1. INTRODUCCIÓN A LA PROGRAMACIÓN

1.1. OBJETIVOS.

Presentar todo el panorama histórico desde los inicios hasta los más recientes avances a fin de que el estudiante comprenda ideas y principios básicos que llevaron al desarrollo de las computadoras modernas.

Que el estudiante conozca la arquitectura de un computador y sus componentes de hardware y software.

1.2. HISTORIA DE LA COMPUTACIÓN

1.2.1. *Antecedentes*

El concepto numérico se considera, generalmente, como anterior al desarrollo de los lenguajes escritos; los primeros registros del hombre son anotaciones sobre la cantidad de granos, animales y demás posesiones personales. Con este fin, el hombre empleaba guijarros, palos y/o marcas en las paredes de las cavernas que habitaba.

El deseo humano de obtener mayor información y mejores comunicaciones, gradualmente fue dejando atrás estos sencillos instrumentos. El ábaco es considerado como el primer instrumento elaborado por el hombre para realizar operaciones aritméticas de manera más eficiente. El ábaco no realiza cálculos de manera autónoma (por si solo), simplemente le permite al hombre realizarlos de manera más eficiente.

La primera máquina que realizaba cálculos de manera autónoma fue inventada en 1642 por el gran matemático y filósofo francés Blas Pascal. La máquina de Pascal (en honor a su inventor), era movida mediante una serie de ruedas dentadas, numeradas del cero al nueve, alrededor de sus circunferencias y era capaz de sumar y restar en forma directa, mostrando un número a través de una ventanita y por este hecho tiene la ventaja de evitar tener que contar, como en el caso del ábaco; además, presenta los resultados en forma más accesible.

Figura 1. Máquina de Pascal (1642)

En 1671 Gottfried Wilhelm Leibniz le adicionó a la máquina de Pascal un cilindro diseñado especialmente para que fuera capaz de multiplicar y dividir de manera directa.

A principios del siglo XIX el ingeniero Joseph Marie Jacquard perfeccionó el concepto de tarjeta perforada, con el cual se podían "programar" las máquinas de tejer para que siguieran un patrón o diseño. Este concepto fue posteriormente utilizado en las máquinas de cómputo para decirles qué debían calcular.

Pero ninguno de estos avances fue antecesor directo de las computadoras electrónicas de hoy. La verdadera precursora de la computadora fue la máquina llamada "motor de diferencias", construida en 1822 por Charles Babbage para calcular algoritmos y tablas astronómicas.

A partir de su trabajo en el motor de diferencias, Babbage diseñó un poderoso instrumento para el cálculo automático. Tal como lo concibió Babbage, este "motor analítico" estaría impulsado por vapor, y trabajaría basado en un programa de planeación almacenado en tarjetas perforadas. Ésta máquina estaba dividida funcionalmente en dos grandes partes: una que ordenaba y otra que ejecutaba las órdenes. La que ejecutaba las órdenes era una versión muy ampliada de la máquina de Pascal, mientras que la otra era la parte clave. La innovación consistía, en que el usuario podía combinando las especificaciones de control, lograr que la misma máquina ejecutara operaciones complejas, diferentes a las hechas antes. Babbage concibió una memoria, un procesador aritmético, los medios de ingresar datos y/o instrucciones, así como una sección de producción que imprimiría los resultados.

Todos estos son los elementos de las computadoras modernas y no se hicieron realidad sino varias generaciones después de ser propuestos por Babagge. Charles Babbage no pudo implementar el motor analítico y murió sin saber que realmente funcionaba.

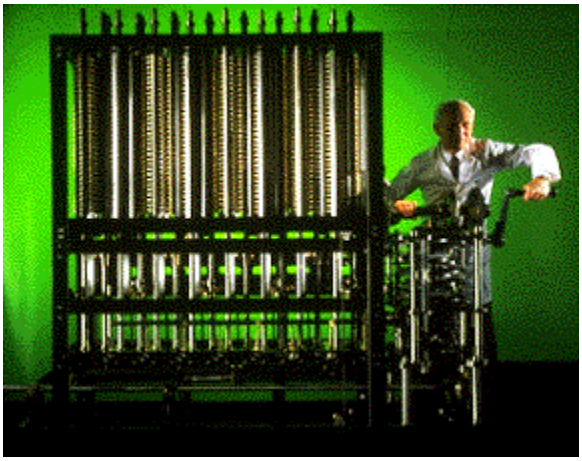


Figura 2. Máquina Diferencial de Charles Babbage

Durante los cien años siguientes, las máquinas activadas por tarjetas perforadas se modificaron, se mejoraron e hicieron más rápidas, pero aún no podían mantener el ritmo de las crecientes necesidades humanas de procesamiento de listas de pagos, cuentas, facturas, análisis de ventas y otros problemas.

En 1937, Howard H. Aiken, un candidato al doctorado en física de Harvard, trabajó en una máquina que podría resolver automáticamente ecuaciones diferenciales. La *International Business Machines* (I.B.M.), hoy una de las más grandes empresas de esta fase de la industria norteamericana, ayudó al inventor a crear la "Calculadora Controlada de Secuencia Automática" conocida como el "Mark I".

El Mark I era un monstruo de cuatro y media toneladas métricas, con 78 aparatos independientes vinculados por unos 800 kilómetros de cable. En tres décimas de segundo podía efectuar sumas y restas de 23 dígitos y en cerca de 6 segundos podía multiplicar números de 23 dígitos. Fue retirado en 1959.

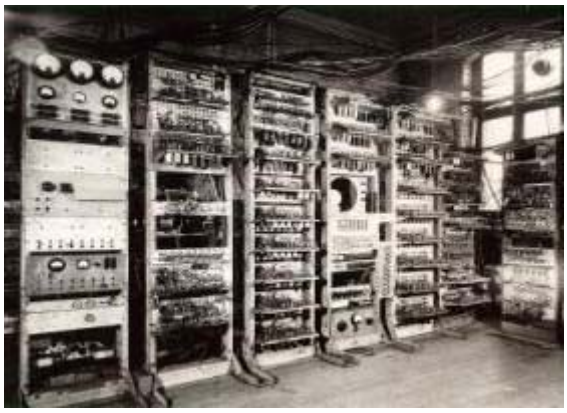


Figura 3. Mark I

Tanto el Mark I como la Segunda Guerra Mundial, desempeñaron un papel clave en el desarrollo de las computadoras. El Mark I aportó los ingredientes tecnológicos básicos mientras que la segunda guerra mundial con sus inmensas demandas de mano de obra y máquinas, creó la necesidad. El resultado fue el "Integrador y Calculador Numérico Electrónico", más conocido como el "ENIAC".

Figura 4. ENIAC

Terminado en 1946, el ENIAC fue creado para el ejército norteamericano en la escuela Moore de Ingeniería Eléctrica, de la Universidad de Pennsylvania. Sus creadores fueron un estudiante graduado, J. Presper Eckert, y un físico, el Dr. John W. Mauchly. Juntos eliminaron la necesidad de las partes que se movían mecánicamente en la computadora central. En su lugar, adaptaron circuitos eléctricos de gatillo "flip-flop" y "pulsaciones" electrónicas para conectar o desconectar tubos al vacío, como interruptores.

Como las interrupciones de este tipo podían hacerse miles de veces más rápido que los aparatos electro-mecánicos, el ENIAC constituyó un gran inicio hacia el desarrollo de las computadoras modernas.

El último paso para completar el concepto de la computadora de hoy, fue el desarrollo del concepto de máquina almacenadora de programas. Este paso se dio a fines de los cuarenta, después de que el célebre matemático hungaro-norteamericano Dr. John Von Neumann sugirió que las instrucciones de operación, así como los datos, se almacenaran de la misma manera en la "memoria" de la computadora. Además, aportó la idea de hacer que la computadora modificara sus propias instrucciones de acuerdo con un control programado. Las ideas de Von Newman fueron fundamentales para los desarrollos posteriores y se le considera el padre de las computadoras. Desde entonces, se ha tratado de modificar, mejorar y apresurar estos conceptos, en fin, de hacer computadoras cada vez más eficientes.

Figura 5. Computador electrónico – IBM 360

Existen básicamente dos tipos básicos de computadoras: las análogas y las digitales. También existen sistemas llamados híbridos que emplean elementos tanto análogos como digitales.
Los fenómenos que se comportan en forma continua reciben el nombre de analógico por ejemplo: la altura de una columna de mercurio en un termómetro clínico, puede variar entre las marcas de treinta y cuarenta y cinco grados y en todo momento puede estar en cualquier punto intermedio de la escala, lo mismo ocurre con un voltaje eléctrico o la rotación angular de un eje. En una computadora análoga los números están representados por cantidades físicas continuamente variables como las anteriores. Tales máquinas tienen aplicaciones físicas e industriales que representan procesos físicos que ocurren con el paso del tiempo.

Existe otro tipo de fenómenos ejemplo: si se averigua la cantidad de ventanas de un edificio se llegará a la conclusión de que son un número exacto como 90 y que no puede haber 90 y media. Estos fenómenos reciben el nombre de digitales porque dan la idea de que se pueden cuantificar con los dedos de la mano. La computadora digital opera con números representados directamente en forma "digital". Tales computadoras son las más extensamente usadas y pueden aplicarse en todos los campos que requieren operaciones aritméticas y manejo de información.

1.2.2. *La fabricación en serie (generaciones de computadores)*

El concepto de generación resulta un tanto especial: aunque tiene relación con la modernización de la tecnología constructiva y de componentes, reviste un carácter marcadamente comercial. Adicionalmente, las fronteras entre generaciones no parecen bien definidas y tal confusión es un síntoma más de las frenéticas y agresivas campañas comerciales llevadas a cabo por las firmas constructoras para colocar sus computadores. Se suelen considerar cinco generaciones:

Generación Cero: En la cuál el hombre construyo máquinas, usando dispositivos mecánicos tales como ruedas dentadas y piñones, para realizar básicamente operaciones aritméticas como suma, resta, multiplicaciones y divisiones. Estas máquinas se conocen con el nombre de **Máquinas Aritméticas**.

Primera Generación: En la cuál el hombre construyó máquinas de cálculo para tareas muy específicas como investigación y militares, usando dispositivos electro-mecánicos como relés y tubos de vacío los cuales dieron paso a los elementos transistorizados. Las máquinas de cómputo de esta generación tenían pocas facilidades de programación. La comunicación se establecía en lenguaje de máquina (lenguaje binario). Estos aparatos eran grandes y costosos.

Figura 6. Folleto promocional de una máquina aritmética

Segunda Generación: En la cuál el hombre construyó computadores (máquinas de cálculo), basados en el transistor. Los computadores de esta generación tienen propósito general, no son usados únicamente para la investigación y el aspecto militar, ya son usados en el arte, la economía y la industria. Se programaban en nuevos lenguajes llamados "de alto nivel".

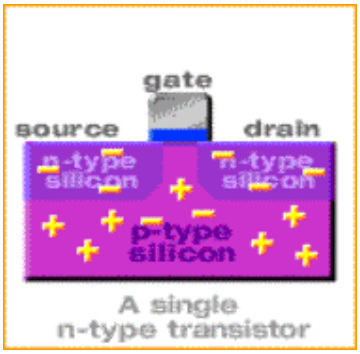


Figura 7. Dibujo esquemático de un transistor

Tercera Generación: En la cual el hombre construye diferentes tipos de computadores, basados en el desarrollo de los circuitos integrados. En esta generación se desarrollan los primeros programas de software de tipo específico.

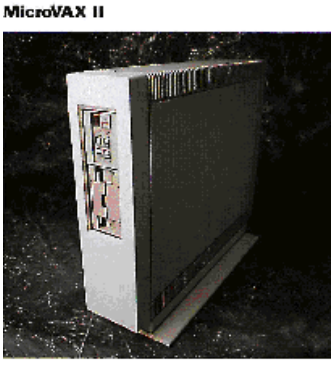


Figura 8. Computador Electrónico -VAX

Cuarta Generación: En la cual el hombre construye computadores de tamaño pequeño pero de gran capacidad, llamados microcomputadores, los cuales están basados en el microprocesador. La aparición del microprocesador se debe a los desarrollos hechos en la tecnología VLSI (Integración de Gran Escala). En esta generación se hacen aplicaciones de software orientadas al usuario final.

