

UNIVERSIDAD NACIONAL DE EDUCACIÓN

Enrique Guzmán y Valle

*Alma Máter del Magisterio Nacional*

FACULTAD DE CIENCIAS

Escuela Profesional de Matemática e Informática



**MONOGRAFÍA**

## **ALGORÍTMICA**

**Introducción, estructura de algoritmos, conceptos, instrucciones con algoritmos, instrumentos que utilizan, aplicaciones prácticas.**

Examen de Suficiencia Profesional Resolución N° 1059-2018-D-FAC

Presentada por:

**Benites Cirilo, Lolo Roberto**

Para optar al Título Profesional de Licenciado en Educación


Especialidad: Matemática e Informática

Lima, Perú  
2018

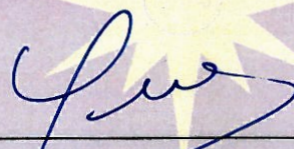
**MONOGRAFÍA****ALGORÍTMICA**

**Introducción, estructura de algoritmos, conceptos, instrucciones con algoritmos, instrumentos que utilizan, aplicaciones prácticas.**

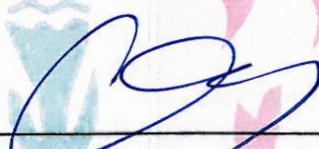
Examen de Suficiencia Profesional Resolución N° 1059-2018-D-FAC

  
\_\_\_\_\_  
**Dr. Huamani Escobar, William Alberto**

Presidente

  
\_\_\_\_\_  
**Dr. Quivio Cuno, Richard Santiago**

Secretario

  
\_\_\_\_\_  
**Dr. Caballero Cifuentes, Lolo José**

Vocal

Línea de investigación: Tecnología y soportes educativos

### Dedicatoria

Dedico con todo mi amor la presente monografía primeramente a Dios, por haberme dado la vida y el día a día, y permitirme alcanzar mis objetivos.

## Índice de contenidos

Portada .....	i
Hoja de firmas de jurado.....	ii
Dedicatoria.....	iii
Índice de contenidos .....	iv
Lista de figuras .....	vi
Introducción .....	vii
<b>Capítulo I. Algorítmica .....</b>	<b>8</b>
1.1 Algorítmica .....	8
1.2 Algoritmos .....	8
1.3 Características y escritura inicial del algoritmo.....	9
1.4 Representación de algoritmos .....	10
1.5 Conversión de algoritmos al computador .....	14
1.6 Metodología para la solución de problemas por medio de computadora .....	15
1.7 Software desarrollador de algoritmos .....	16
<b>Capítulo II. Estructuras de algoritmos y programación.....</b>	<b>19</b>
2.1 Programación convencional.....	20
2.2 Programación modular.....	21
2.3 Programación estructurada .....	22
2.4 Estructuras básicas.....	23
<b>Capítulo III. Instrucciones con algoritmos.....</b>	<b>24</b>
3.1 Datos .....	24
3.2 Constantes y variables .....	26
3.3 Expresiones.....	29
3.3.1 Expresiones aritméticas .....	29
3.3.2 Expresiones lógicas (booleanas) .....	30
3.4 Operadores de relación.....	30
3.5 operadores lógicos.....	31
3.6 Funciones internas .....	32

3.7 Entrada y salida de información .....	33
Aplicación didáctica .....	34
Síntesis .....	45
Apreciación crítica y sugerencias .....	47
Referencias.....	49

## Lista de figuras

Figura 1. Representación de algoritmos .....	10
Figura 2. Reglas básicas para la construcción de diagrama de flujo .....	12
Figura 3. Diagramas N-S.....	13
Figura 4. Ventana de edición.....	16
Figura 5. Barra de herramienta.....	16
Figura 6. PSeInt.....	17
Figura 7. NetBeansIDE6.5.1 .....	18
Figura 8. Relación histórica de las principales metodologías .....	20
Figura 9. Tipos de datos .....	26
Figura 10. Expresiones aritméticas .....	29
Figura 11. Operadores de relación .....	30
Figura 12. Cuatro tipos de datos estándar: enteros, real, lógico, carácter .....	31
Figura 13. Operadores lógicos .....	31
Figura 14. Funciones internas .....	32
Figura 15. Reales o enteros.....	33

## Introducción

El término actual de elección para un procedimiento de resolución de problemas, algoritmo, se usa comúnmente hoy en día para el conjunto de reglas que una máquina (y especialmente una computadora) sigue para lograr un objetivo particular. Sin embargo, no siempre se aplica a la actividad mediada por computadora. El término puede usarse con la misma precisión de los pasos seguidos para hacer una pizza como para el análisis de datos por computadora.

El algoritmo a menudo se combina con palabras que especifican la actividad para la cual se ha diseñado un conjunto de reglas. Un algoritmo de búsqueda, por ejemplo, es un procedimiento que determina qué tipo de información se recupera de una gran masa de datos. Un algoritmo de encriptación es un conjunto de reglas mediante las cuales se codifica información o mensajes para que personas no autorizadas no puedan leerlos.

Aunque atestiguado por primera vez a principios del siglo XX (y, hasta hace poco, utilizado estrictamente como un término de matemáticas e informática), el algoritmo tiene una historia sorprendentemente profunda. Se formó a partir del algoritmo "el sistema de números arábigos", una palabra que se remonta al inglés medio y que en última instancia se deriva del nombre de un matemático persa del siglo IX, abu-Ja'far Mohammed ibn-Mūsa al-Khwarizmi, que realizó un trabajo importante en los campos de álgebra y sistemas numéricos.

## **Capítulo I**

### **Algorítmica**

#### **1.1 Algorítmica**

Llamaremos algorítmica a la ciencia que estudia a los algoritmos, estos se juzgan más comúnmente por su eficiencia y la cantidad de recursos informáticos que requieren para completar su tarea. Una forma común de evaluar un algoritmo es observar su complejidad temporal. Esto muestra cómo crece el tiempo de ejecución del algoritmo a medida que aumenta el tamaño de entrada. Dado que los algoritmos actuales deben funcionar con grandes entradas de datos, es esencial que nuestros algoritmos tengan un tiempo de ejecución razonablemente rápido.

#### **1.2 Algoritmo**

En informática, un algoritmo nos permite resolver una clase de problemas. Pueden realizar cálculos, procesamiento de datos y tareas de razonamiento automatizado.

Hay ciertos requisitos que debe cumplir un algoritmo:

Definitividad: cada paso del proceso se establece con precisión.

Computabilidad efectiva: cada paso en el proceso puede ser realizado por una computadora.



Finita: El programa eventualmente terminará exitosamente.

Algunos tipos comunes de algoritmos incluyen algoritmos de clasificación, algoritmos de búsqueda y algoritmos de compresión. Las clases de algoritmos incluyen Gráfico, Programación dinámica, Clasificación, Búsqueda, Cadenas, Matemáticas, Geometría computacional, Optimización y Varios. Aunque técnicamente no es una clase de algoritmos, las estructuras de datos a menudo se agrupan con ellos.

### **1.3 Características y escritura inicial del algoritmo**

- Las características clave de los algoritmos son:
- Fácil de entender: los algoritmos están ahí para ayudar a los humanos a comprender la solución.
- Corrección: esta es una característica imprescindible para todos los algoritmos.
- Precisión: los pasos se establecen (definen) con precisión.
- Finitud: el algoritmo se detiene después de un número finito de pasos.
- Generalidad: el algoritmo se aplica a toda la distribución posible de entradas como se indica.

