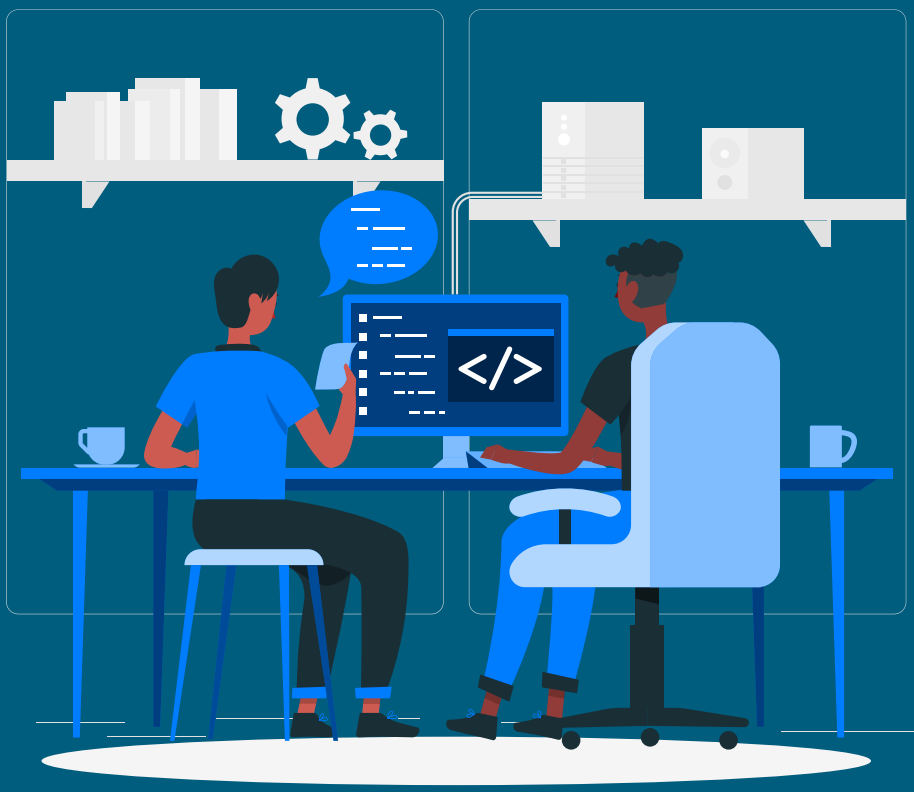


LÓGICA DE PROGRAMACIÓN A PARTIR DEL PENSAMIENTO LÓGICO CON SOFTWARE DFD, PSEINT Y LPP

Mario Dustano Contreras Castro

ediciones USTA

LÓGICA DE PROGRAMACIÓN A PARTIR DEL PENSAMIENTO LÓGICO CON SOFTWARE DFD, PSEINT Y LPP

LÓGICA DE PROGRAMACIÓN A PARTIR DEL PENSAMIENTO LÓGICO CON SOFTWARE DFD, PSEINT Y LPP

Mario Dustano Contreras Castro

ediciones USTA

Contreras Castro, Mario Dustano.
Lógica de Programación a Partir del Pensamiento Lógico con Software Dfd, Pseint y LPP/ Mario Dustano Contreras Castro, Bogotá: Universidad Santo Tomás, 2023.
85 páginas; ilustraciones
Incluye referencias bibliográficas (páginas 85)

E-ISBN: 978-958-782-615-9

1. Lenguajes de programación (Computadores electrónicos) 2. Pensamiento –Lógico 3. Software - algoritmos 4. Lenguajes de programación I. Universidad Santo Tomás (Colombia).

CDD 005.42 CO-BoUST

© Mario Dustano Contreras Castro, autor, 2023
© Universidad Santo Tomás, 2023

Ediciones USTA
Bogotá, D. C., Colombia
Carrera 9 n.° 51-11
Teléfono: (+571) 587 8797, ext. 2991
editorial@usta.edu.co
http://ediciones.usta.edu.co

Corrección de estilo:
Camilo Moreno

Diagramación y diseño de cubierta:
Patricia Montaña D.

E-ISBN: 978-958-782-615-9
Primera edición, julio 2023

Universidad Santo Tomás
Vigilada MinEducación
Reconocimiento personería jurídica: Resolución 3645 del 6 de agosto de 1965, MinJusticia Acreditación Institucional de Alta Calidad Multicampus: Resolución 014525 del 28 de julio de 2022, 8 años, MinEducación

Se prohíbe la reproducción total o parcial de esta obra, por cualquier medio, sin la autorización expresa del titular de los derechos.

CONTENIDO

Lista de figuras

Lista de tablas

Prólogo

Lo que inició como una explicación de un ciclo como un *round point* renovó y removió los cimientos de una pedagogía centrada en la enseñanza de estructuras de lógica de programación de una manera mecanicista y la transformó en un proceso de aprendizaje divertido del uso de la inteligencia lógica. Como consecuencia de este cambio se desarrolló una correlación de experiencias previas que se encuentran directamente vinculadas con la teoría de compiladores en el diseño de autómatas de estados finitos en la orientación y explicación de estructuras de programación como, por ejemplo, la evaluación de condiciones, la selección por casos, los ciclos finitos por conteo, los ciclos por condición de permanencia y los ciclos por condición de salida hacia una nueva mirada de momentos.

Introducción

La división por capítulos tiene como objetivo considerar desde un punto de vista panorámico el pensamiento lógico y su aplicación en autómatas finitos, además, también se considera la aplicación de software en un flujo de datos y después en un pseudolenguaje.

En el primer capítulo se considera el pensamiento lógico como la base para el desarrollo de los diferentes tipos de pensamiento: analítico, convergente y divergente. En el caso del pensamiento analítico se considera la posibilidad de que un enunciado sea verdadero o falso; en este proceso se considera la lógica proposicional. Por otra parte, el pensamiento convergente se entiende como aquel proceso mental que tiene como resultado la respuesta correcta a una pregunta; esta respuesta implica ordenar de manera lógica la información disponible a partir de autómatas finitos como estados, condiciones y comportamientos (transiciones de estados a partir de condiciones). Finalmente se considera el pensamiento divergente entendido como aquellos razonamientos lógicos que implican múltiples soluciones posibles.

En el segundo capítulo se define qué es un algoritmo, además, se explican las proposiciones de lectura, de escritura, de asignación, de ciclo y de selección. Luego, se toma como referente un software de diagramas de flujo para entender cómo funcionan los algoritmos.

En el tercer capítulo se explica cómo el pseudocódigo es una forma detallada de representar por medio de pasos la solución de un algoritmo de forma segmentada (divide y vencerás) utilizando un lenguaje cercano a un lenguaje de programación (LP) con el uso de software como PSeint y LPP. En este contexto, un pseudolenguaje es un lenguaje que puede ser entendido de forma natural por el ser humano y al mismo tiempo puede ser ejecutado por un computador.

El pensamiento lógico

El desarrollo del pensamiento lógico requiere de proposiciones, conceptos y razonamientos coherentes como sucesiones de hechos o ideas y que, además, carezcan de contradicciones. Lo anterior permite obtener los criterios de verdad de la denominada *lógica*.

Pensamiento analítico

Inicialmente evalúa si una proposición o enunciado es verdadera o falsa, lo que significa que no puede ser ambas a la vez. Para conectar dos proposiciones se hace uso de operador de conjunción Y (AND) que solo es verdad cuando las dos proposiciones son verdaderas. Así mismo, el operador de disyunción O (OR) (o disyunción débil) es verdadera si una de las dos proposiciones es verdadera.

Pensamiento convergente

En el caso del pensamiento convergente, la lógica se usa para establecer el conjunto de transiciones (cambios o continuidad) de estado finito como, por ejemplo, el comportamiento de un autómata finito (AF).

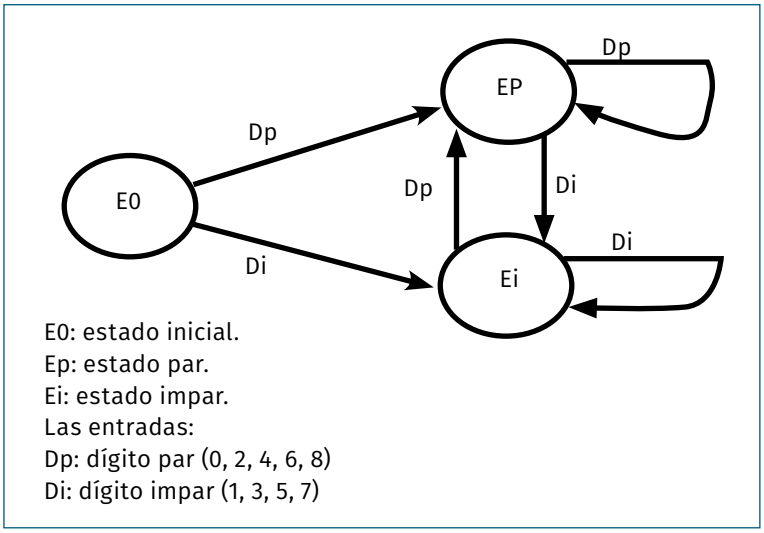


Figura 1. Autómata que evalúa la paridad de un número

Fuente: elaboración propia.

De acuerdo con lo anterior, un número está constituido por un conjunto de dígitos de naturaleza par o impar, lo que permite establecer los estados (inicial, par e impar) y también la naturaleza de los dígitos (par e impar).

	E0	**E1**	**E2**			
E0	D0	01	D2			
E1	D1	D0	D1			
E2	D2	D2	D0			
Las distancias con respecto a 3						
D0	0, 3, 6, 9					
D1	1, 4, 7					
D2	2, 5, 8					
Los residuos con respecto a 3						
E0	Residuo 0					
El	Residuo 1					
E2	Residuo 2					
		D2	D0	D1	D2	D0
Número 56789		2	0	1	2	
Estado	E0	E2	E2	E0	E2	E2

Figura 2. Autómata de estado finito con residuo 3.

Fuente: elaboración propia.

Pensamiento divergente

Se aplican diversos razonamientos lógicos relacionando hechos de forma creativa para obtener una posible solución a una problemática de representación de símbolos.

<símbolos>:- <letras> <dígitos> <operadores><blanco>. Los símbolos pueden ser letras o dígitos o operadores o blanco

<letras>:- <mayúsculas> <minúsculas>. Las letras son mayúsculas o minúsculas

<dígitos>:-0...9. Los dígitos son 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

<operadores>: - <aritméticos> <lógicos> <relación> <bloques> <agrupación> <puntuación> <literal>.

Los operadores son aritméticos o lógicos o de relación o de bloques o de agrupación o de puntuación o literal.

Tabla 1. Tabla de símbolos

SÍMBOLOS	NATURALEZA
A...Z a...z	<letras>
0...9	<dígitos>
+ - * /%	<aritméticos >
=	<asignación>
AND OR	<lógicos>
> < =	<relación>
( ) [ ]	<agrupación>
, ; .	<puntuación>
“ “	<literal>
	<blanco>

Token: como conjunto de símbolos agrupados
Los *tokens* se pueden dividir en:
<Token>:- ><Operandos> <Operadores
<operandos>:- <nombre variables> <nombre funcion><números>
<nombre variable>: - <letra> $\int_{O} < letra >$ <número entero> Ejemplo: n, b1
<nombre método>: - <letra> $\int_{O} < letra >$
<números>:- <números entero> <números reales>
<número enteros>:- $\int_{i} < digito >$ Ejemplo: 22
<números reales>:- <números enteros> . <números enteros> Ejemplo: 9.6
<nombre reservada>: - <todo minúscula>
<operadores>:- ><relación><lógicos><aritméticos<literal>
<Relación>:- < > == (igual) <= >= < > (diferente)
<Lógicos >:- || (or) && (and) ! (negación)
Ejemplo: i<=n && p< >n || k==z
<Aritméticos>:- >:- + - * / % (residuo)
Ejemplo: i++ que es i=i+1
<Literal>:-<char><cadena>
<char>:- ‘<símbolo>’ Ejemplo: ‘a’
<cadena>:-” $\int_{i}^{256}$ <símbolo> “
Ejemplo: “casa”

Representación de la evaluación de una condición

A partir de un árbol de condiciones se establecen las acciones que van a ocurrir cuando se presente o se ausente una condición no obligatoria.

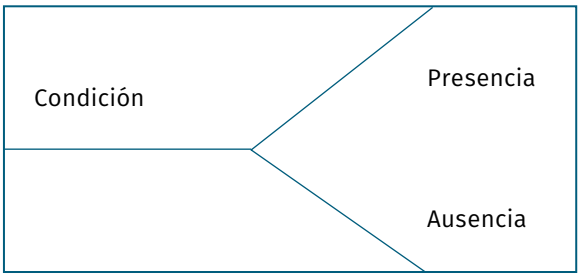


Figura 3. Evaluación de una condición

Fuente: elaboración propia.

Se establece entonces:

SI Condición **ENTONCES**
Proposiciones-presencia-condición
SI-NO
Proposiciones-ausencia-condición
FINSI

Representación de un autómata de repetición fijo – PARA

Se utiliza para la realización de número finito de actividad(es).

Tabla 2. Autómata de repetición fijo – PARA

MOMENTOS	INCREMENTAL	DECREMENTAL
a	Índice = Límite inferior	Índice = Límite superior
b	Índice <= Límite superior	Índice > = Límite inferior
c	Sentencia(s)	Sentencia(s)
d	Índice = Índice + 1 Ir al momento b	Índice = Índice - 1 Ir al momento b

Fuente: elaboración propia.

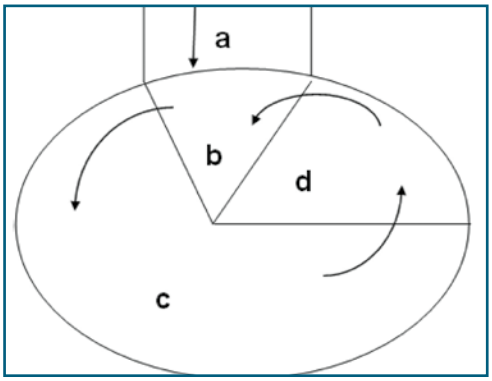


Figura 4. Autómata de repetición fijo – PARA

Fuente: elaboración propia.

Representación de un autómata de repetición controlada – MIENTRAS QUE

Mientras se cumpla condición de continuidad expresada en el momento *b* del autómata, se realizan entonces un número finito de actividades que se encuentra delimitado por el momento *c*. Su representación es la siguiente:

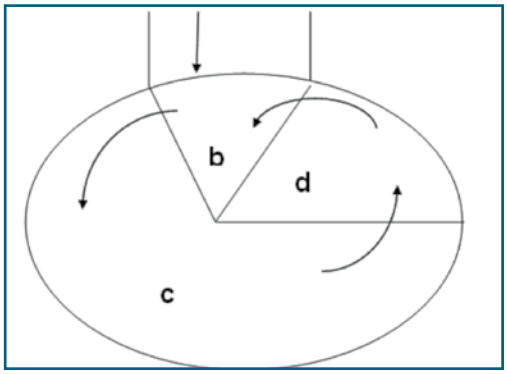


Figura 5. Autómata de repetición - MIENTRAS QUE

Fuente: elaboración propia.

Tabla 3. Autómata de repetición - MIENTRAS QUE

MOMENTOS	¿QUÉ PASA?
b	Si se cumple la condición de permanencia (CP)
c	Sentencia(s)
d	Ir al momento b
e	Condición de salida = negación (CP)

Representación de un autómata que repite una acción - HASTA QUE

Primero se realiza un número finito de actividades que se encuentra expresado en el momento c, para después evaluar una condición de terminación que se encuentra determinada como el momento d. Su representación es la siguiente:

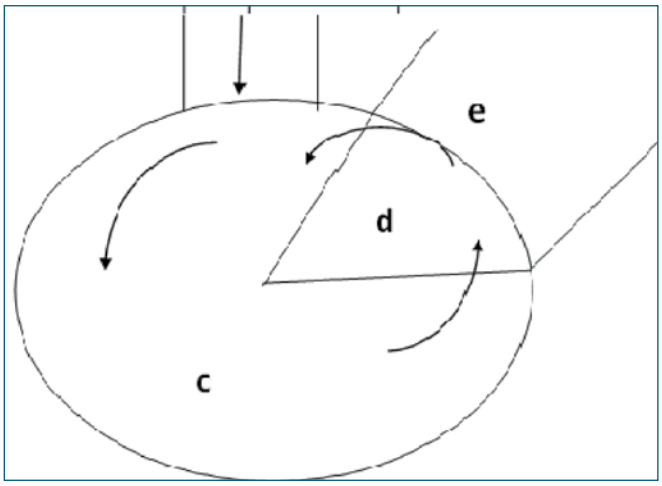


Figura 6. Autómata repita - **HASTA QUE**
Fuente: elaboración propia.

Tabla 4. Autómata de repetición HASTA QUE

MOMENTOS	¿QUÉ PASA?
c	Sentencia(s)
d	Si no se cumple la condición de salida (CS) Ir al momento c o si se cumple la condición de permanencia = negación (CS) ir al momento c
e	CS

Representación de un autómata que replica una acción mientras que se cumple una condición - MIENTRAS QUE

Primero se realiza un número finito de actividades que se encuentra expresado en el momento c, para después evaluar la condición de continuidad que está expresada en el momento d. Su representación se hace de la siguiente manera:

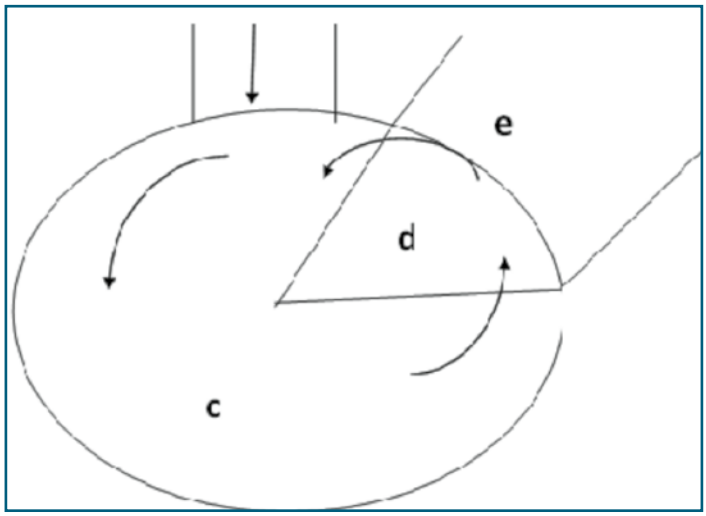


Figura 7. Autómata HAGA MIENTRAS QUE

Fuente: elaboración propia.