Figura 9. El Apple-II, Primer Computador Personal

Quinta Generación: En la cual el hombre construye computadores con más de un procesador, cada uno con una tarea específica como procesamiento de imagen y procesamiento de sonido, y/o con una tarea en común. Adicionalmente se construyen computadores capaces de auto-configurarse, auto-programarse, etc. En esta generación se desarrolla software tanto genérico como específico.

Figura 10. Computador con múltiples procesadores.

1.3. ESTRUCTURA DE UN COMPUTADOR

Un **COMPUTADOR** es una máquina que realiza cálculos de manera automática. Se divide fundamentalmente en dos partes: el hardware y el software. El **HARDWARE** es la parte física de un computador, es decir, la parte que realiza los cálculos. El **SOFTWARE** es la parte lógica del computador, es decir, la parte que le dice al hardware qué hacer. Usando una metáfora se puede decir que: *"Un computador es como un ser humano: el hardware es el cuerpo y el software es la mente"*

1.3.1. *Arquitectura de hardware*

Un computador desde la perspectiva del hardware, está constituido por una serie de dispositivos cada uno con un conjunto de tareas definidas. Los dispositivos de un computador se dividen según la tarea que realizan en: dispositivos de entrada, dispositivos de salida, dispositivos de comunicaciones, dispositivos de almacenamiento y dispositivo de cómputo.

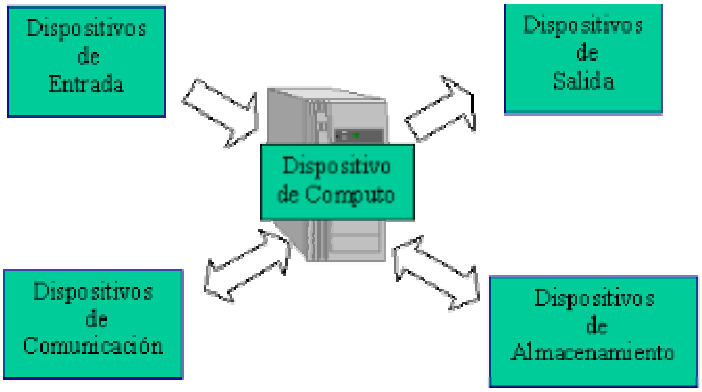


Figura 11. Arquitectura de Hardware

Dispositivos de entrada: Son aquellos que permiten el ingreso de datos a un computador. Entre estos se cuentan: teclados, ratones, scaners, micrófonos, cámaras fotográficas, cámaras de video, controles de juegos, lápices ópticos, y guantes de realidad virtual.

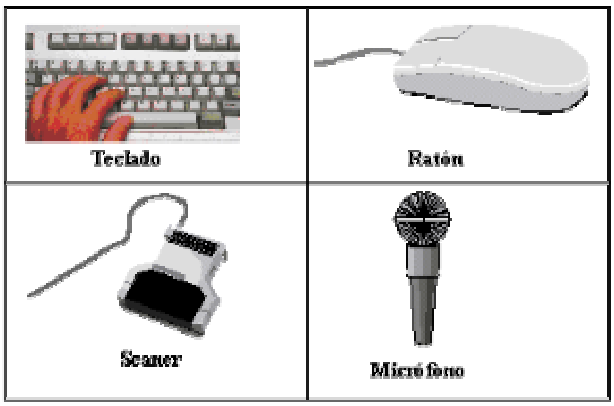


Figura 12. Dispositivos de Entrada

Dispositivos de salida. Son aquellos que permiten mostrar información almacenada o procesada por el computador. Entre otros están: las pantallas de video, impresoras, audífonos, plotters, guantes de realidad virtual, gafas y cascos virtuales.

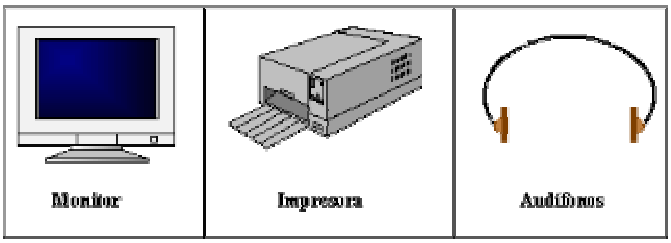


Figura 13. Dispositivos de Salida

Dispositivos de almacenamiento. Son aquellos en los cuales el computador puede guardar información y de los cuales puede obtener información previamente almacenada. Entre otros están los discos flexibles, discos duros, unidades de cinta, CD-ROM, CD-ROM de re-escritura y DVD.

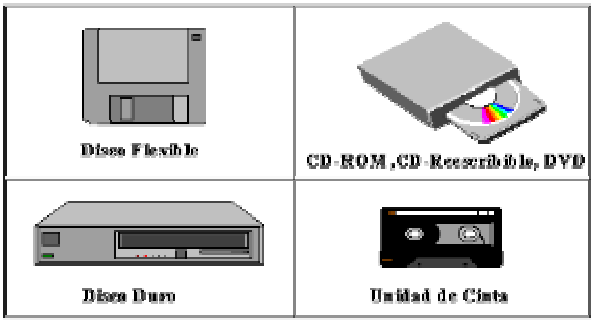


Figura 14. Dispositivos de Almacenamiento.

Dispositivos de comunicación: Son aquellos que le permiten a un computador comunicarse con otros. Entre estos se cuentan los modems y tarjetas de red.

Figura 15. Modem

Dispositivo de cómputo: Es la parte del computador que realiza todos los cálculos y tiene el control sobre los demás dispositivos. Está formado por tres elementos fundamentales: la unidad central de proceso, la memoria y el bus de datos y direcciones.

Figura 16. Diagrama esquemático del dispositivo de computo

La **unidad central de proceso (UCP)[1]:** es el 'cerebro' del computador, está encargada de realizar todos los cálculos, utilizando para ello la información almacenada en la memoria y de controlar los demás dispositivos, procesando las entradas y salidas provenientes y/o enviadas a los mismos. Mediante el bus de datos y direcciones, la UCP se comunica con los diferentes dispositivos enviando y obteniendo tales entradas y salidas.

Para realizar su tarea la unidad central de proceso dispone de una unidad aritmético lógica, una unidad de control, un grupo de registros y opcionalmente una memoria caché para datos y direcciones.

La ***unidad aritmético lógica (UAL)***[2] es la encargada de realizar las operaciones aritméticas y lógicas requeridas por el programa en ejecución, la ***unidad de control*** es la encargada de determinar las operaciones e instrucciones que se deben realizar, el ***grupo de registros*** es donde se almacenan tanto datos como direcciones necesarias para realizar las operaciones requeridas por el programa en ejecución y la *memoria caché* se encarga de mantener direcciones y datos intensamente usados por el programa en ejecución.

[1] La unidad central de proceso es más conocida como CPU por sus siglas en inglés Central Process Unit.

[2] La unidad aritmético lógica es más conocida como ALU por sus siglas en inglés Arithmetic Logic Unit.

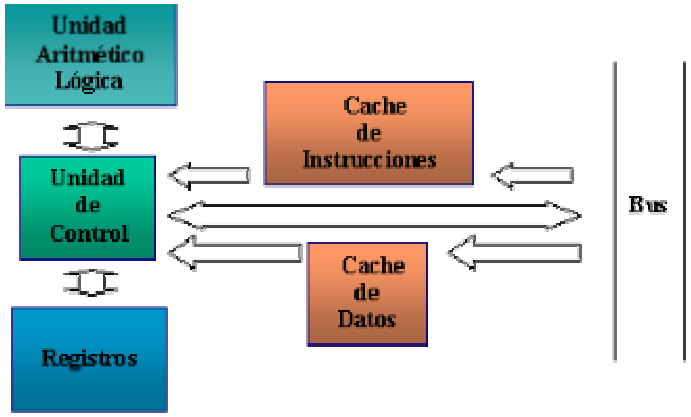


Figura 17. Unidad Central de Proceso.

La **memoria** está encargada de almacenar toda la información que el computador está usando, es decir, la información que es accedida (almacenada y/o recuperada) por la UCP y por los dispositivos.

La unidad de medida de memoria es el byte, constituido por 8 bits (ceros o unos). Cada byte tiene asignada una dirección de memoria, para poder ser accedida por la UCP. Para la interpretación de la información que está en memoria, como datos o comandos o instrucciones, se utilizan códigos que la UCP interpreta para llevar a cabo las acciones deseadas por el usuario.

Existen diferentes tipos de memoria, entre las cuales se encuentran las siguientes:

- **RAM** (Random Access Memory): Memoria de escritura y lectura, es la memoria principal del computador. El contenido solo se mantiene mientras el computador está encendido.

- **ROM** (Read Only Memory): Memoria de solo lectura, es permanente y no se afecta por el encendido o apagado del computador. Generalmente almacena las instrucciones que le permiten al computador iniciarse y cargar (poner en memoria RAM) el sistema operativo.

- **Caché**: Memoria de acceso muy rápido, usada como puente entre la UCP y la memoria RAM, para evitar las demoras en la consulta de la memoria RAM.

El **bus de datos y direcciones** permite la comunicación entre los elementos del computador. Por el bus de datos viajan tanto las instrucciones como los datos de un programa y por el bus de direcciones viajan tanto las direcciones de las posiciones de memoria donde están instrucciones y datos, como las direcciones lógicas asignadas a los dispositivos.

1.3.2. *Arquitectura de software*

Un computador desde la perspectiva del software, está constituido por:

- Un sistema operativo.
- Un conjunto de lenguajes a diferente nivel con los cuales se comunica con el usuario y con sus dispositivos. Entre estos están los lenguajes de máquina, los ensambladores y los de alto nivel.
- Un conjunto de aplicaciones de software.
- Un conjunto de herramientas de software.

Software: Es un conjunto de instrucciones que le dicen al hardware que hacer. El hardware por si solo no puede hacer nada.

Lenguaje de programación: Es un conjunto de reglas y estándares que es utilizado para escribir programas de computador (software), que puedan ser entendidos por él.

Programa: Es la representación de algún software en un lenguaje de programación específico.

1.3.2.1. ***Sistema Operativo***

Es el software encargado de administrar los recursos del sistema. Adicionalmente, ofrece un conjunto de comandos para interactuar con la máquina.

Los sistemas operativos pueden ser escritos en lenguaje de alto nivel (UNIX fue escrito en C), en lenguaje ensamblador y/o en lenguaje máquina. Algunos de los sistemas operativos más conocidos son DOS, UNIX, LINUX y las distintas versiones de Microsoft Windows.

1.3.2.2. ***Lenguajes a diferente nivel***

1.3.2.2.1 Lenguaje de Máquina

Es el único lenguaje que entiende el hardware (máquina) y usa exclusivamente el sistema binario (ceros y unos). Este lenguaje es específico para cada hardware (procesador, dispositivos, etc.).

El programa (tanto códigos de instrucción como datos) es almacenado en memoria. La estructura de una instrucción en lenguaje máquina es la siguiente:

CODIGO ARGUMENTO(S)

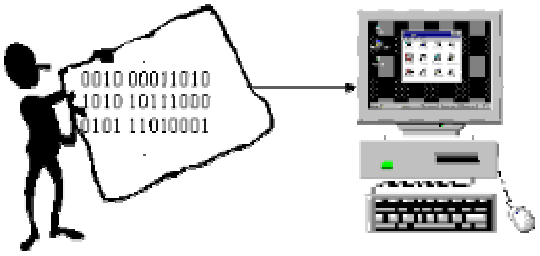


Figura 18. Lenguaje de máquina

1.3.2.2.2 Lenguaje Ensamblador

Es un lenguaje que usa mnemónicos (palabras cortas escritas con caracteres alfanuméricos), para codificar las operaciones. Los datos y/o direcciones son codificados generalmente como números en un sistema hexadecimal. Generalmente es específico (aunque no único) para cada lenguaje de máquina. La estructura de una instrucción en este lenguaje es la siguiente:

MNEMONICO ARGUMENTO(S)

Un **ENSAMBLADOR** es un software, generalmente escrito en lenguaje de máquina, que es capaz de traducir de lenguaje ensamblador a lenguaje de máquina.

