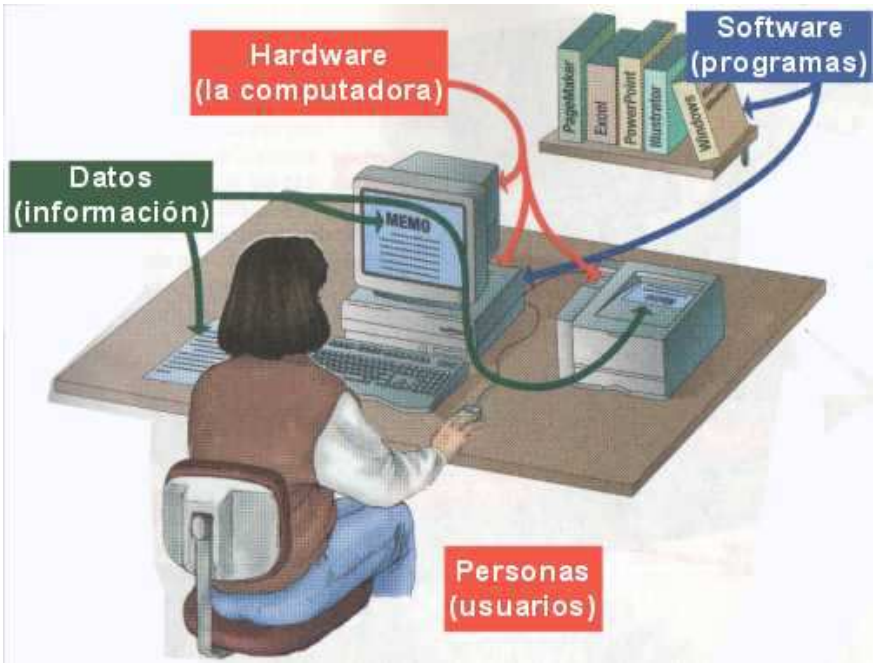
Fundamentos Generales de Programación



DEPTO. L&S

ING. JACQUELINE DEL R. LÓPEZ ALVARADO
MSC. GLORIA TALÍA FLORES QUINTANA

Managua, Nicaragua 2012

FUNDAMENTOS GENERALES DE PROGRAMACIÓN

Ing. Jacqueline del R. López Alvarado

Profesor de la Universidad Nacional de Ingeniería (UNI) del departamento de Lenguajes y Simulación. Facultad de Electrotecnia y Computación (FEC)

MSc. Gloria Talía Flores Quintana

Profesor de la Universidad Nacional de Ingeniería (UNI) del departamento de Lenguajes y Simulación. Facultad de Electrotecnia y Computación (FEC)

Dedicatoria

Al reconocer que nada somos sino por la mismísima voluntad del creador, doy gracias por su maravillosa voluntad de ser parte del desarrollo de esta nuestra amada Nicaragua.

Jacqueline del Rosario López Alvarado.

A mis hijos, Emilio Xavier y Gabriel Alcides, motor de mi vida y superación. De quienes aprendo cada día.

Gloria Talía Flores Quintana.

Contenido Página

Presentación

Objetivo

Este libro tiene como objetivo mostrar los fundamentos básicos de la computadora, tales como:

- Reseña histórica de la Ingeniería en el área de computación, en ella se abordan conceptos básicos sobre la misma, su historia y campos en los que se divide.
- Definiciones elementales del computador y el desarrollo del mismo a través de sus generaciones.
- Sistemas de numeración
- Algoritmos.
- Herramienta practica para visualizar las salidas de los algoritmos.

El libro está dirigido a estudiantes de los primeros años de la carrera de Ingeniería en Computación como apoyo en el desempeño académico.

Organización

El libro está compuesto en siete capítulos, cada uno de ellos cuenta con definiciones precisas sobre el tema abordado, de igual manera al final de cada capítulo se presenta guía práctica de preguntas y ejercicios donde el lector puede repasar conceptos básicos y dar solución a estos.

Capítulo I: Introducción y definiciones básicas de la ingeniera

El capítulo I muestra las definiciones básicas de la ingeniería. Se estudia su historia, el origen de la ingeniería, los campos de la ingeniera actual, las generaciones de la computadora y sus respectivos logros en cada una de ellas. Al final se de este capítulo se incluye una guía de preguntas.

Capítulo II: Organización básica del computador

El capítulo II aborda los conceptos básicos sobre la organización básica del computador, definiciones tales como, dispositivos de entrada y salida, unidad central de procesamiento, memoria, dispositivos de almacenamiento, hardware, software. Al final del capítulo se incorpora una guía de preguntas que deben contestarse en base a la previa lectura de dicho capítulo.

Capítulo III: Sistemas de numeración

El capítulo III define los cuatro principales sistemas de numeración: binario, sistema de numeración decimal, sistema de numeración octal, sistema de numeración hexadecimal, además las operaciones básicas aplicadas a números binarios. Al final del capítulo se presenta una serie de ejercicios permitiendo aplicar la teoría a la práctica.

Capítulo IV: Algoritmos

El capítulo IV da a conocer que el concepto básico de algoritmo, en donde explica las técnicas para el diseño del mismo. Al final de este capítulo se presenta una serie de ejemplos resueltos diseñados como pseudocódigos.

Capítulo V: PSEint

El capítulo V presenta una herramienta para simular instrucciones algorítmicas, aprendidas en el capítulo IV denominada PSEint. Se incluye un manual práctico y el instalador del software.

Capítulo VI: Programación estructurada

El capítulo VI da a conocer conceptos básicos de programación en bajo nivel específicamente en lenguaje C tales como:

- Historia del lenguaje C
- Definiciones básicas del lenguaje C.
- Estructuras de control.

Al final del capítulo se agrega guía de problemas.

Capítulo VII: Arreglos

El capítulo VI detalla el término *arreglo* como nuevo tipo de datos. Describiendo:

- Sintaxis
- Arreglos lineales
- Arreglos bidimensionales
- Cadenas de caracteres

Al final de este capítulo se establecen una serie de ejemplos y ejercicios propuestos.

Capítulo 1

INTRODUCCIÓN A LA INGENIERIA

1.1 HISTORIA DE LA INGENIERIA

La historia de la civilización es de cierto modo, la de la ingeniería: un extenso y dificultoso esfuerzo para lograr que la naturaleza trabaje en armonía con hombre.

Los primeros individuos utilizaron ciertos principios de la ingeniería con la finalidad de conseguir sus alimentos, pieles y construir armas de defensa como hachas, puntas de lanzas, martillos entre otros.

Sin embargo el desarrollo de la ingeniería como tal, se dio con la revolución agrícola, esta surge cuando el hombre cambia de una existencia nómada a otra en un lugar más o menos fijo para cultivar productos y criar animales comestibles. Algunos historiadores piensan que estos cambios ocurrieron primero en Siria e Irán, aproximadamente hacia 8,000 A. de J.C[1].

Los primeros ingenieros fueron arquitectos, especialistas en irrigación e ingenieros militares. Las primeras labores de los ingenieros fue construir muros que rodeaban las ciudades; con el fin de protegerla de los enemigos. Es justo pensar que los antiguos arquitectos precederían a los ingenieros en la satisfacción de esta necesidad. Sin embargo en el diseño y edificación de estructuras de uso público tales como edificios se hizo necesario auxiliarse de las habilidades de la ingeniería.

La innovación de inventos se realizo de forma lenta, las necesidades en el área militar y agrícola tenían mayor prioridad. De igual manera debido a las limitaciones en el campo de la comunicación, la distancia entre las poblaciones era sumamente grande y se podría decir que fue realmente difícil el intercambio de conocimientos, y muchos de los inventos tuvieron que volverse a inventar antes que formaran parte del constante proceso evolutivo de la sociedad de esa época. En cambio las poblaciones aledañas a las rutas principales de comercio desde China a España se desarrollaron mucho más rápido que las demás debido a que les llegaba él Conocimiento de innovaciones de otros distantes lugares.

Esta época se puede definir como la era de los inventos, los mismos dieron inicio a la ingeniería, que entonces como ahora ***"Es el proceso de aplicar el conocimiento científico en bien de la humanidad"*** (aunque otros lo utilizan para destruir). La ciencia y la Ingeniería han avanzado mucho en los últimos tres siglos, sin embargo este desarrollo se vio obstruido antes del siglo XVIII debido a la persecución que se le dio a los hombres de ciencia (acusados de brujería), por la Santa Inquisición.