#### **Escritura inicial del algoritmo**

Un algoritmo es un procedimiento que una computadora o un ser humano sigue para resolver un problema. La división larga es un algoritmo de muestra que muchas personas aprenden a hacer en la escuela. El algoritmo euclidiano, usado para encontrar el máximo divisor común de dos números, es otro ejemplo común.

Un algoritmo informático finalmente se escribe en un lenguaje de programación que la computadora puede entender, pero cuando se desarrolla el algoritmo, los programadores y los científicos informáticos a menudo lo escriben primero de manera

informal como prosa y luego más formalmente en un formato genérico llamado pseudocódigo.

#### 1.4 Representación de algoritmos

**Flujogramas:** Representación gráfica de la realización de un algoritmo.

Los símbolos son los siguientes:


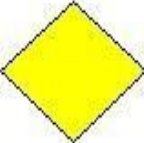


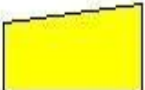
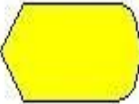
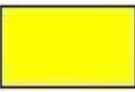

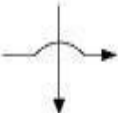

	<b>Inicio/Final</b> Se utiliza para indicar el inicio y el final de un diagrama; de Inicio sólo puede salir una línea de flujo y al final sólo debe llegar una línea		<b>Decisión</b> Indica la comparación de dos datos y dependiendo del resultado lógico (falso o verdadero) se toma la decisión de seguir un camino del diagrama u otro
	<b>Entrada/Salida</b> Entrada/Salida de datos por cualquier dispositivo (scanner, lector de código de barras, micrófono, parlantes, etc.)		<b>Impresora/Documento.</b> Indica la presentación de uno o varios resultados en forma impresa
	<b>Entrada por teclado.</b> Entrada de datos por teclado. Indica que el computador debe esperar a que el usuario teclee un dato que se guardará en una variable o constante		<b>Pantalla</b> Instrucción de presentación de mensajes o resultados en pantalla
	<b>Acción/Proceso</b> Indica una acción o instrucción general que debe realizarse (operaciones aritméticas, asignaciones, etc.)		<b>Conector Interno</b> Indica el enlace de dos partes de un diagrama dentro de la misma página
	<b>Flujo/Flecas de Dirección</b> Indica el seguimiento lógico del diagrama. También indica el sentido de ejecución de las operaciones		<b>Conector Externo</b> Indica el enlace de dos partes de un diagrama en páginas diferentes

Figura 1. Representación de algoritmos. Fuente: Recuperado de <http://elbibliote.com/resources/Temas/>.

**Reglas básicas en la elaboración de un diagrama de flujo:**

- Debe tener un inicio y fin.
- Las conexiones son mediante rectas.
- Los enlaces deben ser precisas.
- Se deben dibujar todos los símbolos visualmente de arriba hacia abajo (diseño top-down) y de izquierda a derecha.
- Debe ser claro y equilibrado.
- Si el diagrama abarca más de una hoja es conveniente enumerarlo e identificar de donde viene y a donde se dirige.

Por ejemplo hallar el producto de varios números positivos introducidos por teclado y el proceso termina cuando se ingrese un número negativo.

1. Iniciar
2. Leer el primer número.
3. Interrogar si es negativo o positivo.
4. Si es negativo se sale y se presenta el producto.
5. Si es positivo, lo multiplicamos y luego leemos un nuevo número, y vuelve al paso

Ejemplo: Calcular salario neto de acuerdo al número de horas trabajadas, precio de la hora de trabajo y descuentos fijos al salario bruto (20 %).

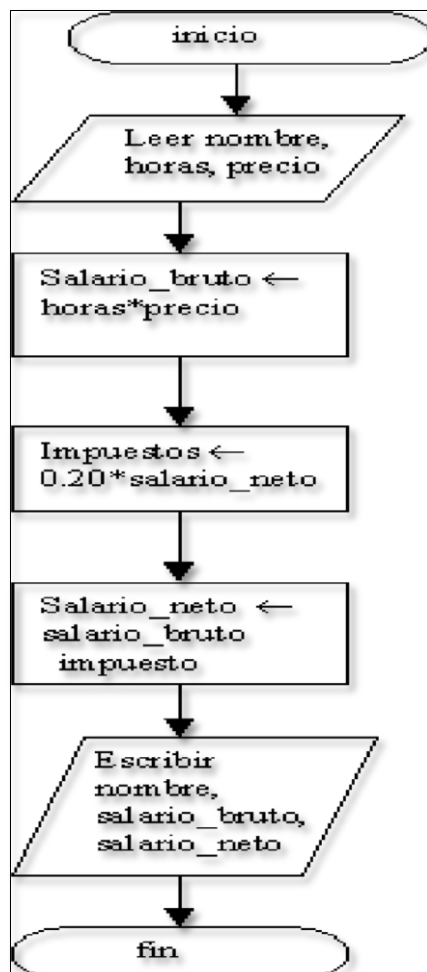


Figura 2. Reglas básicas para la construcción de diagrama de flujo. Fuente: Recuperado de <http://correo.uan.edu.mx/~iavalos/introprog>.

Realizare un diagrama de flujo que permita mostrar en pantalla un mensaje de mayoría o minoría de edad según sea el caso para una persona específica.

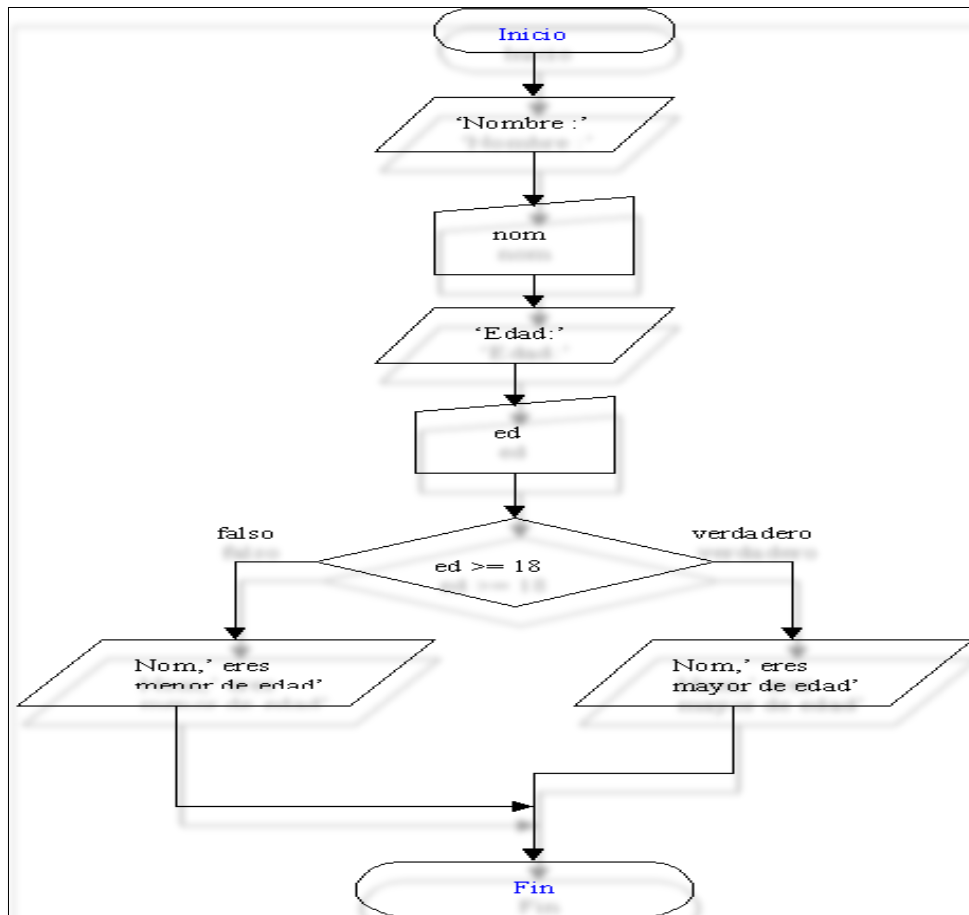


Figura 3. Diagramas N-S. Fuente Recuperado de <https://marcoc76.github.io/segundoperiodo/2019/09/25/algoritmos>.

