

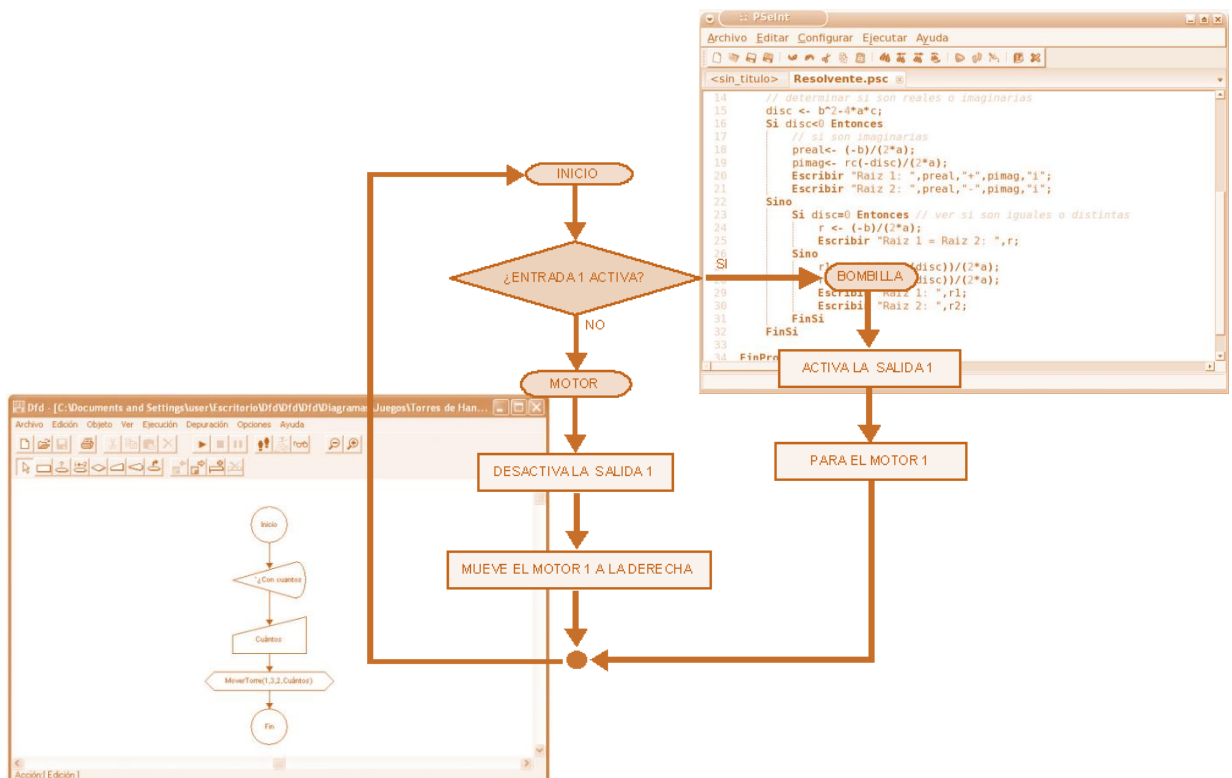
**UNIVERSIDAD NACIONAL DE JUJUY**  
**ESCUELA DE MINAS DR. HORACIO CARRILLO**



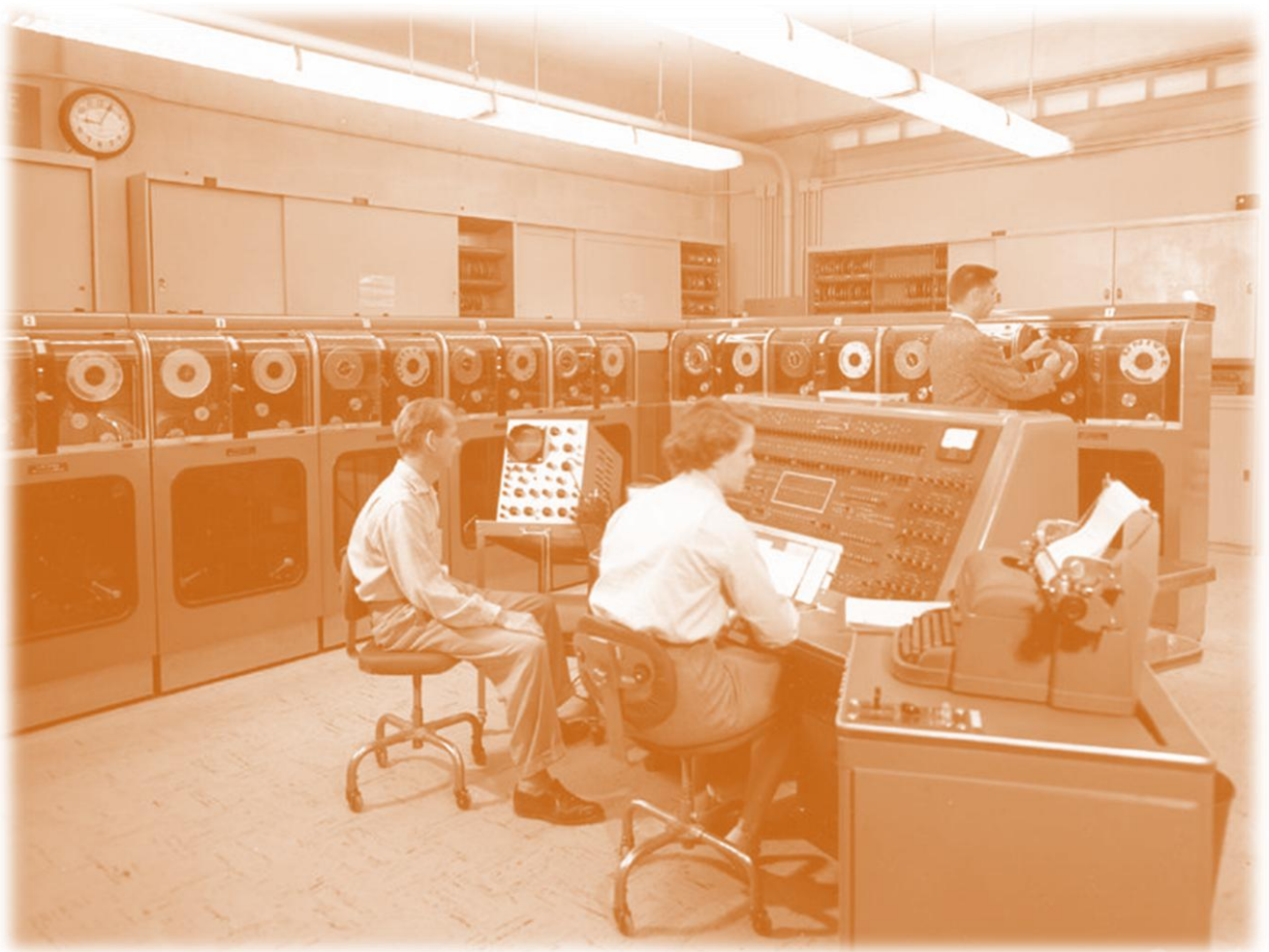
**TÉCNICO INFORMÁTICO**

# PROGRAMACIÓN ESTRUCTURADA

**Año 2016**

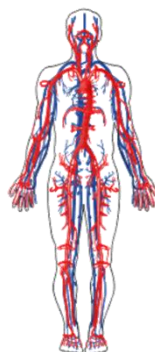


# UNIDAD I: CONCEPTOS BÁSICOS



*¿Qué es un sistema?*

Un sistema es un conjunto de componentes interrelacionados que trabajan juntos para alcanzar un objetivo predefinido. Por ejemplo, el sistema circulatorio humano (Figura 1).