Tabla 5. Autómata de repetición HAGA MIENTRAS QUE

MOMENTOS	¿QUÉ PASA?
c	Sentencia(s)
d	Si se cumple la Condición de permanencia (CP) ir al momento c
e	Condición de salida (CS)

Desarrollo de autómata repetición fijo – PARA

1. La multiplicación de dos variables por medio de una suma; esta se representa como la sumatoria de m en *n* veces.

$$\sum_{i=1}^{n} m = 0 + m + m + m \quad n \text{ veces}$$

MOMENTOS	¿QUÉ PASA?	AUTÓMATA
E0	Estado previo al autómata	S = 0 Lea m Lea n
a	Índice = Límite inferior	i = 1
b	Si se cumple la condición de permanencia (CP)	i <= n
c	Sentencia(s)	S = S + m
d	Ir al momento b	i = i + 1 Ir al momento b
e	Condición de salida (CS) = negación (CP)	Imprimir S

2. Factorial de un número n.

Л i = 1*1*2*3*... n (multiplicadora de i *n* veces)

MOMENTOS	¿QUÉ PASA?	AUTÓMATA
E0	Estado previo al autómata	F = 0 Lea n
a	Índice = Límite inferior	i = 1
b	Si se cumple la condición de permanencia (CP)	i <= n
c	Sentencia(s)	F = F × i
d	Ir al momento b	i = i + 1 Ir al momento b
e	Condición de salida (CS) = negación (CP)	i > Límite superior

3. Generar hasta término enésimo de la serie de Fibonacci

MOMENTOS	¿QUÉ PASA?	AUTÓMATA
E0	Estado previo al autómata	Padre = 0 Abuelo = 1 Lea n
a	Índice = Límite inferior	i = 1
b	Si se cumple la condición de permanencia (CP)	i <= n
c	Sentencia(s)	Escribir “Nieto” Abuelo = Padre Padre = nieto
d	Ir al momento b	i = i + 1 Ir al momento b
e	Condición de salida (CS) = negación (CP)	i > Límite superior

4. Gestión de una variable unidimensional.

Una variable Unidimensional se parte del término magnitud vectorial o sea se posee un número de posiciones.

A

1	2	3	4	5	...	n

MOMENTOS	¿QUÉ PASA?	AUTÓMATA
E0	Estado previo al autómata	Lea n Dimensión enter A [n]
a	Índice = Límite inferior	i = 1
b	Si se cumple la condición de permanencia (CP)	i <= n
c	Sentencia(s)	Lea A[i] Escritura A[i]
d	Ir al momento b	i = i + 1 Ir al momento b
e	Condición de salida (CS) = negación (CP)	i > Límite superior

5. Simulación de una potencia.

   Como base (*b*) y potencia (elevado a la *n*) se realiza por medio de multiplicaciones.

MOMENTOS	¿QUÉ PASA?	AUTÓMATA
E0	Estado previo al autómata	Lea n Lea b k = 1
a	Índice = Límite inferior	i = 1
b	Si se cumple la condición de permanencia (CP)	i <= n
c	Sentencia(s)	k = k × b
d	Ir al momento b	i = i + 1 Ir al momento b
e	Condición de salida (CS) = negación (CP)	Escritura K

Estudio de caso: ciclos dentro de otros ciclos

Cuando existe un autómata dentro de otro en el momento *c*.

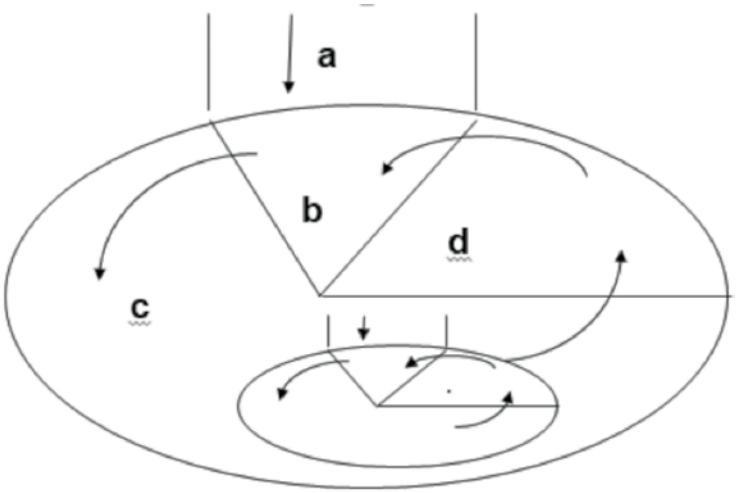


Figura 8. Autómata ciclo anidado

Fuente: elaboración propia.

6. Reloj en horas y minutos.

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata externo	
a1.	Índice = Límite inferior externo	Hora = 0
b1.	Si se cumple la condición de permanencia (CP) Externa	Hora <= 59
c1.	Sentencia(s) del ciclo externo	

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E02	Estado previo al autómata interno	
a2.	Índice = Límite inferior interno	Minuto = 0
b2.	Si se cumple la condición de permanencia (CP) interno	Minuto <= 59
c2.	Sentencia(s) del ciclo interno	
d2.	Ir al momento b2	Minuto = Minuto + 1
e2.	Condición de salida 2	
d1.	Ir al momento b1	Hora = Hora + 1
e1.	Condición de salida 1	

Autómata externo
Autómata interno

7. Operaciones I/O de una variable bidimensional.

Una variable bidimensional se parte dos magnitudes vectoriales o sea se posee un número de posiciones en x y y.

1 2 3 4 . . . n **COLUMNAS**
1
A 2 **FILAS**

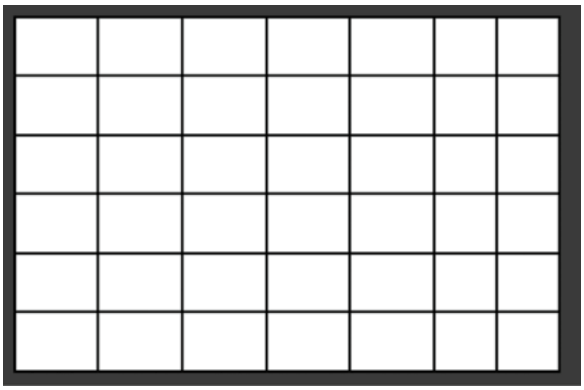


Figura 9. Representación matriz
Fuente: elaboración propia.

Para recorrer una variable bidimensional se requiere de un autómata externo para el recorrido por filas y de un autómata interno para el recorrido por columnas a partir de una fila.

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata externo	Lea n Dimensión entera a [n] [n]
a1.	Índice = Límite inferior externo	i = 1

MOMENTOS	¿QUÉ PASA?	ALGORITMO
b1.	Si se cumple la condición de permanencia (CP) Externa	i <= n
c1.	Sentencia(s) del ciclo externo	
E02	Estado previo al autómata interno	
a2.	Índice = Límite inferior interno	j = 1
b2.	Si se cumple la condición de permanencia (CP) interno	j <= n
c2.	Sentencia(s) del ciclo interno	Lea A [i][i] Escritura A[i][i]
d2.	Ir al momento b2	j = j × 1
e2.	Condición de salida 2	
d1.	Ir al momento b1	i = i + 1
e1.	Condición de salida 1	

Autómata externo
Autómata interno

Triangular inferior

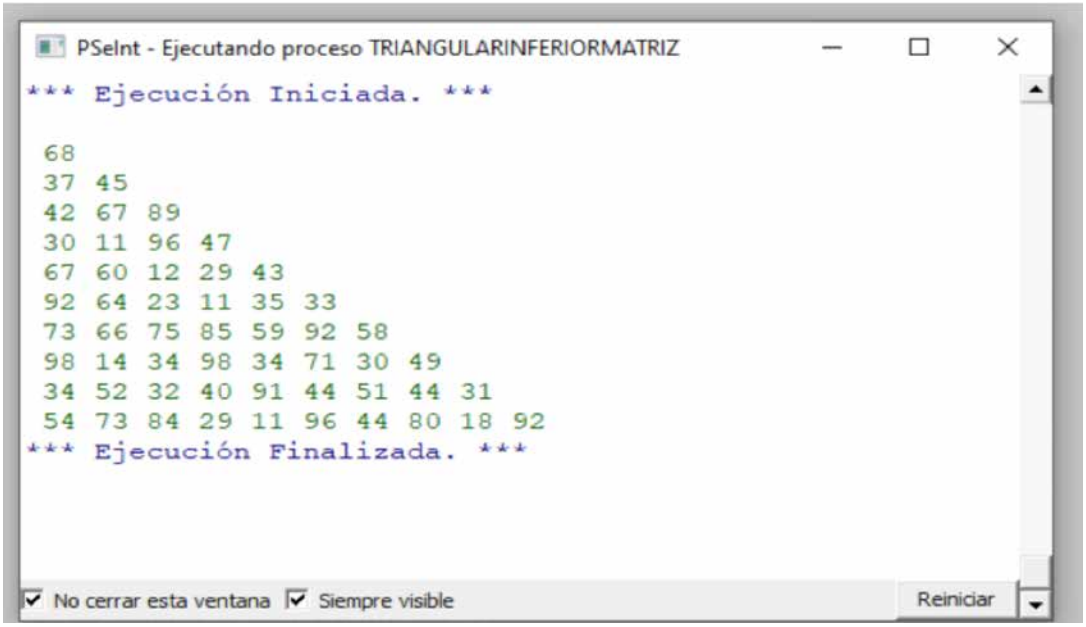


Figura 10. Triangular inferior de una matriz

Fuente: captura de pantalla del programa pseint."

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata externo	Lea n Dimensión entera A [n][n]
a1.	Índice = Límite inferior externo	i = 1
b1.	Si se cumple la condición de permanencia (CP) Externa	i <= n
c1.	Sentencia(s) del ciclo externo	
E02	Estado previo al autómata interno	
a2.	Índice = Límite inferior interno	j = 1
b2.	Si se cumple la condición de permanencia (CP) interno	j <= n
c2.	Sentencia(s) del ciclo interno	Lea A [i][i] Si j < i entonces Escritura A[i][i] FinSi
d2.	Ir al momento b2	j = j × 1
e2.	Condición de salida 2	
d1.	Ir al momento b1	i = i + 1
e1.	Condición de salida 1	

Autómata externo
Autómata interno

Triangular superior

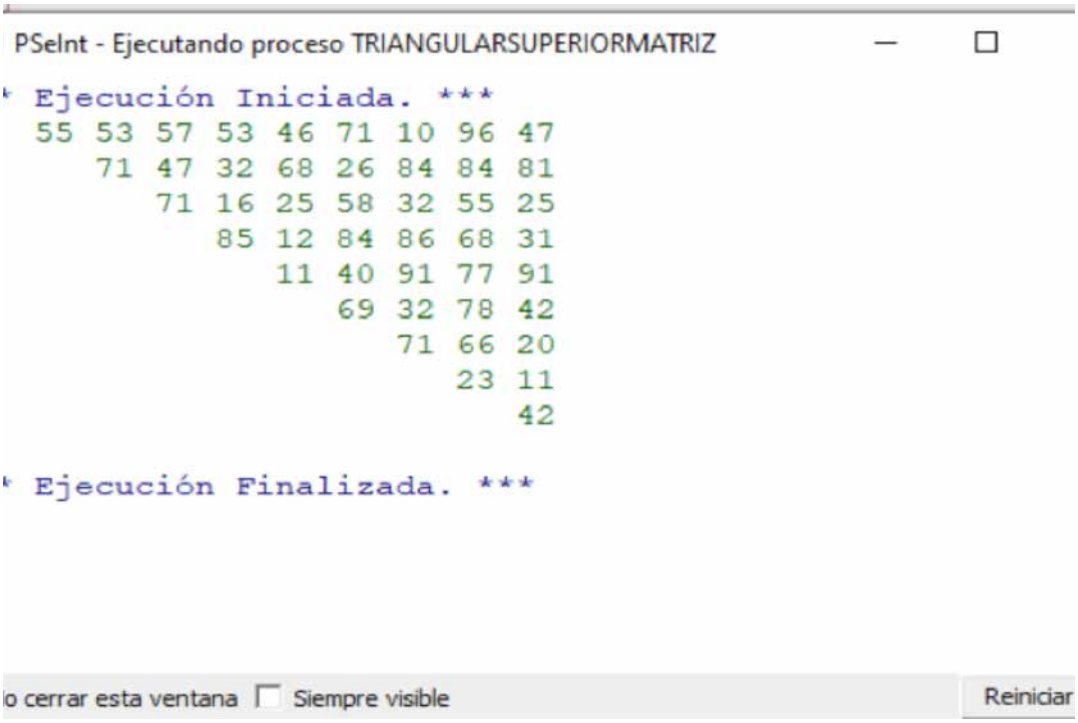


Figura 11. Triangular superior de una matriz

Fuente: captura de pantalla del programa pseint.

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata externo	Lea n Dimensión entera A [n][n]
a1.	Índice = Límite inferior externo	i = 1
b1.	Si se cumple la condición de permanencia (CP) Externa	i <= n
c1.	Sentencia(s) del ciclo externo	
E02	Estado previo al autómata interno	
a2.	Índice = Límite inferior interno	j = 1
b2.	Si se cumple la condición de permanencia (CP) interno	j <= n
c2.	Sentencia(s) del ciclo interno	Lea A [i][i] Escritura A[i][i]
d2.	Ir al momento b2	j = j + 1
e2.	Condición de salida 2	
d1.	Ir al momento b1	i = i + 1
e1.	Condición de salida 1	

Autómata externo
Autómata interno

Diagonal principal

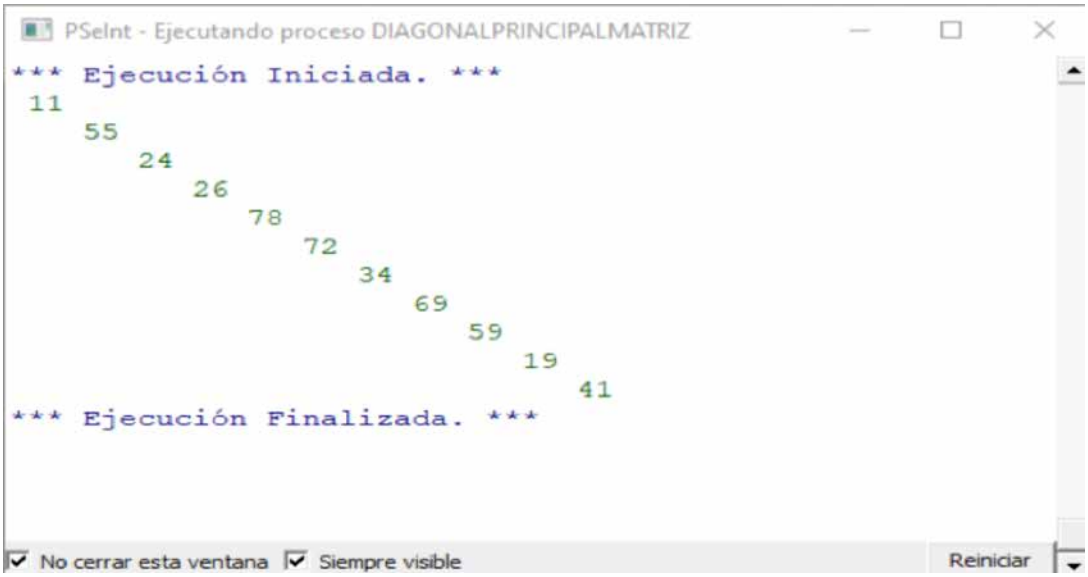


Figura 12. Diagonal principal de una matriz

Fuente: captura de pantalla del programa pseint.

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata externo	Lea n Dimensión entera A [n][n]
a1.	Índice = Límite inferior externo	i = 1
b1.	Si se cumple la condición de permanencia (CP) Externa	i <= n
c1.	Sentencia(s) del ciclo externo	
E02	Estado previo al autómata interno	
a2.	Índice = Límite inferior interno	j = 1
b2.	Si se cumple la condición de permanencia (CP) interno	j <= n
c2.	Sentencia(s) del ciclo interno	Lea A [i][i] Si j = i entonces Escritura A[i][i] FinSi
d2.	Ir al momento b2	j = j + 1
e2.	Condición de salida 2	
d1.	Ir al momento b1	i = i + 1
e1.	Condición de salida 1	

Autómata externo
Autómata interno

Transversal principal

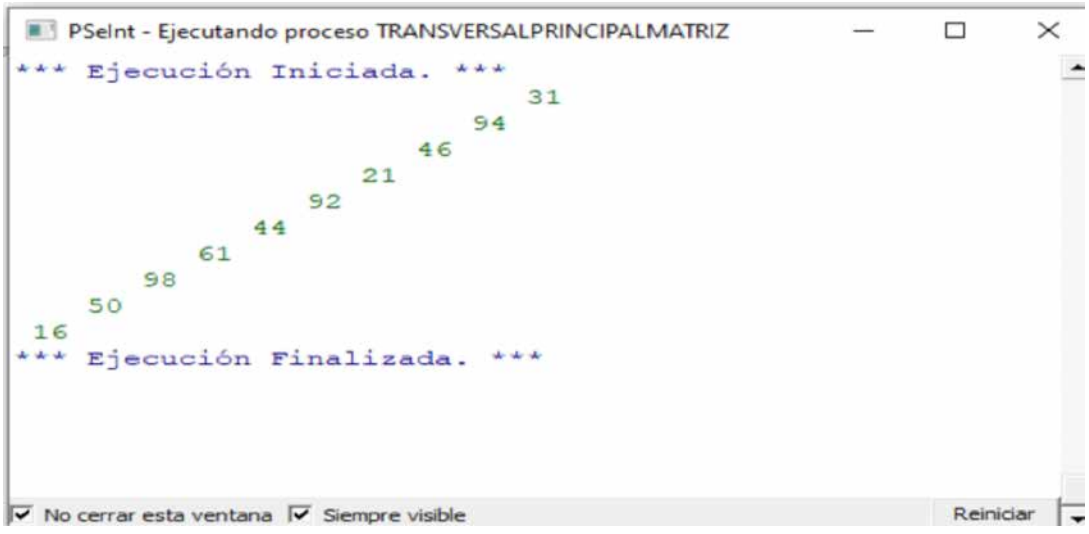


Figura 13. Transversal principal de una matriz

Fuente: captura de pantalla del programa pseint.

