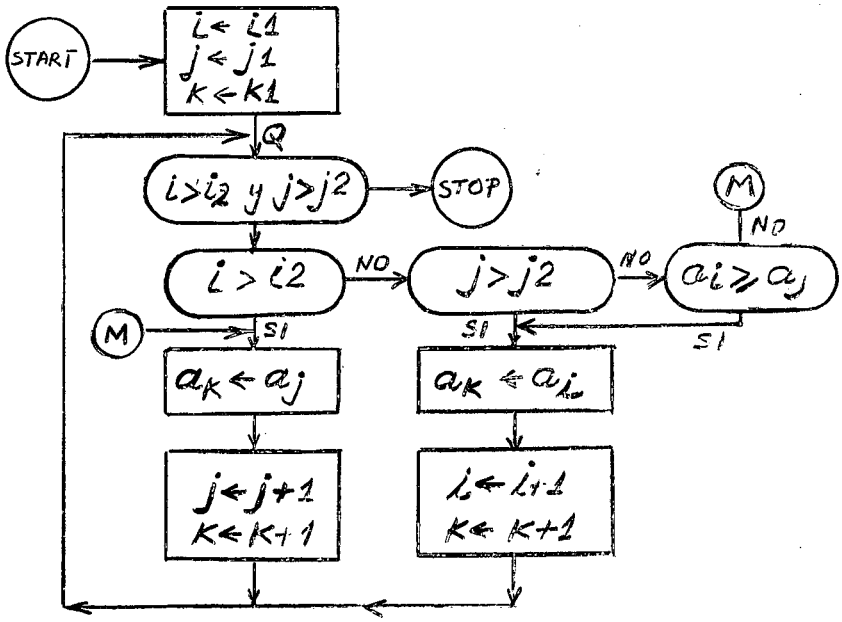
- Programa de mezcla de sucesiones.

Se tiene un arreglo a[1:f].

Suponemos que 1) los valores de los elementos a[i1] ,a[i1+1] ,...
...a[i] ,...a[i2] están en orden creciente.

2) los valores de los elementos a[j1] ,a[j1+1] ,
,...a[j] ,...a[j2] están en orden creciente.
Es j1 > i2

Hacer un programa que mezcle estas dos sucesiones en una única sucesión creciente que comience en a[k1]...(k1 > j2)



16 - Hacer un programa para calcular la transpuesta de una matriz a[i,j]

- Hacer un programa para tabular las funciones de Bessel de órdenes M, R, R+1 (enteros positivos) entre valores xI, xI+ x, ... xF con error menor que ε

La función de Bessel de orden N se define por

$$J_N(x)=\sum_{S=0}^{\infty}\frac{(-1)^N}{S!\,(N+S)}\left(\frac{x}{2}\right)^{N+2S}$$

Obsérvese que se puede partir del término correspondiente a S=0

$$\frac{1}{N!}\left(\frac{x}{2}\right)^N$$

y para hallar cada término debe multiplicarse el anterior por

$$\frac{-1}{(S+1)\,(S+1+N)}\left(\frac{x}{2}\right)^2$$

- Véase que el error es menor que el último término despreciado.
- Si es x=0, N=0 no hay que calcular. Debe ponerse $J_0(0)=1$

- Una aproximación para arctan(x) dada por fracción continua es :

$$\arctan(x)=x\ \cfrac{x^2}{k0+\cfrac{x^2}{k1+\cfrac{x^2}{k2+\cfrac{x^2}{k3+\cfrac{x^2}{k4}}}}}$$

donde k0=0.99999752 k1=-3.00064286

k2=-0.55703890 k3=-17.03715998

k4=-0.20556880

- Completar el siguiente programa para calcular arctan(x)

```
arctan:=-0.20556880 ;
for k:=-17.03715998,
    do  arctan=                    ;
arctan:=
```

- Hacer un programa para resolver por el método de iteración de Gauss-Seidel el sistema $\sum_{j=1}^{n} a_{ij}x_j=b_i \quad (i=1,2,..n)$