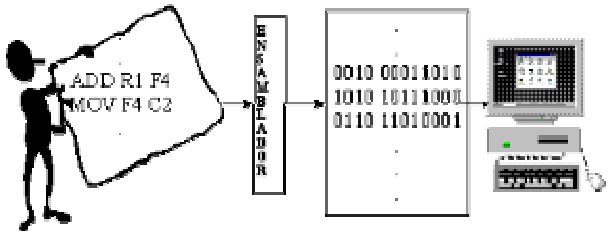


Figura 19. Lenguaje Ensamblador.

1.3.2.2.3 Lenguaje de Alto Nivel

Es un lenguaje basado en una estructura gramatical para codificar estructuras de control y/o instrucciones. Cuenta con un conjunto de palabras reservadas (escritas en lenguaje natural). Estos lenguajes permiten el uso de símbolos aritméticos y relacionales para describir cálculos matemáticos, y generalmente representan las cantidades numéricas mediante sistema decimal.

Gracias a su estructura gramatical, estos lenguajes permiten al programador olvidar el direccionamiento de memoria (donde cargar datos y/o instrucciones en la memoria), ya que este se realiza mediante el uso de conceptos como el de variable.

Los **COMPILADORES** e I**NTERPRETES** son software capaz de traducir de un lenguaje de alto nivel al lenguaje ensamblador específico de una máquina. Los primeros toman todo el programa en lenguaje de alto nivel, lo pasan a lenguaje ensamblador y luego lo ejecutan. Los últimos toman instrucción por instrucción, la traducen y la van ejecutando.

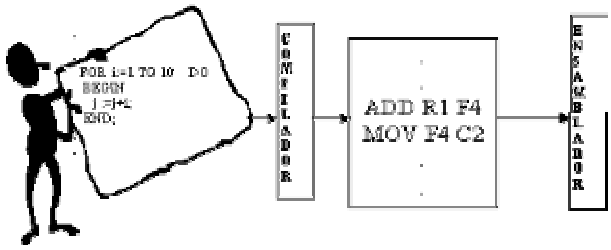


Figura 20. Lenguaje de Alto Nivel.

1.3.2.3. Aplicaciones

Una **APLICACION** es un software construido para que el computador realice una tarea específica y con el cual no se puede construir otro software. Ejemplos de aplicaciones son los procesadores de texto como *Microsoft Word* y *Word Perfect* y las hojas electrónicas de cálculo como *Microsoft Excel* y *Lotus*.

1.3.2.4. Herramientas

Una **HERRAMIENTA** es un software construido especialmente para el desarrollo de nuevo software, (tanto de aplicaciones como de herramientas). Ejemplos de herramientas son los compiladores como Turbo C, Turbo Pascal y Dev C++, las herramientas CASE y los ambientes integrados de desarrollo.

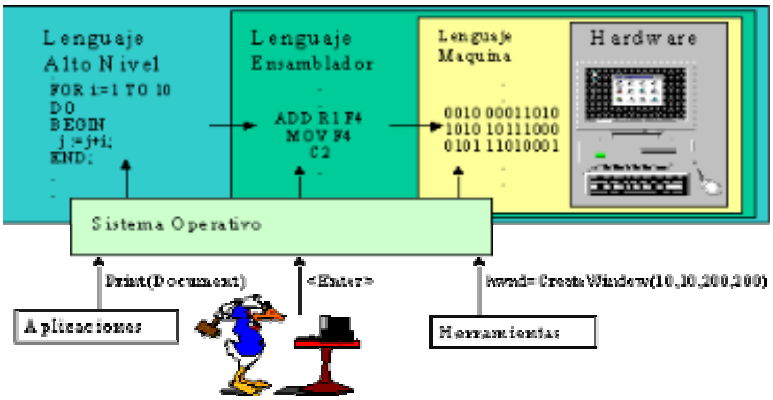


Figura 21. Arquitectura de Software Completa.

1.4. RESUMEN.

Las computadoras son un avance de los sencillos utensilios que el nombre uso para contar a principios de su historia.
El ábaco fué el primer calculador digital.
La máquina de Pascal inventada en 1642 fue la primera máquina calculadora.
La verdadera precursora de las computadoras electrónicas fue el "Motor de Diferencias" construida en 1922 por Charles Babbage.
En 1937 la IBM ayudó a Howard Aiken a crear el Mark I.
El Mark I y la Segunda Guerra Mundial desempeñaron papeles claves en el desarrollo de las computadoras norteamericanas.
El resultado fue el Eniac terminado en 1946 creado por un estudiante graduado Prespert Eckert y el físico John Mauchly.
Afines de los años 40 John Von Nuemann concibió la idea de que en la memoria coexistan datos con instrucciones. Alrededor de este concepto y el de control programado gira toda la evolución posterior de la industria de las computadoras.
Existen computadoras analógicas y digitales. Son computadoras digitales aquellas que manejan la información de manera discreta y son analógicas las que trabajan por medio de funciones continuas, generalmente representación de señales eléctricas.

El computador está constituido por Hardware y Software. El hardware es la parte física y el software la parte lógica.

Los componentes del hardware son: dispositivos de entrada, dispositivos de salida, dispositivos de almacenamiento, dispositivos de comunicación y dispositivo de cómputo (Unidad central de proceso, memoria, bus de datos y direcciones).

Desde la perspectiva de software el computador está constituido por: sistema operativo, conjunto de lenguajes a diferente nivel (lenguajes de máquina, ensambladores y alto nivel), aplicaciones y herramientas.

1.5. BIBLIOGRAFÍA.

- BECERRA C., *Algoritmos: Conceptos Básicos*, 4ª edición, 1998.

1.6. LECTURAS COMPLEMENTARIAS.

Cursos y manuales, artículos, revisiones, bricolaje, compañías de hardware, componentes de la página *http://www.mundopc.net/hardware/*

2. INTRODUCCIÓN A LOS ALGORITMOS

2.1. OBJETIVOS

Presentar una definición informal del concepto de algoritmo que será adoptada para el resto del texto. Igualmente, clarificar los tipos de problemas que pueden ser resueltos de manera algorítmica, y ejemplificar el trabajo que implica resolver un problema mediante un programa de computador. Los ejercicios del final del capítulo ofrecen una oportunidad para que el estudiante consolide estos conceptos y se ejercite en relacionar la información dada en el enunciado de un problema, con la información desconocida.

También, se presentan los elementos básicos de los dos principales formalismos que serán utilizados para especificar algoritmos que se quieren ejecutar en un computador: el pseudocódigo y los diagramas de flujo.

2.2. INTRODUCCIÓN

Diariamente el ser humano trata de darle solución a cada problema que se le presenta, o de mejorar las soluciones disponibles. Para algunos problemas fundamentales ha encontrado soluciones brillantes que consisten en una serie de acciones, que siempre que se realicen de manera ordenada y precisa conducen a la respuesta correcta. Algunas de esas soluciones han requerido el trabajo, la inteligencia y la persistencia de muchas generaciones. Hoy la sociedad cuenta con ese legado de soluciones, además de las máquinas capaces de ejecutarlas precisa y velozmente. Esas máquinas maravillosas, de las cuales trata el capítulo 1, son los computadores.

2.3. ALGORITMOS

La palabra algoritmo se deriva de **Al-khôwarizmi**, un matemático y astrónomo del siglo IX quien al escribir un tratado sobre manipulación de números y ecuaciones, el ***Kitab al-jabr w'almugabala***, usó en gran medida la noción de lo que se conoce hoy como algoritmo.

Un ALGORITMO es una secuencia finita 'bien definida' de tareas 'bien definidas', cada una de las cuales se puede realizar con una cantidad finita de recursos. Se dice que una tarea esta 'bien definida', si se saben de manera precisa las acciones requeridas para su realización. Aunque los recursos que debe utilizar cada tarea deben ser finitos estos no están limitados, es decir, si una tarea bien definida requiere una cantidad inmensa (pero finita) de algún recurso para su realización, dicha tarea puede formar parte de un algoritmo. Además, se dice que una secuencia de tareas está 'bien definida' si se sabe el orden exacto en que deben ejecutarse.

A lo largo de este libro, se considerará solo esta definición informal de algoritmo. En matemáticas se usa una definición formal que está fuera del alcance de este texto.

EJECUTAR un algoritmo consiste en realizar las tareas o instrucciones que lo conforman, en el orden especificado y utilizando los recursos disponibles. Hoy se cuenta con máquinas que realizan esta labor, pero se requiere que los algoritmos que ejecutan se escriban en un lenguaje especial. Usar esos lenguajes especiales para especificar algoritmos se llama programación de computadores.

2.3.1. *Características de un algoritmo*

Las características que debe poseer una secuencia de tareas para considerarse algoritmo son: precisión, de finitud y finitud.

Precisión	De finitud o Determinismo	Finitud
Hay un orden preciso en el cual deben ejecutarse las tareas que conforman el algoritmo.	Todas las veces que se realicen las tareas o pasos de un algoritmo, con las mismas condiciones iniciales, se deben obtener resultados idénticos.	El algoritmo debe terminar en algún momento y debe usar una cantidad finita de recursos.

2.3.2. *Estructura básica de un algoritmo*

En esencia un algoritmo está constituido por los siguientes tres elementos:

Datos	Instrucciones	Estructuras de control
Para almacenar información: datos de entrada, de salida o intermedios.	Las acciones o procesos que el algoritmo realiza sobre los datos.	Las que determinan el orden en que se ejecutarán las instrucciones del algoritmo.

En el capítulo siguiente se presentan de manera completa los conceptos de dato e instrucción, y en el capítulo cuatro se describe el concepto de estructura de control.

2.3.3. *Ejemplos de algoritmos*

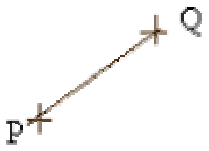
Enseguida se dan varios ejemplos de algoritmos, algunos de los cuales no son susceptibles de ejecutarse por medio de un computador; en cambio, son más bien recetas que se usan para resolver problemas cotidianos.

PROBLEMA UNO: Un estudiante se encuentra en su casa (durmiendo) y debe ir a la universidad (a tomar la clase de programación!), ¿qué debe hacer?
ALGORITMO:

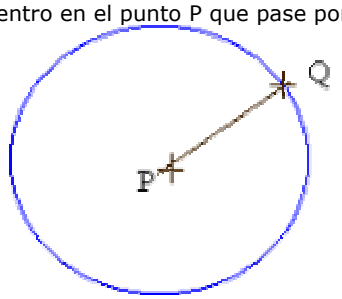
Inicio
PASO 1. Dormir
PASO 2. **Hacer** 1 **hasta** que suene el despertador (o lo llame la mamá).
PASO 3. Mirar la hora.
PASO 4. ¿Hay tiempo suficiente?
PASO 4.1. Si hay, **entonces**
PASO 4.1.1. Bañarse.
PASO 4.1.2. Vestirse.
PASO 4.1.3. Desayunar.
PASO 4.2. Sino,
PASO 4.2.1. Vestirse.
PASO 5. Cepillarse los dientes.
PASO 6. Despedirse de la mamá y el papá.
PASO 7. ¿Hay tiempo suficiente?
PASO 7.1. Si hay, **entonces**
PASO 7.1.1. Caminar al paradero.
PASO 7.2. Sino, Correr al paradero.
PASO 8. Hasta que pase un bus para la universidad **hacer**:
PASO 8.1. Esperar el bus
PASO 8.2. Ver a las demás personas que esperan un bus.
PASO 9. Tomar el bus.
PASO 10. Mientras no llegue a la universidad **hacer**:
PASO 10.1. Seguir en el bus.
PASO 10.2. Pelear mentalmente con el conductor.
PASO 11. Timbrar.
PASO 12. Bajarse.
PASO 13. Entrar a la universidad.
Fin

PROBLEMA DOS: Sean *P=(a,b)* y *Q=(c,d)* los puntos extremos de un segmento de recta. Encontrar un segmento de recta perpendicular al anterior, que pase por su punto medio.
ALGORITMO:

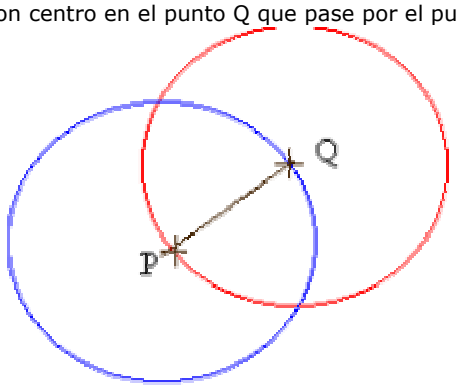
Inicio



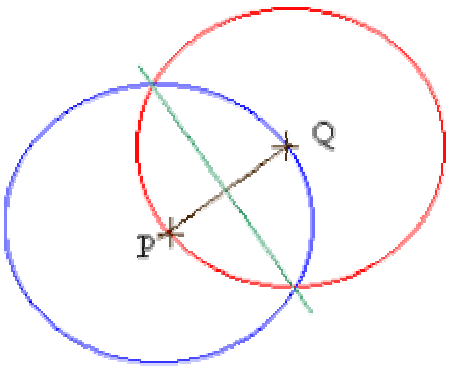
PASO 1. Trazar un círculo con centro en el punto P que pase por el punto Q.