Transversal Principal

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata externo	Lea n Dimensión entera A [n][n]
a1.	Índice = Límite inferior externo	i = 1
b1.	Si se cumple la condición de permanencia (CP) Externa	i <= n
c1.	Sentencia(s) del ciclo externo	
E02	Estado previo al autómata interno	
a2.	Índice = Límite inferior interno	j = 1
b2.	Si se cumple la condición de permanencia (CP) interno	j <= n
c2.	Sentencia(s) del ciclo interno	Lea A [i][i] Si j = (n - i) + 1 entonces Escritura A[i][i] FinSi
d2.	Ir al momento b2	j = j + 1
e2.	Condición de salida 2	
d1.	Ir al momento b1	i = i + 1
e1.	Condición de salida 1	

Autómata externo
Autómata interno

8. Ordenamiento lineal de una variable unidimensional.

Una variable Unidimensional se parte del término magnitud vectorial o sea se posee un número de posiciones.

1	2	3	4	5	...	n

Tabla 6. Autómata de ordenamiento lineal de una variable unidimensional

COMPARADO	COMPARANTE
Posición 1	Posición 2 hasta n
Posición 2	Posición 3 hasta n
Posición 3	Posición 4 hasta n
Se concluye entonces Autómata externo Posición i = 1... n - 1	Autómata interno Posición j = i + 1... n

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata externo	Lea n Dimensión entera A [n] Lectura de vector
a1.	Índice = Límite inferior externo	i = 1
b1.	Si se cumple la condición de permanencia (CP) Externa	i < n
c1.	Sentencia(s) del ciclo externo	
E02	Estado previo al autómata interno	
a2.	Índice = Límite inferior interno	j = 1
b2.	Si se cumple la condición de permanencia (CP) interno	j <= n
c2.	Sentencia(s) del ciclo interno	Si A[i] > A[j] aux = A[i] A[i] = A[j] A[j] = aux FinSi
d2.	Ir al momento b2	j = j + 1
e2.	Condición de salida 2	
d1.	Ir al momento b1	i = i + 1
e1.	Condición de salida 1	

Autómata externo
Autómata interno

9. Ordenamiento burbuja de una variable unidimensional.

Una variable unidimensional parte del término de magnitud vectorial, lo que significa que posee un número determinado de posiciones.

Tabla 7. Autómata de ordenamiento burbuja de una variable unidimensional

CICLO EXTERNO	COMPARADO DESDE	COMPARANTE DESDE	COMPARADO HASTA	COMPARANTE HASTA
1	Posición 1	Posición 2	Posición n - 1	Posición n
2	Posición 1	Posición 2	Posición n - 2	Posición n - 1
3	Posición 1	Posición 2	Posición n - 3	Posición n - 2
n - 1	Posición 1	Posición 2	1	2

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata externo	Lea n Dimensión entera A [n] Lectura de vector
a1.	Índice = Límite inferior externo	i = 1
b1.	Si se cumple la condición de permanencia (CP) Externa	i < n
c1.	Sentencia(s) del ciclo externo	

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E02	Estado previo al autómata interno	
a2.	Índice = Límite inferior interno	j = 1
b2.	Si se cumple la condición de permanencia (CP) interno	Si j <= n - i
c2.	Sentencia(s) del ciclo interno	Si A[j] > A[j + 1] Entonces aux = A[i] A[j] = A[j + 1] A[j + 1] = aux FinSi
d2.	Ir al momento b2	j = j + 1
e2.	Condición de salida 2	
d1.	Ir al momento b1	i = i + 1
e1.	Condición de salida 1	

Autómata externo
Autómata interno

Estudio de caso: ciclo finito con evento

Un *evento* es una acción/hecho que se detecta mediante la evaluación de una condición antes de que suceda.

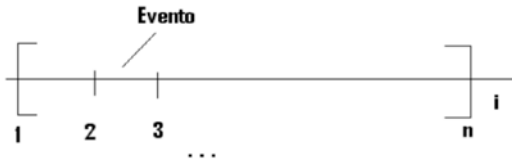


Figura 14. Representación de un evento

Fuente: elaboración propia."

10. Obtener dígitos de un número.

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata	Lea n Dimensión entera A [n] Lea x Tope = 0
b	Si se cumple la condición de permanencia (CP)	Si tope < n AND x > 0 (CP)
c	Sentencia(s)	Tope = Tope + 1 A[Tope] = residuo(x, 10) x = x/10
d	Ir al momento b	
e	Condición de salida (CS) = negación (CP)	

11. Búsqueda de un valor en una variable unidimensional no ordenada.

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata	Lea n Dimensión entera A [n] Lea x Lectura de vector (A) j = 1
b	Si se cumple la condición de permanencia (CP)	Si i < n AND A[i] <> 0 (CP)
c	Sentencias(s)	i = i + 1
d	Ir al momento b	
e	Condición de salida (CS) = negación (CP)	

12. Búsqueda de un valor en una variable unidimensional ordenada.

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata	Lea n Dimensión entera A [n] Lea x Lectura de vector (A) Ordenar vector (A) j = 1
b	Si se cumple la condición de permanencia (CP)	Si i < n AND A[i] <> 0 (CP)
c	Sentencias(s)	i = i + 1
d	Ir al momento b	
e	Condición de salida (CS) = negación (CP)	Si j <= n AND A[i] == x Entonces Escritura x, “Posición “, i SI-NO Escritura x, “No existe” FinSi

13. Cuando un número es primo o no.

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata	Lea n i = 2
b	Si se cumple la condición de permanencia (CP)	Si i < n/2 AND residuo(n,i) <> 0
c	Sentencias(s)	i = i + 1
d	Ir al momento b	
e	Condición de salida (CS) = negación (CP)	Si j <= n/2 Entonces Escritura n, “ no es primo” SI-NO Escritura n, “es primo” FinSi

Desarrollo de autómata de repetición – MIENTRAS QUE

14. Encontrar los dígitos de un número.

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata	Lea n Entero A[n] Lea x Tope = 0
b	Si se cumple la condición de permanencia (CP)	Si Tope < n AND x > 0

c	Sentencias(s)	Tope = Tope + 1
		A[Tope] = residuo (x, 10) // x residuo de 10
		x = x/10

d	Ir al momento b	
e	Condición de salida (CS) = negación (CP)	Escritura vector (A)

Desarrollo de autómata de repetición – HASTA QUE

15. La serie de ULAM se compara

Si el residuo del número *x* con respecto a 2 es 0, entonces se divide por 2 (x/2); si no, entonces se multiplica por tres y se le suma 1 ((x × 3) + 1). El autómata termina cuando el número es igual a 1(condición de salida).

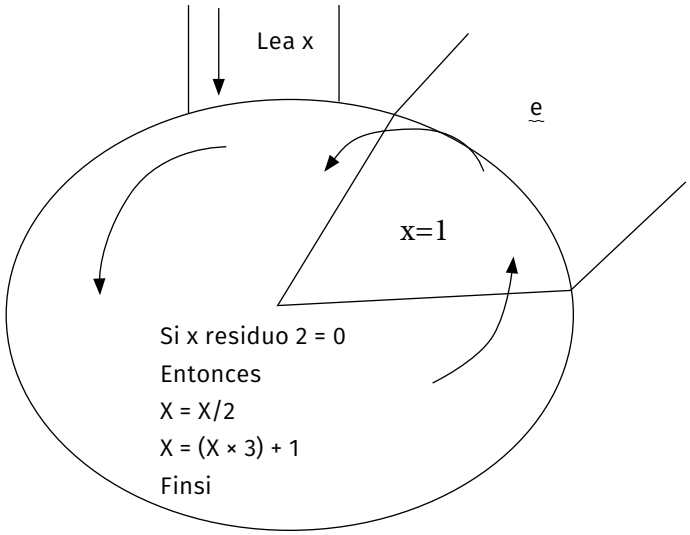


Figura 15. Autómata finito serie ULAM

Fuente: ELABORACIÓN PROPIA.

Tabla 8. Autómata finito serie ULAM

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata	Lea x
c	Sentencias(s)	Si x residuó 2 = 0 entonces x = x/2 Si no x = (x × 3) + 1 FinSi
d	Si no se cumple la condición de salida (CS) Ir al momento c o si se cumple la condición de permanencia = negación (CS) Ir al momento c	Si x diferente de 1 Ir al momento c
e	Condición de salida (CS)	Si x = 1

16. Validar una calificación (nota)

Regla. La calificación (nota) debe estar este entre 1...10.

MOMENTOS	¿QUÉ PASA?	ALGORITMO
c	Sentencias(s)	Lea nota
d	Si no se cumple la condición de salida (Nota < 1 AND nota > 10) Ir al momento c	Si nota > 1 AND nota <= 10 Ir al momento c FinSi
e	Condición de salida (CS)	

17. Partida de tenis de mesa

Regla:
Gana punto jugador 1 si el valor aleatorio generado es 0
Gana punto jugador 2 si el valor aleatorio generado es 1

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata	Lea x
c	Sentencias(s)	Si x residuó 2 = 0 entonces x = x/2 Si no x = (x × 3) + 1 FinSi

MOMENTOS	¿QUÉ PASA?	ALGORITMO
d	Si no se cumple la condición de salida (CS) Ir al momento c o si se cumple la condición de permanencia = negación (CS) Ir al momento c	Si x diferente de 1 Ir al momento c
e	Condición de salida (CS)	Si x = 1

18. Búsqueda binaria de un valor en una variable unidimensional ordenada

Declaración de tres índices: mínimo (extremo inferior), máximo (extremo superior) y mitad (posición a evaluar)
La posición-mitad= (mínimo máximo)/2

MOMENTOS	¿QUÉ PASA?	ALGORITMO
E0	Estado previo al autómata	Entera n = 10 Entera A[n] Lectura vector (A) Ordenamiento vector (A) mínimo = 1 Máximo = n Lea k // valor a buscar
c	Sentencias(s)	Mitad = (mínimo + máximo)/2 Si A[mitad] < k Entonces mínimo = mitado + 1 FinSi Si A[mitad] > k Entonces máximo = mitad - 1 FinSi
d	Si no se cumple la condición de salida (mínimo > máximo OR A[mitad] = k) Ir al momento c	Si (mínimo <= máximo OR A[mitad] <> k) Entonces Ir al momento c FinSi
e	Condición de salida (CS)	Si A[mitad] = k Entonces Escritura k, “ posición “, mitad SI-NO Escritura K, “no existe” FinSi

Algoritmos

Los algoritmos se utilizan para resolver un problema de lógica de programación, por lo tanto, es necesaria una secuencia finita, definida y ordenada (antes o después de) de pasos.

Proposiciones básicas o estructuras de programación de un algoritmo

Proposición de inicio

Mediante la palabra **INICIO**

Proposición declarativa

Nombre de variable con formato
Ejemplo.
Nota: número

Proposición de finalizar

Mediante la palabra **FIN**

Proposición de lectura desde el teclado

LEER <nombrevariable>

Proposición de salida por pantalla

ESCRIBIR <nombrevariable>/<valornumerico>/<valorcadena>

Proposición de asignación de valor a variable

Variable = Variables/Constantes/OperaciónAritmética

Proposición de evaluar condición

Si Condición **ENTONCES**
Proposiciones-presencia-condición
SI-NO
Proposiciones-ausencia-condición
FINSI

Estructura de programación ciclo finito PARA

PARA variableindice= 1 **HASTA** veces
Bloque Proposiciones
FPARA

Estructura de programación ciclo finito MIENTRAS QUE

MIENTRASQUE CondiciónPermanencia **HAGA**
Bloque de Proposiciones
FINMIENTRASQUE

Estructura de programación ciclo finito REPITA HASTA QUE

REPITA
Bloque de Proposiciones
HASTAQUE CondiciónSalida **HAGA**

Estructura de programación de selección

A partir de un valor de una variable se realizan unas sentencias.

SELECCIONCASO <nombrevariable>
CASO ÚnicoValor: <sentencias>
CASO Set / Intervalo: <sentencias>
SI-NO
<sentencias>
FIN SELECCIONCASO

Flujo de datos (fd)

Un flujo de datos (FD) es la visualización de procesamiento de datos a partir de operaciones de lectura y escritura, evaluación de condición, ciclos finitos PARA y ciclos finitos MIENTRAS QUE.

Software Data Flow Diagram (DFD)

El Data Flow Diagram (DFD) es un software libre que ayuda en el diseño y en la ejecución de algoritmos representados en diagramas de flujo (DF).

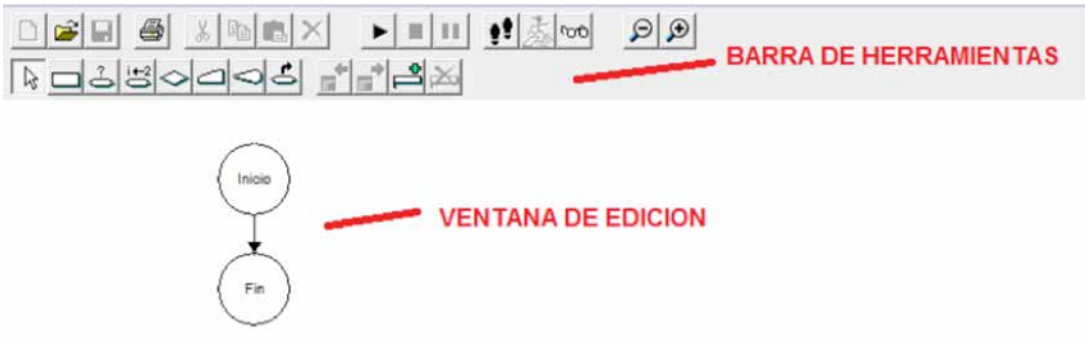


Figura 16. Pantalla inicio software DFD

Fuente: captura de pantalla del programa DFD.

La pantalla de inicio del DFD presenta:

- **Barra de herramientas:** botones de edición, impresión, ejecución, depuración, *zoom*, proposiciones básicas de programación, navegación y subprogramas.
- **Ventana de edición:** cada proposición básica de programación como de subprograma al colocarse sobre la ventana de edición se convierte en objeto de edición. Los botones se editan acorde con su funcionalidad.

Tabla 9. Relación software DFD y algoritmos

SOFTWARE DFD	PROPOSICIÓN	PROPOSICIÓN DE UN ALGORITMO
Inicio	**De Comienzo**	**INICIO**
Fin	**Finalización**	**FIN**
Flujo		
Asignación	**ASIGNACIÓN**	Variable = Variable Variable = Constante Variable = Operación Aritmética

 Lectura por teclado	**LECTURA**	**LEER** variable
 Salida por pantalla	**ESCRITURA**	**ESCRIBIR** Variable
 Decisión (Si entonces)	**EVALUACIÓN CONDICIÓN**	**Si** Condición **ENTONCES** Proposiciones-presencia-condición **SI-NO** Proposiciones-ausencia-condición **FINSI**
 Ciclo para Conteo Finito (Para)	**Repetición Fijo – PARA**	**PARA** variableindice= 1 **HASTA** Veces Bloque Proposiciones **FPARA**
 Ciclo precondición de permanencia (Mientras)	**Repetición Controlada – MIENTRAS QUE**	**MIENTRASQUE** CondiciónPermanencia **HAGA** Bloque de Proposiciones **FINMIENTRASQUE**
	Selección	**SELECCIONCASO** <nombrevariable> **CASO** ÚnicoValor: <sentencias> **CASO** Set / Intervalo: <sentencias> **SI-NO** <sentencias> **FIN SELECCIONCASO**

DFD desarrollados

Factorial

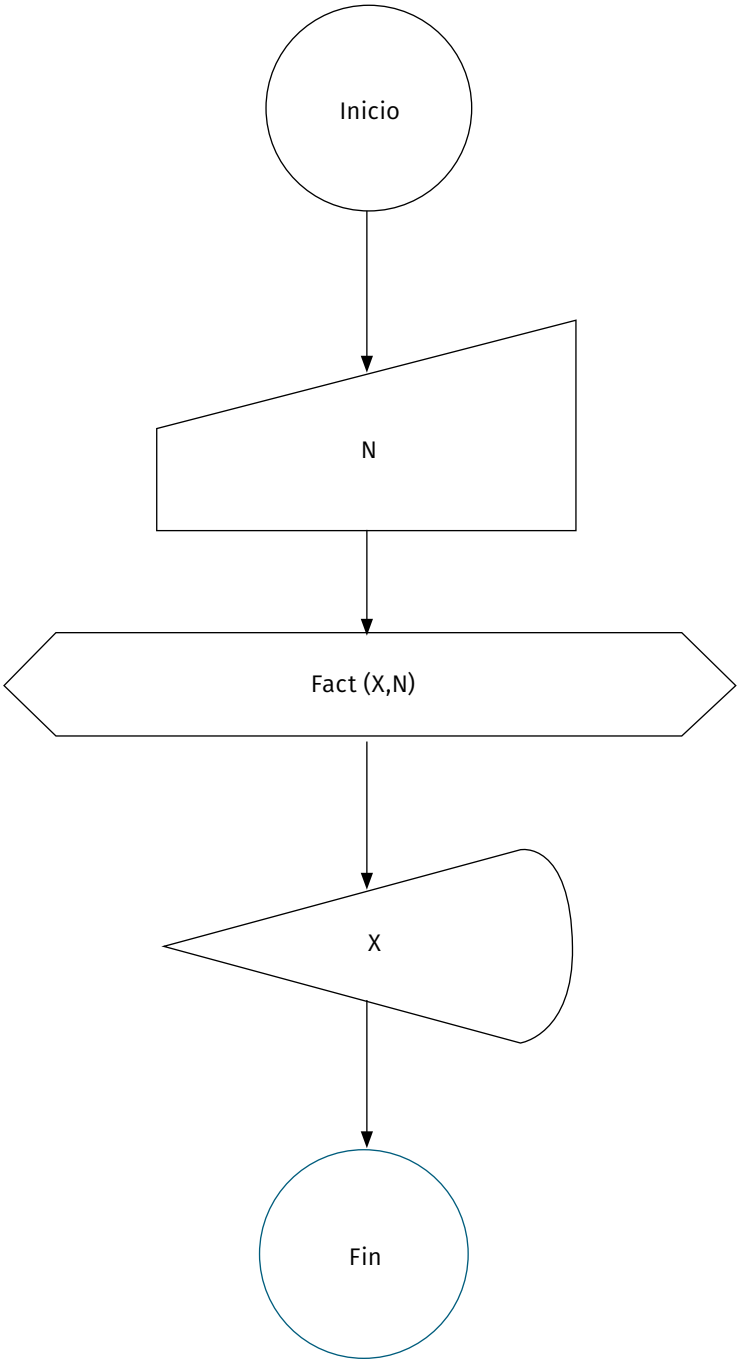


Figura 17. DFD factorial

Fuente: elaboración propia.