### Seudocódigo:

El pseudocódigo es una forma informal de descripción de la programación que no requiere ninguna sintaxis estricta del lenguaje de programación o consideraciones tecnológicas subyacentes. Se utiliza para crear un esquema o un borrador de un programa. El pseudocódigo resume el flujo de un programa, pero excluye los detalles subyacentes. Los diseñadores de sistemas escriben pseudocódigo para garantizar que los programadores entiendan los requisitos de un proyecto de software y alineen el código en consecuencia.

**Descripción:** el pseudocódigo no es un lenguaje de programación real. Por lo tanto, no se puede compilar en un programa ejecutable. Utiliza términos cortos o sintaxis simples en inglés para escribir código para programas antes de que se convierta en un lenguaje de

programación específico. Esto se hace para identificar errores de flujo de nivel superior y comprender los flujos de datos de programación que el programa final va a utilizar. Esto definitivamente ayuda a ahorrar tiempo durante la programación real ya que los errores conceptuales ya se han corregido. En primer lugar, se recopila la descripción y la funcionalidad del programa y luego se utiliza el pseudocódigo para crear declaraciones para lograr los resultados requeridos para un programa. El equipo o programadores del diseñador inspecciona y verifica el pseudocódigo detallado para que coincida con las especificaciones de diseño. Detectar errores o un flujo de programa incorrecto en la etapa de pseudocódigo es beneficioso para el desarrollo, ya que es menos costoso que detectarlos más tarde. Una vez que el equipo acepta el pseudocódigo, se reescribe utilizando el vocabulario y la sintaxis de un lenguaje de programación. El propósito de usar pseudocódigo es un principio clave eficiente de un algoritmo. Se utiliza en la planificación de un algoritmo con el bosquejo de la estructura del programa antes de que tenga lugar la codificación real.

### **Ventajas del pseudocódigo**

- Los programadores entienden el pseudocódigo de todo tipo.
- permite que el programador se concentre solo en la parte del algoritmo del desarrollo del código.
- No se puede compilar en un programa ejecutable. Ejemplo, código Java: `if (i <10) {i ++; }` pseudocódigo: si i es menor que 10, incremente i en 1.

### **1.5 Conversión de algoritmos al computador**

En ciencias de la computación, la conversión de tipo o la conversión de texto se refieren al cambio de una entidad de un tipo de datos a otro. Hay dos tipos de conversión: implícita y

explícita. El término para la conversión de tipo implícito es coerción. La conversión explícita de tipos de alguna manera específica se conoce como conversión. La conversión explícita de tipos también se puede lograr con rutinas de conversión definidas por separado, como un constructor de objetos sobrecargados.

### **Conversión de tipo implícito**

La conversión de tipo implícita, también conocida como coerción, es una conversión de tipo automática realizada por el compilador. Algunos idiomas permiten, o incluso requieren que los compiladores proporcionen coerción.

En una expresión de tipo mixto, un subtipo  $s$  se convertirá en un super tipo  $t$  o algunos subtipos  $s_1, s_2, \dots$  se convertirán en un supertipo  $t$  (quizás ninguno de los  $s_i$  es del tipo  $t$ ) en tiempo de ejecución para que el programa correrá correctamente

## **1.6 Metodología para la solución de problemas por medio de computadora**

Un efecto secundario de haber dado grandes saltos en la informática en las últimas décadas, es la sobreabundancia resultante en herramientas de software creadas para resolver los diversos problemas. La resolución de problemas con las computadoras, en consecuencia, se ha vuelto más exigente; En lugar de centrarse en el problema al conceptualizar estrategias para resolverlos, los usuarios se desvían al buscar aún más herramientas de programación (según esté disponible).

El proceso de resolución de problemas basado en computadora es un trabajo destinado a ofrecer un tratamiento sistemático a la teoría y la práctica de diseñar, implementar y usar herramientas de software durante el proceso de resolución de problemas. Este método se obtiene permitiendo que los sistemas informáticos sean más intuitivos con la lógica humana que con la lógica de la máquina. En lugar de software dedicado a expertos en informática, el autor aboga por un enfoque dedicado a los usuarios

de computadoras en general. Este enfoque no requiere que los usuarios tengan una educación informática avanzada, aunque aboga por una educación más profunda del usuario de la computadora en su lógica de dominio del problema.

## 1.7 Software desarrollador de algoritmos

### DFD

DFD es un programa con herramientas gráficas que ayuda a diseñar algoritmos expresados en diagramas de flujo (DF).

### Inicio de DFD

La pantalla de inicio es la siguiente

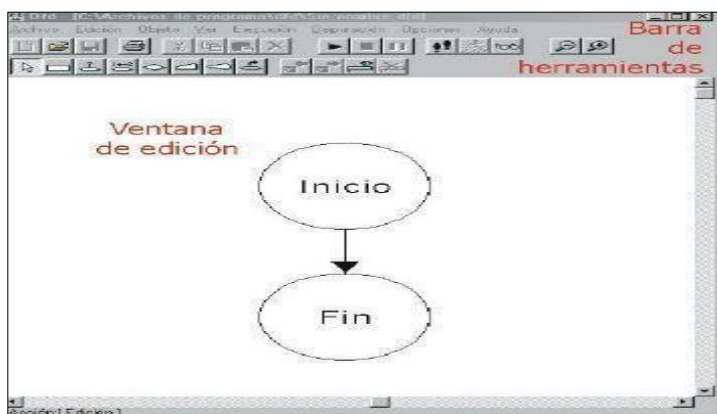


Figura 4. Ventana de edición. Fuente: Autoría propia.

Donde nos fijaremos en la barra de herramientas:



Figura 5. Barra de herramienta. Fuente: Recuperado de [http://www.cca.org.mx/cca/cursos/hbi-webtec/modulos/modulo6/aula/power\\_barras](http://www.cca.org.mx/cca/cursos/hbi-webtec/modulos/modulo6/aula/power_barras).



## PSeInt

PSeInt es una herramienta para aprender programación lógica orientada a estudiantes sin experiencia en programación de computadoras. El uso de un pseudo-lenguaje, intuitivo, puede comenzar a comprender los conceptos y principios básicos de un algoritmo de computadora, también permite exportar a C ++, crear editar y exportar a un diagrama de flujo, ejecutar paso a paso y CODIFICAR y EJECUTAR en TIEMPO REAL.



Figura 6. PSeInt. Fuente: Recuperado de [http://www.cca.org.mx/cca/cursos/hbi-webtec/modulos/modulo6/aula/power\\_barras](http://www.cca.org.mx/cca/cursos/hbi-webtec/modulos/modulo6/aula/power_barras).

## Javanetbeans

NetBeans es un entorno de desarrollo integrado (IDE) basado en Java. El término también se refiere al marco de la plataforma de aplicación subyacente del IDE.