[1] http://medusa.unimet.edu.ve/mecanica/bpii00/Historia/historia.htm

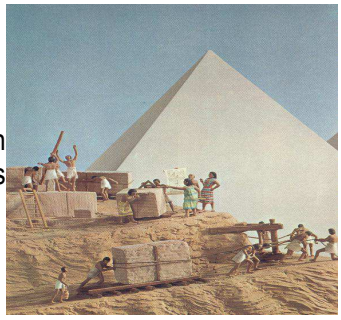
Los campos más importantes de la ingeniería aparecieron de la siguiente manera: militar: *desarrollada para ayudar a satisfacer la necesidad básicas de supervivencia*, civil, mecánica, eléctrica, química, industrial, producción y de sistemas, siendo esta última uno de los campos más nuevo.

Cada periodo de la historia ha tenido distintos climas sociales y económicos, así como presiones que han influido grandemente tanto el sentido como el progreso de la ciencia y de la ingeniería. A continuación se presenta la historia de la ingeniería según las culturas: (Ingeniería Egipcia, Ingeniería Mesopotámica, Ingeniería Griega, Ingeniería Romana, Ingeniería Oriental e Ingeniería Europea).

1.1.1 Ingeniería egipcia

Los egipcios crearon algunas de las obras más grandiosas de la ingeniería de todos los tiempos, por ejemplo el muro de la ciudad de Menfis. Esta antigua ciudad fue la capital de Egipto y se encontraba a 19 Km.(aproximadamente) al norte de donde está situado El Cairo en la actualidad.

Imhotep, hijo de Kanofer, creador del muro de la ciudad de Menfis a quien los historiadores consideran como el primer ingeniero conocido. Construyo de la pirámide de peldaños en Saqqarah, probablemente hacia el 2550 a.C. Por su sabiduría y habilidad, fue elevado a la categoría de dios después de su muerte Realizo una serie de obras en las que tomo en cuenta lo siguiente:



El reinado del Rey Joser fue propicio para el invento de Imhotep: *la pirámide.*

Las habilidades técnicas requeridas para el diseño, organización y control de un proyecto de esta magnitud lo distinguen como una de las proezas más grandes y antiguas de todos los tiempos.

De todas las pirámides, la del faraón Keops fue la mayor.

La Gran Pirámide, como se le conoce ahora, tenía 230.4 m por lado en la base cuadrada y originalmente medía 146.3 m de altura. Contenía unos 2, 300, 000 bloques de piedra, de cerca de 1.1 toneladas en promedio. Teniendo en cuenta el conocimiento limitado de la geometría y la falta de instrumentos de ese tiempo, fue una proeza notable. Estas estructuras masivas de ingeniería sólo son superadas por la Gran Muralla China, entre las obras de la antigüedad.

Además de las pirámides los egipcios construyeron diques, canales y sistemas complejos de irrigación. Cuando la tierra de regadío era más alta que el nivel del río, utilizaban un dispositivo denominado cigüeñal "shaduf" para elevar el agua hasta un nivel desde el cual se dirigía hacia la tierra. El aparato consiste en una cubeta unida mediante una cuerda al extremo largo de un palo apoyado, con un contrapeso en su extremo corto. El operador hacía fuerza en el contrapeso para levantar la cubeta y balancear el palo sobre su fulcro. Lo que parece sorprendente hoy día es que muchos de esos antiguos dispositivos sigan en uso cotidiano en Egipto.[2]

1.1.2 Ingeniería mesopotámica

Otra gran cultura que floreció es Mesopotamia ("Tierra entre ríos"), llamada así por los griegos debido a su posición geográfica (al norte de Irán), entre los ríos Tigris y el Éufrates.

[2] http://medusa.unimet.edu.ve/mecanica/bpii00/Historia/historia.htm

Los historiadores indican que en Mesopotamia se inició la tradición de que un político inaugure la construcción de un edificio público con una palada de tierra.

Los asirios fueron los primeros en emplear armas de hierro. Los asirios también inventaron la torre de asalto, que se convirtió en una pieza estándar del equipo militar durante los dos mil años siguientes, hasta que la invención del cañón la hizo obsoleta. Alrededor de 2,000 A.de J.C., también lograron un avance significativo en el transporte. Aprendieron que el caballo se podía domesticar y servía para cabalgar, lo que les produjo una ventaja militar considerable *(inventaron la caballería).*

1.1.3 Ingeniería griega

Los griegos de Atenas y Esparta, no se destacaron por su creatividad e inventiva, sino por el desarrollo de ideas. Su mayor aporte fue descubrir que la naturaleza tiene leyes generales de comportamiento.

El primer ingeniero reconocido en el mundo griego fue Pytheos, constructor del Mausoleo de Halicarnaso, quien combinó allí tres elementos: el pedestal elevado de la columna, el templo griego y el túmulo funerario egipcio.

Otros ingenieros importantes fueron Dinocrates, el planeador de Alejandría, Sostratus, quien construyó el Faro, Arquímedes, Cresibius y Herón, el inventor de la turbina de vapor

Quienes dirigieron la construcción de esas antiguas estructuras no tenían un título que se pudiera traducir como “ingeniero”. Se les llamaba “arquitekton”, que quiere decir el que había cumplido un periodo como aprendiz en los métodos estándar de construcción de edificios públicos.

Los arquitectos recibían aproximadamente un tercio más de remuneración que los albañiles.

La "Mecánica" fue el primer texto conocido de ingeniería. Aquí se estudiaban conceptos fundamentales de la ingeniería como la teoría de la palanca. También contiene un diagrama que ilustra un tren de tres

engranes, mostrados como círculos, lo que constituye la primera descripción conocida de engranajes. Es más probable que estos no tuvieran dientes, por lo que tuvo que ocurrir mucho deslizamiento antes de que se conociera la ventaja de los dientes y la manera de producirlos.

La mayor aportación de los griegos a la ingeniería fue el descubrimiento de la propia ciencia. Platón y su alumno Aristóteles quizás sean los más conocidos de los griegos por su doctrina de que hay en un orden congruente en la naturaleza que se puede conocer.

Los griegos desarrollaron un estudio llamado "Hybris" (orgullo), que era una creencia en la necesidad de leyes morales y físicas restrictivas en la aplicación de una técnica dominada. En pocas palabras desaprobaban los métodos casi inhumanos que tenían los egipcios para con sus esclavos (cargar monolitos de piedra varios kilómetros de distancia), por eso los griegos no llegaron a construir obras de gran magnitud como Egipto. Sin embargo, lo que los griegos no tuvieron en realizaciones de ingeniería lo compensaron con creces en los campos de arte, literatura, filosofía, lógica y política. Es interesante notar que la topografía, como la desarrollaron los griegos y luego los romanos, se considera como la primera ciencia aplicada de la ingeniería, y fue la única ciencia aplicada durante los veinte siglos siguientes.

Aunque a Arquímedes le conoce mejor por lo que ahora se llama el "Principio de Arquímedes", también era un matemático y hábil ingeniero. Realizó muchos descubrimientos importantes en las áreas de la geometría plana y sólida, tal como una estimación más exacta de PÍ y leyes para encontrar los centros de gravedad de figuras planas. También determinó la ley de las palancas y la demostró matemáticamente. Mientras estuvo en Egipto, inventó lo que se conoce como "el tornillo de Arquímedes". También fue constructor de barcos y astrónomo.

1.1.4 Ingeniería romana

Los ingenieros romanos tenían más en común con sus colegas de las antiguas sociedades de las cuencas hidrográficas de Egipto y Mesopotamia, que con los ingenieros griegos, sus predecesores.

Los romanos utilizaron principios simples, el trabajo de los esclavos y tiempo para producir extensas mejoras prácticas para el beneficio del Imperio Romano. En comparación con las de los griegos, las contribuciones romanas a la ciencia fueron limitadas; sin embargo, sí abundaron en soldados, dirigentes, administradores y juristas notables.

Los romanos aplicaron mucho de lo que les había precedido, y quizá se les puede juzgar como los mejores ingenieros de la antigüedad. Lo que les faltaba en originalidad lo compensaron en la vasta aplicación en todo un imperio en expansión.

En su mayor parte, la ingeniería romana era civil, especialmente en el diseño y construcción de obras permanentes tales como acueductos, carreteras, puentes y edificios públicos. Una excepción fue la ingeniería militar, y otra menor, por ejemplo, la galvanización. La profesión de “architectus" era respetada y popular; en efecto, Druso, hijo del emperador Tiberio, era arquitecto.

Una innovación interesante de los arquitectos de esa época fue la reinvención de la calefacción doméstica central indirecta, que se había usado originalmente cerca de 1200 a. de J.C., en Beycesultan, Turquía.

Otro gran triunfo de la construcción pública durante este periodo fue el Coliseo, que fue el mayor lugar de reunión pública hasta la construcción del Yale Bowl en 1914.