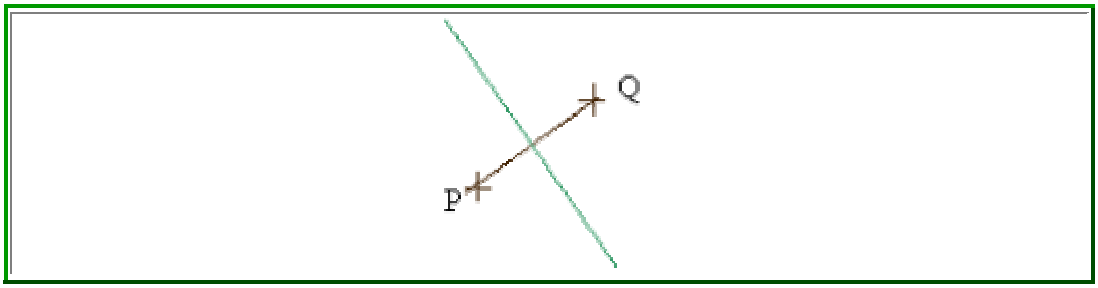
PASO 2. Trazar un círculo con centro en el punto Q que pase por el punto P.



PASO 3. Trazar un segmento de recta entre los puntos de intersección de las circunferencias trazadas en los pasos 1 y 2.



Fin. El segmento de recta trazada es el buscado.



PROBLEMA TRES: Realizar la suma de dos números enteros positivos.

ALGORITMO:

Inicio
PASO 1. En una hoja de papel, escribir los números el primero arriba del segundo, de tal manera que las unidades, decenas, centenas, etc., de los números queden alineadas. Trazar una línea debajo del segundo número.
PASO 2. Empezar por la columna más a la derecha.
PASO 3. Sumar los dígitos de dicha columna.
PASO 4. Si la suma es mayor a 9 anotar el número que corresponde a las decenas encima de la siguiente columna a la izquierda y anotar debajo de la línea las unidades de la suma. Si no es mayor anotar la suma debajo de la línea.
PASO 5. Si hay más columnas a la izquierda, pasar a la siguiente columna a la izquierda y volver a 3.
PASO 6. El número debajo de la línea es la solución.
Fin

PROBLEMA CUATRO: Cambiar la rueda pinchada de un automóvil teniendo un gato mecánico en buen estado, una rueda de reemplazo y una llave inglesa.

ALGORITMO:

Inicio
PASO 1. Aflojar los tornillos de la rueda pinchada con la llave inglesa.
PASO 2. Ubicar el gato mecánico en su sitio.
PASO 3. Levantar el gato hasta que la rueda pinchada pueda girar libremente.
PASO 4. Quitar los tornillos y la rueda pinchada.
PASO 5. Poner rueda de repuesto y los tornillos.
PASO 6. Bajar el gato hasta que se pueda liberar.
PASO 7. Sacar el gato de su sitio.
PASO 8. Apretar los tornillos con la llave inglesa.
Fin

PROBLEMA CINCO: Encontrar los números primos entre 1 y 50.

ALGORITMO:

Inicio
PASO 1. Escribir los números de 1 al 50
PASO 2. Tachar el número 1 ya que no es primo.
PASO 3. Para k entre 2 y el entero más cercano por debajo de la raíz cuadrada de 50, si el número k no esta tachado, tachar los múltiplos del número k, sin tachar el número k.
PASO 4. Los números que no se tacharon son los números primos entre 1 y 50.
Fin

2.3.4. *Representación de Algoritmos*

Cuando se quiere que un computador ejecute un algoritmo es indispensable, por lo menos hasta hoy, representar ese algoritmo mediante algún formalismo. Las técnicas utilizadas más comúnmente para la representación de algoritmos son:

2.3.4.1. **A Diagramas de flujo**

Se basan en la utilización de diversos símbolos geométricos para representar operaciones específicas. Se les llama diagramas de flujo porque los símbolos utilizados se conectan por medio de flechas para indicar la secuencia que sigue la ejecución de las operaciones.

En la sección de Programación Estructurada se explica la forma de construir diagramas de flujo utilizando esta técnica.

2.3.4.2. ***Pseudocódigo***

Es un lenguaje de especificación de algoritmos. Tiene asociado un léxico (conjunto de palabras), una sintaxis (reglas gramaticales) y una semántica precisa (significado), de manera análoga a un lenguaje natural como el castellano. El uso de tal lenguaje hace relativamente fácil el paso de codificación final del algoritmo (esto es, la traducción a un lenguaje de programación).

La ventaja del pseudocódigo es que le permite al programador concentrarse en la lógica y en las estructuras de control del algoritmo que quiere diseñar, sin preocuparse de las reglas de un lenguaje específico de programación, que normalmente incluyen infinidad de detalles. Es también fácil modificar el pseudocódigo si se descubren errores o anomalías en la lógica del programa; además de lo anterior, es fácil su traducción a lenguajes de programación como *Pascal*, *C, Java* o *Basic*.

El pseudocódigo utiliza palabras reservadas (similares a sus homónimos en los lenguajes de programación) para representar las acciones y estructuras de control, tales como *inicio*, *fin*, *si-entonces-sino*, *mientras*, etc.

El formato general de un algoritmo expresado mediante pseudocódigo es el siguiente:

```
[<definición de registros o tipos de datos>]
[constantes <declaración constantes>]
[variables <declaración variables globales del programador>]
[<definición de funciones y procedimientos>]

procedimiento principal()
[constantes <declaración constantes>]
[variables <declaración variables>]
inicio
   /*bloque de instrucciones*/
fin_procedimiento
```

Los detalles completos de este formato no son importantes a esta altura del libro y cada uno de ellos será cubierto adecuadamente en próximos capítulos. Por ahora basta con saber que en las primeras secciones se definen las constantes y variables que se usan dentro del algoritmo; posteriormente se especifican algunos procedimientos y funciones; y finalmente, se escriben las instrucciones que conforman el algoritmo principal.

También conviene mencionar que la información que se manipula dentro de un algoritmo se agrupa en conjuntos llamados TIPOS. Hay unos tipos predefinidos en el pseudocódigo: **entero** (conjunto de valores enteros), **real** (conjunto de valores reales), **caracter** (conjunto de

símbolos representables en el computador) y **booleano** (los dos valores lógicos: falso y verdadero).

2.4. PROBLEMAS

Un algoritmo tiene sentido cuando constituye una forma efectiva de resolver un problema interesante. Se tiene un problema cuando se desea encontrar uno o varios objetos desconocidos (ya sean estos números, símbolos, diagramas, figuras, u otras cosas), que cumplen condiciones o relaciones, previamente definidas, respecto a uno o varios objetos conocidos. De esta manera, solucionar un problema es encontrar los objetos desconocidos de dicho problema.

2.4.1. *Clasificación de problemas*

Los problemas se clasifican, de acuerdo a la existencia de una solución, en solubles, no solubles e indecidibles.

- Un problema se dice **SOLUBLE** si se sabe de antemano que existe una solución para él.
- Un problema se dice **INSOLUBLE** si se sabe que no existe una solución para él.
- Un problema se dice **INDECIDIBLE** si no se sabe si existe o no existe solución algorítmica para él.

A su vez, los problemas solubles se dividen en dos clases: los algorítmicos y los no algorítmicos.

- Un problema se dice **ALGORÍTMICO**[3] si existe un algoritmo que permita darle solución.
- Un problema se dice **NO ALGORÍTMICO** si no es susceptible del resolver mediante un algoritmo.

2.4.2. *Ejemplos de problemas*

PROBLEMA UNO: Sean *P=(a,b)* y *Q=(c,d)* los puntos que definen un segmento de recta. Encontrar un segmento de recta perpendicular al anterior, que pase por su punto medio.

ELEMENTOS CONOCIDOS	Los puntos *P* y *Q*.
ELEMENTOS DESCONOCIDOS	Un segmento de recta.
CONDICIONES	El segmento de recta debe pasar por el punto medio entre *P* y *Q*, y debe ser perpendicular a la recta trazada entre *P* y *Q*
TIPO DE PROBLEMA	Soluble-algorítmico. Es soluble por que ya existe un algoritmo que permite encontrar la solución del mismo. Este algoritmo fue presentado en la sección anterior.

PROBLEMA DOS: De las siguientes cuatro imágenes, ¿cuál es la más llamativa?

[3] En este curso solo se tratarán problemas algorítmicos




ELEMENTOS CONOCIDOS	Las cuatro imágenes.
ELEMENTOS DESCONOCIDOS	La imagen más llamativa.
CONDICIONES	La imagen buscada es la más llamativa para quien resuelve el problema.
TIPO DE PROBLEMA	Soluble-no algorítmico. La solución de este problema existe y es alguna de las cuatro imágenes presentadas; pero no existe un algoritmo que permita determinar cuál es, ya que el concepto de "imagen más llamativa" no está definido de manera precisa.

PROBLEMA TRES: Un granjero tiene cincuenta animales entre conejos y gansos. Si la cantidad de patas de los animales es ciento cuarenta, ¿cuántos conejos y cuántos gansos tiene el granjero?

ELEMENTOS CONOCIDOS	La cantidad total de animales (50), cantidad de patas totales (150), número de patas de los gansos y número de patas de los conejos.
ELEMENTOS DESCONOCIDOS	La cantidad de conejos y la cantidad de gansos.
CONDICIONES	La suma de los conejos y los gansos es igual a 50. La suma de las patas de los conejos y de los gansos es igual a 150.
TIPO DE PROBLEMA	Soluble-algorítmico.

PROBLEMA CUATRO: ¿Existe, en la expansión decimal de π, una secuencia de tamaño *n* para un número natural *n* dado?

ELEMENTOS CONOCIDOS	El número n.
ELEMENTOS DESCONOCIDOS	Un valor de verdad (falso o verdadero).
CONDICIONES	Verdadero si existe en la expansión decimal de π una secuencia de tamaño *n* del número *n*, para todo número natural *n*, Falso en otro caso.
TIPO DE PROBLEMA	Indecidible. Este problema es indecidible por que si en el primer millón de dígitos de π no se encuentra una secuencia como la buscada, nada garantiza que en el siguiente millón de dígitos no se encuentre tal secuencia. Pero si no se encuentra en el segundo millón, nada garantiza que no se encuentre después, o no se encuentre. De esta manera no se puede decidir si existe o no existe tal secuencia.

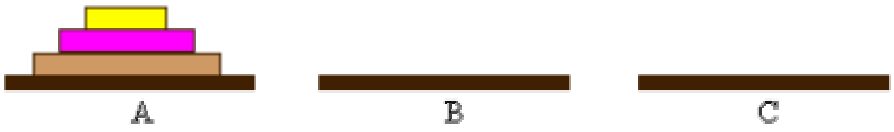
PROBLEMA CINCO: Realizar la suma de dos números naturales

ELEMENTOS CONOCIDOS	Dos números naturales
ELEMENTOS DESCONOCIDOS	Un número natural
CONDICIONES	El número desconocido debe ser igual a la suma de los dos números dados.
TIPO DE PROBLEMA	Soluble-algorítmico. Es soluble por que ya existe un algoritmo que permite encontrar la solución del mismo. Este algoritmo fue presentado en la sección anterior. Además, es un algoritmo que se enseña en la primaria.

PROBLEMA SEIS: Una partícula se mueve en el espacio de manera aleatoria, si en el instante de tiempo *t* se encuentra en la posición *x*, ¿cuál será la posición exacta de dicha partícula *10* segundos después?