Serie de Fibonacci

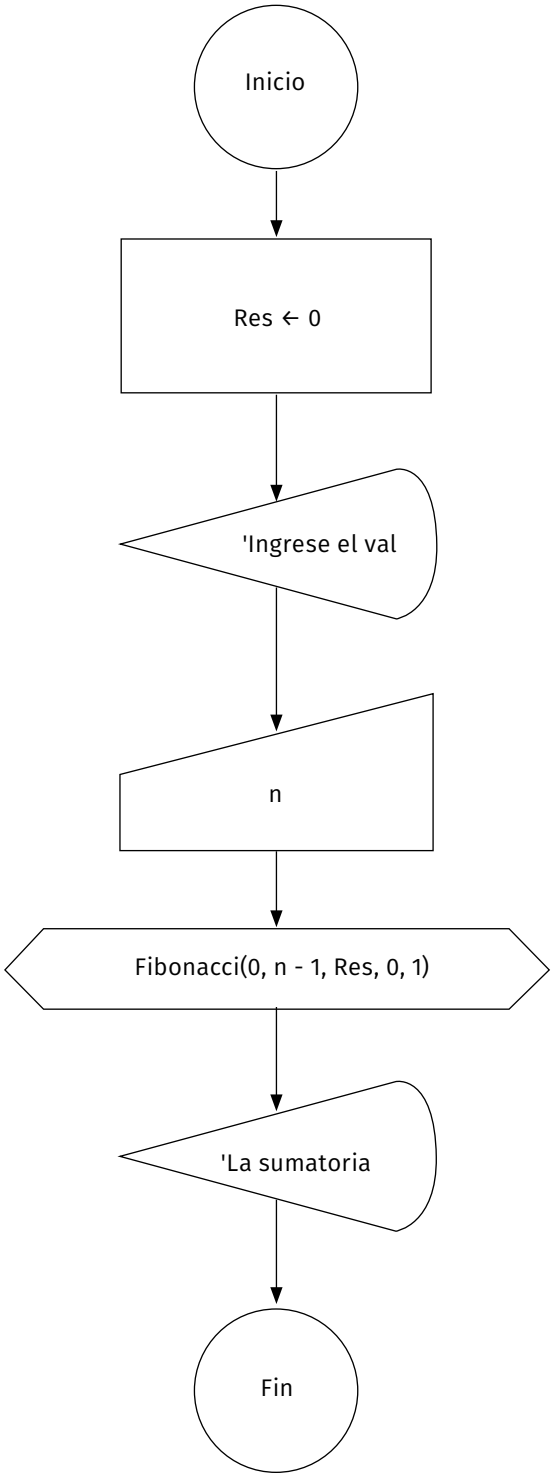


Figura 18. DFD serie Fibonacci

Fuente: elaboración propia.

Potencia

b: base
n: elevado a la...

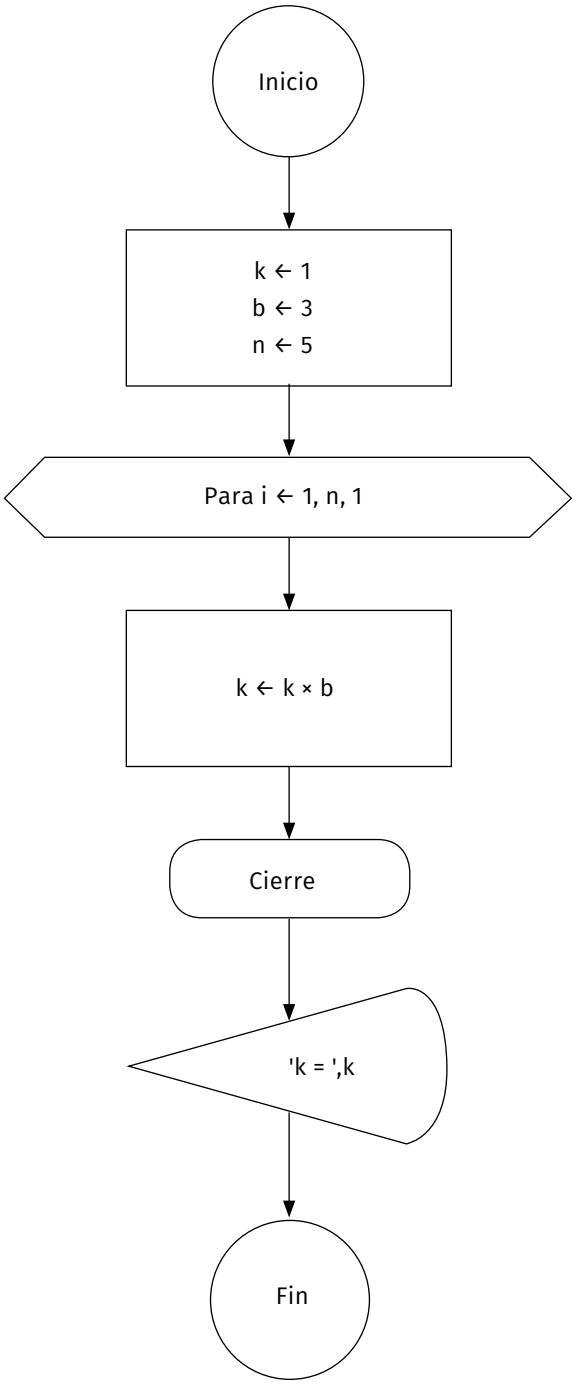


Figura 19. DFD potencia
Fuente: elaboración propia.

Validación nota

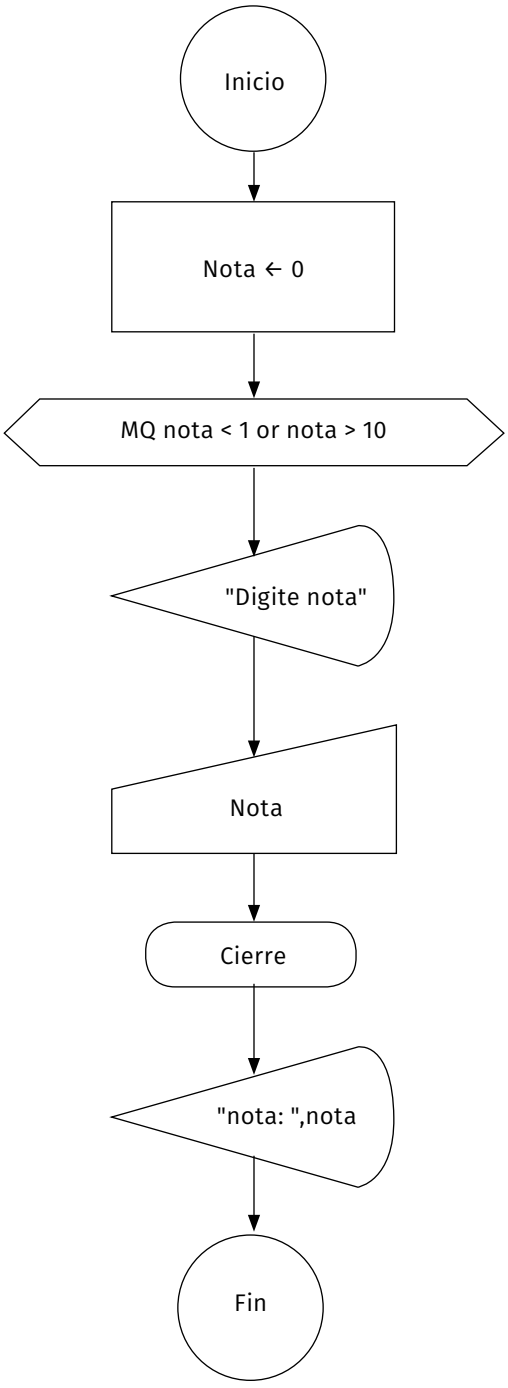


Figura 20. DFD validación nota

Fuente: elaboración propia.

Lectura de un vector

Operadores de cadenas/generador de números aleatorios

DFD	RESULTADO
Random(n)	Genera un número aleatorio entre 0...n-1

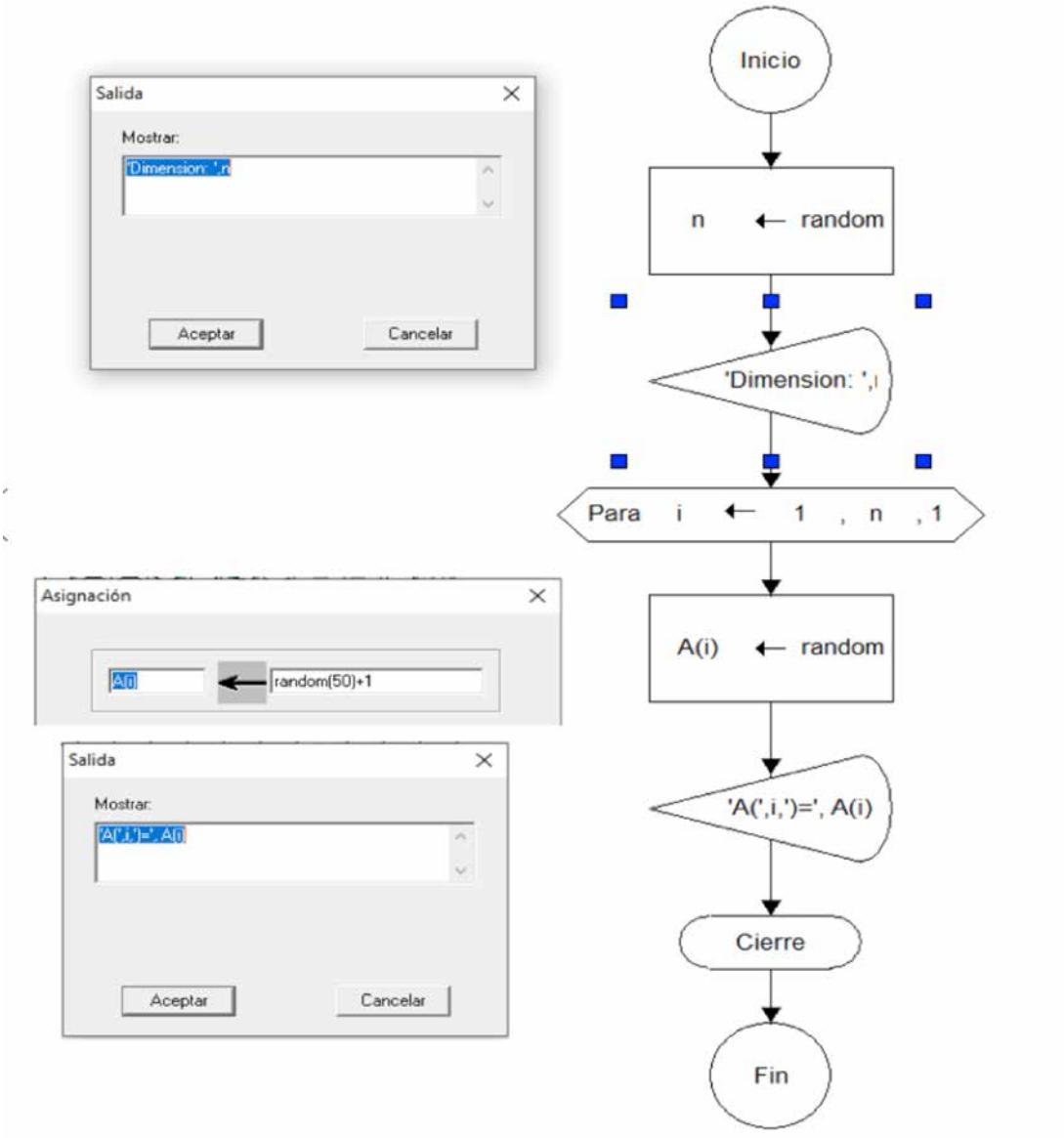


Figura 21. DFD Lectura de un vector

Fuente: elaboración propia y captura de pantalla del programa DFD.

Pseudocódigo

Para la solución de problemas finitos y de lógica programación es necesario iniciar por diagramas de flujo (DF) y luego pasar al pseudocódigo.

Software pseint

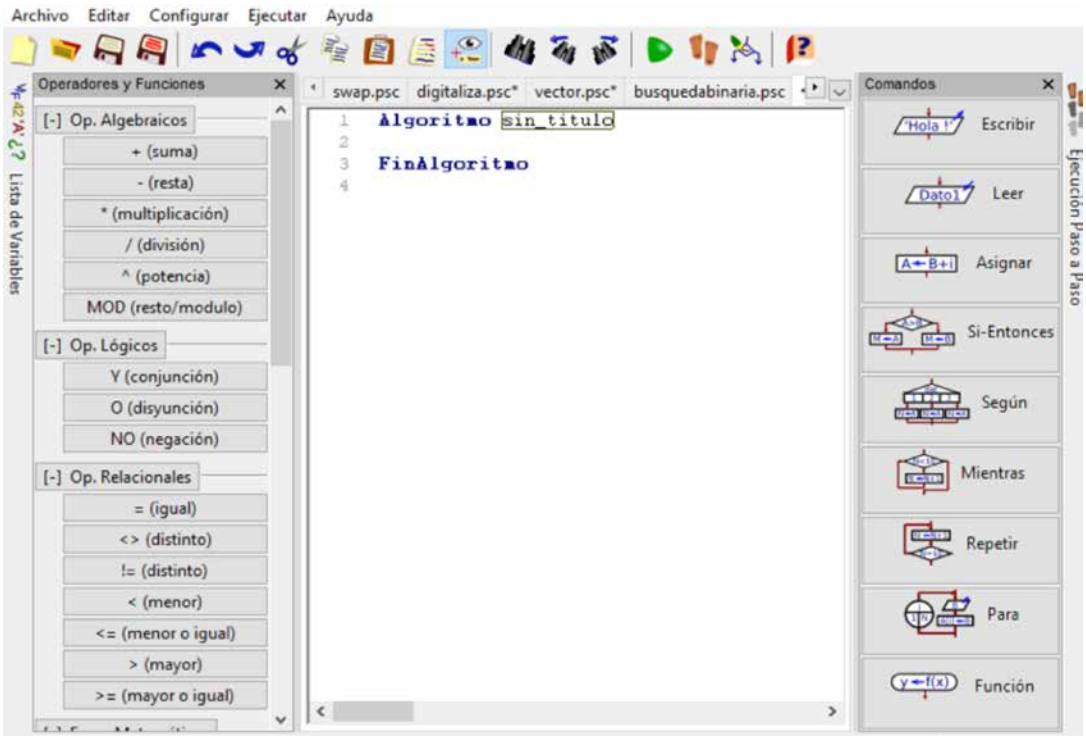


Figura 22. Software Pseint

Fuente: captura de pantalla del programa pseint.

Operadores

Los operadores de Pseint son las siguientes:

OPERADOR	SIGNIFICADO	EJEMPLO
Relacionales		
>	Mayor que	3 > 2
<	Menor que	ABC < abc
=	Igual que	4 = 3
<=	Menor o igual que	a <= b
>=	Mayor o igual que	4 >= 5
<>	Diferente	n <> 5
Lógicos		
& ó Y	Conjunción (y)	(7 > 4) & (2 = 1) // Falso
| ó O	Disyunción (o)	(1 = 1 | 2 = 1) // Verdadero
- ó NO	Negación (no)	- (2 < 5) // Falso
Algebraicos		
+	Suma	Total ← Cantidad 1 + Cantidad 2
-	Resta	Stock ← Disponible - Ventas
×	Múltiplicación	Área ← base × altura
/	División	Porcentaje ← 100 × parte / total
^	Potenciación	Superficie ← 3,41 × Radio ^2
% ó MOD	Módulo (residuo)	Resto ← Número MOD div

Funciones

Las funciones aritméticas de Pseint son las siguientes:

Funciones aritméticas

FUNCIÓN	SIGNIFICADO
RC(X)	Raíz cuadrada de X
ABS(X)	Valor absoluto de X
LN(X)	Logaritmo natural de X
EXP(X)	Fundión exponencial de X
SEN(X)	Seno de X
COS(X)	Coseno de X
TAN(X)	Tangente de X
ASEN(X)	Arcoseno de X
ACOS(X)	Arcocoseno de X
ATAN(X)	Arcotangente de X
TRUNC(X)	Parte entera de X
REDON(X)	Entero más cercano a X
AZAR(X)	Entero aleatorio entre 0 y x - 1

Funciones de cadena

Las funciones de cadena de Pseint son las siguientes:

FUNCIÓN	SIGNIFICADO
LONGITUD(S)	Cantidad de caracteres de la cadena S.
MAYUSCULAS(S)	Retorna una copia de la cadena S con todos sus caracteres en mayúsculas.
MINSCULAS(S)	Retorna una copia de la cadena S con todos sus caracteres en minusculas.
SUBCADENAS(S, X, Y)	Retorna una nueva cadena que consiste en la parte de la cadena S que va desde la posición X hasta la posición Y (incluyendo ambos extremos). Las posiciones utilizan la misma base que los arreglos, por lo que la primer letra será la 0 o la 1 de acuerdo con el perfilo del lenguaje utilizado.
CONCATENAR(S1, 12)	Retorna una nueva cadena resultado de unir las cadenas S1 y S2.
CONVERTIRANUMERO(X)	Recibe una cadena de caracteres que contiene un número y devuelve una variable numérica con el mismo.
CONVERTIRATEXTO(X)	Recibe un real y devuelve una variable numérica con la representación como cadena de caracteres de dicho real.