En este método se ponen como valores iniciales

$$x_i=b_i/a_{ii}$$

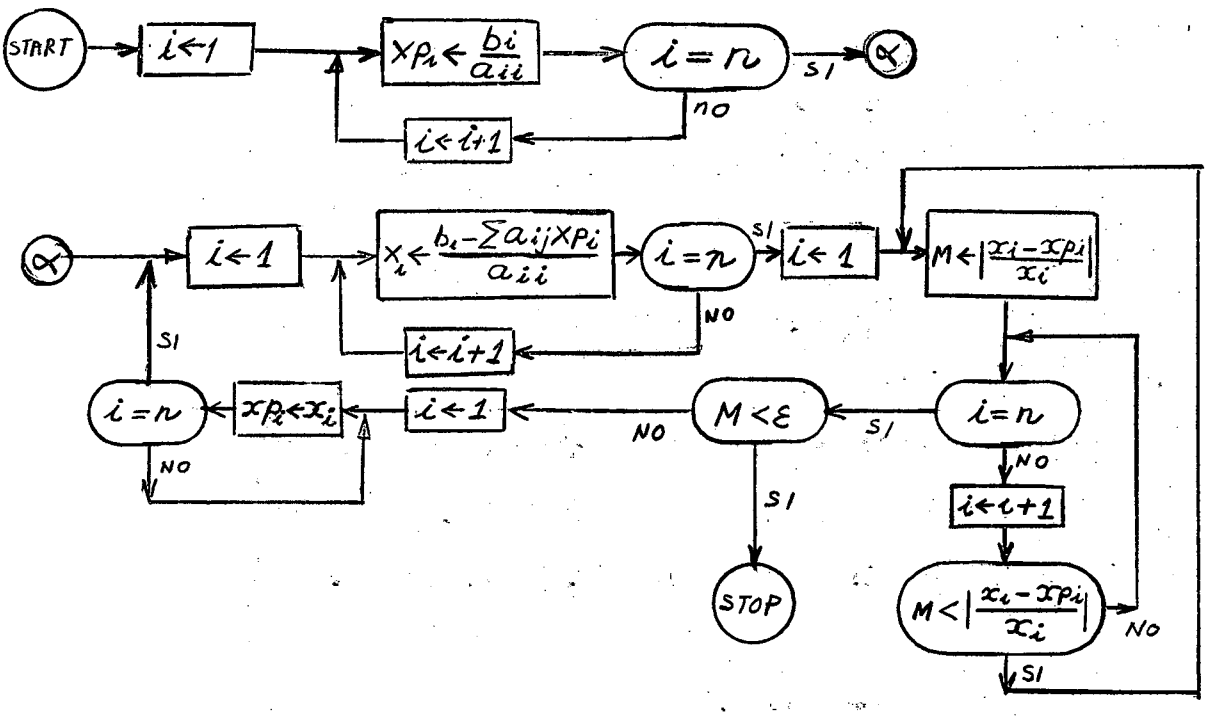
y luego se corrigen por iteraciones sucesivas

$$x_i \leftarrow \frac{b_i-\sum a_{ij}x_j}{a_{ii}}$$

donde la sumatoria se hace de 1 a n con exclusión de $a_{ii}x_i$. El procedimiento se puede detener cuando, después de una iteración completa, es

$$M=\max\left|\frac{x^{(k)}-x^{(k-1)}}{x^{(k)}}\right|$$

donde el índice k se refiere a la iteración.


START
i←1
xp_i ← b_i / a_ii
i = n
SI
NO
i←i+1
∝
i←1
x_i ← (b_i − Σ a_ij x_pi) / a_ii
i = n
SI
NO
i←i+1
i←1
M ← |(x_i − x_pi) / x_i|
i = n
SI
NO
i←i+1
M < |(x_i − x_pi) / x_i|
NO
SI
M < ε
SI
NO
i←1
xp_i ← x_i
i = n
NO
SI
STOP

- Un método de generar números pseudo-aleatorios distribuídos con igual probabilidad en un intervalo A,B consiste en lo siguiente:

a) La primera vez que se desea generar un número se hace:

$$R=\text{número de 11 cifras} < 2^{35}=34359738368$$

Las veces siguientes se debe proceder así:

b) Conservando el R que quedó en el último uso de la subrutina hacer

c) $R \leftarrow 5R$ y ejecutar sucesivamente las instrucciones:

d) Si $R \geqslant 137438953472$ restar a R dicho número

e) Si $R \geqslant 68719476736$ restar a R dicho número

f) Si $R \geqslant 34359738368$ restar a R dicho número

g) $NA=R/3459738368x(B-A)$

Escribir la subrutina NA(A,B)

- Se desea calcular la integral $\int_{x1}^{x2}\int_{y1}^{y2} F(x,y)\,dxdy$

por el método de Montecarlo. La función F se mantiene entre 0 y M.