Los ingenieros romanos aportaron mejoras significativas en la construcción de carreteras, principalmente por dos razones: una, que se creía que la comunicación era esencial para conservar un imperio en expansión, y la otra, porque se creía que una carretera bien construida duraría mucho tiempo con un mínimo de mantenimiento. Se sabe que las carreteras romanas duraban hasta cien años antes de que necesitaran reparaciones mayores.

Es apenas hasta fechas recientes que la construcción de carreteras ha vuelto a la base de “alto costo inicial - poco mantenimiento”.

Quizá el triunfo más conocido en la construcción de carreteras de la antigüedad es la Vía Apia, que se inicio en 312 a. de J.C., y fue la primera carretera importante recubierta de Europa.

Los acueductos romanos se construyeron siguiendo esencialmente el mismo diseño, que usaba arcos semicirculares de piedra montados sobre una hilera de pilares. Cuando un acueducto cruzaba una cañada, con frecuencia requería niveles múltiples de arcos. Uno de los mejor conservados de la actualidad es el Pont du Gard en Nimes, Francia, que tiene tres niveles. El nivel inferior también tenía una carretera.

Los romanos usaron tubería de plomo y luego comenzaron a sospechar que no eran salubres. Sin embargo, el envenenamiento por plomo no se diagnosticó específicamente sino hasta que Benjamín Franklin escribió una carta en 1768 relativa a su uso.

El emperador Claudio hizo que sus ingenieros intentaran en 40 d. de J.C., drenar el lago Facino a través de un túnel, usando el desagüe para irrigación. Se hicieron dos intentos por lograr esto, el segundo tuvo mejores resultados que el primero.

Aproximadamente en 200 d. de J.C., se inventó un ariete[3] llamado “ingenium” para atacar las murallas. Muchos años después se llamó al operador del ingenium, “ingeniator”, que muchos historiadores creen que fue el origen de la palabra ingeniero.

La ingeniería romana declinó después de 100 d. de J.C., y sus avances fueron modestos. Este se considera uno factores que contribuyó a la caída del Imperio Romano.

Una innovación durante este periodo fue la invención del alumbrado público en la ciudad de Antioquía, aproximadamente hacia el año 3~0 d. de J.C. La caída de Roma es sinónimo del fin de los tiempos antiguos.

1.1.5 Ingeniería oriental

Después de la caída del Imperio Romano, el desarrollo ingenieril se trasladó a la India y China. Los antiguos hindúes eran diestros en el manejo del hierro y poseían el secreto para fabricar buen acero desde antes de los tiempos de los romanos.

Aproximadamente en 700 d. de J.C., un monje de Mesopotámica llamado Severo Sebokht dio a conocer a la civilización occidental el sistema numérico indio, que desde entonces hemos llamado números arábigos.

[3] Un ariete es tan sólo un tronco grande y pesado, cargado por varias personas e impulsado con fuerza contra un obstáculo. El ímpetu del ariete es suficiente para dañar el objetivo. Normalmente lleva incorporada al tronco la cabeza de un carnero para aprovechar su cornamenta enroscada en forma de círculo.

Invenciones en esta época:

1. El sistema numérico indio, que desde entonces hemos llamado números arábigos. aproximadamente en 700 d. de J.C., por un monje de Mesopotamia llamado Severo Sebokht.

2. La Gran Muralla de China. La cual media de distancia de un extremo a otro del muro aproximadamente 2 240 Kilometros, de altura aproximadamente 10 metros y 8 metros de espesor en la base, y se reduce hasta aproximadamente 5 metros en la parte superior. A lo largo de esta parte corre un camino pavimentado.

3. La construcción de puentes de puentes, con características únicas, destacándose los puentes más antiguos que fueron de suspensión, con cables hechos de fibra de bambú.

4. La invención del papel.

5. El desarrollo de maquinaria de engranaje[4] desde fechas muy antiguas.

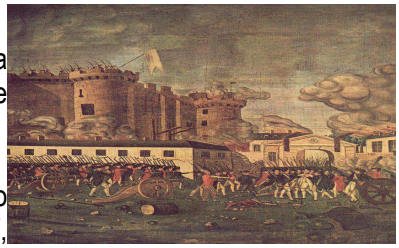
6. Descubrimiento de la brújula, que rápidamente se extendió, para ser de uso común alrededor de 1200 d. de J.C.

Luego los árabes aprendieron de los chinos el método de fabricación del papel, y lo produjeron en grandes cantidades. A partir de entonces aumentó notablemente la comunicación de las ideas. La química progresó mucho como ciencia en Arabia y también se aprendió y extendió con rapidez el proceso para hacer pólvora.

1.1.6 Ingeniería europea

La Edad Media, a la que a veces se le conoce como el período medieval, abarcó desde aproximadamente 500 hasta 1500 d. de J.C., pero por lo general se denomina Oscurantismo al período que media entre el año 600 y el 1000 d. de J.C. Durante este período no existieron las profesiones de ingeniero o arquitecto, de manera que esas actividades quedaron en manos de los artesanos, tales como los albañiles maestros. La literatura en esta época era predominantemente de naturaleza religiosa, y quienes tenían el poder no daban importancia a la ciencia e ingeniería.

Inventos y genios de esa época:



- El cañón, fue el invento que contribuyó a la terminación de la forma de vida con castillos rodeados de murallas y que apareció en Alemania en el siglo XIV.[5]

- La invención de los anteojos en 1286, y el incremento considerable en las obras impresas en Europa en el siglo XV, fueron dos acontecimientos trascendentales en la expansión del pensamiento ingenieril. Desde luego, otro factor importante en todo momento es la actitud de una sociedad hacia una profesión. Durante el

[4] Un engranaje sirve para transmitir movimiento circular mediante contacto de ruedas dentadas: http://es.wikipedia.org/wiki/Engranaje
[5] http://www.monografias.com/trabajos81/evolucion-historica-ingenieria/evolucion-historica-ingenieria2.shtml

- Renacimiento, los ingenieros nuevamente fueron miembros de una profesión respetada y eran bien remunerados.

- La reconstrucción de la Basílica de San Pedro, realizada por Miguel Ángel Buonarroti, al que se le conoce simplemente como Miguel Ángel, quien en 1514 por decisión del Papa Paulo III sustituyo al arquitecto Bramante después de su muerte, terminando así la reconstrucción.

- Entre los logros artísticos se encuentra Leonardo de Vinci (*artista - científico*). Dominó la astronomía, anatomía, aeronáutica, botánica, geología, geografía, genética y física. Sus estudios de física abarcaron todo lo que se conocía en su tiempo. Tenía una curiosidad científica que alguna vez le causó problemas. El Papa León X lo despidió cuando supo que aprendía anatomía humana disecando cadáveres. Leonardo de Vinci uno de los grandes genios de todos los tiempos, anticipó muchos adelantos del futuro; por nombrar algunos: la máquina de vapor, la ametralladora, cámara oscura, el submarino y el helicóptero. Investigador impulsivo, y jamás resumía su investigación para beneficio de otros a través de la publicación. En sus cuadernos hacía la anotación de sus investigaciones de derecha a izquierda, posiblemente por comodidad, debido a que era zurdo.

- Otro gran genio de ese tiempo fue Galileo, quien a la edad de 25 años fue nombrado profesor de matemáticas en la Universidad de Pisa. Estudió mecánica, descubrió la ley fundamental de la caída de los cuerpos y estudió el comportamiento del movimiento armónico del péndulo. Dictó conferencias sobre astronomía en Padua y Florencia, y posteriormente fue acusado ante la Inquisición, en 1633, debido a su creencia de que el Sol era el centro de nuestro universo y no la Tierra como se pensaba.

- Simón Stevin en Holanda, a fines de la década de 1500 realizo uno de los descubrimientos más importantes en la historia de la ingeniería mecánica conocido como el “triángulo de fuerzas", el cual permitió a los ingenieros manejar fuerzas resultantes que actuaban en los miembros estructurales.

- Entre 1640, Fermat y Descartes descubrieron independientemente la geometría analítica.

- William Oughtred, sacerdote inglés, aproximadamente en 1622, diseñó la primera regla de cálculo basada en la suma de logaritmos para obtener el producto de dos números.

- Descartes y Leibmz descubrieron en forma independiente el cálculo diferencial. Newton descubrió el cálculo integral, y luego describió la relación recíproca entre los cálculos diferencial e integral. Sus descubrimientos ocurrieron en Woolsthorpe, aproximadamente en 1665.

- En 1771 un pequeño grupo de ingenieros, a los que se llamaba frecuentemente para dar su testimonio sobre proyectos de puertos y canales, formó la Sociedad de Ingenieros. John Smeaton, director del grupo, fue el primero en darse el título de ingeniero “civil" para señalar que su incumbencia no era militar. Esta sociedad se constituyó en la Institution of Civil Engineering en 1828, iniciando con ello una especialización dentro de la ingeniería.