ELEMENTOS CONOCIDOS	Posición en el instante de tiempo t, lapso de tiempo transcurrido (10 Seg.).
ELEMENTOS DESCONOCIDOS	Una posición en el espacio.
CONDICIONES	La partícula se mueve en el espacio de manera aleatoria.
TIPO DE PROBLEMA	Insoluble. No se puede solucionar por que no existe forma de predecir la posición de la partícula, pues su movimiento es aleatorio, es decir, se mueve de manera arbitraria.

PROBLEMA SIETE: Un robot debe trasladar una pila (montón) de cajas desde un punto A hasta un punto C usando, si lo requiere, un punto intermedio B. Las cajas inicialmente están todas en el punto A, ordenadas como muestra la siguiente figura. En cada traslado, el robot sólo puede tomar una caja que esté en la cima de un montón y llevarla a uno de los otros dos puntos. Además, en ningún momento puede poner una caja sobre otra más pequeña. Mirando la figura, ¿cómo puede el robot pasar las tres cajas apiladas en el lugar A, al lugar C.?



ELEMENTOS CONOCIDOS	Número de cajas, posición inicial y posición destino, número de lugares de apilamiento.
ELEMENTOS DESCONOCIDOS	Una secuencia de traslados.
CONDICIONES	La secuencia de traslados solicitada debe respetar las reglas enunciadas: no se puede poner una caja sobre otra más pequeña, solo se puede tomar una caja a la vez y solo la que este más arriba en una pila de cajas.
TIPO DE PROBLEMA	Soluble-algorítmico.

2.5. PROGRAMACIÓN ESTRUCTURADA

La programación estructurada es un estilo de programación de los años sesentas en el cual, la estructura de un algoritmo se hace tan clara como sea posible utilizando tres formas de organizar sus instrucciones:

1. Secuencia
2. Selección
3. Iteración

Estos tres tipos de estructuras de control pueden combinarse para producir programas de computador tan complejos y largos como se quiera.

La lógica y propósito de un algoritmo estructurado puede comprenderse al leerlo de arriba hacia abajo. Esto facilita que el creador de un programa lo comparta con otros programadores, lo dé a conocer, o delegue en otros la labor de modificarlo.

Un programa (o algoritmo) estructurado tiene segmentos o componentes claramente definidos. Además, en cada componente está explícito un punto de inicio o entrada, y un punto de finalización o salida. Tal segmento de denomina un ***programa propio***.

2.5.1. *Teoría de la Programación Estructurada*

Teorema de la estructura.

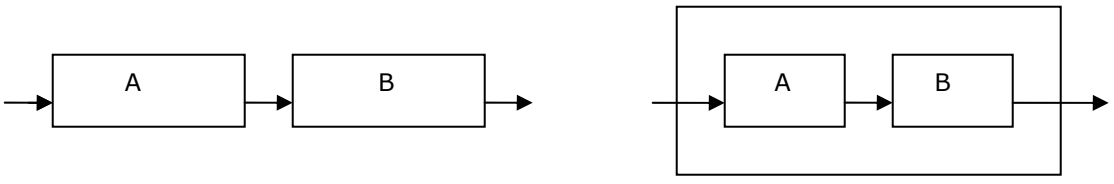
Cualquier programa propio se puede escribir usando solamente las tres estructuras de control: secuencia, selección e iteración.

Un programa propio contempla dos propiedades básicas:

1. Tiene exactamente un punto de entrada y uno de salida
2. Entre el punto de entrada y el de salida hay trayectorias que conducen a cada parte del programa; esto significa que no existen grupos de instrucciones que se ejecuten indefinidamente, ni instrucciones que jamás se vayan a ejecutar.

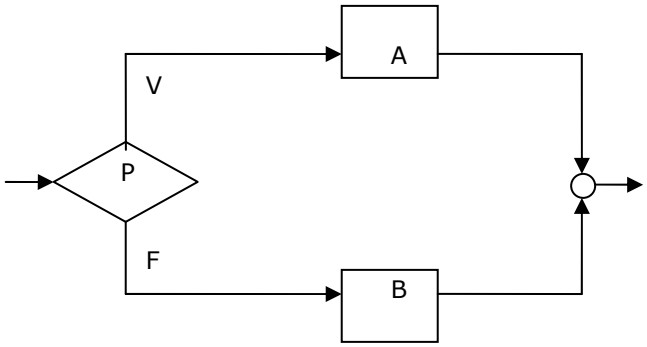
Las tres estructuras de control se ilustran a continuación.

Secuencia: Las instrucciones del programa se ejecutan en el orden en el cual ellas aparecen en el texto del programa como se indica en el siguiente diagrama:

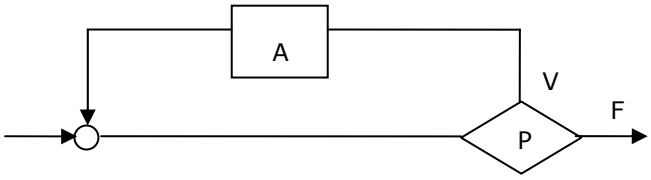


A y B pueden ser instrucciones básicas o compuestas. A y B deben ser ambos programas propios en el sentido ya definido de entrada y salida. La combinación de A y B es también un programa propio y que tiene también una entrada y una salida.

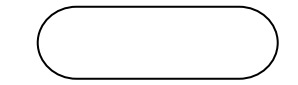
Selección: permite escoger entre dos grupos de instrucciones, de acuerdo a la evaluación de una condición o predicado lógico. Se conoce como estructura SI–ENTONCES–SINO. En el siguiente diagrama, P representa un predicado y, A y B representan grupos de instrucciones. Debe ser claro que si P es falso, se realiza el conjunto de instrucciones B; de lo contrario se realiza A. Las flechas horizontales de más a la izquierda y de más a la derecha representan los puntos de entrada y salida, respectivamente.



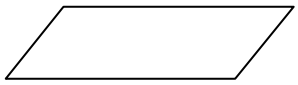
Iteración: Esta forma de control permite repetir varias veces una instrucción o conjunto de instrucciones hasta cuando deje de cumplirse una condición (predicado lógico).
Se conoce como la estructura HACER – MIENTRAS. En el siguiente diagrama, P representa un predicado y A representa un grupo de instrucciones. Debe ser claro que siempre que P sea verdadero, se realiza el conjunto de instrucciones A; de lo contrario se termina la iteración. Las flechas horizontales de más a la izquierda y de más a la derecha representan los puntos de entrada y salida, respectivamente.



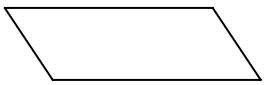
Algunos de los símbolos utilizados en las estructuras para conformar los diagramas de flujo son:



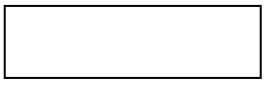
Representa el inicio y el fin de un programa



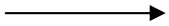
Entrada de datos(lectura)



Salida de datos (impresión)



Proceso (cualquier tipo de operación)

 Indicador de dirección o flujo

2.6. EJERCICIOS

2.6.1. *Ejercicios de Problemas*

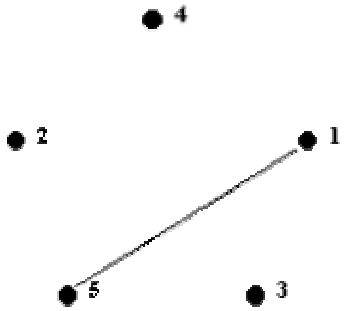
Para los siguientes problemas, determine las variables conocidas, desconocidas, las condiciones y el tipo de problema. Para aquellos problemas algorítmicos desarrollar adicionalmente un algoritmo que permita encontrar una solución.

1. Se tienen dos jarras (*A* y *B*) de capacidades 3 y 7 litros respectivamente, sobre las cuales se pueden efectuar las siguientes acciones: Llenar totalmente cualquiera de las dos jarras, vaciar una de las dos jarras en la otra hasta que la jarra origen este vacía o hasta que la jarra destino este llena y vaciar el contenido de una jarra (este llena o no) en un sifón. ¿Cómo se puede dejar en la jarra A un solo litro utilizando solamente las anteriores acciones?

2. Es cierta o no es cierta la siguiente frase: "Esta frase no es cierta".

3. Si Juan tiene el doble de la edad de Pedro y la suma de las edades de los dos es 33 años, ¿Cuántos años tiene Juan y cuántos tiene Pedro?

4. ¿Qué figura se forma al unir los puntos marcados con números consecutivos con una línea?



5. Calcular de manera exacta el número de átomos del universo.

6. Calcular el costo de una serie de productos comprados en el supermercado.

7. Determinar quien es el mejor jugador de fútbol de toda la historia.

8. Construir un barco de papel.

2.6.2. *Ejercicios de algoritmos*

Para los siguientes problemas construir un algoritmo que los solucione.

EJERCICIO UNO: Buscar en el directorio telefónico, el número de:

a. José González Pérez
b. Pedro Gómez Bernal.
c. Escribir un algoritmo que sirva para buscar a cualquier persona.

EJERCICIO DOS: Calcular el número de días entre las fechas:

a. Enero 17 de 1972 y Julio 20 de 1973
b. Febrero 2 de 1948 y Agosto 11 de 1966

c. Escribir un algoritmo que sirva para calcular la cantidad de días entre dos fechas cualesquiera.

EJERCICIO TRES: Solicitar en préstamo algún libro de una biblioteca.

EJERCICIO CUATRO: Hacer una caja de cartón con tapa de:

a. 20 cm de largo, por 10 cm de ancho y 5 cm de alto.
b. 10 cm de largo, por 30 cm de ancho y 15 cm de alto.
c. Escribir un algoritmo que sirva para construir una caja de cartón con tapa de cualquier tamaño.

EJERCICIO CINCO: Construir un avión de papel.

EJERCICIO SEIS: Calcular manualmente la división de cualquier par de números naturales. El resultado también debe ser un número natural. Escribir un algoritmo para calcular el residuo de la división.

EJERCICIO SIETE: Un juego muy famoso entre dos niños es el de *adivina mi número*, el cual consiste en que cada niño trata de adivinar el número pensado por el otro niño. Dicho número generalmente está entre 1 y 100. Las reglas del juego son las siguientes:

a. Cada niño posee un turno en el que trata de averiguar el número del otro.
b. En su turno el primer niño pregunta si un número que dice es el pensado por el segundo.
c. Si el número que ha dicho el primer niño es el que pensó el segundo, este último debe informarle al primero que ganó.
d. Si el número no es el segundo niño debe decir si su número pensado es menor o mayor al que el primer niño dijo.
e. Luego el segundo niño tiene su turno y de esta manera se van intercalando hasta que alguno de los dos gane. Desarrollar un algoritmo para jugar adivina mi número.

EJERCICIO OCHO: Una balanza se encuentra en equilibrio cuando el producto de la carga aplicada sobre el brazo derecho por la longitud de este brazo, es igual al producto de la carga aplicada sobre el brazo izquierdo por la longitud de este otro brazo. Determinar si la balanza se encuentra en equilibrio si:

a. La longitud del brazo izquierdo es 3 m, la del derecho es 2 m, la carga aplicada al brazo izquierdo es 5 Kg y la carga aplicada al derecho es 7 Kg.
b. La longitud del brazo izquierdo es 4 m, la del derecho es 2 m, la carga aplicada al brazo izquierdo es 4 Kg y la carga aplicada al derecho es 4 Kg.
c. Desarrollar un algoritmo que sirva para cualquier conjunto de valores para las longitudes de los brazos y las cargas aplicadas.

EJERCICIO NUEVE**:** Pasar un número entero positivo de base a diez a binario. Hacer también un algoritmo que haga la transformación contraria.

3. METODOLOGÍA DE SOLUCIÓN DE PROBLEMAS CON EL COMPUTADOR

3.1. OBJETIVO

El objetivo de este capítulo es : lograr que el lector entienda cuál es la metodología que se usará en la solución de los problemas. Para esto, se explica primero cuáles son los pasos de la metodología, y luego se describen algunos ejemplos de su aplicación a pequeños problemas de programación.