**Figura 1. Sistema circulatorio humano.**

*¿Qué es un sistema de procesamiento de información?*

Un *sistema de procesamiento* de información (Figura 2) es un sistema que transforma datos brutos en información organizada, significativa y útil. Un sistema de procesamiento de información consta de 3 componentes: entrada (datos), procesador (métodos de transformación de información) y salida (información procesada).



**Figura 2. Sistema de procesamiento de información.**

Para llevar a cabo el procesamiento de la información se requiere de un conjunto de instrucciones que especifiquen la secuencia de operaciones a realizar, en orden, para resolver un problema específico o clase de problemas. A este conjunto de instrucciones se denomina *algoritmo* (Figura 3).



**Figura 3. Un sistema de información utiliza un algoritmo para realizar la transformación de la entrada.**

Cuando el procesador de información es una computadora, el algoritmo debe expresarse como programa. Un programa se escribe en algún lenguaje de programación y la tarea de expresar un algoritmo como programa se denomina *programación*. Los pasos de un algoritmo se traducen a instrucciones de programa, que indican las operaciones a realizar por la computadora.

*¿Qué es una computadora?*

Una computadora es un dispositivo electrónico utilizado para procesar información y obtener resultados. Los componentes físicos de una computadora, que permiten introducir datos y obtener información se denominan *hardware*. El conjunto de programas y datos que controlan el funcionamiento de una computadora se denomina *software*.

El hardware consta básicamente de los siguientes componentes:

- unidad central de proceso (UCP)
- memoria central o principal
- dispositivos de almacenamiento secundario
- periféricos o dispositivos de entrada, salida y entrada/salida

#### Unidad Central de Proceso

La UCP (CPU en inglés, Central Processing Unit) dirige y controla el proceso de información realizado por la computadora. La UCP procesa o manipula información almacenada en memoria; puede recuperar información desde memoria (instrucciones o datos de programa). También puede almacenar los resultados de procesos en memoria. Es decir, que la UCP puede leer y escribir datos en memoria.

La UCP consta de 2 componentes principales:

Unidad de Control (UC), cuya función es coordinar las actividades de la computadora y determinar qué operaciones deben realizarse y en qué orden; además de controlar y sincronizar todo el proceso de la computadora.

Unidad Aritmético-Lógica, cuya función es realizar las operaciones aritméticas y lógicas, tales como suma, resta, multiplicación, división y comparaciones.

#### Memoria Central

La memoria central o principal de una computadora se utiliza para almacenar información. En general, la información almacenada en memoria puede ser de 2 tipos: instrucciones de programa o datos con los que operan las instrucciones. Para que un programa se ejecute debe cargarse (load) en memoria principal, asimismo los datos utilizados en el procesamiento también deben ser cargados.

La memoria se organiza en unidades de almacenamiento individual (celdas) denominadas registros o palabras. Un registro o palabra es un conjunto, generalmente, de 8 bits al que se llama byte (octeto). Un bit es la unidad mínima de información que puede almacenarse y representa un valor cero o uno. Cada registro o palabra de memoria tiene asociado 2 conceptos: dirección y contenido. La dirección de una palabra indica su posición en la memoria, y permite especificar que byte será leído o escrito. El contenido de una palabra de memoria hace referencia al valor almacenado en esa posición.

### Memoria Secundaria

Para ejecutar un programa es necesario que éste se cargue en la memoria principal. La memoria tiene capacidad limitada (tamaño) y pierde su contenido cuando la computadora se apaga o sufre la interrupción del suministro de energía eléctrica. Es por ello que los dispositivos de almacenamiento masivo se hacen necesarios para guardar tanto datos como programas de modo permanente. Los discos rígidos, cintas magnéticas (tape backup), soportes ópticos y dispositivos extraíbles en general, constituyen las memorias auxiliares, externas o secundarias. En comparación con éstas, la memoria principal es más rápida (en la lectura y escritura de datos) y costosa, pero también volátil (se borra al no tener energía).