- Thomas Savery idea la máquina de vapor, aunque otros anteriores a él aportaron ciertos adelantos menores en ese campo. En 1698 recibió una patente por un dispositivo operado por vapor para drenar minas; lo anunció en un libro que escribió más tarde, y que intituló Tire Mines Friend. En 1712, Thomas Newcomen mejoró mucho la máquina de vapor, la que también se usaba para bombear agua de una mina. Estas primeras máquinas eran muy deficientes, aunque representaban el desarrollo inicial de la energía a partir de máquinas térmicas.

- Robert Boyle estudió la elasticidad del aire y descubre la ley que relaciona la temperatura, presión y volumen, que hoy día lleva su nombre. Robert Hooke experimentó con la elasticidad de los metales y descubrió la ley de la elasticidad que también lleva su nombre. Christian Huygens determinó las relaciones de la fuerza centrípeta y Sir Isaac Newton estableció las tres leyes básicas del movimiento.

- En 1804, Richard Trevithick fue el primero en lograr que una locomotora de vapor corriera sobre rieles. Más tarde demostró que las ruedas lisas podían correr sobre rieles lisos si las pendientes no eran demasiado excesivas. George Stephenson a los treinta y dos años, construyó su primera locomotora de vapor.

- La invención de los automóviles y aeroplanos en los Estados Unidos fueron factores significativos en el desarrollo ingenieril del siglo XX. Los inventos de Tomás Edison, iniciaron la industria de la energía, y el invento de Lee De Forest de la "válvula electrónica" (tubo al vacío), dieron considerable ímpetu a la industria de las comunicaciones.

1.2 EL ORIGEN DE LA INGENIERIA

La Ingeniería apareció con el primer ser humano. Se puede hablar de Ingeniería desde el primer momento en que se dio forma a una piedra para convertirla en una herramienta, o cuando los primeros humanos usaron la energía de forma consciente al encender una hoguera. Desde entonces, el desarrollo de la misma ha ido equivalente con el de la Humanidad. [6]

En la antigüedad hubo grandes logros entre los cuales están:

- La construcción de canales y acueductos, que hicieron posible la aparición de ciudades y la expansión de la agricultura.
- La sustitución de la energía humana por otros tipos de energía.
- El uso de bueyes y, posteriormente con la aparición del arado de caballos (que eran más rápidos y eficientes que los bueyes), permitió al hombre disponer de nuevas fuentes motrices.
- El reemplazo de la energía animal por la mecánica, dando inicio al periodo que se conoce como Revolución Industrial.

Invenciones en esta época:

- Creaciones arquitectónicas como las catedrales.
- el reloj de contrapeso y la imprenta, inventada por Gutenberg en 1450.
- Galileo Galiley, conocido por sus observaciones astronómicas y por su declaración de que objetos de diferentes masas se ven sometidos a la misma "tasa" de caída, éste intentó desarrollar teorías tensionales para estructuras, aunque sus predicciones fueron erróneas al no considerar la elasticidad de los materiales, poco tiempo después Robert Hooke publicó el primer artículo sobre elasticidad (1678) que sentó las bases de la actual teoría de la elasticidad.
- Otto Von Guericke inventó la primera bomba de aire en 1672.
- El desarrollo de un cilindro con un pistón móvil sería crucial para el posterior desarrollo del "motor de fuego". Sólo faltaba mover el pistón con energía calorífica. Esto lo consiguió Denis Papin en 1691, sentando las bases del motor de vapor que, en 1705, Thomas Newcomen puso en práctica. Su motor era útil y práctico, pero lento e ineficiente. Tuvieron que pasar casi 70 años hasta que James Watt

[6] http://www.arqhys.com/origen-de-la-ingenieria.html

- (1736-1819) presentara su máquina de vapor (1774), base de la Revolución Industrial. El motor de vapor cambió radicalmente las factorías existentes hasta entonces, basadas en molinos de agua o de viento. A partir de ese momento, las fábricas podían situarse prácticamente en cualquier lugar. El desarrollo de fábricas trajo consigo la necesidad de combustible en grandes cantidades que, además, proporcionara suficiente poder calorífico para fundir hierro. La solución a esta necesidad la proporcionó el carbón.

- Sin duda alguna uno de los grandes logros de ese periodo fue la construcción del ferrocarril de costa a costa de los Estados Unidos (1.862-1.869).

- En el caso de la eléctrica, se situarían sus comienzos en 1831, llegando hasta nuestros días. Aunque se habían realizado experimentos antes (Oersted, Ampére), fue Michael Faraday quien formuló el principio fundamental en el cual se basa toda la industria de generación eléctrica actual: *"se puede inducir corriente eléctrica a partir de cambios en un campo magnético".*

- El desarrollo del telégrafo en 1835 por Samuel F.B. Hore la cual sentó las bases de lo que hoy conocemos como ingeniería de Telecomunicación. En esa misma década aparecieron los primeros motores eléctricos aunque pesados, con poca autonomía y poco eficientes.

- Aparición del alumbrado eléctrico (Thomas Edison, 1879), y para 1890 ya se habían desarrollado modernos generadores con lo que todo estaba dispuesto para que la industria pudiera hacer uso de la energía eléctrica esto dispara la demanda de electricidad.

A comienzos del siglo XX, se entra en una dinámica de desarrollos no conocida hasta entre los cuales: *el teléfono, la aparición de los aviones*. Esto permitió grandes contribuciones a la ingeniería, plasmadas en trabajos tales como los de Nikola Tesla, Thomas Edison o Stephen Timoshenko. De hecho, se han producido dos desarrollos que han afectado profundamente a la ingeniería y sin duda tendrán una gran repercusión en el futuro: la aparición de la mecánica cuántica y la teoría de la relatividad (Albert Einstein y otros). De manera igual el avance en la electrónica (tubos de vacío) y posteriormente de estado sólido, con la consecuencia de la invención del microprocesador y a partir de él, de la informática como herramienta de ingeniería.

1.3 LOS CAMPOS DE LA INGENIERIA ACTUAL

1.3.1 Ingeniería aeronáutica y aeroespacial

La ingeniería aeronáutica es un área que investiga, diseña, manufactura y mantiene el buen funcionamiento de aviones, misiles y satélites espaciales. Se relaciona con los temas científicos de la Aerodinámica, Materiales, Tecnología, Estructuras de aviones y Mecánica de fluidos.

Debido al desarrollo de la industria aeroespacial, actualmente se habla más de "Ingeniería Aeroespacial" que de "Ingeniería Aeronáutica", aunque también se escucha el término "Ingeniería Aeronáutica y del Espacio".

Los nuevos ingenieros aeroespaciales tienen un perfil profesional muy demandado desde finales del siglo XX y principios del XXI.[7]

[7] http://es.wikipedia.org/wiki/Aeron%C3%A1utica

Algunos campos de la ingeniería aeronáutica son la aerodinámica, la propulsión, el control y el análisis de estructuras.

La ingeniería aeroespacial se ocupa del vuelo fuera de la atmósfera, aunque el término se usa más para designar todas las actividades comprendidas en un amplio sector que para denotar un campo específico de la ingeniería. Así, el papel de los ingenieros electrónicos y de software es fundamental en la construcción de vehículos aeroespaciales, pero igualmente vital es la participación de ingenieros especialistas en diseño mecánico, en estructuras ultraligeras, en materiales avanzados, en robótica, etc.

1.3.2 Ingeniería agrónoma

Denominada también como ingeniería agronómica, es el conjunto de conocimientos de diversas ciencias aplicadas, que rigen la práctica de la agricultura y la ganadería.

Tiene como objetivo mejorar la calidad de los procesos de producción y transformación de productos agrícolas y alimentarios; fundamentada en principios científicos y tecnológicos; estudia los factores físicos, químicos, biológicos, económicos y sociales que influyen o afectan al proceso productivo. Su objeto de estudio es el fenómeno complejo o proceso social del agroecosistema, entendido éste como el modelo específico de intervención del hombre en la naturaleza, con fines de producción de alimentos y materia prima.[8]

El ingeniero agrónomo aplica principios de ingeniería a la resolución de problemas tales como la producción, almacenamiento y procesado de alimentos, la gestión de residuos agrícolas, irrigación y drenaje, y otras actividades propias de la agricultura e industrias cercanas. La ingeniería agrónoma requiere una comprensión general de aspectos tales como biología, gestión del suelo etc., además de conocimientos básicos de los fundamentos de la ingeniería.

Algunas ramas de esta ingeniería son:

- Ingeniería de la alimentación.
- Regadíos.
- Maquinaria.
- Estructuras.
- Ingeniería forestal.
- Medio ambiente.