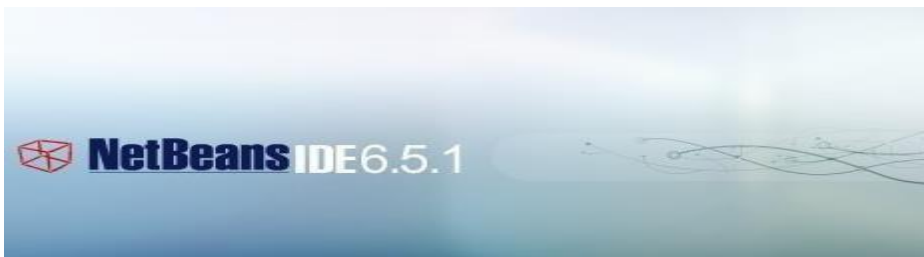
El IDE está diseñado para limitar los errores de codificación y facilitar la corrección de errores con herramientas como NetBeans FindBugs para localizar y corregir problemas comunes de codificación de Java y Debugger para administrar código complejo con vigilancia de campo, puntos de interrupción y monitoreo de ejecución. Aunque NetBeans IDE está diseñado específicamente para desarrolladores de Java, también es compatible con C / C ++, PHP, Groovy y HTML5, además de Java, JavaScript y JavaFX.

Las herramientas y capacidades de NetBeans IDE incluyen un editor de texto rico en funciones con herramientas de refactorización y plantillas de código, vistas de alto nivel y granulares de aplicaciones, un diseño GUI de arrastrar y soltar, y versiones usando

integración inmediata con herramientas tales como como Git. NetBeans IDE puede ejecutarse en cualquier sistema operativo que admita una JVM compatible, incluidos Linux, Windows y OS X.

La plataforma subyacente NetBeans admite la creación de nuevas aplicaciones y un mayor desarrollo de las aplicaciones existentes utilizando componentes de software modular. Como una aplicación que se ejecuta en la plataforma NetBeans, el IDE de NetBeans es extensible y puede ampliarse para admitir nuevos idiomas.

El IDE y la Plataforma fueron convertidos a código abierto por Sun Microsystems en 2000. Oracle continúa patrocinando el proyecto NetBeans desde que adquirió Sun en 2010. .



*Figura 7.* NetBeansIDE6.5.1. Fuente: Recuperado de <https://jmguimera.blogspot.com/2016/09/instalar-netbeans-en-ubuntu>.

## **Capítulo II**

### **Estructuras de Algoritmos y Programación**

Joyanes (2003) “Una metodología de programación no es más que un tipo de técnica para resolver algunos requisitos dados usando lenguajes de programación” (p.130).

Por ejemplo, quiero agregar el número 10 a sí mismo.

Entonces, el ejemplo anterior es un ejemplo de un problema porque es muy difícil calcular este tipo de cálculos manualmente una y otra vez, por lo que para resolver el requisito dado anteriormente mediante programación con una solución permanente, cada lenguaje de programación tiene su propia metodología y estilo de codificación para Resuelve el problema dado.

#### **Relación histórica de las principales metodologías**

Algunos lenguajes de programación emplean varios paradigmas, que, en este caso, se llama multi-paradigma. Dos de los paradigmas de programación más populares incluye procedimental o estructurada y Programación Orientada a Objetos.

Estos dos ejecutan los lenguajes más potentes y populares que conocemos, incluidos, entre otros, Java, C, Python y C++.

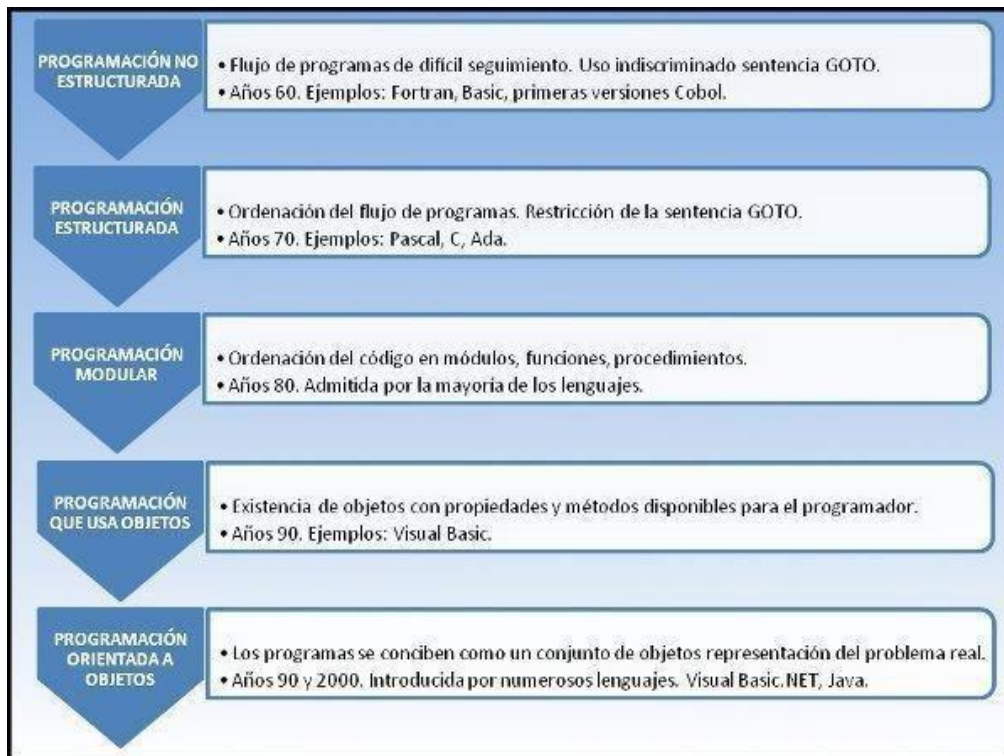


Figura 8. Relación histórica de las principales metodologías. Fuente: Autoría propia.

Las principales metodologías de programación son:

- Programación Convencional (Sin Metodología).
- Programación Modular
- Programación Estructurada.
- Programación Orientada a Objetos

## 2.1 Programación convencional

Un lenguaje de programación en el que toda la lógica del programa se escribe como un solo bloque continuo (ininterrumpido o ininterrumpido) se denomina programación no estructurada.

Los siguientes son los puntos clave de los lenguajes de programación no estructurados:

Es un tipo de técnica de resolución de problemas, en la que resolvemos el problema en términos de mucho código. Si los requisitos aumentan, entonces el código también aumenta.

- No hay reutilización del código existente.
- Encontrar un error en un programa es muy difícil.
- La sintaxis y la estructura son muy difíciles de entender y recordar.
- La modificación es muy difícil.
- Los lenguajes no estructurados solo permiten tipos de datos básicos, como números, cadenas y matrices.
- Las declaraciones son generalmente una en cada línea.

Por ejemplo:

JOSS, FOCAL, MUMPS, TELCOMP, COBOL, FORTRAN, etc.

## **2.2 Programación modular**

La programación modular es una técnica de diseño de software que enfatiza la separación de la funcionalidad de un programa en módulos independientes e intercambiables, de modo que cada uno contiene todo lo necesario para ejecutar solo un aspecto de la funcionalidad deseada.

### **Concepto de modularización**

Uno de los conceptos más importantes de la programación es la capacidad de agrupar algunas líneas de código en una unidad que se puede incluir en nuestro programa. La redacción original para esto fue un subprograma. Otros nombres incluyen: macro, subrutina, procedimiento, módulo y función. Vamos a utilizar el término función para eso es lo que se llaman en la mayoría de los lenguajes de programación predominantes de hoy.