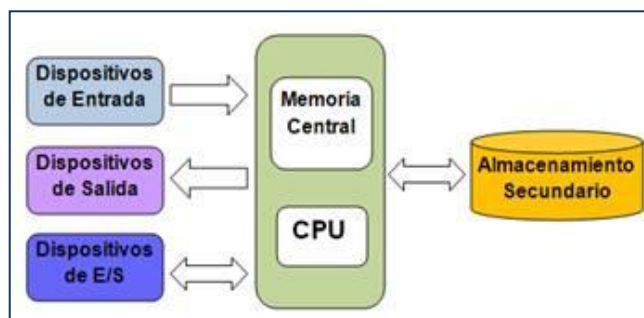
### Dispositivos de entrada, salida y entrada/salida

Estos dispositivos permiten la comunicación entre el usuario y la computadora. Los dispositivos o periféricos de entrada, sirven para introducir datos para el procesamiento. Los datos se leen desde la entrada y se almacenan en memoria principal. Los dispositivos de entrada convierten la información de entrada para que pueda ser almacenada en la memoria principal. Ejemplos de dispositivos de entrada son: teclado, mouse, lápiz óptico, lectores de códigos de barras, etc.

Los dispositivos de salida permiten presentar los resultados del procesamiento de datos. Ejemplos de dispositivos de salida son: monitor, impresoras, plotters (trazadores gráficos), parlantes, etc.

Los dispositivos de entrada/salida cumplen tanto la función de ingreso de datos como la de presentación de resultados. Por ejemplo, las impresoras multifunción permiten escanear imágenes (entrada) y obtener resultados impresos (salida).

La figura 4 presenta la estructura física general de una computadora.



**Figura 4. Estructura física de una computadora.**

El componente lógico de una computadora, el software, permite especificar las operaciones que se realizarán sobre el hardware. En particular el software de sistemas se ocupa de proporcionar servicio a otros programas y administrar los recursos (memoria, los discos, los dispositivos, etc.) y tiempo de una computadora. Ejemplo de software de sistemas son los sistemas operativos, editores de texto, compiladores/intérpretes y programas de utilidad.

## Resolución de Problemas con Computadora

Como ya se mencionó, un algoritmo especifica el conjunto de pasos que debe seguirse para resolver un problema. En el contexto de la Informática un algoritmo se traduce a un programa mediante algún lenguaje de programación, este proceso se denomina *programación*. Por lo tanto, el objetivo de la programación es proporcionar soluciones basadas en computadora que resuelvan problemas considerados engorrosos o difíciles para una persona. La metodología necesaria para resolver problemas mediante computadoras se llama *metodología de la programación*. El punto de partida de esta metodología es el algoritmo. Todo algoritmo debe cumplir con las siguientes características:

- un algoritmo debe ser preciso, es decir, debe indicar el orden de realización de cada paso,
- un algoritmo debe ser definido, es decir, si un algoritmo se sigue 2 veces (considerando el mismo conjunto de datos de entrada) los resultados que se obtengan deben ser los mismos.
- un algoritmo debe ser finito, es decir, si se sigue un algoritmo éste debe finalizar en algún momento (debe tener un número finito de pasos)

Los pasos para obtener una solución basada en computadora son:

- 1.- Análisis del problema
- 2.- Diseño del algoritmo
- 3.- Codificación
- 4.- Compilación y ejecución
- 5.- Verificación y depuración
- 6.- Documentación y mantenimiento

A continuación se describen los pasos enunciados.

### Análisis del Problema

En este paso se debe definir claramente el problema a resolver, la información que se utilizará para calcular la solución (datos de entrada), el resultado o solución deseada (datos de salida) y el objetivo del algoritmo que alcanzará la solución.

### Diseño del Algoritmo

En la fase de diseño se determina cómo el algoritmo llevará a cabo el procesamiento de la información de entrada para obtener el resultado deseado. Entonces el diseñador descompone el problema en subproblemas y éstos a su vez en subproblemas más simples, aumentando en cada nivel de descomposición el detalle de descripción del algoritmo. El enfoque de dividir el problema en subproblemas se denomina *diseño descendente* (top-down design) y la descripción más detallada de los pasos del algoritmo en cada nivel de descomposición se denomina *refinamiento sucesivo*.