Relacionada con la ingeniería agrónoma, se encuentra la biotecnología en procesos basados en microbios y enzimas. Ejemplos industriales son el procesador de alimentos y la producción de vitaminas, hormonas y antibióticos

1.3.3 Ingeniería biomédica

La ingeniería biomédica es el resultado de la aplicación de los principios y técnicas de la ingeniería al campo de la medicina. Ésta se dedica fundamentalmente al diseño y construcción de productos sanitarios y tecnologías sanitarias tales como:

- Equipos médicos
- Prótesis

[8] http://es.wikipedia.org/wiki/Agronom%C3%ADa

- Dispositivos médicos
- Dispositivos de diagnóstico (imagenología médica) y de terapia.[9]

La ingeniería biomédica interviene en la gestión o administración de los recursos técnicos ligados a un sistema de hospitales, lo cual combina la experiencia de la ingeniería con las necesidades médicas para obtener beneficios en el cuidado de la salud tales como el cultivo de tejidos, la producción de determinados fármacos por lo cual suele considerarse parte de la bioingeniería.

La ingeniería biomédica tiene sus orígenes en tres campos:

1. *Bioingeniería*, o aplicación de conceptos ingenieriles a fenómenos biológicos.
2. *Ingeniería médica*, desarrollo de instrumentación, órganos artificiales, ortopedia, etc.
3. *Ingeniería clínica*, que se ocupa en general de la instrumentación necesaria para la operación de un hospital moderno.

1.3.4 Ingeniería química

Esta área aplica principios químicos, físicos y de ingeniería para resolver problemas y proporcionar compuestos de todo tipo, desde productos farmacéuticos hasta combustibles, pasando por productos químicos industriales. En esta rama se toma en cuenta el control de la contaminación y la conservación de la energía. Algunas áreas presentes y futuras de la ingeniería química incluyen la industria de semiconductores, la gestión medioambiental, procesado de fuentes de energía moderna y tradicional, desarrollo de pinturas, prevención de la corrosión, entre otras.

El Ingeniero químico es el profesional que se encarga de planear, diseñar, construir y administrar las plantas de procesos químicos. Estas permiten producir, a partir de materias primas disponibles, productos intermedios, finales y/o materiales necesarios para la vida moderna, tales como alimentos, fertilizantes y pesticidas, combustibles, materiales de construcción, fibras sintéticas, plásticos, pigmentos y colorantes, medicamentos, bebidas, jabones, etc.

El Ingeniero químico puede desempeñar cualquiera de las siguientes funciones:

- ✓ Manejo y control de la producción en las industrias de procesos.
- ✓ Planificación, desarrollo y administración de las industrias de procesos.
- ✓ Control de la calidad de materias primas y procesos de la Industria Química.
- ✓ Docencia e investigación en las ciencias de la Ingeniería Química. [10]

1.3.5 Ingeniería civil y de la construcción

La ingeniería civil es la rama de la ingeniería que aplica los conocimientos de física, química, cálculo y geología a la elaboración de infraestructuras, obras hidráulicas y de transporte.[11]

[9] http://es.wikipedia.org/wiki/Ingenier%C3%ADa_biom%C3%A9dica
[10] http://fiq.uni.edu.ni/carr_perfil.php
[11] http://www.ftc.uni.edu.ni/ingcivil.html

El Ingeniero civil, es un profesional capacitado para utilizar apropiadamente los materiales y la energía, aplicando tecnologías para transformarlos en obras para beneficio de la comunidad. Tales como:

- ✓ Sistemas viales
- ✓ Sistemas sanitarios
- ✓ Sistemas hidráulicos
- ✓ Sistemas estructurales
- ✓ Sistemas de transporte
- ✓ Sistemas de protección y conservación ambiental.

El Ingeniero civil tiene amplios conocimientos de físicas, matemáticas, humanidades y puede realizar las siguientes funciones:

- ✓ Planificación, estudios, diseño y construcción de obras de Ingeniería.
- ✓ Sistemas y métodos constructivos, inspección, supervisión y control de obras.
- ✓ Operación y mantenimiento de obras y sistemas.
- ✓ Dirección y/o Administración en Empresas de Ingeniería Civil.
- ✓ Utilización de sistemas informáticos, aplicados a la Ingeniería.
- ✓ Docencia e investigación.

1.3.6 Ingeniería del software

La ingeniería en software es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iníciales de la especificación del sistema, hasta el mantenimiento de este después de que se utiliza. Esta ingeniería trata con áreas muy diversas de la informática y de las ciencias de la computación, tales como construcción de compiladores, sistemas operativos, o desarrollos Intranet/Internet, abordando todas las fases del ciclo de vida del desarrollo de cualquier tipo de sistemas de información y aplicables a infinidad de áreas: negocios, investigación científica, medicina, producción, logística, banca, control de tráfico, meteorología, derecho, Internet, Intranet, etc.

De este modo, podemos resumir las funciones de un Ingeniero de Software en los siguientes puntos:

- Gestionar los proyectos software.
- Realizar la gestión de configuración para los proyectos software.
- Asegurar la calidad en los proyectos software.
- Desarrollar sistemas software.
- Diseñar software empleando tanto las técnicas estructuradas como las técnicas orientadas a objeto.
- Utilización de herramientas CASE (Computer Aided Software Engineering; y en su traducción al Español significa Ingeniería de Software Asistida por Computación), estas herramientas permitirán organizar y manejar la información de un proyecto informático. Permitiéndole a los participantes de un proyecto, que los sistemas (especialmente los complejos), se tornen más flexibles, más comprensibles, mejorando la comunicación entre los participantes, realizar estrategias de prueba y validar sistemas software.
- Mantener el software y aplicar la ingeniería inversa para actualizar desarrollos software.
- Diseñar interfaces de usuario por ordenador.
- Realizar la documentación de usuario correspondiente al proyecto software.

1.3.7 Ingeniería de telecomunicaciones

La ingeniería de telecomunicaciones en una rama de la ingeniería que resuelve problemas de transmisión y recepción de señales e interconexión de redes. El término telecomunicación se refiere a la comunicación a distancia a través de la propagación de ondas electromagnéticas. Esto incluye muchas tecnologías, como radio, televisión, teléfono, comunicaciones de datos y redes informáticas. La definición dada por la Unión Internacional de Telecomunicaciones (ITU, *International Telecommunication Union*) para telecomunicación es toda emisión, transmisión y recepción de signos, señales, escritos e imágenes, sonidos e informaciones de cualquier naturaleza, por hilo, radioelectricidad, medios ópticos u otros sistemas electromagnéticos.[12]

En función de la especialidad elegida durante la carrera, los ingenieros de Telecomunicación pueden ser especialistas en Radio, Transmisión, Informática, Automática, Microelectrónica, etc.

Las funciones más frecuentes de un Ingeniero de Telecomunicación en el ámbito laboral son la realización de proyectos tales como, el diseño de circuitos integrados, proyectos de telefonía móvil, instalación de redes telemáticas, diseño de software de control y aplicaciones, instrumentación, así como la de investigación y desarrollo que se lleva a cabo principalmente en grandes empresas de telecomunicación, informática y electrónica. También aparecen las funciones gerenciales, es decir, aquellas en las que el ingeniero aplica los conocimientos adquiridos a lo largo de su vida profesional en combinación con otras de carácter administrativo para dirigir empresas relacionadas con la actividad de las telecomunicaciones.

1.3.8 Ingeniería eléctrica

Se puede decir que el término ingeniería eléctrica es muy amplio, incluyendo actividades tales como la generación y transmisión de energía, motores eléctricos, sistemas eléctricos para todo tipo de edificios e instalaciones, etc. Tiene mucha relación con ella la ingeniería electrónica que, aunque por sí misma constituye una especialización distinta, tradicionalmente está ligada a la eléctrica y a las telecomunicaciones en sus campos de potencia y microelectrónica respectivamente.

1.3.9 Ingeniería medioambiental

La **ingeniería ambiental** es la rama de la ingeniería que estudia los problemas ambientales de forma integrada, teniendo en cuenta sus dimensiones ecológicas, sociales, económicas y tecnológicas, con el objetivo de promover un desarrollo sostenible.

La ingeniería ambiental garantiza mediante la conservación y preservación de los recursos naturales, una mejor calidad de vida para la generación actual y para las generaciones futuras. Ésta a su vez, ve como necesidad la proporción de una serie de soluciones para enfrentar la actual crisis ecológica que vive el planeta. El ingeniero ambiental debe saber reconocer, interpretar y diagnosticar impactos negativos y positivos ambientales, evaluar

[12] http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_telecomunicaci%C3%B3n

el nivel del daño ocasionado en el ambiente (en el caso de un impacto negativo) y proponer soluciones integradas de acuerdo a las leyes medioambientales vigentes, con el fin de tener una mejor calidad de vida.[13]

1.3.10 Ingeniería de organización industrial

La ingeniería de organización es un campo interdisciplinar con aplicaciones industriales, de servicios, comerciales y de gestión. Según el Instituto Americano de Ingeniería de Organización (AIIE), el ingeniero de organización se ocupa del diseño, mejora e instalación de sistemas integrados de personas, materiales y energía. Algunos campos de la ingeniería de organización son la planificación y gestión de la producción, la ingeniería y el diseño de plantas industriales, la gestión de la calidad, el establecimiento de objetivos, entre otros.