Las funciones son importantes porque nos permiten tomar programas grandes y complicados y dividirlos en piezas manejables más pequeñas. Debido a que la función es una parte más pequeña del programa general, podemos concentrarnos en lo que queremos que haga y probarlo para asegurarnos de que funcione correctamente. En general, las funciones se dividen en dos categorías:

Control del programa: funciones que se utilizan simplemente para subdividir y controlar el programa. Estas funciones son exclusivas del programa que se está escribiendo. Otros programas pueden usar funciones similares, tal vez incluso funciones con el mismo nombre, pero el contenido de las funciones es casi siempre muy diferente.

Tarea específica: funciones diseñadas para ser utilizadas con varios programas. Estas funciones realizan una tarea específica y, por lo tanto, se pueden usar en muchos programas diferentes porque los otros programas también necesitan realizar la tarea específica. Las funciones de tareas específicas a veces se denominan bloques de construcción. Debido a que ya están codificados y probados, podemos usarlos con confianza para escribir un programa grande de manera más eficiente.

El programa principal debe establecer la existencia de funciones utilizadas en ese programa. Dependiendo del lenguaje de programación, hay una forma formal de:

Definir una función (su definición o el código que ejecutará)

Llamar a una función  
Declarar una función (un prototipo es una declaración a un compilador)

### **2.3 Programación estructurada**

Un lenguaje de programación en el que se escribe toda la lógica del programa dividiéndolo en unidades o módulos más pequeños se denomina lenguaje de programación estructurado.

- Los siguientes son los puntos clave de los lenguajes de programación estructurados:

- Encontrar un error en un programa es fácil.
- La sintaxis y la estructura son muy simples de entender y recordar.
- La modificación es fácil.
- No hay reutilización del código existente tanto como se esperaba.

Por ejemplo, C y así sucesivamente.

## 2.4 Estructuras básicas

Siguiendo el teorema del programa estructurado, todos los programas se consideran compuestos por estructuras de control:

"Secuencia"; sentencias ordenadas o subrutinas ejecutadas en secuencia.

"Selección"; se ejecuta una o varias declaraciones dependiendo del estado del programa. Esto generalmente se expresa con palabras clave como `if..then..else..endif`.

"Iteración"; se ejecuta una declaración o bloque hasta que el programa alcanza un cierto estado, o se han aplicado operaciones a cada elemento de una colección. Esto generalmente se expresa con palabras clave como `while`, `repeat`, `for` o `do..hasta`. A menudo se recomienda que cada bucle solo tenga un punto de entrada (y en la programación estructural original, también solo un punto de salida, y algunos idiomas lo imponen).

Recursión; una declaración se ejecuta al llamarse repetidamente hasta que se cumplan las condiciones de terminación. Si bien es similar en la práctica a los bucles iterativos, los bucles recursivos pueden ser más eficientes computacionalmente y se implementan de manera diferente como una pila en cascada.

## **Capítulo III**

### **Instrucciones con Algoritmos**

#### **3.1 Datos**

##### Definición de datos

La definición de datos define un dato particular con las siguientes características.

- Atómica: la definición debe definir un concepto único.
- Rastreable: la definición debe poder asignarse a algún elemento de datos.
- Preciso: la definición no debe ser ambigua.
- Claro y conciso: la definición debe ser comprensible.
- Objeto de datos
- Objeto de datos representa un objeto que tiene datos.

##### **Tipo de datos**

El tipo de datos es una forma de clasificar varios tipos de datos, como enteros, cadenas, etc., que determina los valores que se pueden usar con el tipo de datos correspondiente, el tipo de operaciones que se pueden realizar en el tipo de datos correspondiente. Hay dos tipos de datos:

- Tipo de datos incorporado
- Tipo de datos derivados



### **Tipo de datos incorporado**

Los tipos de datos para los que un idioma tiene soporte incorporado se conocen como tipos de datos incorporados. Por ejemplo, la mayoría de los idiomas proporcionan los siguientes tipos de datos integrados.

- Enteros
- Booleano (verdadero, falso)
- Flotante (números decimales)
- Carácter y cadenas

### **Tipo de datos derivados**

Los tipos de datos que son independientes de la implementación, ya que se pueden implementar de una u otra forma, se conocen como tipos de datos derivados. Estos tipos de datos normalmente se crean mediante la combinación de tipos de datos primarios o integrados y las operaciones asociadas en ellos.

Por ejemplo

- Lista
- Formación
- Apilar
- Cola
- Operaciones básicas

Los datos en las estructuras de datos son procesados por ciertas operaciones. La estructura de datos particular elegida depende en gran medida de la frecuencia de la operación que debe realizarse en la estructura de datos.

- Atravesar
- buscando
- Inserción
- Supresión
- Clasificación
- Fusionando

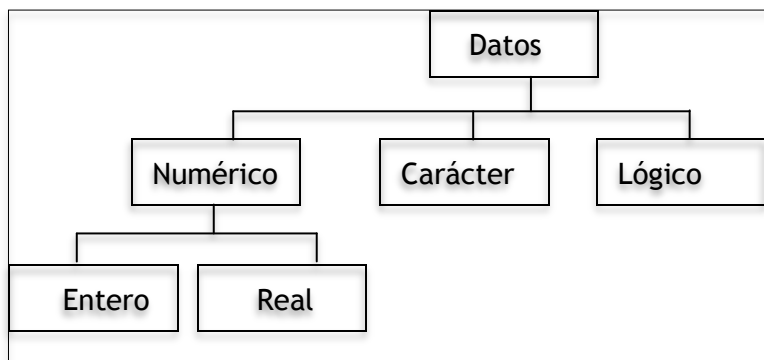


Figura 9. Tipos de datos. Fuente Autoría propia.

### 3.2 Constantes y variables

#### Variables

Una variable es algo que puede cambiar de valor. Una variable podría ser el número de palabras en diferentes páginas de este folleto, la temperatura del aire cada día o las calificaciones de los exámenes que se otorgan a una clase de niños en edad escolar.

Una variable podría compararse con una caja de almacenamiento cuyo contenido a menudo puede cambiar. El cuadro o variable debe tener un nombre para distinguirlo de los demás. De acuerdo con las reglas, el nombre de la variable debe comenzar con una letra y puede ser seguido por hasta cinco caracteres (solo letras o números).

Las variables son de diferentes tipos. Usted especifica el tipo y el nombre de cada variable que piensa usar en la parte superior de su programa, después de la instrucción

PROGRAM y antes de cualquier otra línea ejecutable. Las líneas comentadas no son ejecutables, por lo que pueden aparecer en cualquier parte del programa.

Si se omiten las declaraciones de variables, el compilador hará ciertas suposiciones, por ejemplo, que cualquier variable que comience con las letras I, J, K, L, M, N es INTEGER. La falta de especificación a menudo conduce a errores de programa y se recomienda encarecidamente que siempre se declaren los tipos de variables. Los datos numéricos se pueden separar en números enteros y reales.

### **Enteros**

Los enteros son números enteros, aquellos sin punto decimal (por ejemplo, 7, 4563, 99) y se almacenan en variables enteras.

La forma general de la declaración de variables enteras es:

INTEGER name1, name2

Ejemplo:

ENTERO ENTERO, PROMEDIO, SUMA

### **Reales**

Los reales son números fraccionarios, aquellos que incluyen un punto decimal, (por ejemplo, 0.2, 653.46, 1.0) y se almacenan en variables reales. Los números reales no se pueden almacenar con precisión. La precisión de una variable real depende de la computadora.

La forma general de la declaración de una variable real es:

Nombre real, nombre2

Ejemplo

FRACT REAL, MEDIO, STDDEV

## Caracteres

Las variables de caracteres contienen uno o más caracteres (por ejemplo, G u OXFORD).