Las ventajas más importantes del diseño descendente son:

- el problema se comprende más fácilmente al dividirse en partes más simples denominadas módulos (conjunto de acciones que tienen un punto de entrada y un punto de salida)
- las modificaciones en los módulos son más fáciles
- la comprobación del problema se puede realizar fácilmente

El diseño abarca:

- diseño descendente
- refinamiento por pasos
- herramientas de programación para la representación de algoritmos

Las herramientas de programación permiten representar un algoritmo a través de técnicas que independizan al diseñador de la sintaxis de un lenguaje de programación específico. Los métodos más usuales para representar algoritmos son:


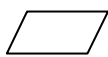

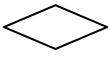

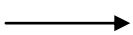






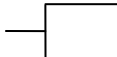
- diagramas de flujo,

- diagramas N-S,
- pseudocódigo,
- lenguaje español (una descripción narrativa) o
- fórmulas

### Diagramas de Flujo

Un *diagrama de flujo* (flowchart) es una técnica de representación de algoritmos que utiliza símbolos (cajas) estándar y que tiene los pasos del algoritmo escritos en esas cajas unidas por flechas, denominadas líneas de flujo, que indican la secuencia en que se deben ejecutar. La tabla 1 presenta los símbolos más utilizados en los diagramas de flujo.

**Tabla 1. Símbolos del Diagrama de Flujo.**

SÍMBOLO	FUNCIÓN
	Terminal, representa el "inicio" y el "fin" de un programa.
	Entrada/salida, representa cualquier tipo de introducción de datos en la memoria desde los periféricos, "entrada", o registro de la información procesada en un periférico, "salida."
	Proceso, representa cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transferencia, etc.
	Decisión/Decisión Múltiple, indica operaciones lógicas o de comparación entre datos, y en función del resultado determina cuál de los caminos alternativos del programa seguir.
	Conector, permite enlazar 2 partes de un ordinograma a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página del diagrama.
	Indicador de dirección o línea de flujo, señala el sentido de la ejecución de las operaciones.
	Línea conectora, permite conectar 2 símbolos.
	Conector, permite conectar 2 puntos del organigrama situado en páginas diferentes.
	Llamada a subrutina o un proceso predeterminado. Una subrutina es un módulo independiente del programa principal, que recibe una entrada procedente de dicho programa, realiza una tarea determinada y regresa, al terminar, al programa principal.
	Pantalla, se utiliza en ocasiones en lugar del símbolo E/S.
	Impresora, se utiliza en ocasiones en lugar del símbolo E/S.
	Teclado, se utiliza en ocasiones en lugar del símbolo E/S.
	Comentarios, permite añadir comentarios a los símbolos del diagrama de flujo, se pueden dibujar a cualquier lado del símbolo.

### Diagramas Nassi-Schneiderman (N-S)

El diagrama N-S de Nassi-Schneiderman (también conocido como diagrama de Chapin) es como un diagrama de flujo en el que se omiten las flechas de unión y las cajas son contiguas. Las acciones sucesivas se escriben en cajas sucesivas y, como en los diagramas de flujo, se pueden escribir diferentes acciones en una caja.

### Pseudocódigo

El pseudocódigo es un lenguaje de especificación de algoritmos. El uso de tal lenguaje hace el paso de codificación final relativamente fácil. La ventaja del pseudocódigo es que en su uso, en la planificación de un programa, el

programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje de programación específico. Otra ventaja del pseudocódigo es que puede ser traducido fácilmente a los lenguajes estructurados.

El pseudocódigo utiliza un conjunto de palabras (palabras reservadas) para representar las estructuras de control que implementan las acciones que el algoritmo debe realizar. La escritura de pseudocódigo exige normalmente la indentación de diferentes líneas. Además se pueden incluir comentarios en el pseudocódigo que permitan documentarlo, éstos se indican precediéndolos por el símbolo "//".