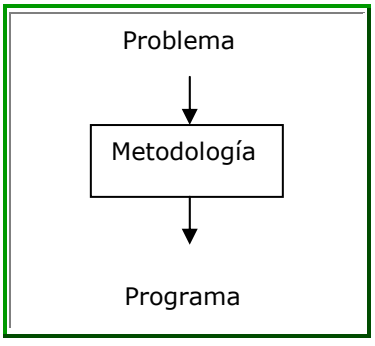
3.2. INTRODUCCIÓN

A través del tiempo, y mediante la práctica constante, se ha desarrollado en ingeniería un conjunto muy amplio de métodos o formas estructuradas de proceder ante ciertas clases de problemas que se presentan con frecuencia. Este conjunto de métodos disponibles para afrontar un problema se llama metodología.

En ingeniería de sistemas se han venido depurando varias metodologías para programar computadores de manera adecuada y efectiva. Estas metodologías no se deben suponer como fórmulas mágicas, sino más bien como formas de proceder que, seguidas con disciplina, conducen a buenos resultados, en la mayoría de los casos. Programar sin metodología se traduce en invertir mucho más tiempo del necesario, en sufrir muchos dolores de cabeza y muy seguramente, en obtener resultados de baja calidad.

El desarrollo de un programa que resuelva un problema es generalmente un reto intelectual importante. Desde hace más o menos 50 años se están haciendo programas para resolver problemas de diversa índole. Este ejercicio continuado y prolífico ha dado lugar a la formulación de *metodologías de programación*. En este libro se sigue una metodología sencilla que se espera que el estudiante adopte como forma de proceder para afrontar la mayoría de los problemas de programación a que se enfrente.

Esta metodología es un conjunto o sistema de métodos, principios y reglas que permiten enfrentar de manera sistemática el desarrollo de un programa que resuelva un problema algorítmico. Esta metodología se estructura como una secuencia de pasos que parten de la definición del problema y culminan con un programa que lo resuelve.



A continuación se presentan de manera general los pasos de una metodología:

Análisis del problema	Con la cual se busca comprender totalmente el problema a resolver.
Especificación del problema	Con la cual se establece de manera precisa cuáles son los datos de entrada, los de salida y qué condiciones debe cumplir cada uno de ellos.
Diseño del algoritmo	En esta etapa se construye un algoritmo que cumpla con la especificación formulada en el paso anterior

Prueba del algoritmo y refinamiento	Consiste en comprobar experimentalmente que el algoritmo funciona bien y corregir los posibles errores que se detecten.
Codificación	Se traduce el algoritmo a un lenguaje de programación.
Prueba y Verificación	Se realizan pruebas del programa implementado para determinar su efectividad en la resolución del problema.

NOTA: LOS EJEMPLOS INCLUIDOS EN ESTA SECCIÓN EN PSEUDICÓDIGO SON SOLO PARA ILUSTRAR LA METODOLOGÍA, NO SE ESPERA QUE EL ESTUDIANTE LOS COMPRENDA

3.3. ANÁLISIS DEL PROBLEMA

Es una perogrullada, pero hay que decirla: **antes de poder dar solución adecuada a un problema hay que entenderlo completamente**. En el primer paso de la metodología el programador debe alcanzar claridad acerca de lo que le están pidiendo resolver. Concretamente, debe determinar de manera clara y concisa lo siguiente:

1. *Los objetos, datos conocidos o datos de entrada*. Aquellos elementos de información total o parcial que se encuentran en el enunciado del problema, y que son útiles en la búsqueda de los objetos desconocidos.
2. *Los objetos, datos desconocidos o datos de salida*. Aquellos datos que el algoritmo debe entregar como solución, al final de su ejecución.
3. *Las condiciones*. De una parte están las condiciones que deben cumplir los datos de entrada. De otra parte están las propiedades que deberán cumplir los datos de salida, que son normalmente relaciones con los datos de entrada. Estas relaciones establecen una dependencia de los datos de entrada.

Ejemplo. Sean los puntos *P=(a,b)* y *Q=(c,d)* que definen una recta, encontrar un segmento de recta perpendicular a la anterior que pase por el punto medio de los puntos dados.

OBJETOS CONOCIDOS	Los puntos *P* y *Q*.
OBJETOS DESCONOCIDOS	Un segmento de recta
CONDICIONES	Los puntos P y Q son diferentes El segmento de recta pedido debe pasar por el punto medio entre *P* y *Q*, y debe ser perpendicular a la recta trazada entre *P* y *Q*.

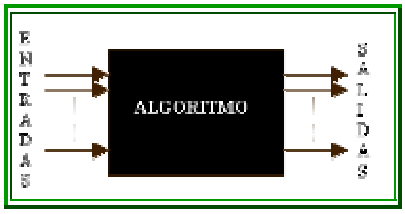
3.4. ESPECIFICACION DEL PROBLEMA

Después de entender totalmente el problema a resolver (lo cual se consigue con la etapa de análisis), se debe realizar su especificación que corresponde a una formalización del análisis. La especificación de un problema se hace mediante una descripción clara y precisa de:

1. Las entradas que recibirá el algoritmo pedido. Las entradas corresponden a los objetos conocidos. Se debe indicar la descripción, cantidad y tipo de las mismas.
2. Las salidas que el algoritmo pedido proporcionará. Las salidas corresponden a los objetos desconocidos del problema. Se debe indicar la cantidad, descripción y tipo de las mismas.
3. La dependencia que mantendrán las salidas obtenidas con las entradas recibidas. Por una parte, se debe hacer claridad sobre las condiciones o propiedades que deben cumplir los datos de entrada. De otro lado, se describe claramente como dependen las salidas de las entradas.

Esta descripción puede estar acompañada de un diagrama de caja negra como el de la siguiente figura. En ella, a cada entrada y salida se le pone un nombre adecuado y el algoritmo pedido se

representa con una caja, para indicar que no se conoce, por ahora, nada acerca de su estructura interna.



Ejemplo 1. Construir un algoritmo que calcule el promedio de 4 notas.

ESPECIFICACION:

Entradas	N_1,N_2,N_3,N_4 (notas parciales) de tipo Real.
Salidas	Final (nota final) de tipo Real.
Condiciones	N_1,N_2,N_3,N_4 deben ser números que corresponden a notas válidas $$Final = \frac{N_1 + N_2 + N_3 + N_4}{4}$$

DIAGRAMA DE CAJA NEGRA:

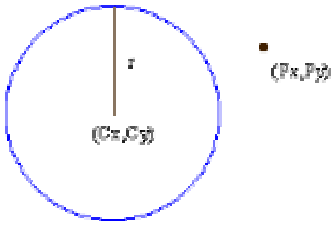
Ejemplo 2. Construir un algoritmo que determine el mayor de tres números enteros dados.

ESPECIFICACIÓN:

Entradas	A,B y C. Tres números de tipo Real.
Salidas	Mayor (valor mayor) de tipo Real.
Condiciones	Mayor debe ser el valor máximo de A, B y C. Formalmente: $Mayor = A \circ Mayor = B \circ Mayor = c$, y $Mayor \geq A$ y $Mayor \geq B$ y $Mayor \geq c$

DIAGRAMA DE CAJA NEGRA:

Ejemplo 3. Determinar si un punto está dentro de un círculo.



ESPECIFICACIÓN:

Entradas	Cx (abscisa del centro del círculo) de tipo Real. Cy (ordenada del centro del círculo) de tipo Real. r (radio del círculo) de tipo Real. Px (coordenada x del punto) de tipo Real. Py (coordenada y del punto) de tipo Real.
Salidas	***Pertenece*** de tipo Booleano, (indica si el punto está dentro o fuera del círculo).
Condiciones	r>0 (**ojo!**: condición sobre un dato de entrada) Pertenece = Verdadero, si el punto está dentro del círculo. Pertenece = Falso, si el punto está fuera del círculo. Formalmente: $Pertenece = \begin{cases} Verdadero & si\ r \geq \sqrt{(Cx-Px)^2+(Cy-Py)^2} \\ Falso & en\ otro\ caso \end{cases}$

DIAGRAMA DE CAJA NEGRA:

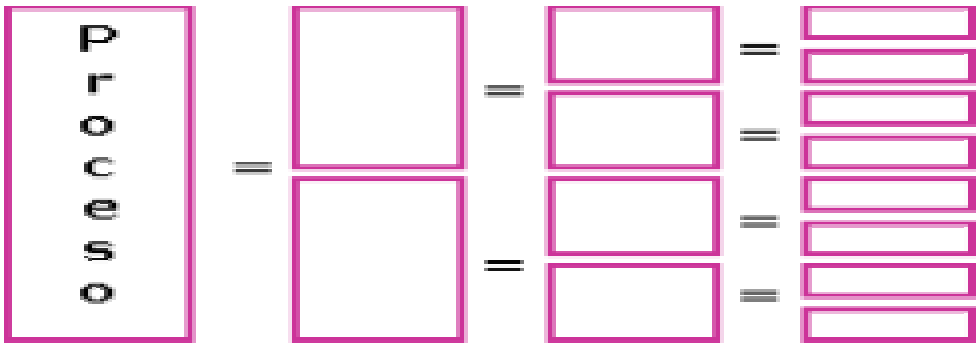
3.5. DISEÑO ESTRUCTURADO DE ALGORITMOS

En la fase de diseño del algoritmo se especifica un conjunto de instrucciones capaz de obtener datos de salida correctos, a partir de datos válidos de entrada. Normalmente, un algoritmo tiene una estructura no trivial, es decir, pueden identificarse en él, partes o componentes muy claramente delimitados que interactúan durante la ejecución. Una primera estrategia para diseñar el algoritmo que solucione el problema consiste en identificar los componentes de esa

estructura, para luego diseñarlos individualmente, y por último ensamblarlos en el programa completo. Esta forma de proceder se denomina *diseño top-down* porque consiste en solucionar un problema dividiéndolo en otros más pequeños, solucionando esos problemas resultantes y finalmente, ensamblando las soluciones de esos subproblemas para obtener la solución completa del problema original.

3.5.1. *División*

Consiste en identificar subprocesos dentro del proceso completo que debe llevar a cabo el algoritmo buscado. Está subdivisión se realiza de manera repetida hasta llegar al nivel de instrucción básica. La siguiente figura ilustra el modo de proceder.



En la etapa final de subdivisión, los problemas a resolver son tan sencillos que ya no se pueden dividir más, sino que simplemente se pueden resolver usando las instrucciones básicas disponibles en el pseudocódigo: asignación, lectura o escritura, secuencia, selección, repetición.

Problema: Realizar un programa que lea una serie de *n* números enteros y determine si la suma de los mismos es un cuadrado perfecto.

ESPECIFICACIÓN DEL PROBLEMA

Entradas	n (número de datos a considerar) $dato_1$, $dato_2$, ..., $dato_n$ (los números enteros)
Salidas	***Es_cuadrado*** de tipo Booleano, (indica si la suma de los datos de entrada es un cuadrado perfecto).
Condiciones	n>0 **Es_cuadrado** tiene valor verdadero si la suma de los n números es un cuadrado perfecto. De lo contrario, tendrá valor falso.

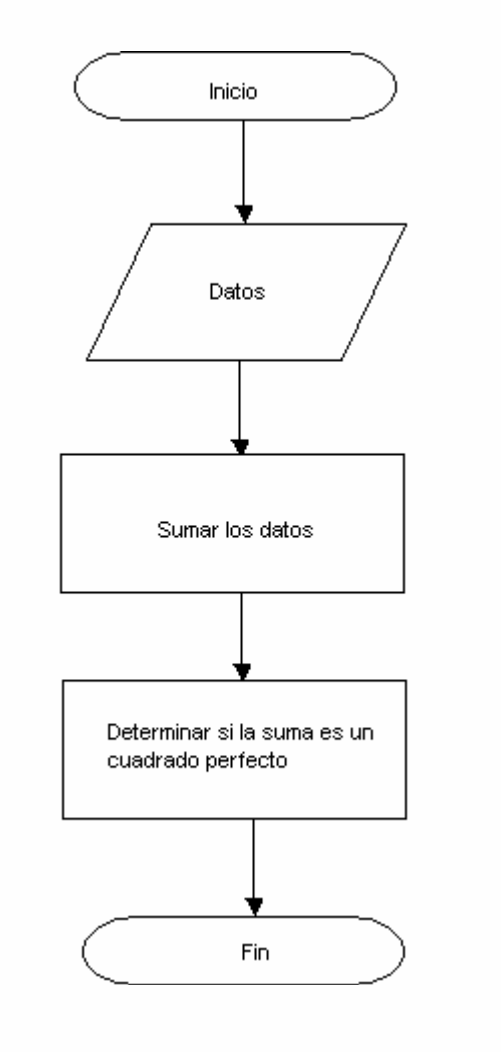
Diagrama de caja negra:

DISEÑO DEL ALGORITMO:

El algoritmo que soluciona este problema es muy sencillo. Sin embargo, para el propósito de ilustrar la técnica de subdivisiones sucesivas (diseño top-down o refinamiento a pasos) se realizan tres etapas de división, cada una de las cuales tiene un diagrama de flujo asociado.