Las formas generales son:

CARÁCTER name1, name2 donde name1 y name2 son 1 carácter cada uno.

CARÁCTER \* n name1, name2 donde name1 y name2 son n caracteres de longitud cada uno.

CARÁCTER name1 \* n1, name2 \* n2 donde name1 es de longitud n1 y name2 es de longitud n2.

## Constantes

Las constantes son cantidades cuyos valores no cambian durante la ejecución del programa. En FORTRAN pueden ser de tipo numérico o de caracteres.

## Precisión doble

Los valores de datos reales se denominan comúnmente datos de precisión única porque cada constante real se almacena en una única ubicación de memoria. Esto generalmente da siete dígitos significativos para cada valor real. En muchos cálculos, particularmente aquellos que involucran iteraciones o secuencias largas de cálculos, la precisión simple no es adecuada para expresar la precisión requerida. Para superar esta limitación, FORTRAN proporciona el tipo de datos de doble precisión. Cada precisión doble se almacena en dos ubicaciones de memoria, proporcionando así el doble de dígitos significativos.

La forma general de la declaración de una variable de doble precisión es:

DOBLE PRECISIÓN name1, name2

Ejemplo:

DOBLE RAÍZ DE PRECISIÓN, VELO

### 3.3 Expresiones

Expresión: una expresión es una combinación de operadores, constantes y variables. Una expresión puede consistir en uno o más operandos y cero o más operadores para producir un valor.

El resultado de la expresión aritmética es de tipo numérico; el resultado de la expresión relacional y de una expresión lógica es de tipo lógico: el resultado de una expresión es de tipo carácter.

#### 3.3.1 Expresiones aritméticas.

Las expresiones aritméticas son análogas a las fórmulas matemáticas.

Los símbolos  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $^$ ,  $-$  o  $**$  y las palabras clave `div` y `mod` se conocen como operadores aritméticos. En la expresión  $5 + 3$  se denominan operando. El valor de la expresión  $5 + 3$  como resultado de la expresión.

Operador	Significado	Tipos de operandos	Tipos de resultado
$-, ^, **$	Exponenciación	Entero o Real	Entero o Real
$+$	Suma	Entero o Real	Entero o Real
$-$	Resta	Entero o Real	Entero o Real
$*$	Multiplicación	Entero o Real	Entero o Real
$/$	División	Real	Real
<code>Div</code>	División entera	Entero	Entero
<code>Mod</code>	Modulo (resto)	Entero	Entero

Figura 10. Expresiones aritméticas. Fuente: Autoría propia.

- $5 \times 7$  se representa por  $5 * 7$
- $6/4$  se representa por  $6 / 4$
- $3^7$  se representa por  $3 ^ 7$

### 3.3.2 Expresiones lógicas (booleanas).

Se denominan por el matemático George Boole.

Se forman combinando constantes lógicas, variables lógicas y otras expresiones lógicas, utilizando los operadores lógicos not, and y or, y los operadores relacionales:  $=$ ,  $<$ ,  $>$ ,  $<=$ ,  $>=$ ,  $<>$ .

### 3.4 Operadores de relación

Son los que realizan comparaciones de valores. Sirven para condicionar los algoritmos.

Su formato es:

Operadores de relación	
Operador	Significado
$<$	Menor que
$>$	Mayor que
$=$	Igual que
$<=$	Menor o igual que
$>=$	Mayor o igual que
$<>$	Distinto de

Figura 11. Operadores de relación Fuente: Autoría propia.

Los operadores de relación se pueden aplicar a cualquiera de los cuatro tipos de datos estándar: enteros, real, lógico, carácter.

N1	N1	Expresión lógica	Resultado
3	6	$3 < 6$	Verdadero
0	1	$0 > 1$	Falso
4	2	$4 = 2$	Falso
8	5	$8 \leq 5$	Falso
9	9	$9 \geq 9$	Verdadero
5	5	$5 = 5$	Falso

Figura 12. Cuatro tipos de datos estándar: enteros, real, lógico, carácter. Fuente Autoría propia.

### 3.5 Operadores lógicos

De acuerdo a la tabla de verdad recoge el funcionamiento de dichos operadores.

A	Negación de a
V	F
F	V

Operador lógico	Expresión lógica	Significado
No (not)	No p (not p)	Negación de p
Y (and)	P y q (p and q)	Conjunción p y q
O (or)	P o q (p or q)	Disyunción de p y q

A	b	A y b	A o b
Verdad	Verdad	Verdad	Verdad
Verdad	Falso	Falso	Verdad
Falso	Verdad	Falso	Verdad
falso	falso	Falso	Falso

Figura 13. Operadores lógicos. Fuente: Autoría propia.

### 3.6 Funciones internas

Son aquellas que se encuentran dentro de los lenguajes de programación ya establecidos y son utilizadas mayormente en los algoritmos.

Función	Descripción	Tipo de argumento	Resultado
Abs (x)	Valor absoluto de x	Entero o real	Igual que argumento
Arctan(x)	Arco tangente de x	Entero o real	Real
Cos(x)	Coseno de x	Entero o real	Real
Exp (x)	Exponencial de x	Entero o real	Real
Ln (x)	Logaritmo neperiano de x	Entero o real	Real
Log10(x)	Logaritmo decimal de x	Entero o real	Real
Redondeo(x) (round(x))*	Redondeo de x	Real	Entero
sen(x) (sin(x))*	Seno de x	Entero o real	Real

Cuadrado(x) (sqr(x))*	Cuadrado de x	Entero o real	Igual que argumento
Raíz2(x) (sqrt(x))*	Raíz cuadrada de x	Entero o real	Real
trunc(x)	Truncamiento de x	Entero o real	Entero

Figura 14. Funciones internas. Fuente Autoría propia.

#### Terminología en inglés

Las funciones aceptan argumentos reales o enteros y sus resultados de la tarea que realice la función.



Expresión	Resultado
Raíz2(25)	5
Redondeo(6,6)	7
Redondeo(3,1)	3
Redondeo(-3,2)	-3
Trunc(5,6)	5
Trunc(3,1)	3
Cuadrado(4)	16
Abs(9)	9
Abs(-1,2)	12

Figura 15. Reales o enteros. Fuente Autoría propia.

### 3.7 Entrada y salida de información

Los cálculos que realizan las computadoras requieren para ser útil la entrada de los datos necesarios para ejecutar las operaciones que posteriormente se convertirán en resultados, es decir, salida.