El método consiste en elegir dos coordenadas x,y al azar dentro del recinto de integración y calcular F(x,y)

Luego se elige un número z al azar entre 0 y M.

Si es $F(x,y) \leq z$ se considera que la elección ha tenido éxito.

Después de N elecciones exitosas una estimación de la integral es

Integral ← N/número total de elecciones.

- El procedimiento clásico de Von Neumann y Golstine para ordenar una serie de números consiste en lo siguiente:

  a) ordenar secuencias de largo 2 (pares sucesivos)

  b) mezclar las secuencias ordenadas obtenidas obteniendo secuencias de largo doble

  c) repetir el procedimiento b) hasta que quede una sola secuencia.

Ejemplo:

1 4 8 2 9 7 6 5 7 2 1 3 4

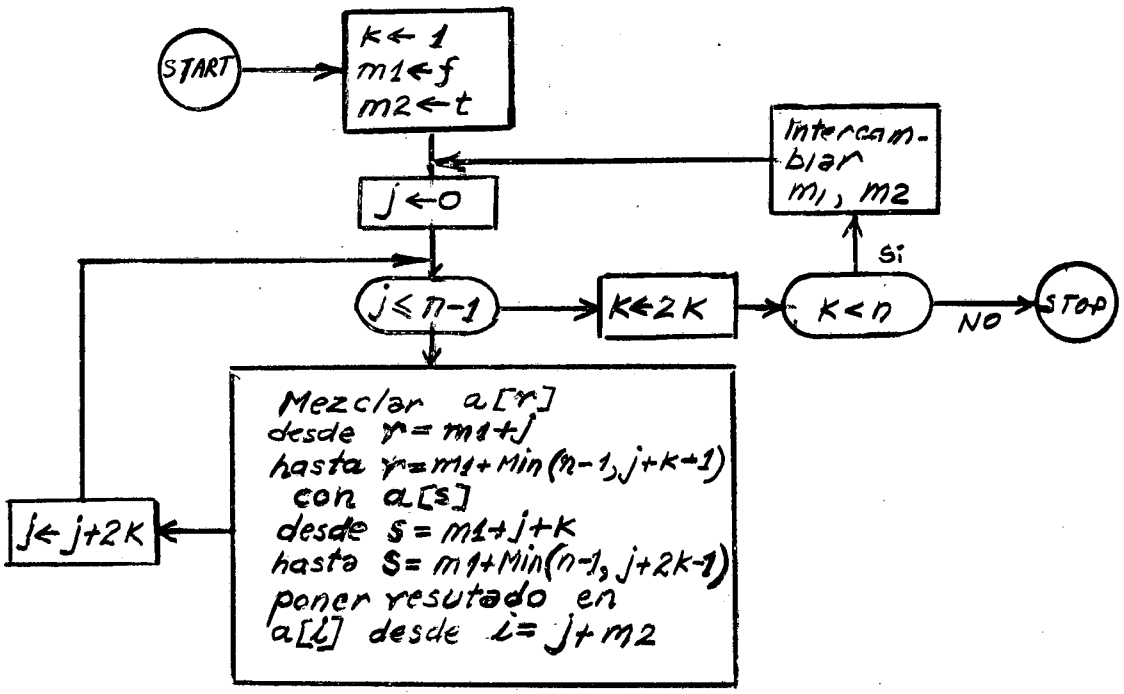
1 4 28 79 56 27 13 4

1 2 4 8 5 6 7 9 1 2 3 7 4

1 2 4 5 6 7 8 9 1 2 3 4 7

1 1 2 2 3 4 4 5 6 7 7 8 9

- Usando como procedimiento la subrutina mezcla programada anteriormente hacer un programa que realice el procedimiento de clasificación descrito. Se supone que el arreglo original está en $a[f], a[f+1] \ldots$. Al final puede estar en las mismas posiciones (si hay un número par de ordenaciones) o en $a[t], a[t+1] \ldots$ (si hay número impar de ordenaciones). Distinguir estos dos casos mediante el valor de la variable lógica E.


START
k ← 1
m1 ← f
m2 ← t
intercambiar m1, m2
j ← 0
Si
j ≤ n-1
k ← 2k
k < n
NO
STOP
Mezclar a[r]
desde r = m1+j
hasta r = m1+Min(n-1, j+k-1)
con a[s]
desde s = m1+j+k
hasta s = m1+Min(n-1, j+2k-1)
poner resutado en
a[i] desde i = j+m2
j ← j+2k