

# **BONDADES DE LA LOGICA ESTRUCTURADA EN LA PROGRAMACION DE COMPUTADORES**

FRANKLIN J. VALVERDE DELGADO

Especialista en Sistemas de Información, Universidad EAFIT  
-ICESI. Ingeniero Electricista, Universidad del Valle, Profe-  
sor ICESI, Univalle, Liceo Benálcazar, Asesor de Sistemas.

Con este escrito quiero invitar a quienes diseñan algoritmos de manera tradicional para que cambien su forma de trabajo, y a quienes ya usan la lógica estructurada para que continúen haciéndolo. Igual invitación quiero hacer a los profesores de los cursos de algoritmos, con respecto al tipo de lógica que enseñan a sus alumnos.

Quienes aprendimos a programar hace varios años, lo hicimos en lógica libre, propia de artistas. Sin embargo, la computación siempre ha sido una ciencia desde que empezó y como tal, la investigación ha hecho buenos aportes tales como la lógica estructurada.

La lógica estructurada implica orden, ya que es un método de trabajo, un estilo

propio de la ciencia, y logra que todos aquellos que diseñan algoritmos puedan interpretar fácilmente los realizados por otros, lo que no pasa con la lógica libre. Quien hace algoritmos en lógica libre sabe muy bien que a él mismo al cabo de unos días no le va a ser fácil interpretar sin mucho esfuerzo lo que allí se tiene.

Con la enseñanza de la lógica estructurada he tenido muy buenos resultados tanto a nivel de pregrado como de postgrado en la Universidad del Valle. A nivel de pregrado he comenzado el curso dictando lógica estructurada (pseudocódigo) con estudiantes que inician sus estudios en programación, y los resultados han sido muy buenos. A nivel de postgrado dicté el mismo curso a estudiantes que inician los

prerrequisitos en el magister de Ingeniería Industrial y de Sistemas y los resultados fueron aceptables; que no hayan sido mejores se debió a que la mayoría había recibido en su pregrado cursos de programación en lógica libre; es natural que la gente se resista al cambio; sin embargo, los que aceptaron cambiar, terminaron olvidándose de la lógica libre.

He querido confrontar las dos técnicas con la ayuda de un subprograma escrito en Fortran IV y sugerido por Jhon Narrn, analista de Hewlett Packard (\*)

El subprograma genera una a una todas las posibles permutaciones de un número M. Dichas permutaciones se generan invocando m! (m factorial) veces el subprograma.

DESCRIPCION DEL SUBPROGRAMA  
Su uso:

CALL PERMU (M, NPER, L)

M: Tamaño de la permutación, el cual debe mantenerse constante durante la ejecución del programa.

NPER: Arreglo unidimensional que contiene la permutación.

L: Variable de control de entrada y salida al subprograma. Es igual o menor que cero cuando se genera la primera permutación, la cual será modificada automáticamente por el subprograma. Cuando el subprograma ha generado el ciclo completo de las m! posibles permutaciones, regresará al programa principal el valor L = -1; esto significa que la primera vez que se llama, L debe ser cero o menor que cero y cada vez que se invoque el subprograma, L deberá validarse, hasta que sea -1.

Código FORTRAN IV del Subprograma

```
SUBROUTINE PERMU (M, NPER, I)
DIMENSION NPER (10)
```

```
IF (I) 1,1,3
2 DO 2 K = 1,M
2 NPER (K) = K
I = 1
10 RETURN
3 J = M
11 LT = NPER (I)
LI = 2
4 IF (LI - J) 5,5,6
5 NPER (LI - 1) = NPER (LI)
LI = LI + 1
GOTO 4
6 NPER (J) = LT
IF (NPER (J) - J) 10, 7,10
7 J = J - 1
IF (J - 1) 8,8, 11
8 I = -1
RETURN
END
```

Aquellos que conocen el FORTRAN, estarán de acuerdo en que, tal como está el subprograma, el mejor algoritmo visto en un diagrama de flujo es el que se muestra en la figura 1. Seguidamente presento el algoritmo en lógica estructurada, autodocumentado.