### Codificación

Codificación es la escritura en un lenguaje de programación de la representación del algoritmo desarrollada en las etapas precedentes. Dado que el diseño de un algoritmo es independiente del lenguaje de programación utilizado para su implementación, el código puede ser escrito con igual facilidad en un lenguaje o en otro.

Para realizar la conversión del algoritmo en un programa se deben sustituir las palabras reservadas en español por sus homónimos en inglés, y las operaciones/instrucciones indicadas en lenguaje natural expresarlas en el lenguaje de programación correspondiente.

Es conveniente aclarar que los lenguajes de programación se clasifican básicamente en:

- lenguajes máquina,
- lenguajes de bajo nivel (ensamblador) y
- lenguajes de alto nivel

Los lenguajes máquina son aquellos que están escritos en lenguajes directamente inteligibles por la computadora, ya que sus instrucciones son cadenas binarias (secuencias de 0's y 1's) que especifican una operación y las posiciones de memoria implicadas en la operación. Las instrucciones en lenguaje máquina dependen del hardware de la computadora, y por tanto, serán diferentes de una computadora a otra. Esto lleva a que el programador requiera de un profundo conocimiento de la estructura interna de la computadora para la que escribe el programa.

Los lenguajes de bajo nivel son más fáciles de utilizar que los lenguajes máquina, pero, al igual que ellos, dependen del hardware de la computadora. El lenguaje de bajo nivel por excelencia es el ensamblador. Las instrucciones en ensamblador son conocidas como nemotécnicos (por ejemplo, ADD es una instrucción de suma). Estos nemotécnicos se utilizan en lugar de los códigos máquina de las instrucciones correspondientes. Un programa en lenguaje ensamblador no es directamente ejecutable por la computadora, sino que requiere una traducción a lenguaje máquina.

Los lenguajes de alto nivel son los más utilizados por los programadores y están diseñados para que las personas escriban y entiendan los programas fácilmente. Además los programas escritos en lenguajes de alto nivel son independientes del hardware de la computadora, por lo que se dice que son portables o transportables (pueden ser ejecutados en diferentes computadoras con poco o ninguna modificación). Los programas en lenguajes de alto nivel deben ser traducidos por programas traductores (compiladores/intérpretes) para poder ser ejecutados.

### Compilación y Ejecución

Una vez que el algoritmo se ha traducido en un programa fuente (generalmente en un lenguaje de alto nivel), es preciso introducirlo en memoria mediante el teclado y almacenarlo en disco. Esta operación se realiza con un programa editor, posteriormente el programa fuente se convierte en un archivo de programa que se graba en disco.

El programa fuente debe ser traducido a lenguaje máquina. Este proceso se realiza con el compilador y el sistema operativo que se encarga de la compilación. Si tras la compilación se presentan errores (errores de compilación) en el

programa fuente, es preciso volver a editar el programa, corregir los errores y compilar de nuevo. Esto se repite hasta que no se producen errores, obteniéndose el programa objeto que todavía no es ejecutable directamente. Suponiendo que no existen errores en el programa fuente, se debe instruir al sistema operativo para que realice la fase de montaje o enlace (link), que carga el programa objeto con las librerías del programa compilador. El montaje genera un programa ejecutable. La Figura 5 ilustra el proceso de compilación y ejecución de un programa.

Cuando el programa ejecutable se ha creado, se puede ejecutar. Suponiendo que no se producen errores durante la ejecución se obtendrán los resultados del programa.