1.3.11 Ingeniería naval

El ingeniero naval, es conocido en varios países como Arquitecto Naval, éste se encarga del diseño, planificación, proyecto y construcción buques, embarcaciones, y artefactos flotantes como pudieran ser plataformas petrolíferas e incluso campos eólicos offshore[14]. La ingeniería naval abarca las funciones de ingeniería incluyendo el proyecto creativo del buque y artefactos flotantes, la investigación aplicada, el desarrollo técnico en los campos de diseño y construcción y la administración de los centros de producción de material flotante (astilleros). Así como también del mantenimiento y reparación de estos.

El ingeniero naval posee conocimientos de muchos campos de la ingeniería, tales como generación y transporte de energía eléctrica, fabricación de motores navales y su instalación, estructuras metálicas dinámicas, logística, actividad portuaria, organización industrial, gestión de flotas y navieras, etc. Ésta ingeniería combina los campos del transporte, exploración, ingeniería militar y obtención de recursos naturales.

1.3.12 Ingeniería de materiales

El ingeniero de materiales se ocupa del desarrollo de nuevos materiales o la mejora de los ya existentes a partir del conocimiento de las propiedades y comportamiento de los mismos. Es decir, el ingeniero de materiales investiga las propiedades mecánicas, eléctricas, térmicas, electrónicas de los materiales, además de desarrollar los métodos de producción de los mismos. Algunas especialidades dentro de la ingeniería de materiales son la ingeniería metalúrgica, la ingeniería de materiales cerámicos y la ingeniería de materiales compuestos.

Los coches, la ropa y el calzado, el equipo deportivo, los ordenadores o las prótesis y dispositivos biomédicos se fabrican con materiales cada vez más modernos, incluso basados en la nanotecnología. En estos campos, como en muchos otros, un nuevo material ha sido la clave que ha permitido desarrollar nuevos productos y aplicaciones. Así ha sucedido con los materiales compuestos en aeronáutica y en el deporte de alta competición[15].

[13] http://es.wikipedia.org/wiki/Ingenier%C3%ADa_ambiental

[14] Un campo eólico es una agrupación de aerogeneradores que transforman la energía eólica en energía eléctrica. Tom
http://es.wikipedia.org/wiki/Parque_e%C3%B3lico.

[15] http://ingenierodemateriales.blogspot.com/2011/05/ingenieria-de-materiales.html

1.3.13 Ingeniería de diseño y análisis mecánico

La característica fundamental del ingeniero de diseño mecánico es la realización de cálculos basados en la aplicación de las matemáticas y leyes físicas.

Esto se denomina análisis que se entiende como la obtención de una respuesta a un problema por medio de sucesivos pasos de razonamiento, cada uno de los cuales se basa en los resultados del precedente. El ingeniero de diseño mecánico desarrolla máquinas para resolver problemas en los más diversos campos de la actividad humana como: Máquinas herramienta, automóviles, vehículos ferroviarios, maquinaria de construcción, aeronáutica, etc.

1.3.14 Ingeniería de fabricación

El ingeniero de fabricación planifica, desarrolla y optimiza los procesos de producción, incluyendo los métodos de fabricación y el diseño de herramientas y equipos para producción. Típicas áreas de competencia son el diseño de herramientas, la planificación de la fabricación, el control numérico, la fabricación asistida por ordenador, la automatización de la producción, robótica, desarrollo de tecnologías avanzadas de fabricación. Ejemplos de esto último pueden encontrarse en la soldadura por láser y por haz de electrones, los nuevos métodos adhesivos, los nuevos materiales para herramientas de fabricación de materiales compuestos, etc.[16]

1.3.15 Ingeniería energética

Este ingeniero se ve involucrado en el desarrollo y aplicación de nuevas fuentes de energía como la eólica, la solar, la proveniente de las mareas, etc. Las técnicas que utilizan habitualmente se basan en la aplicación de la termodinámica, mecánica de fluidos, aerodinámica y transmisión de calor.

1.3.16 Ingeniería de minas

El ingeniero de minas trabaja con depósitos minerales de todo tipo, desde el momento de su descubrimiento, durante su evaluación y explotación. Trabajan en otras áreas tales como la investigación, la seguridad, diseño o la gestión.

[16] http://www.ingeniaritza-bilbao.ehu.es/p224-content/es/contenidos/informacion/cesi_profesion_del_ingeniero/es_campos/fabricacion.html

1.3.17 Ingeniería nuclear

Esta rama de la ingeniería comprende la generación, control y uso de todos los aspectos relacionados con la energía de origen nuclear. Abarca el estudio de las interacciones nucleares, moleculares y radiactivas, además de sus aplicaciones a problemas para el beneficio y el interés de la sociedad. Éstos contribuyen a la vitalidad y la salud de la sociedad gracias a su uso extendido para la generación eléctrica e industrial o la diagnosis médica, y como herramienta indispensable para la investigación científica en campos tales como la investigación farmacéutica o los estudios medioambientales. [17]

1.3.18 Ingeniería petrolífera

La ingeniería petrolífera es la parte de la ingeniería, combina métodos científicos y prácticos orientados al desarrollo y utiliza técnicas para descubrir, explotar, desarrollar, transportar, procesar y tratar los hidrocarburos desde su estado natural, en el yacimiento, hasta los productos finales o derivados. Se encarga además de la planificación, desarrollo y producción de campos petrolíferos y de gas.

1.3.19 Ingeniería de sistemas

El ingeniero de sistemas se encarga de analizar un sistema real y comprobar si se comporta según fue diseñado. Las técnicas usadas habitualmente son la estadística y probabilidad, teoría de control, modelización de sistemas y programación.

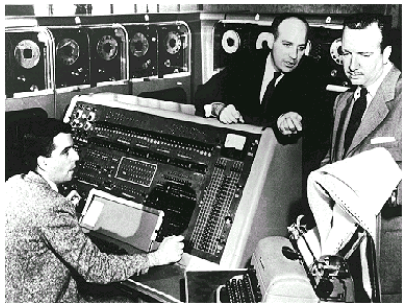
El Ingeniero de Sistemas, posee dominios y competencias en el análisis, diseño y programación de sistemas empresariales, que aplica las técnicas de las ciencias gerenciales, de la Informática y negocios o administración de empresas, con una formación humanística, básica y profesional sólida, que contribuye a su desempeño profesional con calidad, en cualquier campo del área en la que labore dentro de la empresa de bienes y servicios, dando respuesta a las demandas planteadas por la gerencia. [18]

1.3.20 Ingeniería en computación

El ingeniero en computación implementa aplicaciones avanzadas desde la definición de la aplicación hasta la programación, cooperando con ingenieros u otros profesionales que plantean problemas. Además asesora y califica a los demás usuarios para utilizar eficientemente el equipo de computación. El ingeniero en computación puede desarrollarse en diversos ámbitos, entre algunos se puede mencionar: análisis, diseño y desarrollo de diversos tipos de software, administración de centros de cómputo, empresas con procesos automatizados, desarrollo web, empresa de auditaje informático.[19]

[17] http://mit.ocw.universia.net/Nuclear-Engineering/index.htm
[18] http://www.ies.uni.edu.ni/Ingeneriasistema.html
[19] http://www.ies.uni.edu.ni/computacion.html

1.4 HISTORIA DE LA COMPUTADORA



Por siglos los hombres han tratado de usar fuerzas y artefactos de diferente tipo para realizar sus trabajos, para hacerlos más simples y rápidos. La historia conocida de los artefactos que calculan o computan, se remonta a muchos años antes de Jesucristo.[20]

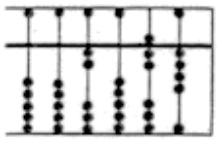
Dos principios han coexistido con la humanidad en este tema. El primero es usar cosas para contar, ya sea los dedos, piedras, semillas, etc. El segundo es colocar esos objetos en posiciones determinadas. Estos principios se reunieron en el ábaco, instrumento que sirve hasta el día de hoy, para realizar complejos cálculos aritméticos con enorme rapidez y precisión.

El Abaco

Al principio, los primeros dispositivos para contar fueron las manos humanas y los dedos. Al contar largas cantidades (más de lo que 10 dedos humanos podían representar) se vio la necesidad de utilizar varios artículos naturales como piedrecillas y ramitas las cuales se utilizaron para contar. Los comerciantes quienes negociaban artículos, no solo necesitaban una buena forma para contar lo comprado y lo vendido, sino también para calcular el costo de esos artículos. Hasta que los números fueron inventados, los dispositivos para contar eran usados para hacer cálculos todos los días.