La estructura de un pseudocódigo

La estructura general es la siguiente:

Algoritmo sin título

Instrucciones

FinAlgoritmo

Recomendaciones

Primera: es obligatorio darle un nombre al programa. Por ejemplo, si estamos sumando números entonces:

Algoritmo **sumanumeros**

FinAlgoritmo

Segunda: se recomienda usar el comando “Guardar Como” y elegir el nombre del programa, por ejemplo, el algoritmo **sumanumeros** debe guardarse como **sumanumeros**

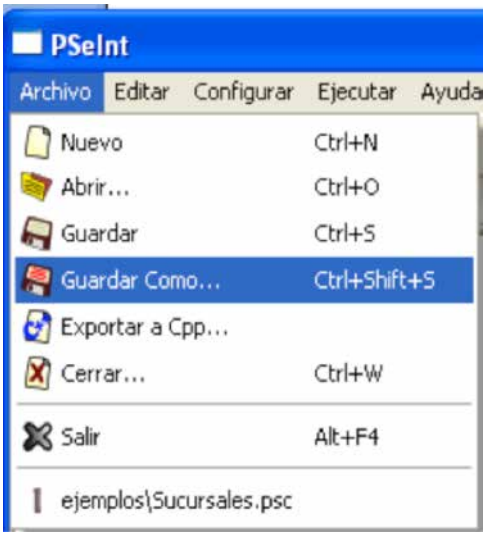


Figura 23. Menú archivo

Fuente: captura de pantalla del programa Pseint.

Tercera: ubicarse en la línea de código y después pulsar un clic en el comando de la barra de comandos.

Figura 24. Barra de comandos Pseint

Fuente: captura de pantalla del programa Pseint.

Cuarta: editar lo que aparece entre { }.

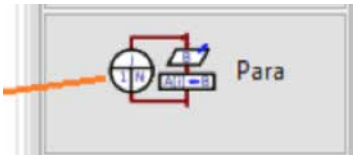
Quinta: utilizar la cláusula *definir* nombrevariable *como* formato (número/carácter/ lógico) para declaración de variables. Por ejemplo:

definir m,k como número;

Sexta: definir un arreglo (vector o matriz) con número de posiciones haciendo uso de la sentencia Dimension.

Dimension <identificador> (<maxl>,...,<maxN>);

Desarrollo de un pseudocodigo



1. La suma de valores.

```
Algoritmo sumatienda
sumatoria<-0
Para k<-1 Hasta 6 Con Paso 1 Hacer
Escribir “Valor Artículo “,k, “:”
Leer valor
sumatoria<-sumatoria+valor
Fin Para
Escribir “sumatoria:”,sumatoria
FinAlgoritmo
```

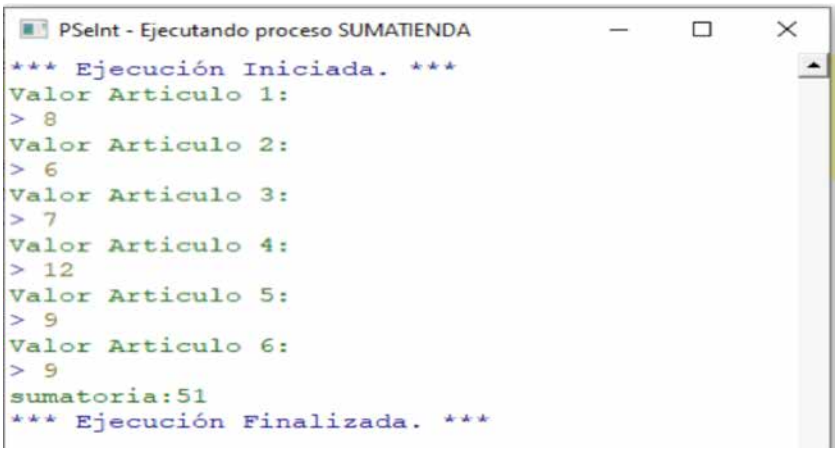


Figura 25. Ejecución sumatoria de valores

Fuente: captura de pantalla del programa PSeint.

2. El Factorial de un número

m
Л i = 1*1*2*3*....m (Multiplicatoria de k m veces)
k=1 m - Veces
Algoritmo factorial
fact<-1
Escribir " Número de términos:"
Leer m
fact<-1
Para k<-1 Hasta m Con Paso 1 Hacer
fact<-fact*k
Fin Para
Escribir "factorial de:",m," es:",fact
FinAlgoritmo

Figura 26. Ejecución de factorial
Fuente: captura de pantalla del programa PSeint.

3. Generar los enésimos términos de la serie de Fibonacci

Algoritmo Fibonacci
Definir semilla, m, fruto, planta como Entero;
semilla<-0; fruto<-1
Escribir " lea término: "; Leer m;
Escribir "Serie de Fibonacci";
Para i<-1 Hasta m Con Paso 1 Hacer
planta<-semilla+fruto;
Escribir planta;
fruto<-semilla;
semilla<-planta;
FinPara
FinAlgoritmo

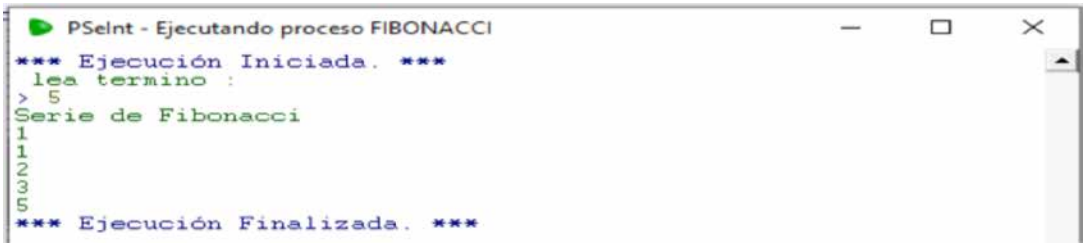


Figura 27. Ejecución de serie de Fibonacci

Fuente: captura de pantalla del programa PSeint.

4. Lectura y escritura de un vector

```
Algoritmo LecturaEscrituraMatriz
Definir x , z,m como Entero;
m<-4;
Dimension D(m,m);
Escribir “Lectura/Escritura Matriz: “;
Para x<-1 Hasta m Con Paso 1 Hacer
Para z<-1 Hasta m Con Paso 1 Hacer
D(x,z)<-azar(70);
Escribir “D(“,x,”,”,z,”):”,D(x,z);
FinPara
FinPara
FinAlgoritmo
```

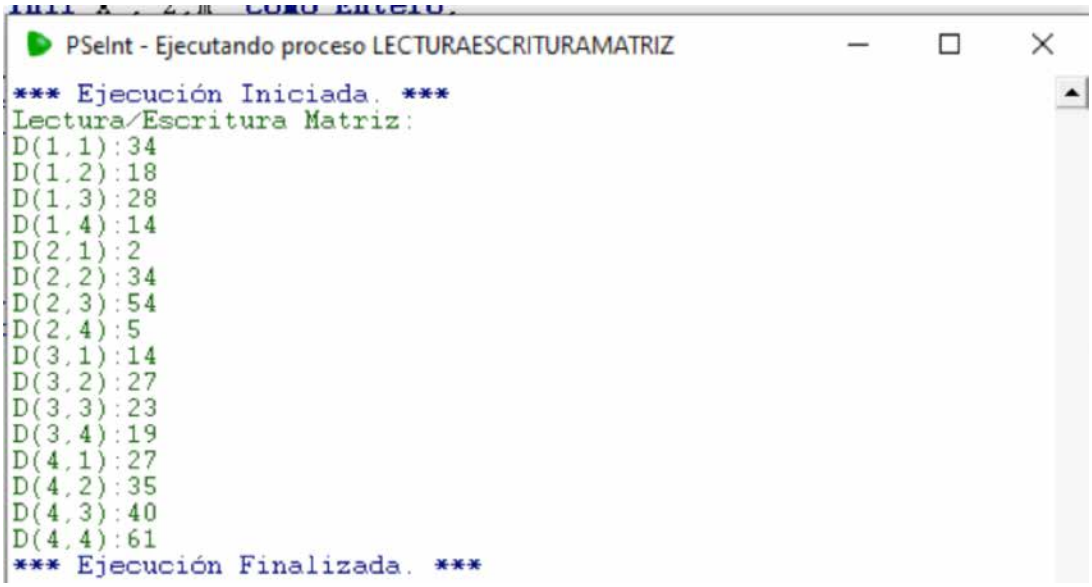



Figura 28. Ejecución lectura de un vector

Fuente: captura de pantalla del programa PSeint.

5. Simulación de una potencia

Como base (b) y potencia (elevado a la n) se realiza por medio de multiplicaciones
Algoritmo potencia
Escribir “Base:”
Leer base
Escribir “potencia:”
Leer m
k<-1
Para i<-1 Hasta m Con Paso 1 Hacer
k<-k*base
Fin Para
Escribir “base:”,base,” elevado a la “,m,” es:”,k
FinAlgoritmo

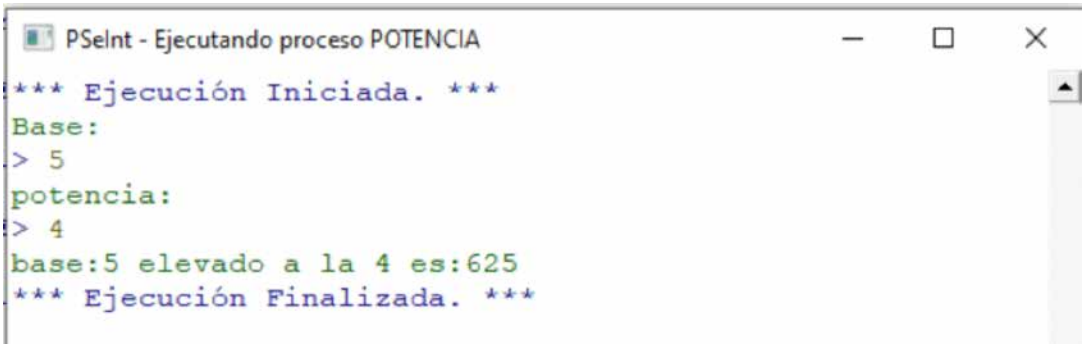


Figura 29. Ejecución de simular una potencia
Fuente: captura de pantalla del programa PSeint.

6. Lectura y Escritura de una matriz

Algoritmo LecturaEscrituraMatriz
Algoritmo LecturaEscrituraMatriz
Definir x , z,m como Entero;
m<-4;
Dimension D(m,m);
Escribir “Lectura/Escritura Matriz: “;
Para x<-1 Hasta m Con Paso 1 Hacer
Para z<-1 Hasta m Con Paso 1 Hacer
D(x,z)<-azar(70);
Escribir “D(“,x,”,”,z,”):”,D(x,z);
FinPara
FinPara
FinAlgoritmo

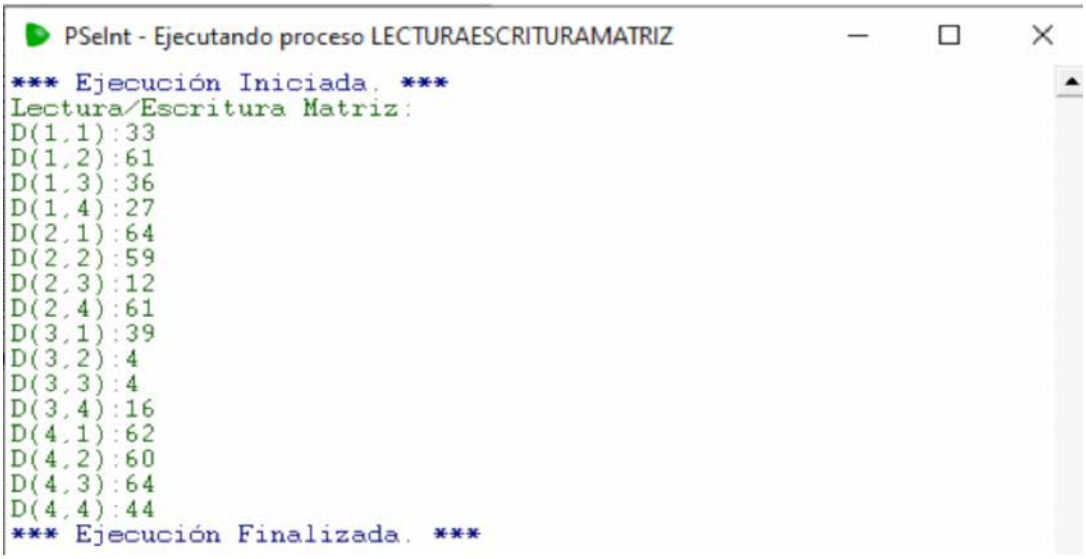


Figura 30. Ejecución de lectura de una matriz

Fuente: captura de pantalla del programa PSeint.

7. Ordenamiento de un vector

```
Algoritmo OrdenarVector
m<-10;
Dimension D(m);
Escribir “Vector Leído: “;
Para x<-1 Hasta m Con Paso 1 Hacer
D(x)<-azar(50);
Escribir “D(“,x,”):”,D(x);
FinPara
Para x<-1 Hasta m-1 Con Paso 1 Hacer
Para z<-x+1 Hasta m Con Paso 1 Hacer
Si D(x)>D(z) Entonces
k<-D(x);
D(x)<-D(z);
D(z)<-k;
FinSi
FinPara
FinPara
Escribir “Vector Ordenado: “;
Para x<-1 Hasta m Con Paso 1 Hacer
Escribir “D(“,x,”):”,D(x);
FinPara
finAlgoritmo
```

Figura 31. Ejecución ordenamiento vector

Fuente: captura de pantalla del programa PSeint.

8. Digitalización de un número


```
Algoritmo DigitosDeNumero
Definir x, m, n como Entero;
Escribir “Digite número”;
Leer k;
m<-k; n<-0
Mientras m>0 Hacer
n<-n+1
m<-trunc(m/10)
Fin Mientras
Dimension B(n)
m<-k
Para i<-1 Hasta n Con Paso 1 Hacer
B(i)<-m % 10
m<-trunc(m/10)
Fin Para
Para i<-1 Hasta n Con Paso 1 Hacer
Escribir B(i);
```

FinPara
FinAlgoritmo

Figura 32. Ejecución digitalizar número
Fuente: captura de pantalla del programa PSeint.

9. Búsqueda de un valor en una variable unidimensional no ordenada

```
Algoritmo Buscarenvector
Definir i,x,m como Entero;
m<-10;
Dimension D(m);
Escribir “Escritura Vector: “;
Para x<-1 Hasta m Con Paso 1 Hacer
D(x)<-azar(50);
Escribir “D(“,x,”):”,D(x);
FinPara
Escribir “Digitar Número a Buscar en vector: “;
Leer k;
x<-1;
Mientras x<=m y D(x)<>k Hacer
x<-x+1;
FinMientras
Si x>m Entonces
Escribir k, “ no existe”
SiNo
Escribir k,” está en la posición: “,x
Fin Si
FinAlgoritmo
```

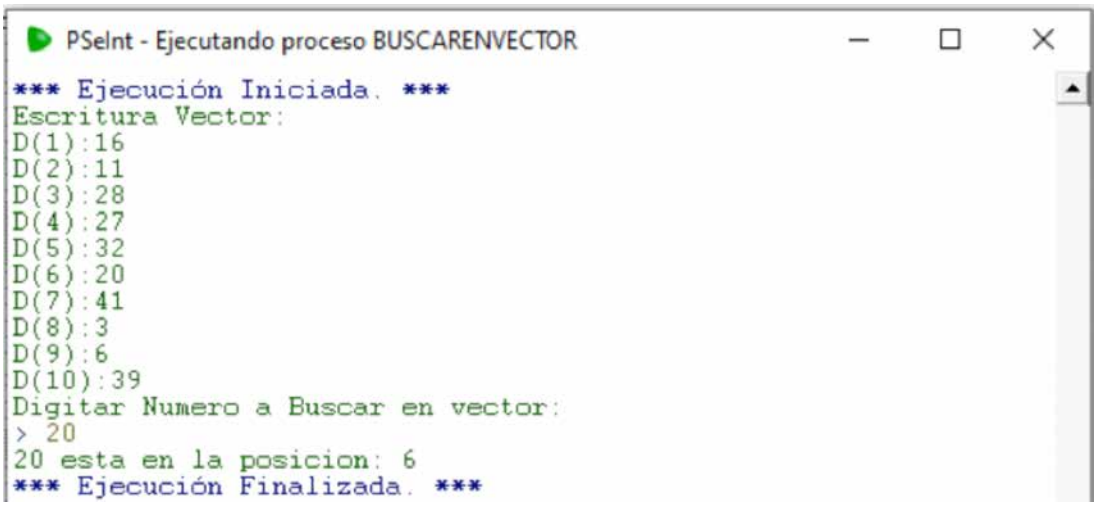


Figura 33. Búsqueda de número en un vector sin ordenar

Fuente: captura de pantalla del programa PSeint.