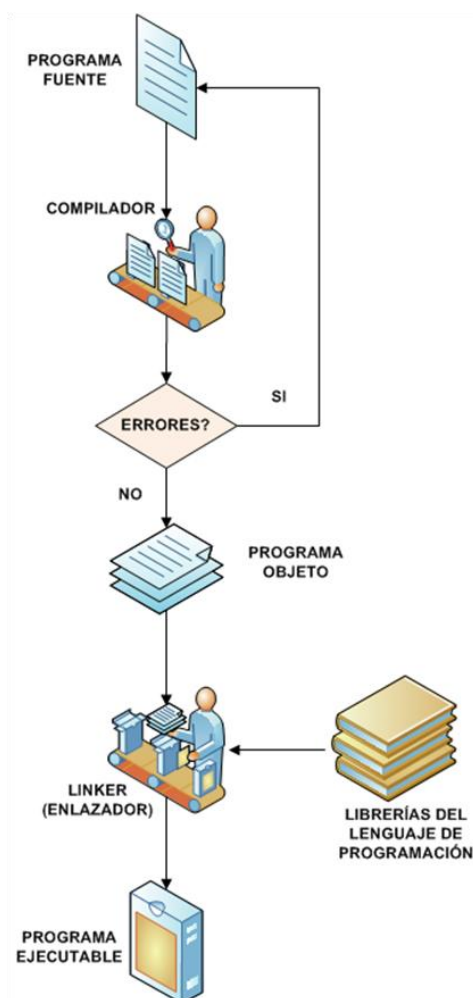


Figura 5. Proceso de compilación y ejecución de un programa.

### Verificación y Depuración

La verificación de un programa es el proceso de ejecución del programa con una amplia variedad de datos, llamados datos de test o prueba, que determinarán si el programa tiene errores ("bugs"). Para realizar la verificación se debe desarrollar una amplia gama de datos de test: valores normales de entrada, valores extremos de entrada que comprueben los límites del programa y valores de entrada que comprueben aspectos especiales del programa.

La depuración es el proceso de encontrar los errores del programa y corregir o eliminar dichos errores.

Cuando se ejecuta un programa, se pueden producir tres tipos de errores:

- 1.- *Errores de compilación.* Se producen normalmente por el uso incorrecto de las reglas del lenguaje de programación y suelen ser errores de sintaxis. Si existe un error de sintaxis, la computadora no puede comprender la instrucción, no se obtendrá el programa objeto y el compilador imprimirá una lista de todos los errores encontrados durante la compilación.



2.- *Errores de ejecución.* Estos errores se producen por instrucciones que la computadora puede comprender pero no ejecutar. Ejemplos típicos son: división por cero y raíces cuadradas de números negativos. En estos casos se detiene la ejecución del programa y se imprime un mensaje de error.

3.- *Errores lógicos.* Se producen en la lógica del programa y la fuente del error suele ser el diseño del algoritmo. Estos errores son los más difíciles de detectar, ya que el programa puede funcionar y no producir errores de compilación ni de ejecución, y sólo puede advertirse el error por la obtención de resultados incorrectos. En este caso se debe volver a la fase de diseño del algoritmo, modificar el algoritmo, cambiar el programa fuente, compilar y ejecutar una vez más.

### Documentación y Mantenimiento

La documentación de un problema consta de las descripciones de los pasos a dar en el proceso de resolución de un problema. La importancia de la documentación debe ser destacada por su decisiva influencia en el producto final. Programas pobremente documentados son difíciles de leer, más difíciles de depurar y casi imposibles de mantener y modificar.

La documentación de un programa puede ser interna y externa. La *documentación interna* es la contenida en las líneas de comentarios que aparecen en el programa fuente. La *documentación externa* incluye análisis, diagramas de flujo y/o pseudocódigos, manuales de usuario con instrucciones para ejecutar el programa y para interpretar los resultados.

La documentación es vital cuando se desea corregir posibles errores futuros o bien cambiar un programa. Tales cambios se denominan *mantenimiento* del programa. Después de cada cambio la documentación debe ser actualizada para facilitar cambios posteriores. En la práctica, se suele numerar las sucesivas versiones de los programas 1.0, 1.1, 2.0, 2.1, etc. (si los cambios introducidos son importantes, se varía el primer dígito [1.0, 2.0, ...], en caso de pequeños cambios sólo se varía el segundo dígito [2.0, 2.1, ...]).