El Ábaco se considera el primer dispositivo mecánico de contabilidad que existió, éste fue inventado en China unos 2.500 años AC, más o menos al mismo tiempo que apareció el soroban[21], una versión japonesa del ábaco.

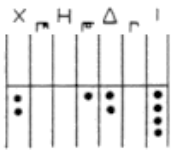
Japanese



Soroban

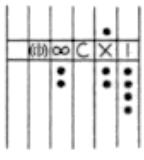
En general el ábaco, en diferentes versiones era conocido en todas las civilizaciones de la antigüedad. En China y Japón, su construcción era de alambres paralelos que contenían las cuentas encerrados en un marco, mientras en Roma y Grecia consistía en una tabla con surcos grabados.

Greek

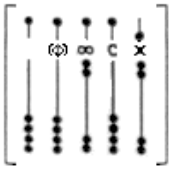


Salamis

Roman

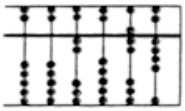


Calculi



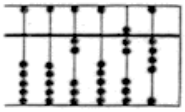
Hand-abacus

Chinese



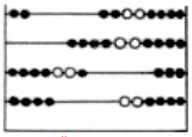
Suan-pan

Japanese



Soroban

Russian

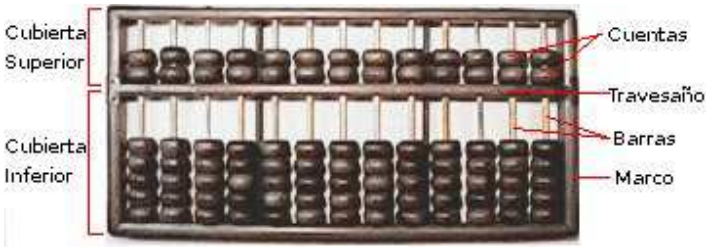


Ščët

[20] galeon.com/histoyevoldelpc/descargas/generacionea1.doc
[21] En Japón, el ábaco es llamado *soroban*

El ábaco estándar puede ser usado para efectuar adiciones, sustracciones, divisiones y multiplicaciones; el ábaco también puede ser usado para extraer la raíz cuadrada y la raíz cubica.

El ábaco es típicamente construido de varios tipos de maderas duras y viene en variados tamaños. El marco del ábaco tiene una serie de barras verticales en cuales un número de cuentas de madera son permitidas para mover libremente. A través de una barra horizontal (travesaño) la cual separa el marco interior en dos secciones, conocida como la cubierta superior y la cubierta inferior.[22]



- **Otros inventos**

RESUMEN DE LOS INVENTOS QUE DIERON PASO A LA COMPUTACION		
Año	Inventor	Aportes
1614	John Napier	Creación de un método que reduce el trabajo de cálculos matemáticos, en una tabla de logaritmos.
1642	Blas Pascal	Diseña la Pascalina, una máquina para calcular basada en ruedas dentadas las cuáles sumaban y restaban hasta seis cifras.
1694	Liebnist	Perfeccionó la Pascalina que dividía, multiplicaba y sacaba raíz cuadrada.
1804	Joseph Marie Jaquard	En Francia crea una máquina que tejía diseños en tela a través de una tarjeta perforada, que después se implementó en la automatización de datos.
1834	Charles Babbage	Marca el principio y el funcionamiento del ordenador a través de la cual resolvía polinomios de hasta 8 términos, ha dicha máquina se le llamó Diferencial.
1854	Georges Boole	Desarrolla un sistema para representar las proporciones lógicas por medio de signos matemáticos de modo de determinar una serie de reglas en las cuales la proporción se manifestaba si era verdadera o falsa, dio origen a la lógica simbólica, en su honor se le denominó Booleana al álgebra que hace operaciones lógicas.
1880	Herman Hollerit	Aplica el sistema de tarjetas perforadas dándole un uso estadístico, realiza el censo de 1890 en Estados Unidos y el crea un sistema en el que aplicó el criterio de una cinta perforada, algunas de estas tarjetas fueron programadas por la condesa de Lovelance Lady Augusta Ada Byron quien se considera la primera programadora de tarjetas perforadas, esto duró para su perfeccionamiento 2 años y una vez logrado, obtenía la información a gran velocidad, sustituyó cintas por tarjetas, fundó la Sociedad de Informática y de el surge la compañía IBM.

[22] http://www.ee.ryerson.ca/~elf/abacus/espanol/intro.html

1938	Shannon	Aplica el álgebra Booleana para crear una unidad básica de información.
1944	VonNewman	Pone en funcionamiento la idea de un programa interno y desarrolla el fundamento teórico para la construcción de un ordenador.

Primeras computadoras

En 1937 Howard Aiken, estudió en Harvard, ideó una gigantesca calculadora llamada Mark I o Automatic Secuence Controled, la tarea de esta máquina se realizó en IBM, su funcionamiento se fundaba en el RELE que es un dispositivo electrónico que permite abrir y cerrar un circuito.

En 1943 funciona el primer ordenador llamado ENIAC (Electronic Numerical Integrator And Calculator), creado por Mauchly Eckert Goldfine. Estaba compuesto de válvulas que se quemaba constantemente. Su peso era de

130 toneladas ocupaba un sótano de la universidad tenía más de 18,000 tubos de vacío consumía 200 kw pero efectuaba alrededor de cinco mil operaciones aritméticas por segundo.

En 1945 Eckert y Mauchly mismos que trabajaron en la máquina anterior construyeron la EDVAC (Electronical Discrete Variable Automatic Computer) realizaba operaciones aritméticas con números binarios y almacenaba instrucciones.

En 1948 William Shockley creó la primera computadora de transistores superando rápidamente a los bulbos ya que estos ocupan menos peso, tamaño y energía eléctrica, superando ahorra por mil millones de veces a la máquina de Babbage.

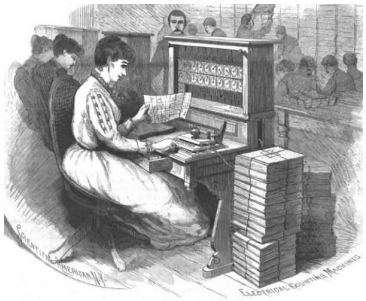
En 1951 la Cía Remington Rand fundada por Eckert y Mauchly desarrollan la primera computadora comercial en 1951 las características son el uso de cinta magnéticas; usaba un lenguaje particular, usaban bulbos y manejaba memorias de ferrita.

En 1960 se crea la regla de cálculo y surge el primer antecedente de un dispositivo analógico.

En 1962-1963 surgen los discos magnéticos, las computadoras (como las actuales), se reduce su tamaño, surgen los chips orgánicos y la micro tecnología que alcanza su mayor esplendor en 1968 con el descubrimiento del microscopio electrónico[23].

1.5 ETAPAS DE LA EVOLUCION DE LA COMPUTADORA

1.5.1 Primera generación (1950 a 1959)



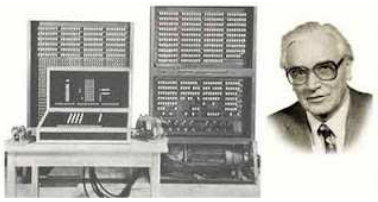
En esta generación se caracteriza por la construcción de maquinas, válvulas y programación en ***lenguaje maquina***, esta etapa abarca la década de 1950.

Herman Hollerith Funda la empresa I.B.M.
En los comienzos de la industria de la computación se desconocía las capacidades y alcances. Funcionaba por tubos de vacío y mediante

[23] hanskelsen.gdl.iteso.mx/INFOR1.doc

programación en lenguaje máquina o lenguaje binario, tenían velocidad limitada, eran máquinas grandes y costosas. Un ejemplo de esta generación fue la ***UNIVAC I***, Remington Rand 1103, IBM 701, Burrought 220, la más exitosa fue la 650 de IBM.

1.5.2 Segunda generación (1959-1964)



Aparece por primera vez la aplicación del transistor, los sistemas operativos, los lenguajes de alto nivel y los discos magnéticos.

Estas computadoras eran un poco más pequeñas, con más capacidad de memoria que las anteriores, y con menos costo, por nadie sabía con precisión para que pudieran ser útiles, la forma de comunicación con estas nuevas computadoras es mediante lenguajes más avanzados que el lenguaje de máquina, y que reciben el nombre de "lenguajes de alto nivel" o lenguajes de programación.

Se podían comunicar entre sí, aumentan su velocidad, estaban formadas por circuitos de transistores, se programan en nuevos lenguajes de alto nivel. Además, en 1951, Maurice Wilkes inventa la microprogramación, que simplifica mucho el desarrollo de las CPU.