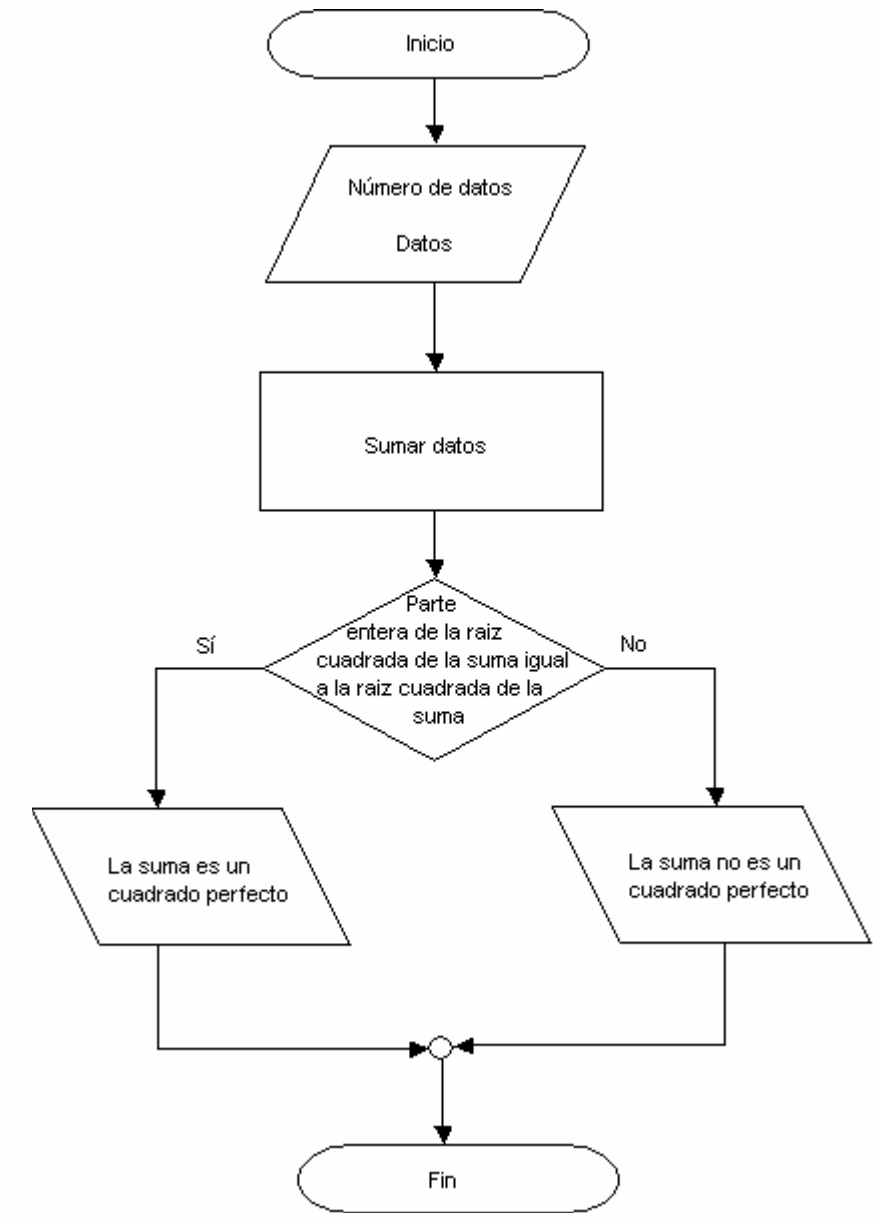
En la primera fase de división se considera que el algoritmo consiste, grosso modo, en leer los datos de entrada, sumar los n números, y determinar si el resultado de la suma es un cuadrado perfecto.

Primera Iteración



En la segunda fase de división se hace explícito que el proceso de lectura de los datos consiste en **(i)** obtener el número de enteros que se van a procesar y en **(ii)** leer cada uno estos n enteros. Adicionalmente, se aclara de qué manera se verificará que la suma es un cuadrado perfecto (se comprueba si la raíz cuadrado es un entero). Por último se deja especificado lo que el algoritmo escribirá como resultado final.

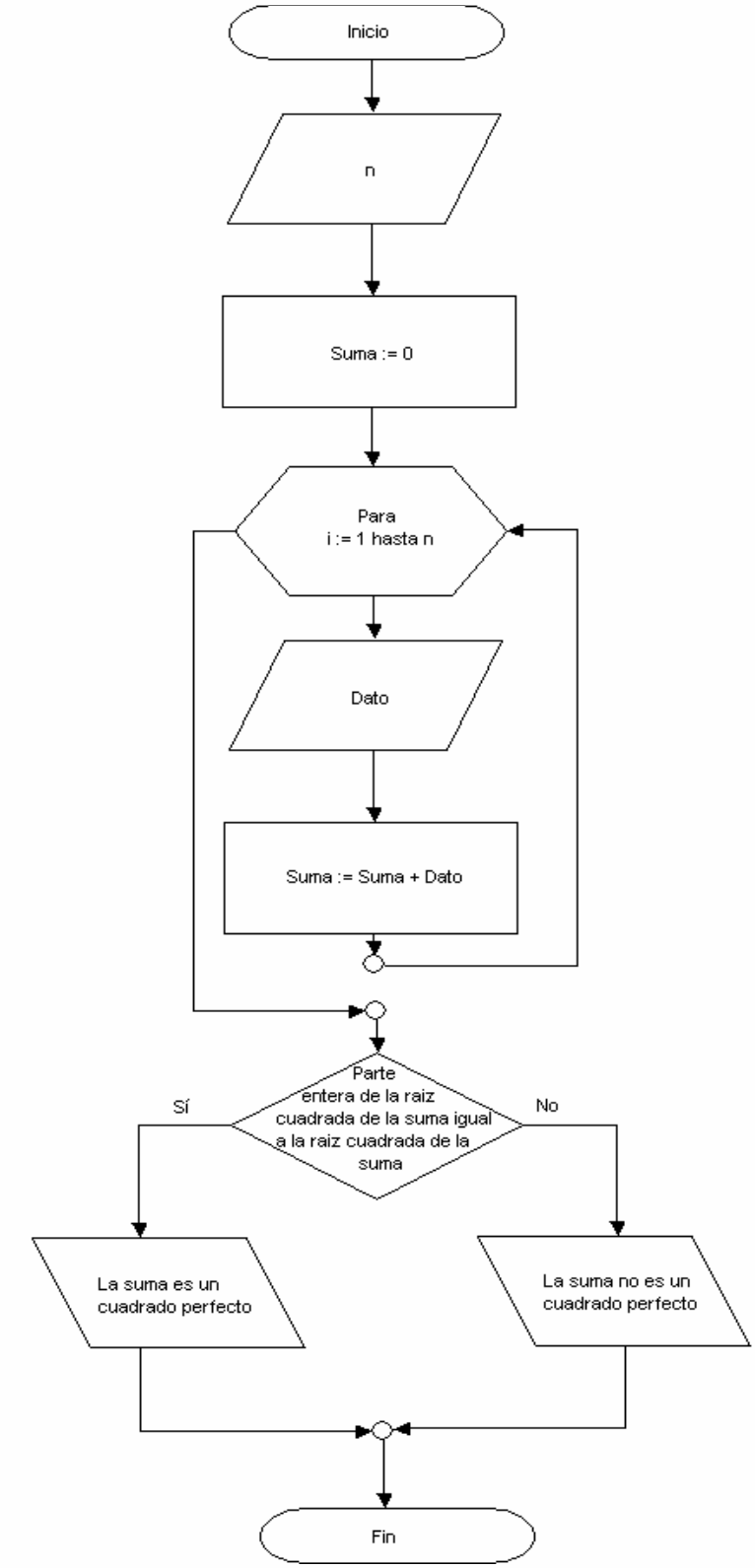
Segunda Iteración



En la tercera fase de división se logra describir todo las acciones del algoritmo con instrucciones básicas. En este punto ya no es necesario ni posible hacer más subdivisiones. El algoritmo está diseñado completamente.

Aunque el lector no entienda todos los detalles del lenguaje con el cual se describe el algoritmo, se le invita a entender la lógica del mismo.

Tercera Iteración (final en este ejemplo)



En seguida se muestra la traducción del diagrama de flujo anterior a pseudocódigo. Nuevamente se aclara que los detalles de esta forma de describir algoritmos se estudian en los próximos capítulos.

Pseudocódigo

```
procedimiento principal()
variables
 n : entero
 suma : entero
 dato : entero
 i : entero
inicio
 escribir ( "Número de enteros a considerar:")
 leer( n)
 suma := 0
 Para( i:=1 hasta n) hacer
    escribir("ingrese un número entero:" )
    leer( dato)
    suma := suma + dato
 fin_para
 si(piso( raiz2( suma ) ) = raiz2(suma ))
  Entonces
      escribir ("La suma de los números es un cuadrado perfecto")
 sino
     escribir ("La suma de los números no es un cuadrado perfecto")
 fin_si
fin_procedimiento
```

3.5.2. *Definición de abstracciones*

Durante el proceso de división es posible identificar qué secuencias de pasos se utilizan más de una vez en diferentes partes del proceso completo. Los lenguajes de programación brindan la posibilidad de escribir solamente una vez estas secuencias de pasos, mediante el uso de unas **abstracciones o subprogramas** llamados procedimientos y funciones. Lo que debe hacer el programador es:

- Recolectar estas secuencias de pasos en funciones y procedimientos, según sea el caso.
- Documentar cada función y procedimiento especificando claramente:
    - El propósito de la función (o procedimiento).
    - El nombre, tipo y propósito de cada argumento.
    - El resultado que produce ese subprograma (o efectos laterales).

El uso de funciones y procedimientos evita tener que escribir más de una vez las secuencias de pasos que se repiten. Pero lo más importante es que permiten describir más claramente la lógica del programa.

Problema: Desarrollar un programa que determine si un número dado se encuentra en alguno de dos conjuntos finitos de enteros.

Recuerde que lo primero que recomienda la metodología propuesta es entender completamente el problema y hacer su especificación.
En este ejercicio se usan arreglos para guardar los dos conjuntos de números. El uso de arreglos para representar conjuntos, requiere que se realicen ciertas validaciones que garanticen la validez como conjunto, por ejemplo, que en el arreglo no esté un mismo elemento dos veces (en un conjunto un elemento está solo una vez). Los arreglos son un tipo de estructura de datos que

se cubre en un capítulo posterior. Teniendo en cuenta estos razonamientos, se puede especificar el problema:

ESPECIFICACIÓN DEL PROBLEMA

Entradas	dos conjuntos, A y B; y el número entero.
Salidas	***bandera*** de tipo Booleano, (indica si el número está o no en alguno de los dos conjuntos).
Condiciones	si el entero dado está en alguno de los conjuntos dados el valor de verdad de ***bandera*** será verdadero. Será falso en caso contrario.

Diagrama de caja negra:

donde,

A: es un conjunto de enteros.
B: es un conjunto de enteros.
elemento: es el entero a comprobar si está en alguno de los conjuntos.
bandera : es un booleano que indica si el elemento está o no está.

$$bandera = \begin{cases} V \text{ si } elemento \in A \vee elemento \in B \\ F \text{ en otro caso} \end{cases}$$

Diseño del algoritmo

Primera iteración. De manera general, el algoritmo debería tener los siguientes pasos

Inicio
1. Leer los dos conjuntos.
2. Leer entero.
3. Determinar si el entero está en alguno de los dos conjuntos
4. Imprimir el resultado.
Fin

Segunda iteración. En esta segunda fase se aclara en qué consiste leer los conjuntos.