10. Búsqueda de un valor en una variable unidimensional ordenada

```
Algoritmo BuscarenvectorOrdenado
Definir i,x,m como Entero;
m<-10;
Dimension D(m);
Escribir “Escritura Vector Leído: “;
Para x<-1 Hasta m Con Paso 1 Hacer
D(x)<-azar(99)+1;
Escribir “D(“,x,”):”,D(x);
FinPara
Para x<-1 Hasta m-1 Con Paso 1 Hacer
Para z<-x+1 Hasta m Con Paso 1 Hacer
Si D(x)>D(z) Entonces
k<-D(x);
D(x)<-D(z);
D(z)<-k;
FinSi
FinPara
FinPara
Escribir “Escritura Vector Ordenado: “;
Para x<-1 Hasta m Con Paso 1 Hacer
Escribir “D(“,x,”):”,D(x);
FinPara
Escribir “Digitar Número a Buscar en vector: “;
Leer k;
x<-1;
```

```
Mientras x<=m y D(x)<>k Hacer
x<-x+1;
FinMientras
Si x>m Entonces
Escribir k, “ no existe”;
SiNo
Escribir k, “ está en la posición: “,x;
Fin Si
FinAlgoritmo
```

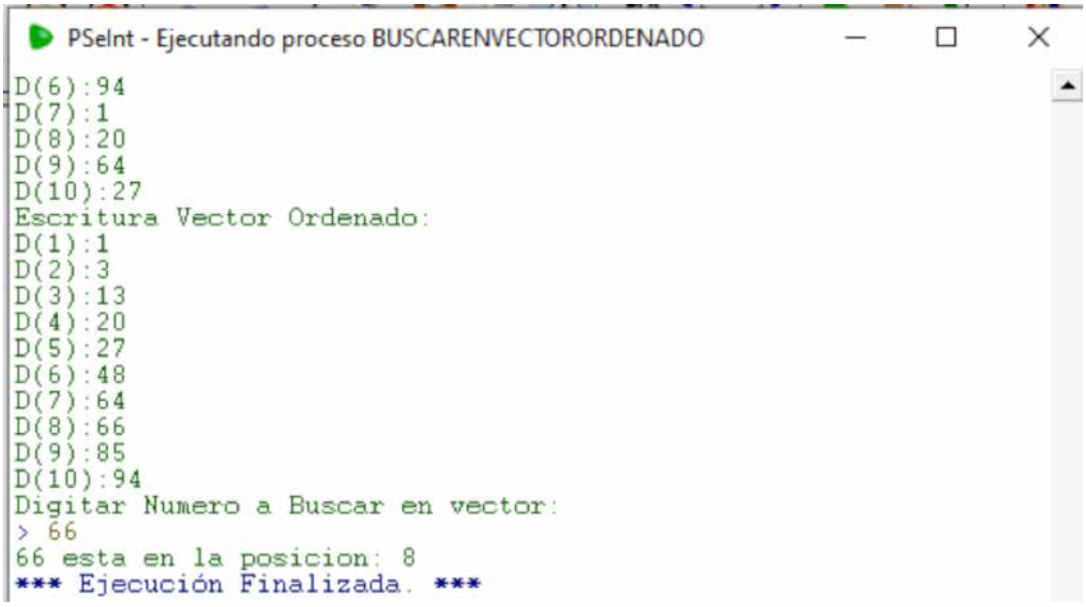


Figura 34. Búsqueda de número en un vector ordenado

Fuente: captura de pantalla del programa PSeint.

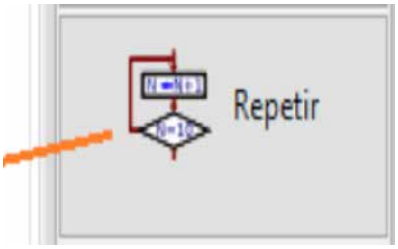
11. Comprobar si el número es primo o no.

```
Algoritmo NumeroPrimo
Escribir “Digite Número: “;
Leer m;
j<-2;
Mientras j<=m/2 y m%j<>0 Hacer
j<-i+1;
FinMientras
Si j<=m/2 Entonces
Escribir m, “ No es primo “;
Sino
Escribir m, “ Es Primo”;
FinSi
finAlgoritmo
```

Figura 35. Evaluar si un número es primo

Fuente: captura de pantalla del programa PSeint.

12. Encontrar Intervalo




```
Algoritmo limite
Repetir
k<-azar(500);
Escribir k
Hasta Que k>=200 y k <=300
Escribir “Salida: “,k
FinAlgoritmo
```


Figura 36. Encontrar intervalo

Fuente: captura de pantalla del programa PSeint.

13. Simular un juego de volleyball

Regla:
Si el valor aleatorio generado (*random*) es igual 0 gana un punto el jugador 1, de lo contrario, gana un punto el jugador 2

```
Algoritmo juegovolleyball
equipo1<-0;
equipo2<-0;
Repetir
Si azar(2)=0 Entonces
Escribir “gana punto Equipo 1”;
equipo1<-equipo1+1;
Sino
Escribir “gana punto Equipo 2”;
equipo2<-equipo2+1;
FinSi
Hasta Que (equipo1>=25 | equipo2>=21) y abs(equipo1-equipo2)>=2
Si equipo1>equipo2 Entonces
Escribir “gana Equipo 1 con “, equipo1,” contra “, equipo2;
Sino
Escribir “gana Equipo 2 con “, equipo1,” contra “, equipo2;
FinSi
finAlgoritmo
```

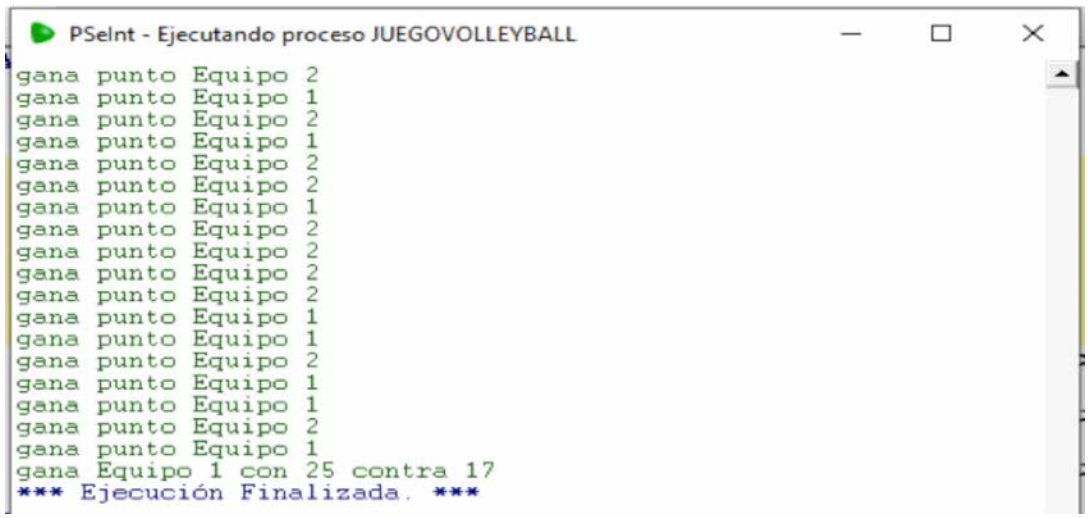



Figura 37. Simular juego de volleyball

Fuente: captura de pantalla del programa PSeint.

14. Búsqueda Binaria

```
Algoritmo BuscarVectorOrdenado
m<-10
Dimension D(m)
Escribir “Vector Leído: “
Para x<-1 Hasta m Con Paso 1 Hacer
D(x)<-azar(50)//se genera aleatoriamente un número entre 1 y 50
Escribir “D[“,x,”]:”,D(x)
FinPara
Para x<-1 Hasta m-1 Con Paso 1 Hacer
Para z<-i+1 Hasta m Con Paso 1 Hacer
Si D(x)<D(z) Entonces
k<-D(x)
D(x)<-D(z)
D(z)<-k
FinSi
FinPara
FinPara
Escribir “Vector Ordenado:”
Para x<-1 Hasta m Con Paso 1 Hacer
Escribir “D(“,x,”):”,D(x)
FinPara
Escribir “Leer Valor a Buscar”
Leer k
minimo<-1
maximo<-m
Repetir
mitad<-trunc((minimo+maximo)/2)
Si D(mitad)<k Entonces
minimo<-mitad+1
FinSi
Si D(mitad)>k Entonces
maximo<-mitad-1
FinSi
Hasta Que minimo > maximo | D[mitad] = k
Si D[mitad] = k Entonces
Escribir k,” esta en la posición: “ ,mitad
Sino
Escribir k, “no existe”
FinSi
finAlgoritmo
```

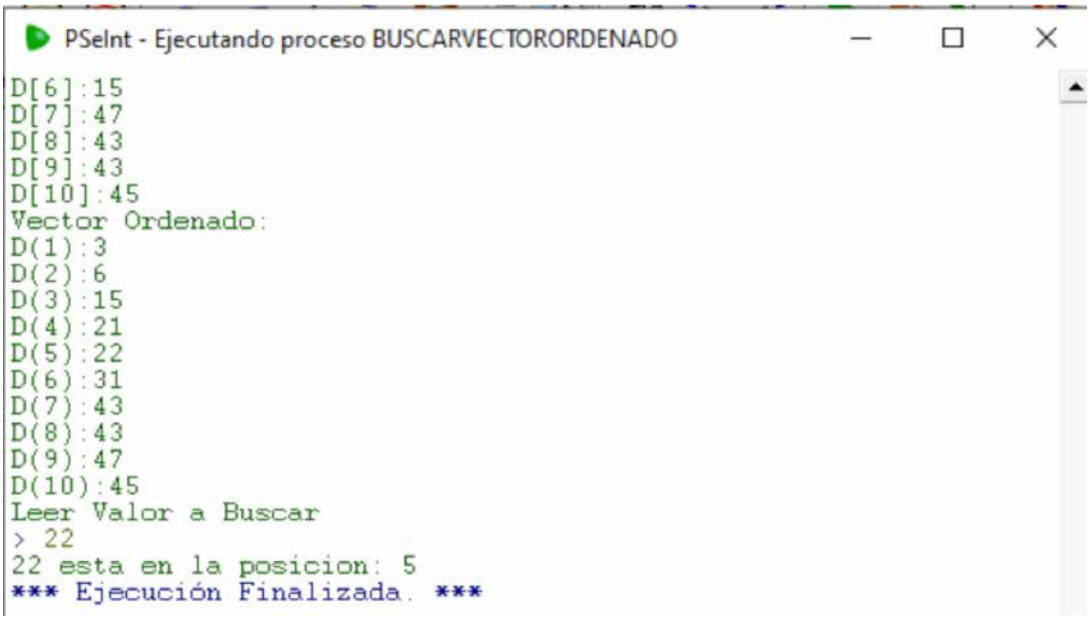


Figura 38. Búsqueda binaria en un vector

Fuente: captura de pantalla del programa PSeint.

15. Identificar estación acorde con mes

mes>=11 and edad<=12 and mes=1:: invierno
mes>=2 and mes<=4 :: primavera
mes>=5 and mes<=7 :: verano
mes>=8 and mes<=10 :: otoño
Algoritmo seleccionmes
Para i<-1 Hasta 3 Con Paso 1 Hacer
mes<-azar(12);
Escribir “Mes: “,mes;
Segun mes Hacer
11, 12, 1: Escribir “ Invierno “;
2.3,4:

Escribir “ Primavera “;
5.6,7:
Escribir “ Verano “;
De Otro Modo:
Escribir “Otoño”;
FinSegun
FinPara
FinAlgoritmo

Figura 39. Selección de meses
Fuente: captura de pantalla del programa pseint.

Otras sentencias secuenciales del Pseint

Borrar Pantalla
Sintaxis: Borrar Pantalla;
Esperar Tecla
Sintaxis: Esperar Tecla;
Esperar n segundos
Sintaxis: Esperar n Segundos

Programación Modular pseint

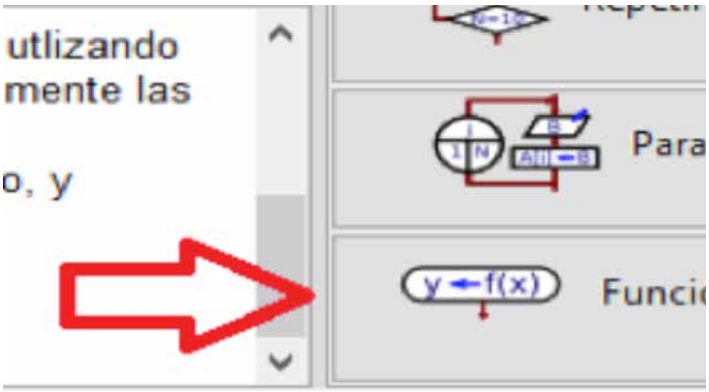


Figura 40. Opción programación modular
Fuente: captura de pantalla del programa pseint.

Función variable_de_retorno <- Nombre (Argumentos)
Fin Función

Análisis de un subalgoritmo PSEINT

El nombre (argumentos) se coloca la lista de argumentos separados por el signo coma (,). Así mismo, *nombre* es el nombre del subalgoritmo. Si no existen argumentos, se coloca solamente ().

Además, no existe la cláusula de retorno.
Cuando se requiere retornar un valor, se hace uso de una *variable de retorno*.
Al quitar la cláusula "variable_de_retorno <" no se retorna ningún valor.
Nota. Los vectores como matrices son argumentos por referencia
El llamado a un subAlgoritmo tiene la siguiente sintaxis:
Nombre_subAlgoritmo(Argumentos)
Si no existen argumentos se coloca solamente ().
Ahora, dos subAlgoritmos con argumentos por valor

Modular.psc

```
1  SubAlgoritmo k <- factorial ( n por valor )
2      k<-1
3      Para i<-1 Hasta n Con Paso 1 Hacer
4          k<-k*i
5      Fin Para
6  Fin SubAlgoritmo
7
8  Algoritmo Modular
9      Escribir "Digite Numero:"
10     Leer n
11     Escribir "Resultado: ",factorial(n)
12 FinAlgoritmo
13
14
15
16
```

Figura 41. Programación modular PSeint
Fuente: captura de pantalla del programa PSeint.

SubAlgoritmo k <- factorial (n por valor)
k<-1
Para i<-1 Hasta n Con Paso 1 Hacer
k<-k*i
Fin Para
Fin SubAlgoritmo
Algoritmo Modular
Escribir "Digite Número:"
Leer n
Escribir "Resultado: ",factorial(n)
FinAlgoritmo

Ejemplo:
main ()
Cuerpo
Entero b,n
Lea b
Lea n
Escriba Potencia(n)
FinCuerpo
Función Potencia (b,n)
Cuerpo
Entero i,k=1
Para i=1 hasta n Haga
k=k*b
FinPara
Retorno k
FinCuerpo

Ejemplos de PSEINT

Primer ejemplo. Función potencia

funcion k<-potencia (b Por valor,n por valor)
k<-1;
Para i<-1 Hasta n Con Paso 1 Hacer
k<-k*b;
Fin Para
Fin SubAlgoritmo
Algoritmo potencias
b=azar(6)+2;
n= azar(10)+2;
Escribir b," elevado ",n," = ",potencia(b,n);
FinAlgoritmo

Segundo ejemplo. Digitalizar un número

Función s <-imprimir (B por referencia,n por valor)
s<-" "
Para i<-1 Hasta n Con Paso 1 Hacer
s<-concatenar(s," "+ConvertirATexto(B(i)))
Fin Para
Fin Función
Función digital (B por referencia,m por valor,n por valor)
Para i<-1 Hasta n Con Paso 1 Hacer
B(i)<-m % 10

```
m<-trunc(m/10)
Fin Para
Fin Función
Función c <-cuantos ( m por valor )
c<-0
Mientras m>0 Hacer
c<-c+1
m<-trunc(m/10)
Fin Mientras
Fin Función
Algoritmo digitaliza
Escribir “Digite valor”
Leer x
n<-cuantos(x)
Dimension A(n)
digital(A,x,n)
Escribir imprimir(A,n)
Esperar Tecla
FinAlgoritmo
```

Tercer ejemplo. Valor único en un vector


```
Función valorunico(B por referencia,max por valor)
Repetir
j=1
k=azar(50)+1
Mientras j<max && B(j)<>k Hacer
j<-j+1
Fin Mientras
Hasta Que j=max
B(j)<-k
FinFuncion
Función s <-imprimir ( B por referencia,n por valor )
s<-” “
Para i<-1 Hasta n Con Paso 1 Hacer
s<-concatenar(s,ConvertirATexto(B(i)))
s<-concatenar(s,” “)
Fin Para
Fin Función
Función valorunico(B por referencia,max por valor)
Repetir
j=1
k=azar(50)+1
Mientras j<max && B(j)<>k Hacer
```

```
j<-j+1
Fin Mientras
Hasta Que j=max
B(j)<-k
FinFuncion
Función s <-imprimir ( B por referencia,n por valor )
s<-” “
Para i<-1 Hasta n Con Paso 1 Hacer
s<-concatenar(s,ConvertirATexto(B(i)))
s<-concatenar(s,” “)
Fin Para
Fin Función
Algoritmo vectorvalorunico
Escribir “Digite Dimension”
Leer m
Dimension C(m)
Para i<-1 Hasta m Con Paso 1 Hacer
valorunico(C,i)
Fin Para
Escribir “Vector Generado:”
Escribir imprimir(C,m)
FinAlgoritmo
```

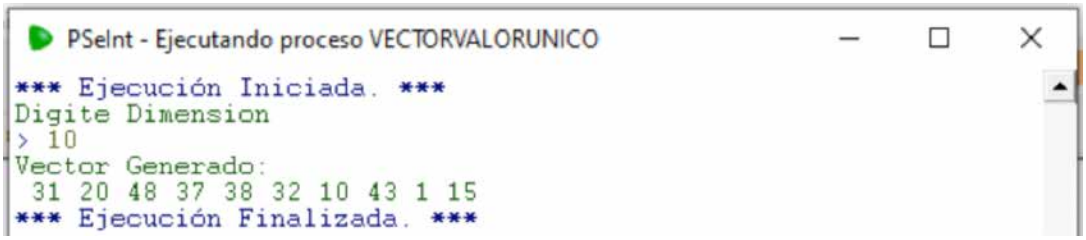



Figura 42. Vector con números únicos

Fuente: captura de pantalla del programa PSeint.

Cuarto ejemplo. Ordenamiento de un vector

```
Función valorunico(B por referencia,max por valor)
Repetir
j=1
k=azar(50)+1
Mientras j<max && B(j)<>k Hacer
j<-j+1
Fin Mientras
Hasta Que j=max
B(j)<-k
```

FinFuncion
Función s <-imprimir (B por referencia,n por valor)
s<-" "
Para i<-1 Hasta n Con Paso 1 Hacer
s<-concatenar(s,ConvertirATexto(B(i)))
s<-concatenar(s," ")
Fin Para
Fin Función
Algoritmo vectorvalorunico
Escribir "Digite Dimension"
Leer m
Dimension C(m)
Para i<-1 Hasta m Con Paso 1 Hacer
valorunico(C,i)
Fin Para
Escribir "Vector generado:"
Escribir imprimir(C,m)
FinAlgoritmo

Figura 43. Ordenamiento de vector
Fuente: captura de pantalla del programa PSeint.