**Leer** (lista de variables de entrada)

**Escribir** (lista de expresiones de salida)

**I. Título: “Herramientas de programación” Datos informativos:**

Institución educativa :

Componente :

Grado y sección : 1er - nivel secundario

Duración : 45 minutos

Fecha : 13 de octubre

Profesor :

**II. Logro de aprendizaje de la sesión**

Define y reconoce las herramientas del algoritmo

**III. Organización de los aprendizajes**

Conocimientos	Capacidades	Actitudes
Desarrollo de un algoritmo con PSEINT. Solución de un problema de proceso de datos con el Software PSEINT	<b>Razonamiento y demostración</b>  - Identifica datos y analiza sus relaciones  - Representa e interpreta los datos en el diagrama de flujo.	- Muestra seguridad y perseverancia al resolver problemas y comunicar resultados.  - Toma la iniciativa para formular preguntas  - Perseverancia en la búsqueda de soluciones.  - Valora aprendizajes desarrollados en el área como parte de su proceso formativo

**IV. Secuencia didáctica**

Tiempo	Situación de aprendizaje	Estrategias metodológicas			Evaluación		
		Intervención didáctica	Actividad de aprendizaje	Recursos didácticos	Criterio	Indicadores	Instrumentos
5	<b>Inicio</b>	Grupos cooperantes          Exposición de ideas	<b>Motivación</b> Me presento, para comenzar con el tema los alumnos haciendo una repaso de los temas anteriores, a cada alumno se le entrega una ficha con el problema que consiste en: Calcular la suma de dos números naturales. De esta manera se permite al alumno prepararse para comenzar con la clase.	Computador       Proyector multimedia       Pizarra	Actitud ante el área	Toma la iniciativa para formular preguntas.	Ficha de observación.          Guía de Laboratorio
40	<b>Proceso</b>		Culminando el inicio, se procederá a desarrollar un breve resumen del tema (herramientas de programación), con participación de los alumnos, se plantea desarrollar un algoritmo para hallar la solución empleando la idea de algoritmos.	Plumones       Computador			

10	Salida		Realizan el algoritmo y lo plasman en código en el Software PSEINT.  Ejecutan el programa y comprueban los resultados del algoritmo.  Trabajan una hoja de práctica.				
----	--------	--	--	--	--	--	--

## Guía de laboratorio: diagramas de flujo

Apellido y Nombre: .....

Profesor:

Fecha:            Grado:

Un diagrama de flujo es un tipo de diagrama que representa un proceso que utiliza diferentes símbolos que contienen información sobre pasos o una secuencia de eventos. Cada uno de estos símbolos está vinculado con flechas para ilustrar la dirección del flujo del proceso.

Los diagramas de flujo son una metodología utilizada para analizar, mejorar, documentar y administrar un proceso o programa. Los diagramas de flujo son útiles para:

1. Ayudar a comprender las relaciones entre los diferentes pasos del proceso.
2. Recopilar datos sobre un proceso en particular.
3. Ayudando con la toma de decisiones
4. Medir el desempeño de un proceso
5. Representar la estructura de un proceso.
6. Seguimiento del flujo del proceso
7. Destacando pasos importantes y eliminando los pasos innecesarios

Un diagrama de flujo en informática generalmente tiene los siguientes tipos de símbolos para representar un proceso o programa:

Rectángulo / círculo ovalado / redondeado: representa cualquier proceso que tenga una actividad de inicio y fin.

**Rectángulos:** Representa una actividad o paso del proceso.

**Diamantes:** se utilizan cuando hay que tomar una decisión o responder una pregunta, como Sí / No o Verdadero / Falso. El camino a seguir está determinado por la respuesta a la pregunta.





**Líneas de flecha:** se utilizan para mostrar el flujo de control de un paso a otro. También indican el progreso de un paso a otro.





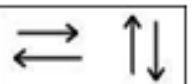



**Paralelogramos:** se utilizan para representar entrada / salida.

Los diagramas de flujo se usan comúnmente en el desarrollo de planes de negocios, el diseño de algoritmos y la determinación de pasos para la resolución de problemas.

Muchos programas de software están disponibles para diseñar diagramas de flujo.

Algunos de los programas de software más utilizados son SmartDraw, Visio (diseñado para PC) y OmniGraffle (diseñado para Mac).

	<b>Inicio o fin del programa</b>
	<b>Pasos, procesos o líneas de instrucción de programa de computo</b>
	<b>Operaciones de entrada y salida</b>
	<b>Toma de decisiones y Ramificación</b>

	Conector para unir el flujo a otra parte del diagrama
	Cinta magnética
	Disco magnético
	Conector de página
	Líneas de flujo
	Anotación
	Salida a pantalla
	Envía datos a la impresora

## Símbolos gráficos

Los símbolos gráficos son utilizados específicamente para operaciones aritméticas y relaciones condicionales.

Símbolo	Acción	Símbolo	Acción
+	Sumar	<sup>3</sup>	Mayor o igual que
-	Menos	£	Menor o igual que
*	Multiplicación	<sup>1</sup> o $\nabla$	Diferente de
/	División		Si
±	Más o menos		No
=	Equivalente a		True
>	Mayor que		False
<	Menor		

## Reglas para la creación de Diagramas

Para asegurarse de que un diagrama de flujo funciona, debe seguir algunas reglas básicas de construcción:

Cada diagrama de flujo debe tener un solo objeto Start.

El flujo de control siempre debe ingresar un objeto desde la parte superior.

El flujo de control siempre debe dejar un objeto desde abajo (a excepción de los objetos de Decisión, que permiten que el flujo de control salga desde el lado).

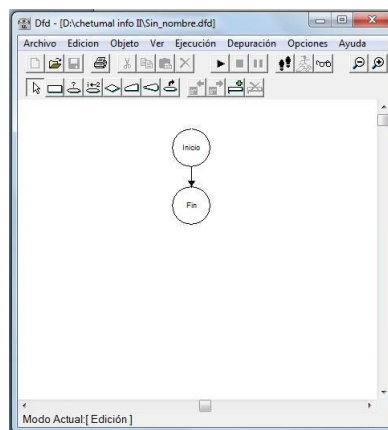


El flujo de control no debe dividirse. Puede usar objetos de decisión para dar al flujo de control una selección de rutas a seguir, pero solo puede seguir una de estas rutas.

El cursor no-go se usa para mostrarle dónde no puede soltar objetos o completar enlaces.

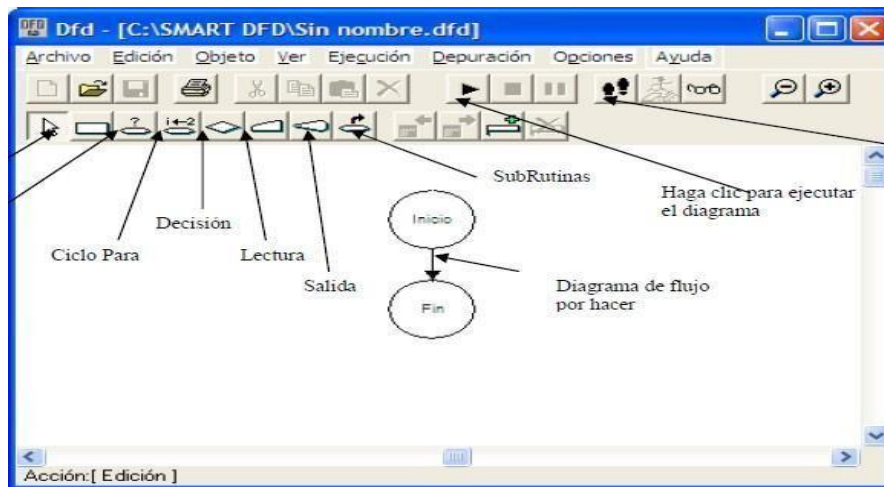
## Ejercicio

Realizar la adición de dos números naturales.