PROCEDIMIENTO PERMU (M, NPER, I)

```
W = 1
SI (I <= 0) ENTONCES
/* Genera la 1a. permutación */
EJECUTE K = 1, M
NPER (K) = K
SIGUIENTE K
I = 1
W = 0
FIN SI
J = M
```

```
/* La variable W controla el siguiente
ciclo para que */
/* genere las permutaciones de la segun-
da en adelante */
HACER MIENTRAS (I < 0 y J > y W = 1)
LT = NPER (I)
/* El siguiente ciclo intercambia las posi-
ciones en el */
/* arreglo NPER */
```

```
EJECUTE LI = 2, J
NPER (LI - 1) = NPER (LI)
```

```
SIGUIENTE LI
NPER (J) = LT
/* Validamos el valor de la posición J en
NPER con */
/* el valor de J */
SI (NPER (J) = J) ENTONCES
```

```
J = J - 1
/* Validamos el final de las permutacio-
nes */
SI (J = 1) ENTONCES
I = -1
```

```
FIN SI
DE OTRO MODO
/* Bloqueo del ciclo, se generó otra
permutación */
```

```
W = 0
FIN SI
FIN HACER MIENTRAS
RETORNAR
FIN PROCEDIMIENTO PERMU
```

Como puede apreciarse, la lógica estructurada utiliza tres estructuras básicas de control: ciclos, comparaciones e instrucciones; con ellas se construye cualquier algoritmo por largo que sea.

Sin pretender convencer al lector de la bondad de la lógica estructurada sino que se convenza solo, puede apreciarse de los anteriores modelos:

Primero, el diagrama de flujo implica dibujos que quitan tiempo y además las planillas que se consiguen en el mercado no permiten a veces colocar las instrucciones completamente, obligando a hacer contracciones en los nombres de las varia-

bles que hacen perder la claridad; y qué decir cuando tal diagrama es largo (más de dos páginas).

Quede claro que los diagramas de flujo también pueden cumplir las normas de la técnica estructurada.

Segundo, el diseño en pseudocódigo, por su naturaleza en prosa, tiene una lógica natural de principio a fin con una entrada y una salida al final del algoritmo, lo que facilita la lectura e interpretación del problema.

Cualquier persona con una formación básica en computación puede saber, con una lectura rápida, qué hace el algoritmo. Incluso puede detectar errores y presentar sugerencias para mejorarlo.

Para completar lo expuesto, analicemos el procedimiento invocándolo desde el algoritmo principal con un valor típico de M, por ejemplo 3.

Deseamos pues generar las 3! = 6 permutaciones del número 123 (123 es la primera).

ALGORITMO PERMUTACIONES

```
L = 0
M = 3
HACER MIENTRAS (L >= 0)
LLAMAR PERMU (M, NPER, L)
SI (L >= 1) ENTONCES
EJECUTE I = 1, M
IMPRIMA NPER (I)
SIGUIENTE I
FIN SI
FIN HACER MIENTRAS
FIN ALGORITMO PERMUTACIONES
```

Aclaremos que la variable I que aparece en el algoritmo principal y la del procedimiento (subalgoritmo) son locales, o sea que la I del principal no es la misma I del subalgoritmo.

También hay que diferenciar el algoritmo del código (lenguaje de programación) ya que en éste último es donde se deben declarar las variables y se dimensionan los arreglos.

(\*) Tomado del libro Introducción a la Programación de Computadores, publicado por la Universidad del Valle División de Ingeniería.

Realizando una prueba de escritorio al algoritmo y al subalgoritmo, encontramos los siguientes resultados:

PRIMER LLAMADO

(M = 3, NPER, I = 0) NPER = **0 0 0**

W = 1

NPER (1) = 1

NPER (2) = 2

NPER (3) = 3

I = 1

W = 0

SEGUNDO LLAMADO

(M = 3, NPER, I = 1) NPER = **1 2 3**

W = 1

J = 3

LT = 1

LI = 2

NEPR (1) = 2

LI = 3

NPER (2) = 3

LI = 4

NPER (3) = 1

W = 0

TERCER LLAMADO

(M = 3, NPER, I = 1) NPER = **2 3 1**

W = 1

J = 3

LT = 2

LI = 2

NPER (1) = 3

LI = 3

NPER (2) = 1

LI = 4

NPER (3) = 2

W = 0

CUARTO LLAMADO

(M = 3, NPER, I = 1) NPER = **3 1 2**

W = 1

J = 3

LT = 3

LI = 2

NPER (1) = 1

LI = 3

NPER (2) = 2

LI = 4

NPER (3) = 3

J = 2

LT = 1

LI = 2

NPER (1) = 2

LI = 3

NPER (2) = 1

W = 0

QUINTO LLAMADO

(M = 3, NPER, I = 1) NPER = **2 1 3**

W = 1

J = 3

LT = 2

LI = 2

NPER (1) = 1

LI = 3

NPER (2) = 3

LI = 4

NPER (3) = 2

W = 0

SEXTO LLAMADO

(M = 3, NPER, I = 1) NPER = **1 3 2**

W = 1

J = 3

LT = 1

LI = 2

NPER (1) = 3

LI = 3

NPER (2) = 2

LI = 4

NPER (3) = 1

W = 0

SEPTIMO LLAMADO

(M = 3 NPER, I = 1) NPER = **3 2 1**

W = 1

J = 3

LT = 3

LI = 2

NPER (1) = 2

LI = 3

NPER (3) = 3

J = 2

LT = 2

LI = 2

NPER (1) = 1

LI = 3

NPER (2) = 2

J = 1

I = -1

A continuación se encontrará el algoritmo y el procedimiento codificados en BASIC y FORTRAN-77 para que se puedan comparar estos códigos con el código FORTRAN IV presentado antes y el lector saque sus propias conclusiones.

CODIGO BASIC.

```
10 REM "PROGRAMA PRINCIPAL
PERMUTACIONES
20 L = 0
30 INPUT "ENTRE M"; M
40 DIM NPER (M)
50 IF (NOT (L >= 0)) THEN GOTO 140
60 GOSUB 160
70 IF (NOT (L >= 1)) THEN GOTO 120
80 FOR I = 1 TO M
90 PRINT NPER (I);
```

100 NEXT I

110 PRINT

120 REM ENDIF

130 GOTO 50

140 REM ENDDO

150 END

160 REM "SUBPROGRAMA PERMUTA-
CIONES

170 W = 1

175 I = L

180 IF (NOT (I <= 0)) THEN GOTO 240

190 FOR K = 1 TO M

200 NPER (K) = K

210 NEXT K

220 I = 1

230 W = 0

240 REM ENDIF

250 IF (NOT (I > 0 AND J > 1 AND

W <= 1)) THEN GOTO 410

260 LT = NPER (I)

270 FOR LI = 2 TO J

280 NPER (LI - 1) = NPER (LI)

290 NEXT LI

300 NPER (J) = LT

310 IF (NOT (NPER (J) = J)) THEN GOTO
370

320 J = J - 1

330 IF (NOT (J = 1)) THEN GOTO 350

340 I = -1

350 REM ENDIF

360 GOTO 390

370 REM ELSE

380 W = 0

390 REM ENDIF

400 GOTO 250

410 REM ENDDO

420 L = I

430 RETURN

NOTA: El código BASIC en general no posee las estructuras básicas de control

DO WHILE, IF... THEN.. ELSE, lo que hace necesaria la implementación de tales estructuras.

CODIGO FORTRAN 77

?BEGIN JOB PERMUTACIONES

?COMPILE PMUTA FORTRAN 77 GO;

?FORTRAN DATA CARD

INTEGER L,M,NPER

DIMENSION NPER (1 $\phi$ )

L = 0

READ (5, 1 $\phi\phi$ ) M

1 $\phi\phi$  FORMAT (1)

1 IF (.NOT. (L. GE.  $\phi$ )) GOTO 2

CALL PERMU (M, NPER, L)

IF (L.GE. 1) THEN

DO 1 $\phi$  I = 1, M

WRITE (6, 2 $\phi$ )(NPER (I), I = 1, M)

1 $\phi$  CONTINUE

END IF

GOTO 1

2 CONTINUE

2 $\phi$  FORMAT (T 25, 312)

END

C\*\*\*\*\*SUBPROGRAMA PERMU \*\*\*

SUBROUTINE PERMU (M, NPER,I)

INTEGER M, I, W, K, NPER, J, LI, LT

DIMENSION NPER (1 $\phi$ )

W = 1

IF (I. LE.  $\phi$ ) THEN

DO 2 $\phi\phi$  K = 1, M

NPER (K) = K

2 $\phi\phi$  CONTINUE

I = 1

W =  $\phi$

END IF

C\*\*\*\*\*CONTINUACION SUBPROGRAMA PERMU \*\*\*\*\*

J = M

3 $\phi\phi$  IF (.NOT (I.GT.  $\phi$ . AND. J. GT. 1 AND.

W. GT. 1)) GOTO 4 $\phi\phi$

LT = NPER (1)

DO 31 $\phi$  LI = 2, J

NPER (LI - 1) = NPER (LI)

31 $\phi$  CONTINUE

NPER (J) = LT

IF (NPER (J). EQ. J) THEN

J = J - 1

IF (J.EQ. 1) THEN

I = -1

END IF

ELSE

W =  $\phi$

END IF

GOTO 3 $\phi\phi$

4 $\phi\phi$  CONTINUE

RETURN

END