Software LPP

En el Software LPP, los pseudocódigos tienen la siguiente estructura general:

Zona para Declaración de variables
inicio
acción 1;
acción 1;
.
acción n;
fin

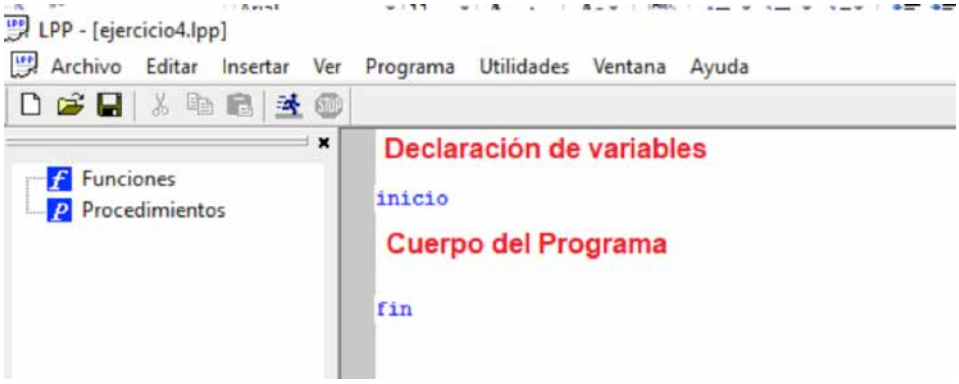


Figura 44. Presentación LPP

Fuente: captura de pantalla del programa Pseint.

Recomendación 1. Se recomienda utilizar el comando "Guardar Como" para guardar el documento con el nombre del programa, por ejemplo, el algoritmo "sumanumeros" se debe guardar como "sumanumeros".

Figura 45. Menú archivo LPP

Fuente: captura de pantalla del programa Pseint.

Recomendación 2. Posicionarse en la línea de código donde se va a colocar un bloque. Después, utilizar la herramienta "Menú Insertar".

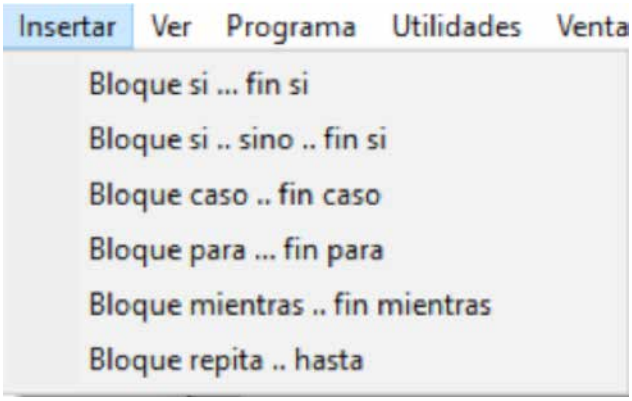


Figura 46. Menú insertar LPP

Fuente: captura de pantalla del programa Pseint.

Recomendación 3. Lo que se puede editar de un bloque es lo que aparece entre /*Condición*/ por ejemplo:

```
Entero edad
inicio
Si /*Condición*/ Entonces
/*Instrucciones*/
Fin Si
Fin
```

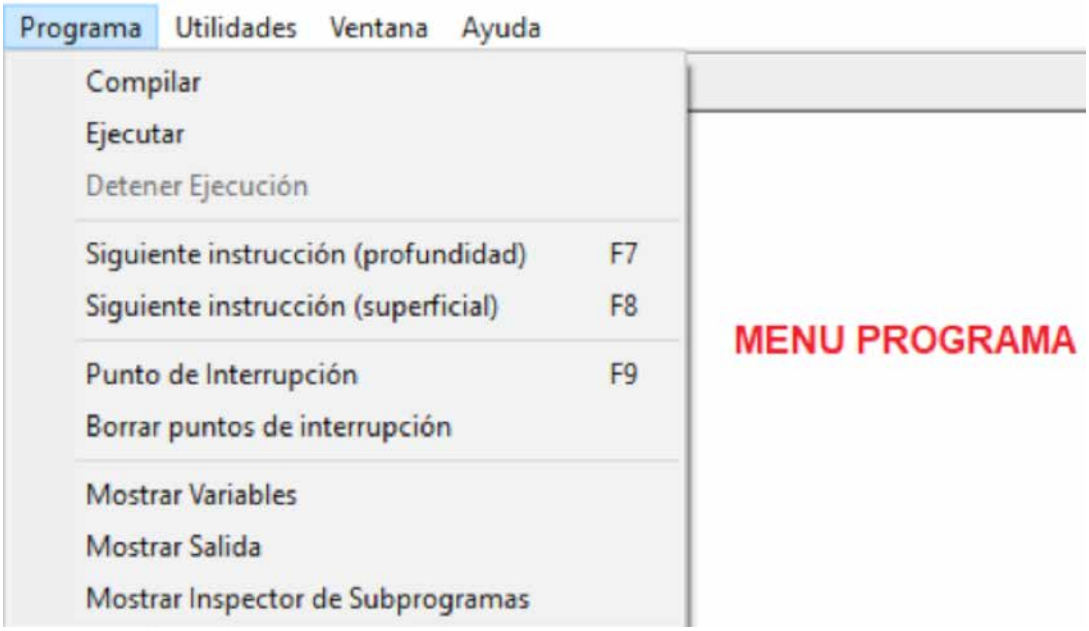


Figura 47. Menú Programa LPP

Fuente: captura de pantalla del programa PSeInt.

Funciones y procedimientos para manejo de texto

- Car (número): retorna el carácter de acuerdo con el parámetro valor numérico ASCII.
- Valor_Ascii (carácter): retorna el código Ascii del carácter del parámetro.
- Longitud (cadena): retorna la cantidad de caracteres de la cadena.

Funciones y procedimientos para aspecto en pantalla

- Llamar Posicionar_Cursor(n1, n2): coloca cursor en el *x* y *y* de la pantalla.

- Llamar Nueva_Linea (): equivalente a un salto de línea.
- Llamar Limpiar_Pantalla(): borra el contenido de la pantalla.
- Llamar Color_Texto(número): colocar un color de texto acorde con número de parámetro.
- Llamar Color_Fondo (número): colocar un color de fondo acorde con número de parámetro.
- Llamar Obtener_Caracter(): pausa hasta que se pulse tecla.

Funciones y procedimientos para uso matemático

- Llamar Inicializar_Aleatorio () y Aleatorio (): la primera inicializa números randómicos (semilla) y la segunda lo genera un número real entre 0 y 1.

Declaracion de Variables

Siempre que se necesita hacer un programa se deben declarar variables para poder almacenar datos.

Los tipos de datos simples soportados son los siguientes:

<números enteros>:- $\int_{1} < digito >$

<Real>:- <números enteros> . <números enteros>
<cadena>:-<letras><caracteres alfanuméricos>
<booleano>:-<verdad><falso>

Ejemplos
Entero n
Real Pi: Pi se declara de tipo real
cadena[25] Nombre: se declara una cadena que guarda 25 caracteres.

Arreglos

- Dimensionamiento

Arreglo[dimensión] de tipo <nombre-variable>, para declarar un vector
Arreglo[filas, columnas] de tipo <nombre-variable>, para declaración de una matriz.
Ejemplo:
Declaración Vector
Arreglo[5] de entero A

La Declaración de una Matriz
Arreglo[5.5] de entero B
Sintaxis Si
Formato 1:
Si /*Condición*/ Entonces
/*Instrucciones*/
Fin Si
Formato 2:
Si /*Condición*/ Entonces
/*Instrucciones*/
Sino
/*Instrucciones*/
Fin SiSi /*Condición*/ Entonces
/*Instrucciones*/
Fin Si

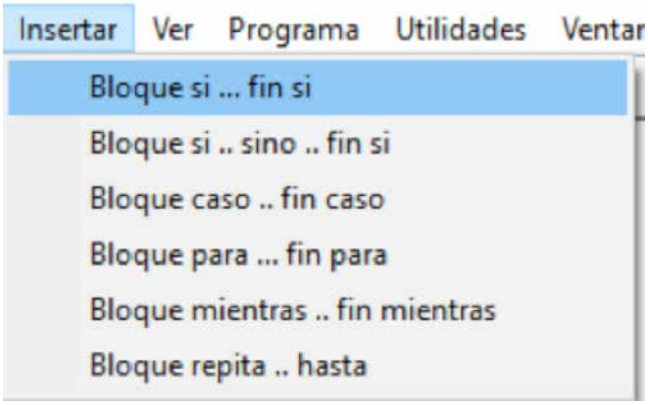


Figura 48. Evaluación condición LPP
Fuente: captura de pantalla del programa LPP.

Programa en lpp

Entero a,b,aux
inicio
a<-Aleatorio()*99+1
Escriba “Valor de a:”,a
Llamar nueva_linea()
b<-Aleatorio()*99+1
Escriba “Valor de b:”,b
Llamar nueva_linea()
Escriba “se realiza intercambio..”
Llamar nueva_linea()
aux<-a // aux variable intermedia

a<-b // primer momento de intercambio
b<-aux //segundo momento de intercambio
Escriba “Valor de a:”,a
Llamar nueva_linea()
Escriba “Valor de b:”,b
fin

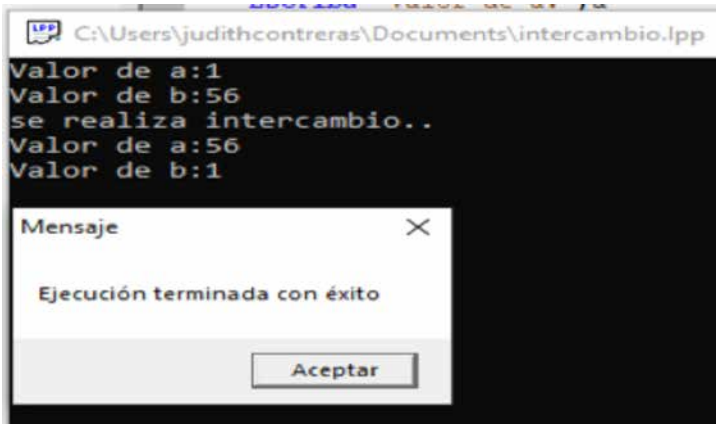


Figura 49. Programa intercambio LPP
Fuente: captura de pantalla del programa LPP.

- Bloque para... fin para

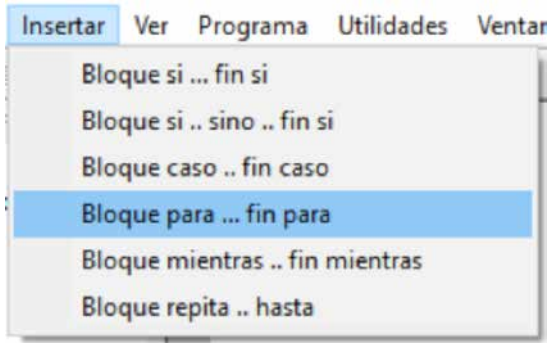


Figura 50. Bloque para... fin para
Fuente: captura de pantalla del programa LPP.

Para /*Variable*/ <- /*Valor Inicial*/ Hasta /*Valor Final*/ Haga
/*Instrucciones*/
Fin Para

- Ejemplos Bloque para... fin para

Entero A,B,i,s
inicio
s<-0
Escriba “ lea A:”

Lea A
Escriba " lea B:"
Lea B
Para i<-1 Hasta A Haga
s<-s+B
Fin Para
Escriba "El resultado de la multiplicación de ",A," por ",B, " es: ",s
fin

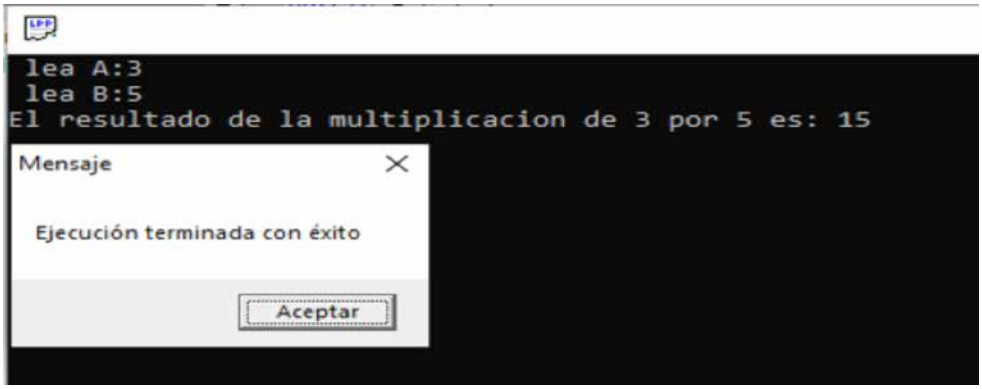


Figura 51. Multiplicación de A × B en LPP
Fuente: captura de pantalla del programa LPP.

Entero fact,m,k
inicio
fact<-1
Escriba " Número de Términos:"
Lea m
fact<-1
Para k<-1 Hasta m Haga
fact<-fact*k
Fin Para
Escriba "factorial de:",m," es:",fact
fin

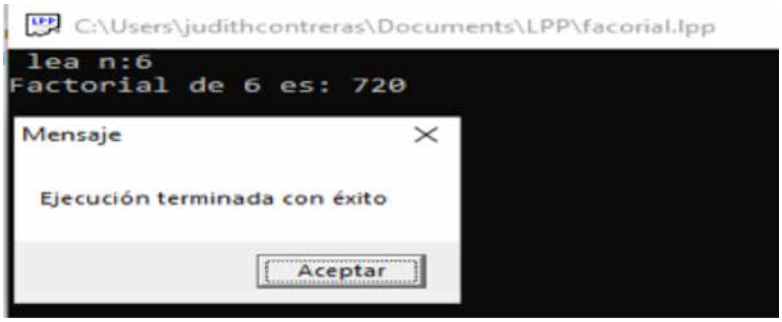


Figura 52. Factorial en LPP
Fuente: captura de pantalla del programa LPP.

```
Entero Nieto, Padre, Abuelo, n, i
inicio
Padre<-0
Abuelo<-1
Escriba " lea término n:"
Lea n
Escriba "Serie de Fibonacci..."
Llamar nueva_linea()
Para k<-1 Hasta n Haga
Nieto<-Padre+Abuelo
Escriba Nieto, " "
Abuelo<-Padre
Padre<-Nieto
Fin Para
Fin
```

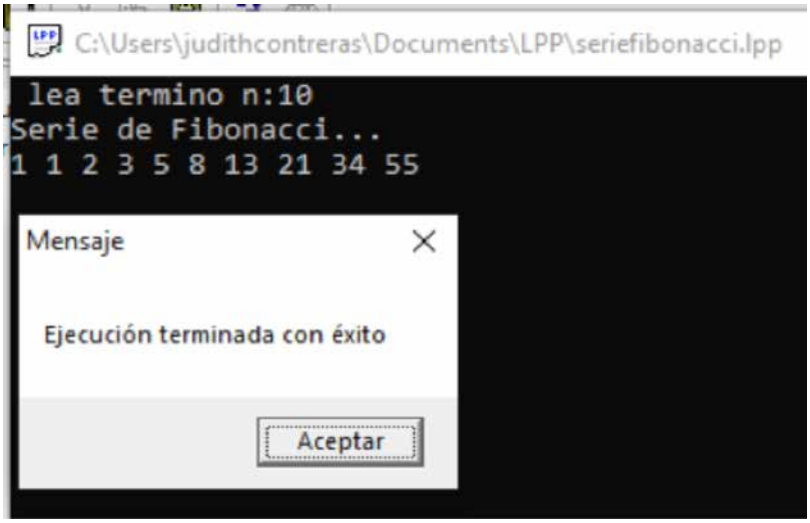


Figura 53. Serie de Fibonacci en LPP

Fuente: captura de pantalla del programa LPP.

```
Arreglo[5] de entero D
Entero m,k
inicio
m<-5
Escriba "Lectura Vector: "
Llamar nueva_linea()
Para k<-1 Hasta m Haga
Escriba "D[",k,"]:"
Lea D[k]
Fin Para
Para k<-1 Hasta m Haga
```

Llamar nueva_linea()
Escriba “D[“,k,”]:”,D[k]
Fin Para
fin

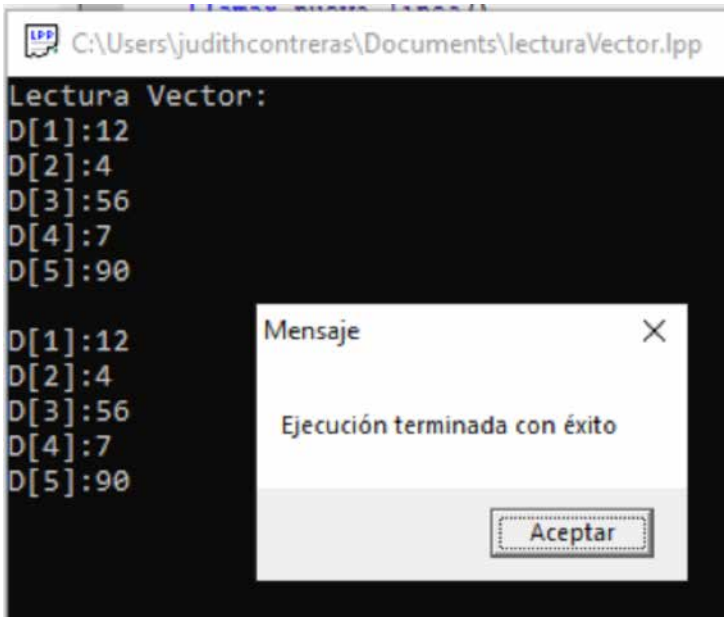


Figura 54. Lectura de un vector en LPP
Fuente: captura de pantalla del programa LPP.