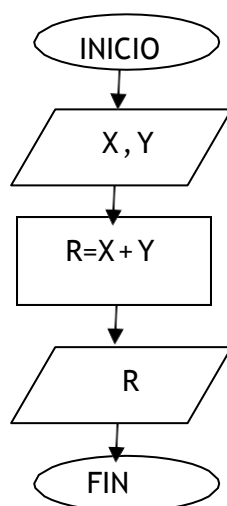
### Pasos de la práctica:

1. Escritorio de Windows.
2. Abrir el programa **DFD que aparece en el escritorio).**
3. Se muestra a continuación:

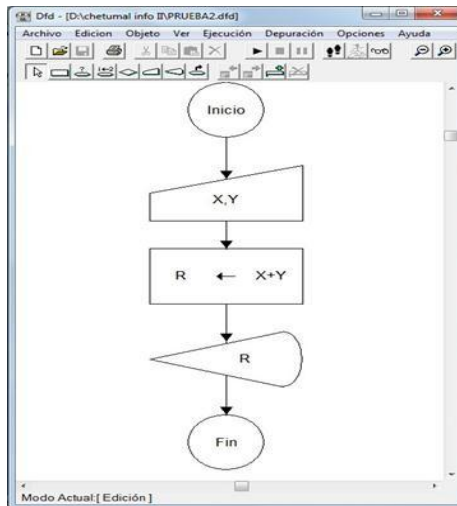


La forma en que se hace un diagrama es seleccionando primero el símbolo que quiere agregar al diagrama, después se lleva el puntero del ratón hacia la posición en que quiere colocar el símbolo en el sistema, una vez allí, se hace clic para colocarlo, por último se hace doble clic en el símbolo que ya colocó en su posición para agregarle la información que lleva en su interior.

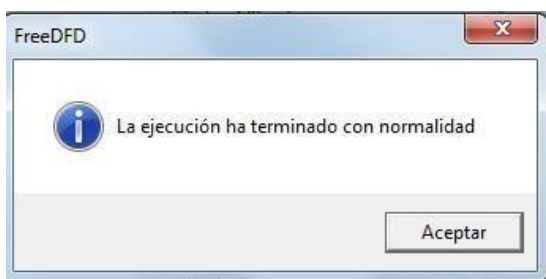
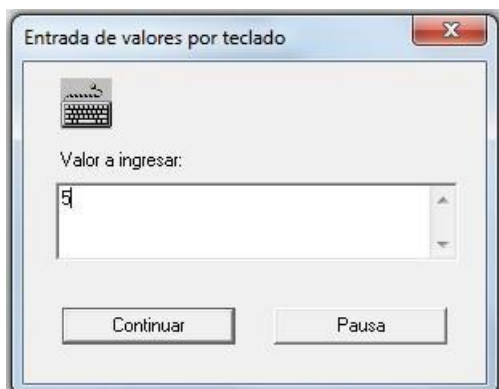
#### 4. Realizar el diagrama de flujo



5. En el programa Smart-DFD quedará así:



6. Ejecute el diagrama y cuando solicite un valor, introduzca 5, posteriormente aparecerá la misma ventana, introduzca el número 7, observe la ilustración siguiente:



Haga clic en el botón **Continuar**

7. Se muestra la ventana:

Resultados de la Ejecución.



8. Al finalizar de forma correcta saldrá la siguiente ventana. Revisión por parte del asesor.

## Síntesis

Este apéndice resume los conceptos clave cubiertos en cada uno de los capítulos de la monografía. Se describe primero los conceptos básicos del algoritmo. Una pila es una estructura de datos que proporciona acceso de último en entrar, primero en salir a los elementos.

Uno puede implementar una pila en una lista o matriz vinculada, aunque puede necesitar cambiar el tamaño de la matriz si se llena. Una cola es una estructura de datos que proporciona acceso de primera entrada, primera salida a los elementos. Él / ella puede implementar una cola en una lista vinculada o matriz circular, aunque puede necesitar cambiar el tamaño de la matriz si se llena. Muchos algoritmos usan árboles, por lo que es importante recordar al menos las propiedades de árbol más básicas. Se pueden modelar muchos problemas con los árboles de decisión. Los árboles de juego son un tipo especial de árbol de decisión. La monografía también describe algoritmos de red, algoritmos de cadena, algoritmos criptográficos, teoría de la complejidad, algoritmos distribuidos y rompecabezas de entrevistas.

Algoritmos y estructuras de datos: una estructura de datos contiene datos en alguna disposición. Un algoritmo produce un resultado. Necesita un algoritmo para construir y usar una estructura de datos.

El pseudocódigo es un texto que se parece mucho al código pero no a ningún lenguaje de programación en particular. Debe traducirlo a un lenguaje de programación real antes de poder ejecutarlo.

Metas algorítmicas: para ser útil, un algoritmo debe ser correcto, sostenible y eficiente.

La notación Big O describe la relación entre el número de entradas y los requisitos de tiempo de ejecución o memoria a medida que aumenta el tamaño del problema. La notación Big O ignora los múltiplos constantes y considera solo la función de más rápido crecimiento que describe el rendimiento.

### **Apreciación crítica y sugerencias**

Para muchas personas, la programación es virtualmente un fenómeno de la naturaleza. Parece intemporal, grabado en piedra, difícil de cambiar y tal vez no necesita cambiar. Pero la educación de programación de ayer, que tenía una base práctica, ya no es viable. El aprendizaje de memoria de los procedimientos lógicos ya no tiene el valor claro que alguna vez tuvo.

La amplia disponibilidad de herramientas tecnológicas para la computación significa que las personas dependen menos de sus propios poderes de computación. Al mismo tiempo, las personas están mucho más expuestas a los números y las ideas cuantitativas y, por lo tanto, necesitan lidiar con la programación en un nivel más alto que hace 20 años. Sin embargo, muy pocos estudiantes abandonan la escuela primaria y secundaria con el conocimiento de programación, la habilidad y la confianza adecuados para que cualquiera esté satisfecho de que todo está bien en matemáticas escolares. Además, ciertos segmentos de la población no están bien representados entre aquellos que tienen éxito en el aprendizaje de programación. El fracaso generalizado de aprender programación limita las posibilidades individuales y obstaculiza el crecimiento nacional. Nuestras experiencias, debates y revisión de la literatura nos han convencido de que la programación exige un cambio sustancial. Reconocemos que dicho cambio debe llevarse a cabo de manera cuidadosa y deliberada, para que cada estudiante tenga la oportunidad y el apoyo necesarios para dominarlas.

En este capítulo, presentamos conclusiones y recomendaciones para ayudar a mover a la nación hacia el cambio necesario.

Dentro del mismo problema abarcado en este trabajo, se recomienda utilizar el PSeint especializado en la implementación de algoritmos que den solución a los problemas de programación de información, para así comparar los resultados obtenidos manualmente y por el paquete computacional. Además, se recomienda implementar el curso de algoritmos, para que todos los estudiantes de educación superior tengan la capacidad de programar.



## Referencias

- Brena, R. (1997). *Lenguajes Formales y Autómatas*, Centro de Inteligencia Artificial, Instituto Tecnológico y de Estudios Superiores de Monterrey, Monterrey, México: Editorial Campus Monterrey.
- Brookshear, J.(1993).*Teoría de la Computación, Lenguajes formales, Autómatas y Complejidad*, Addison Wesley Iberoamericana. España: Editorial Addison-Wesley Iberoamericana.
- Carrasco, M. (2000). *Programación con Visual Basic*. El Salvador: Editorial Macro.
- Castañeda, J. (2001). *Programación con Visual Basic*. Lima, Perú: Editorial Megabyte.
- Joyanes, A. (2003). *Metodología de la programación*. México: Editorial McGraw-Hill.
- Joyanes A. (2003). *Fundamentos de programación, algoritmos y estructura de Datos*. México: Editorial McGraw-Hill.
- Joyanes, A. (2004). *Fundamentos de programación, algoritmos y estructura de datos*. México: Editorial McGraw-Hill.
- Monreal, J. (2007). *El Mundo De La Computación Tomo 3 Y 4*”, España: Editorial Oceano.
- Tremblay, B. y Bunt, R. (1987). *Introducción a las ciencias de las computadoras* New York, NY, Estados Unidos: Editorial. McGraw-Hill.