### **Partes de un Programa**

Los componentes básicos de un programa son instrucciones y datos. Las instrucciones o acciones representan operaciones que ejecutará una computadora al *interpretar* un programa. Los lenguajes de programación brindan un conjunto de instrucciones que permiten escribir programas (soluciones algorítmicas) para resolver problemas específicos. Los datos son los valores que deben ser procesados por la computadora (que ejecuta el programa) para obtener resultados o información. Los datos pueden ser constantes (si no cambian) o variables (si sufren modificaciones durante la ejecución del programa).

### **Documentación de Algoritmos**

Un programa bien documentado es más fácil de leer y mantener. Para ello la *documentación* es fundamental. La mayoría de los lenguajes de programación proporcionan algún mecanismo de documentación, por ejemplo, a través de comentarios en el programa (documentación interna). Es recomendable utilizar un comentario general para el objetivo del programa en cuestión, que refleje la especificación del problema a resolver de la forma más completa posible. Además deben describirse todos los objetos de datos que se utilicen en el programa así como fecha de creación, autor, etc. Cuando se realizan modificaciones a un programa es necesario que se actualicen los comentarios para reflejar los cambios llevados a cabo sobre el código.

## Corrección de Algoritmos

Un algoritmo es correcto cuando cumple con su especificación, esto significa que cumple con los requerimientos propuestos. Para determinar cuáles son esos requerimientos se debe tener una especificación completa, precisa y libre de ambigüedades (imprecisiones) del problema a resolver antes de escribir el mismo.

Una de las formas de determinar si un programa es correcto consiste en realizar pruebas con un juego de datos reales que permitan establecer si cumple con su especificación.

## Eficiencia de Algoritmos

Un problema puede tener varias soluciones algorítmicas. Sin embargo, el uso de los recursos de la computadora (tiempo, memoria) que ejecuta cada una de las soluciones puede ser muy diferente.

La eficiencia se define como una medida de la calidad de los algoritmos que está asociada a la utilización óptima de los recursos de la computadora que ejecuta el algoritmo.

## Mantenimiento de Programas

El mantenimiento de un programa involucra básicamente dos tareas: corregir errores y modificar código. Los errores pueden ocurrir antes que el programa esté terminado o bien pueden ser descubiertos más tarde (cuando ya está en uso). En estas situaciones se debe identificar los errores, determinar las posibles causas y corregir el código correspondiente. Las modificaciones del código, en general, están asociadas a la necesidad de agregar funcionalidades a los programas o mejorar el rendimiento actual de éstos.

Según la naturaleza de los cambios que deban realizarse en un programa, el mantenimiento puede ser:

- **Correctivo:** una vez entregado, un programa aún puede contener errores que el usuario descubre al utilizarlo. El mantenimiento correctivo tiene por objetivo localizar y eliminar estos errores.
- **Preventivo:** este tipo de mantenimiento consiste en la modificación del software para mejorar sus propiedades sin alterar su funcionalidad (las tareas que realiza). Por ejemplo, se pueden incluir sentencias que comprueben la validez de los datos de entrada, reestructurar los programas para mejorar su legibilidad, o incluir nuevos comentarios que faciliten su comprensión.
- **Adaptativo:** este tipo de mantenimiento consiste en la modificación de un programa debido a cambios en el entorno (hardware o software) en el cual se ejecuta. Este mantenimiento permite al software adaptarse a las nuevas condiciones en las que debe trabajar.
- **Perfectivo:** el mantenimiento perfectivo se define como el conjunto de actividades para mejorar o añadir nuevas funcionalidades requeridas por el usuario.

La experiencia demuestra que un programa bien diseñado reduce el costo de mantenimiento del software.

## Reusabilidad de Código

En el desarrollo de soluciones basadas en computadora es común que se escriban componentes de programa de propósito general. Esto hace posible que cada vez que se crea un nuevo programa se usen los componentes diseñados anteriormente, reduciéndose así los costos, tiempo y esfuerzo invertidos en la creación de software.