Entero k,base,m,j
inicio
k<-1
Escriba “base: “
Lea base
Llamar nueva_linea()
Escriba “Elevado a la: “
Lea m
Llamar nueva_linea()
Para j<-1 Hasta m Haga
k<-k*base
Fin Para
Escriba “Respuesta:”,k
fin

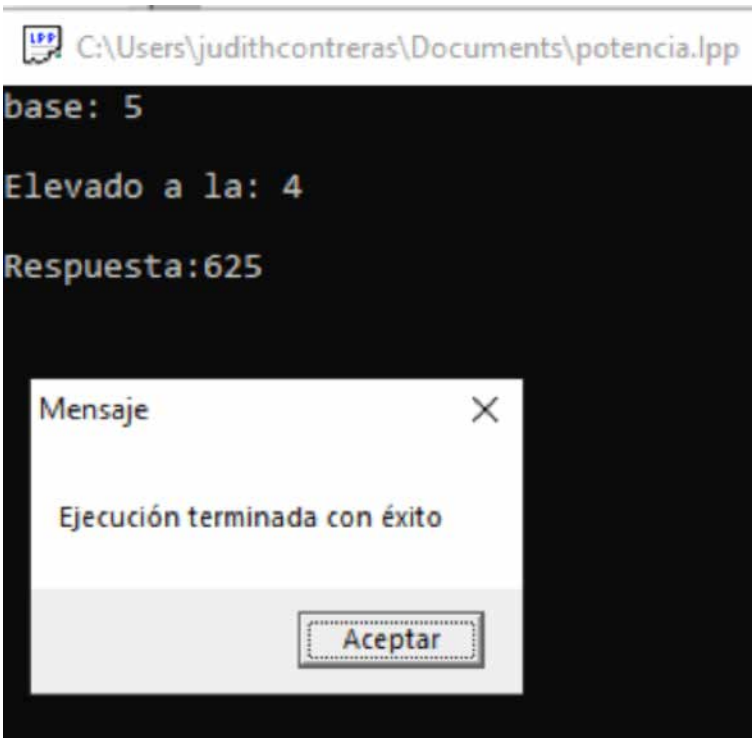


Figura 55. Simulación de potencia en LPP

Fuente: captura de pantalla del programa LPP.


```
Arreglo[6] de entero A
Entero n,i,j,k
inicio
n<-100 000+aleatorio()*900 000+1
k<-n
Llamar Posicionar_Cursor(1.2)
Escriba “Número a Digitalizar: “,n
Para i<-1 Hasta 6 Haga
A[i]<-k mod 10
k<-k/10
Fin Para
Llamar Posicionar_Cursor(1.4)
Escriba “Dígitos de “,n
Para i<-1 Hasta 6 Haga
Llamar Posicionar_Cursor(i*4.5)
Escriba A[i]
Fin Para
Fin
```

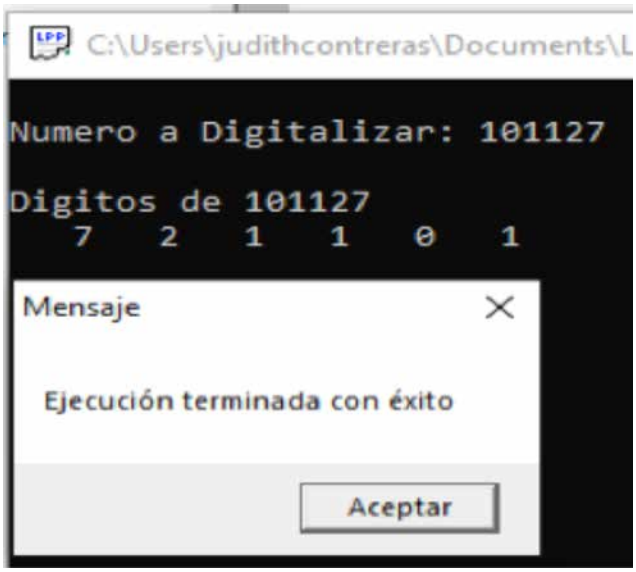


Figura 56. Digitalización de un número en LPP

Fuente: captura de pantalla del programa LPP

```
Arreglo[6.6] de entero D
Entero m,x,z
inicio
m<-6
// Lectura Matriz:
Para x<-1 Hasta m Haga
Para z<-1 Hasta m Haga
D[x,z]<- aleatorio()*20+1
Fin Para
Fin Para
// Escritura Matriz:
Escriba “Matriz: D[“,m,”,”,m,”]:”
Llamar nueva_linea()
Para x<-1 Hasta m Haga
Para z<-1 Hasta m Haga
Llamar Posicionar_Cursor(z*4.2+x)
Escriba D[x,z]
Fin Para
Fin Para
Fin
```

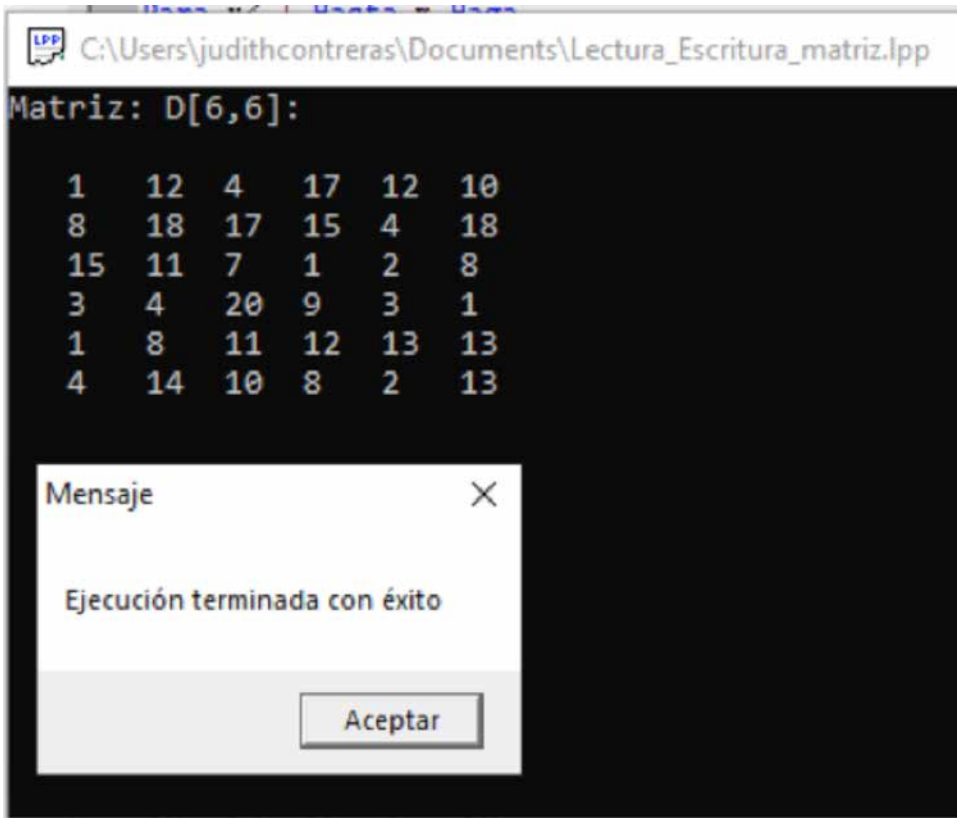


Figura 57. Lectura/Escritura de un vector en LPP

Fuente: captura de pantalla del programa LPP.

```
Arreglo[10] de entero A
Entero n,i,j,k
inicio
n<-10
Para i<-1 Hasta n Haga
A[i]<-aleatorio()*20+1
Fin Para
Llamar Posicionar_Cursor(1.2)
Escriba “Vector Leído: “
Para i<-1 Hasta n Haga
Llamar Posicionar_Cursor(i*4.3)
Escriba A[i]
Fin Para
Llamar Posicionar_Cursor(i*4.3)
Para i<-1 Hasta n-1 Haga
Para j<-i+1 Hasta n Haga
Si A[i]>A[j] Entonces
k<-A[i]
A[i]<-A[j]
```

A[j]<-k
Fin Si
Fin Para
Fin Para
Llamar Posicionar_Cursor(1.4)
Escriba “Vector Ordenado: “
Para i<-1 Hasta n Haga
Llamar Posicionar_Cursor(i*4.5)
Escriba A[i]
Fin Para
Fin

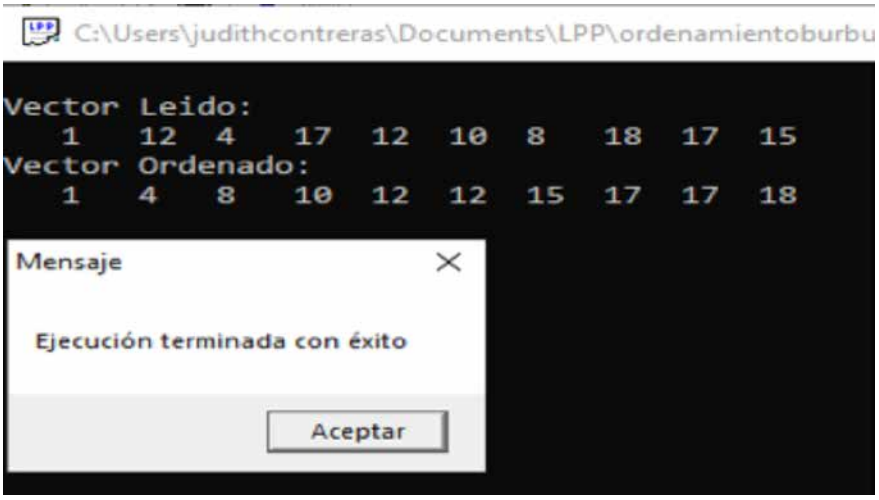


Figura 58. Ordenamiento de un vector en LPP
Fuente: captura de pantalla del programa LPP.

- Bloque mientras...fin mientras

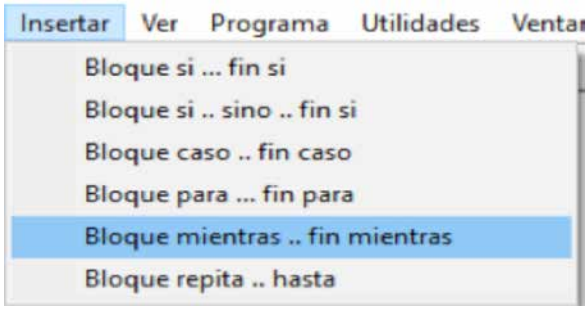


Figura 59. Bloque mientras...fin mientras
Fuente: captura de pantalla del programa LPP.

```
Mientras /*Condición*/ Haga
/*Condiciones*/
Fin Mientras
```

Ejemplos Bloque mientras...fin mientras

```
Arreglo[6] de entero A
Entero n,i,j,x,m
inicio
x<-aleatorio()*1 000 000+1
m<-x
n<-0
Llamar Posicionar_Cursor(1.2)
Escriba “Número a Digitalizar: “,k
Mientras m>0 Haga
n<-n+1
A[n]<-m mod 10
m<-m/10
Fin Mientras
Llamar Posicionar_Cursor(1.4)
Escriba “Dígitos “
Para i<-1 Hasta n Haga
Llamar Posicionar_Cursor(8+(i*4),4)
Escriba A[i]
Fin Para
Fin
```

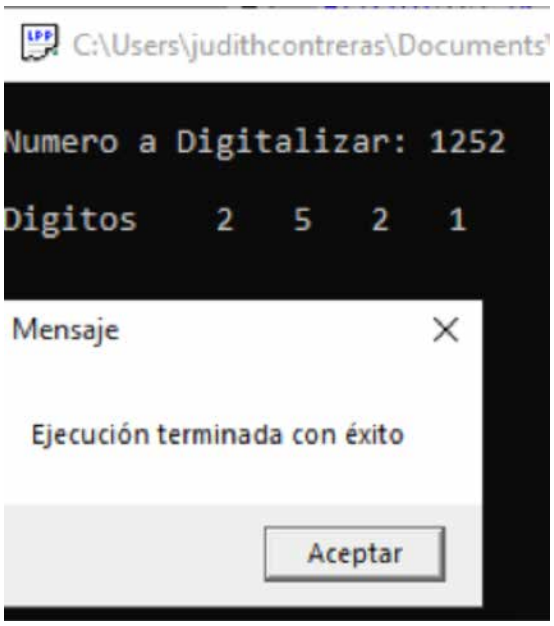



Figura 60. Digitalizar un número en LPP

Fuente: captura de pantalla del programa LPP.

```
Arreglo[6.6] de entero D
Entero m,x,z
inicio
m<-6
// Lectura Matriz:
Para x<-1 Hasta m Haga
Para z<-1 Hasta m Haga
D[x,z]<- aleatorio()*20+1
Fin Para
Fin Para
// Escritura Matriz:
Escriba “Matriz: D[“,m,”,”,m,”]:”
Llamar nueva_linea()
Para x<-1 Hasta m Haga
Para z<-1 Hasta m Haga
Llamar Posicionar_Cursor(z*4.2+x)
Escriba D[x,z]
Fin Para
Fin Para
Fin
```

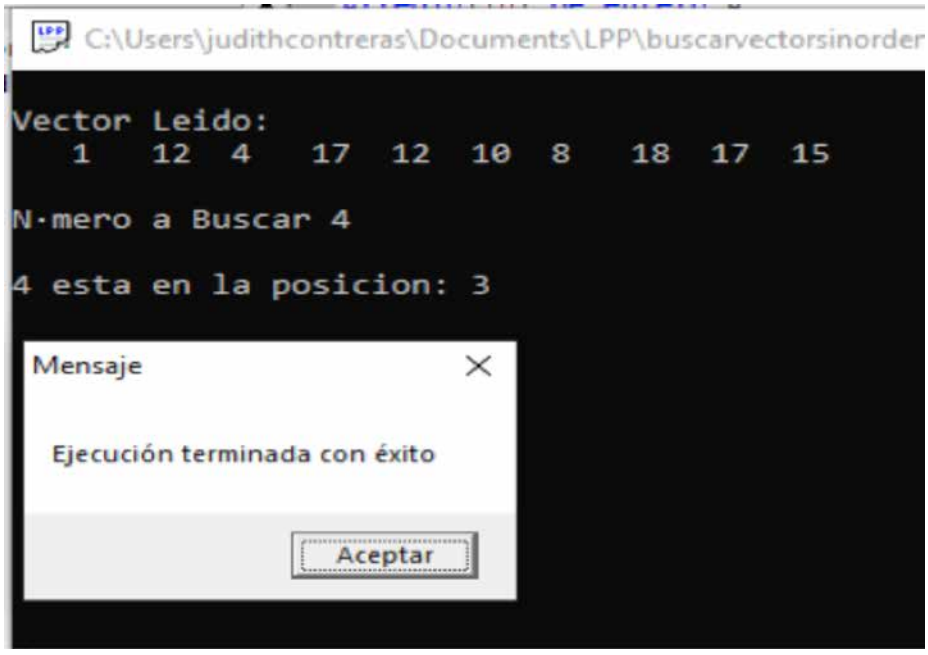


Figura 61. Buscar un número en un vector en LPP

Fuente: captura de pantalla del programa LPP.

- Bloque repita...hasta

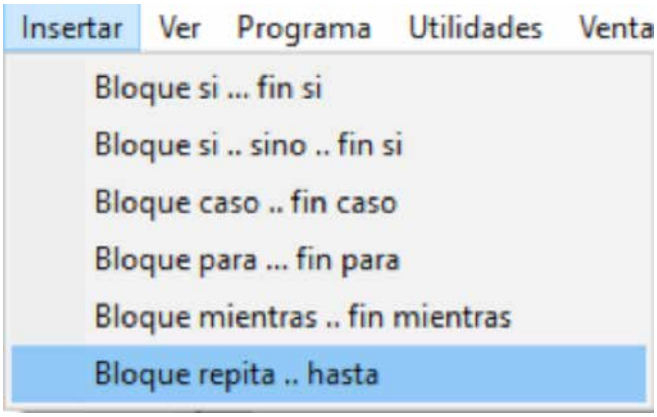


Figura 62. Bloque repita hasta en LPP

Fuente: captura de pantalla del programa LPP.

Repita
/*Instrucciones*/
Hasta /*Condición*/

Ejemplo repita...hasta

Entero nota
inicio
Repita
Escriba “Digite Número a Evaluar: “
Lea nota
Hasta (nota>=1) Y (nota<=10)
fin

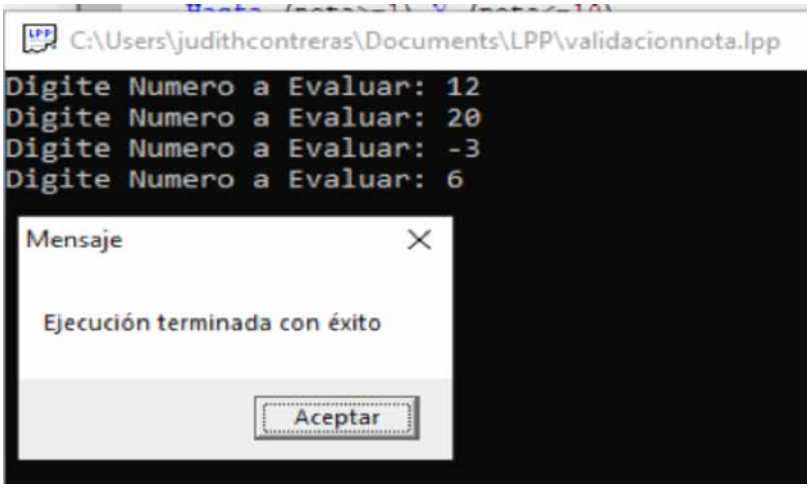


Figura 63. Repita nota hasta

Fuente: captura de pantalla del programa LPP.

Referencias

Arguello, N. (2021). *Investigación PSeint*. Universidad de Guayaquil. Docsity. https://bit.ly/3GV9zdW

Castillo, R. (2021). *Programación en LPP* [PDF]. Rincón Digital. https://bit.ly/3kEv6QV

Cooper, C. D. (2022). *Languages and machines* [PDF]. Coopernotes. https://bit.ly/3H5x1Fu

Cormen, T. H. (2021). *Algorithms Unlocked*. Cambridge. MIT Press. https://bit.ly/3kBqRFA

Hilal, L. (2021). *Razonamientos para la lógica proposicional* [PDF]. Adriana Suarez. https://bit.ly/3JaLlzs

Sobre el autor

Mario Dustano Contreras Castro

Es ingeniero de sistemas por la Universidad Autónoma de Colombia, magíster en Edumática por la Universidad Autónoma de Colombia, especialista en Multimedia Educativa por la Universidad Antonio Nariño (Colombia). Actualmente se desempeña como profesor de tiempo completo en la Universidad Santo Tomas.

Correo electrónico: mariocontreras@usta.edu.co

ORCID: https://orcid.org/0000-0002-2560-6426

Esta obra se editó en Ediciones
USTA. Tipografías de la familia Fira
Sans y Share Tech.
2023

Modular

Esta obra plantea una relación entre la lógica como ciencia formal, que estudia la estructura o formas del pensamiento lógico (analítico, divergente, convergente), y su desarrollo, a partir de Autómatas de Estado Finito con la lógica de programación. Del mismo modo, analiza el desarrollo de la lógica de programación con la tipología de variables como: constantes, estructura de un algoritmo de programación, operaciones de entradas y salidas y ciclos a partir de autómatas. Además, relaciona el algoritmo de programación Software Diagrama de Flujo (DFD) y el Software de algoritmos de programación (PSeint y LPP).

UNIVERSIDAD SANTO TOMÁS
SEDE PRINCIPAL