1. Leer primer conjunto consiste en:
1.1. Leer un dato.
1.2. Determinar si el elemento no está en el primer conjunto.
1.3. Si no está, agregar el dato al primer conjunto. Si ya está, mostrar un mensaje de error.
1.4. Preguntar si el usuario desea ingresar un nuevo elemento al primer conjunto.
1.5. Si el usuario desea ingresar un nuevo elemento volver a 1.1. Si no, continuar.
2. Leer segundo conjunto se divide en:

2.1. Leer un dato.
2.2. Determinar si el elemento no está en el segundo conjunto.
2.3. Si no está, agregar el dato al segundo conjunto. Si ya está, mostrar un mensaje de error.
2.4. Preguntar si el usuario desea ingresar un nuevo elemento al segundo conjunto.
2.5. Si el usuario desea ingresar un nuevo elemento volver a 2.1. Sino, continuar.
3. Leer entero.
4. Determinar si el entero está en el primer conjunto.
5. Determinar si el entero está al segundo conjunto.
6. Imprimir el resultado.

Tercera Iteración. Se utiliza pseudocódigo para especificar todas las operaciones necesarias.

```
procedimiento principal()
variables
  i, j, n, m, elemento : entero
  continuar : carácter
  bandera : booleano
  A : arreglo [100] de entero
  B : arreglo [100] de entero
/* el codigo siguiente lee el conjunto A*/
Inicio
n := 0
escribir("Desea ingresar elementos al conjunto A (S/N):")
leer( continuar)
mientras( continuar = 'S' | continuar = 's') hacer
   escribir( "Ingrese el elemento al conjunto A:")
   leer( elemento)
/* el codigo siguiente prueba si el elemento esta en el conjunto A */
   i := 0
   mientras( i<n & A[i]<> elemento) hacer
            i := i+1
   fin_mientras
   si( i = n) entonces
       A[n] := elemento
       n := n+1
   Sino
       escribir( "Error: el elemento ya esta en el conjunto A" )
   fin_si
   escribir( "Desea ingresar mas elementos al conjunto A (S/N)" )
   leer( continuar)
fin_mientras
/* el codigo siguiente lee el conjunto B */
m := 0
escribir("Desea ingresar mas elementos al conjunto B (S/N)" )
leer( continuar)
mientras( continuar = 'S' | continuar = 's') hacer
     escribir( "Ingrese el elemento al conjunto B:")
     leer( elemento)
/* el codigo siguiente prueba si el elemento esta en el conjunto B */
     i := 0
     mientras i<m & B[i] <> elemento hacer
         i := i+1
     fin_mientras
     si (i = m) entonces
         B[m] := elemento
          m := m+1
     sino
         escribir( "Error: el elemento ya esta en el conjunto B" )
     fin_si
     escribir( "Desea ingresar mas elementos al conjunto B (S/N)" )
```

```
    leer( continuar)
fin_mientras
/* el codigo siguiente lee un elemento a probar */
escribir( "Ingrese el dato que desea buscar en los conjuntos" )
leer( elemento)
/* el codigo siguiente prueba si el elemento esta en el conjunto A */
bandera := falso
i := 0
mientras( i<n & A[i] <> elemento) hacer
    i := i+1
fin_mientras
si( i < n) entonces
    bandera := verdadero
fin_si
/* el codigo siguiente prueba si el elemento esta en el conjunto B */
i := 0
mientras( i<m & B[i] <> elemento) hacer
    i := i+1
fin_mientras
si(i < m) entonces
    bandera := verdadero
fin_si
/* el codigo siguiente determina si el elemento esta en alguno de los dos conjuntos */
si(bandera = verdadero) entonces
    escribir( "El dato dado esta en alguno de los dos conjuntos" )
sino
    escribir( "El dato dado NO esta en ninguno de los conjuntos" )
fin_si
fin_procedimiento
```

Abstracción: En el código obtenido mediante la fase de división se puede apreciar la existencia de porciones de código que no son idénticas, pero son muy parecidas y funcionalmente equivalentes. Este es el caso de las porciones de código que permiten leer los conjuntos A y B, y las porciones de código que permiten determinar si un elemento está en el conjunto A y en el conjunto B. Identificadas estas redundancias, se puede crear un procedimiento que permita leer un conjunto cualquiera, y una función que verifique si un elemento está en un conjunto dado.

Primero, la función **pertenece** es un subprograma que se define así:

```
pertenece: Arreglo[100] de Entero x Entero x Entero -> Booleano
( A    ,          ,   n    ,   e ) Þ V si e = A[i] para algún i
Þ F en otro caso
```

Se puede observar que esta función además de recibir el arreglo de datos y el elemento, recibe un entero adicional *n*. Este entero se utiliza para indicar el tamaño del conjunto dado. Esta función se codifica como sigue:

```
funcion pertenece( A :arreglo[100] de entero, n :entero, e :entero ):booleano
 variables
  bandera :booleano
  i : entero
 inicio
/* el codigo siguiente prueba si el elemento está en el conjunto */
  i := 0
  mientras( i<n & A[i] <> e) hacer
          i := i+1
  fin_mientras
  si( i = n) entonces
       bandera := falso
```

```
 si_no
       bandera := verdadero
 fin_si
retornar bandera
fin_funcion
```

Segundo, el procedimiento **leer_conjunto** es un subprograma que se define así:

```
Procedimiento leer_conjunto( var A :arreglo [100] de entero, var n :entero)
   variables


   i :entero
   elemento :entero
   continuar :caracter
inicio
/* el codigo siguiente lee el conjunto */
 n := 0
 escribir( "Desea ingresar mas elementos al conjunto  (S/N)?")
 leer( continuar)
 mientras ( n<100 & (continuar = 'S' | continuar = 's'))  hacer
    escribir( "Ingrese el elemento al conjunto " )
    leer( elemento)

/* el código siguiente prueba si el elemento esta en el conjunto y de no ser así lo adiciona */
    si no (pertenece( A, n, elemento) ) entonces
         A[n] := elemento
         n := n+1
    sino
        escribir( "Error: el elemento ya esta en el conjunto " )
    fin_si
    escribir( "Desea ingresar mas elementos al conjunto (S/N)?")
    leer( continuar)
 fin_mientras
fin_procedimiento
```

Este procedimiento recibe un arreglo de enteros y una variable entera llamada **n**. Lo que hace es leer unos números (máximo 100) y guardarlos en el arreglo. También lleva la cuenta de cuántos datos ha leído, y almacena esta cifra en la variable **n**.

Otro aspecto importante que se puede destacar en este procedimiento es que usa la función ***pertenece*** para determinar si se debe o no adicionar el elemento leído al conjunto.

De esta manera el algoritmo principal se puede presentar como sigue:

```
procedimiento principal()
variables
  n :entero
  m :entero
  elemento :entero
  A :arreglo [100] de entero
  B :arreglo [100] de entero
inicio
/* el codigo siguiente lee el conjunto A */
leer_conjunto( A, n )
/* el codigo siguiente lee el conjunto B */
leer_conjunto ( B, m)
/* el codigo siguiente determina si el elemento esta en alguno de los dos conjuntos */
si pertenece( A, n, elemento ) | pertenece( B, m, elemento ) entonces
   escribir( "El dato dado esta en alguno de los dos conjuntos")
sino
    escribir( "El dato dado NO esta en los conjuntos dados" )
fin_si
fin_procedimiento
```

Se puede apreciar la reducción de líneas de código del programa y la facilidad de lectura de cada uno de estos algoritmos (función, procedimiento y algoritmo principal), con respecto al algoritmo inicial realizado sin abstracción. Se deja al lector la escritura del programa completo, utilizando las reglas descritas en esta sección.

3.6. CODIFICACION

Cuando ya se ha diseñado completamente el algoritmo y se tiene escrito en algún esquema de representación (pseudo-código o diagrama de flujo), el siguiente paso es codificarlo en el lenguaje de programación definido para tal fin.

En este momento es cuando el programador interactúa con el computador mediante la herramienta de software de que disponga para codificar en el lenguaje seleccionado.

EJEMPLO. Tómese como base el pseudocódigo desarrollado en la sección anterior. El programa en C++ para este pseudocódigo sería:

```
#include <iostream.h>

bool  pertenece( int A[100, int n, int e)
{
   bool bandera;
    int i;
/* el codigo siguiente prueba si el elemento esta en el conjunto */
 i = 0;

   while(> i<n && A[i] != e ) {
        i = i+1;
   };
   if( i == n ) {
        bandera = false;
   }
   else {
   bandera = true;
   };
 return bandera;
}

void leer_conjunto( intA[100], int & n,)
{
int i;
int elemento;
char continuar;

/* el codigo siguiente lee el conjunto */
 n = 0;
 cout << "Desea ingresar mas elementos al conjunto (S/N)";
 cin >> continuar;
 while( continuar == 'S' || continuar == 's' ) {
     cout << "Ingrese el elemento que quiere agregar :";
     cin >> elemento;

/* el codigo siguiente prueba si el elemento esta en el conjunto y de no ser así lo adiciona */
    if( !pertenece( A, n, elemento ) ) {
          A[n] = elemento;
          n = n+1;
    }
    else {
         cout << "Error: el elemento ya esta en el conjunto";
```

```
  };
  cout << "Desea ingresar mas elementos al conjunto (S/N)" ;
  cin>> continuar;
 };
}

void main()
{
 int n, m, elemento;
 int A[100];
 int B[100];
/* el codigo siguiente lee el conjunto A */
 leer_conjunto( A, n);
/* el codigo siguiente lee el conjunto B */
 leer_conjunto( B, m)

/* el codigo siguiente determina si el elemento esta en alguno de los dos conjuntos */
 if( pertenece(A, n, elemento) || pertenece(B, m, elemento) ) {
 cout << "El dato dado esta en alguno de los dos conjuntos";
 }
 else {
 cout<< "El dato dado NO esta en los conjuntos dados";
 };
}
```

3.6.1. *Prueba de escritorio*

La prueba de escritorio es una herramienta útil para entender qué hace un determinado algoritmo, o para verificar que un algoritmo cumple con la especificación sin necesidad de ejecutarlo.

Básicamente, una prueba de escritorio es una ejecución 'a mano' del algoritmo, por lo tanto se debe llevar registro de los valores que va tomando cada una de las variables involucradas en el mismo.

A continuación se muestra un ejemplo de prueba de escritorio del siguiente algoritmo:

```
procedimiento principal()
variables
  suma, entrada, menor :entero
inicio
leer( entrada)
menor := entrada
suma:= 0
mientras (entrada <> 0) hacer
   si (entrada < menor) entonces
      menor =entrada
   fin_si
   suma:= suma + entrada
   leer( entrada)
fin_mientras
escribir( "valor menor:" )
escribir( menor)
escribir( "Suma total:")
escribir( suma)
fin_procedimiento
```

INSTRUCCIÓN	entrada	menor	suma	Pantalla
leer (entrada)	10			
menor := entrada		10		
suma :=0			0	
suma :=suma + entrada			10	
leer (entrada)	7			
menor := entrada		7		
suma:=suma + entrada			17	
leer(entrada)	9			
suma :=suma + entrada			26	
leer (entrada)	0			
escribir (**"**valor menor:**")**				Valor Menor
escribir (menor)				7
escribir (**"**Suma:**")**				Suma:
escribir (suma)				26

3.7. EJERCICIOS

En cada uno de los siguientes problemas, use la metodología presentada en este capítulo para resolverlo. Los algoritmos los puede especificar usando lenguaje natural (español).

EJERCICIO UNO: Construir un algoritmo que determine si un número dado es primo.

EJERCICIO DOS: Construir un algoritmo que determine si dos números son primos relativos o no. Dos números son primos relativos si no tienen un divisor común mayor que 1.

EJERCICIO TRES: Construir un algoritmo que determine si una fecha, especificada mediante tres números (día, mes y año), es una fecha válida o no.

EJERCICIO CUATRO: Construir un algoritmo que lea un entero y escriba todos sus divisores propios, en orden decreciente.

EJERCICIO CINCO: Construir un algoritmo que lea las longitudes de tres segmentos de recta y determine si con ellos se puede construir un triángulo.

EJERCICIO SEIS: Un rectángulo queda determinado si se conocen las coordenadas de su vértice inferior izquierdo, su ancho y su largo. Construir un algoritmo que lea los datos relacionados con dos rectángulos y determine si estos se intersectan o no.