La competencia fue más clara al salir la serie 5000 de Burrouhs la ATLAS de la universidad de Manchester, la Philco 212, las Control Data Corporation. Luego IBM perfecciona sus máquinas y saca la 7090. La RCA con el modelo 601 maneja el lenguaje COBOL.

1.5.3 Tercera generación (1964-1975)

Esta generación marca la aparición de circuitos integrados.

Las computadoras de la tercera generación surgieron con el desarrollo de los circuitos integrados (pastillas de silicio) en las cuales se colocan miles de componentes electrónicos, en una integración en miniatura. Las computadoras se hicieron más pequeñas, más rápidas, desprendían menos calor y eran energéticamente más eficientes.

El ingeniero Jack S. Kilby (nacido en 1928) de Texas Instruments, descubre en 1958 el primer circuito integrado (Chip), la cual se maneja por medio de lenguaje de control de sistemas operativos.

Antes del advenimiento de los circuitos integrados, las computadoras estaban diseñadas para aplicaciones matemáticas o de negocios, pero no para las dos cosas. Los circuitos integrados permitieron a los fabricantes de computadoras incrementar la flexibilidad de los programas, y estandarizar sus modelos.

La IBM 360 es una de las primeras computadoras comerciales que usó circuitos integrados, podía realizar tanto análisis numéricos como administración ó procesamiento de archivos.

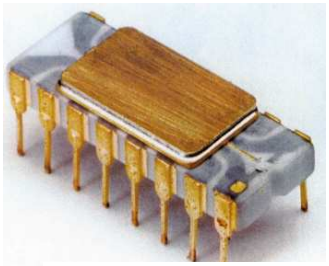
Se empiezan a utilizar los medios magnéticos de almacenamiento, como cintas magnéticas de 9 canales, paquetes de discos magnéticos, se usa el sistema operativo OS.
Japón produce la computadora Fujitsu comparables con las máquinas más grandes de IBM o CDC. Las grandes máquinas reciben el nombre de Mainframes.

Surgen las minicomputadoras, que no son tan costosas, pero disponen de una gran capacidad de proceso. Aparece la empresa Hewlett-Packard, Simens.

1.5.4 Cuarta generación (1975 a 1980)

Surge el microprocesador y la aplicación de la informática a través de datos, imagen, las telecomunicaciones y la robótica.

El reemplazo de las memorias con núcleos magnéticos por las de chips de silicio, así como la colocación de varios componentes en un chip son los dos avances característicos en esta generación. El tamaño reducido del microprocesador y de chips hizo posible la creación de las computadoras personales (PC).



Chip Intel 4004

En 1971, Intel Corporation, que era una pequeña compañía fabricante de semiconductores ubicada en Silicon Valley, presenta el primer microprocesador o chip de 4 bits, este fue el primer microprocesador del mundo, creado en un simple chip, que en un espacio de aproximadamente 4 x 5 mm contenía 2 250 transistores. Era un CPU de 4 bits y también fue el primero que se comercializó. Este desarrollo impulsó la calculadora de Busicom y dio camino a la manera para dotar de «inteligencia» a objetos inanimados, así como la computadora personal.

Esta generación de computadoras se caracterizó por grandes avances tecnológicos realizados en un tiempo muy corto. En 1977 aparecen las primeras microcomputadoras, entre las cuales, las más famosas fueron las fabricadas por Apple Computer, Radio Shack y Commodore Busíness Machines. IBM se integra al mercado de las microcomputadoras con su Personal Computer (figura 1.15), de donde les ha quedado como sinónimo el nombre de PC, y lo más importante; se incluye un sistema operativo estandarizado, el MS- DOS (MicroSoft Disk Operating System).

Las principales tecnologías que dominan este mercado son:

IBM y sus compatibles llamadas clones, fabricadas por infinidad de compañías con base en los procesadores 8088, 8086, 80286, 80386, 80486, 80586 o Pentium, Pentium II, Pentium III y Celeron de Intel y en segundo término Apple Computer, con sus Macintosh y las Power Macintosh, que tienen gran capacidad de generación de gráficos y sonidos gracias a sus poderosos procesadores Motorola serie 68000 y PowerPC, respectivamente. Este último microprocesador ha sido fabricado utilizando la tecnología RISC (Reduced Instruc tion Set Computing), por Apple Computer Inc., Motorola Inc. e IBM Corporation, conjuntamente.

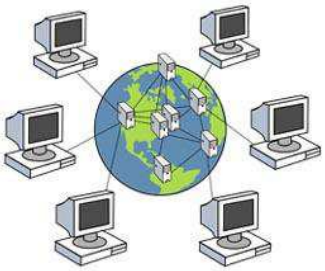
Los sistemas operativos han alcanzado un notable desarrollo, sobre todo por la posibilidad de generar gráficos a gran des velocidades, lo cual permite utilizar las interfaces gráficas de usuario (Graphic User Interface, GUI), que son pantallas con ventanas, iconos (figuras) y menús desplegables que facilitan las tareas de comunicación entre el usuario y la computadora, tales como la selección de comandos del sistema operativo para realizar operaciones de copiado o formato con una simple pulsación de cualquier botón del ratón (mouse) sobre uno de los iconos o menús.

1.5.5 Quinta generación y la inteligencia artificial (1980-1989)

En esta generación se pretende integrar el software con el hardware.

Se desarrolla el software y los sistemas que se manejan las computadoras. Se desea alcanzar el nivel de comunicarse con la computadora mediante lenguaje natural y no los de códigos o lenguajes de control especializados. También lograr un procesamiento paralelo mediante arquitectura y diseños especiales, circuitos de gran velocidad. Así como manejo de sistemas de inteligencia artificial, funcionando con superconductores y fibras ópticas. Como también con pantallas de tercera dimensión, sonido estéreo y colores más claros.

1.5.6 Sexta generación (1990 hasta la fecha)



Las computadoras de esta generación cuentan con arquitecturas combinadas paralelo / vectorial, con cientos de microprocesadores vectoriales trabajando al mismo tiempo; se han creado computadoras capaces de realizar más de un millón de millones de operaciones aritméticas de punto flotante por segundo (teraflops); las redes de área mundial (Wide Area Network, WAN), comunicación a través de fibras ópticas y satélites, con anchos de banda impresionantes. Las tecnologías de esta generación ya han sido desarrolladas o están en ese proceso. Algunas de ellas son: inteligencia / artificial distribuida; teoría del caos, sistemas difusos, holografía, transistores ópticos, etcétera.

1.6 CLASE PRÁCTICA

Historia de la ingeniería:

1. Crear mapas conceptuales para:

- Representar la historia de la ingeniería
- Ingeniería egipcia, mesopotámica, griega.
  Recuerde que para realizar un mapa conceptual debe realizar:
  - Análisis de la unidad
  - Identificación de idea principal y secundarias
  - Elaboración de lista de conceptos como base del mapa conceptual.

2. Enumere los avances que tuvieron los ingenieros romanos en comparación con los griegos.
3. ¿Cuál fue la innovación "*más interesante*" de los arquitectos romanos en esa época?
4. ¿Cuál fue el triunfo de la construcción pública más destacada en la ingeniería romana?
5. ¿Dónde aportaron mejoras significativas los romanos y cuáles fueron las razones de dichas mejoras?
6. Explique sobre los acueductos romanos.
7. Escriba sobre otros inventos que se produjeron en este periodo.
8. ¿En qué eran diestros los hindúes?
9. Investigue sobre "números arábigos"
10. ¿Qué es la muralla china?
11. ¿Quiénes fueron los primeros constructores de puentes? Argumente.
12. ¿Que otros inventos se les atribuye a los chinos?
13. ¿Qué invento contribuyo a la terminación de los castillos rodeados de murallas y quienes lo crearon?
14. Enumere los inventos más destacados por los europeos.
15. Enumere algunos de los genios europeos más destacados.

Campos de la Ingeniería actual:

1. Elabore un cuadro sinóptico en donde se establezcan los campos de la ingeniería aeronáutica y aeroespacial, agrónoma, biomédica, química, civil y de la construcción, del software.
2. Elabore un cuadro sinóptico en donde se establezcan los campos de la ingeniería de telecomunicaciones, eléctrica, medioambiental, de organización industrial, naval, de materiales.
3. ¿Qué actividades desarrolla un ingeniero de diseño y análisis mecánico?
4. ¿Cuál es la diferencia entre las actividades de un ingeniero en sistemas a un ingeniero en computación?

Historia de la computación:

1. Elabore un resumen sobre la historia de la computadora. Establézcalo en un cuadro sinóptico.
2. Elabore un cuadro sinóptico sobre las generaciones de la computadora.

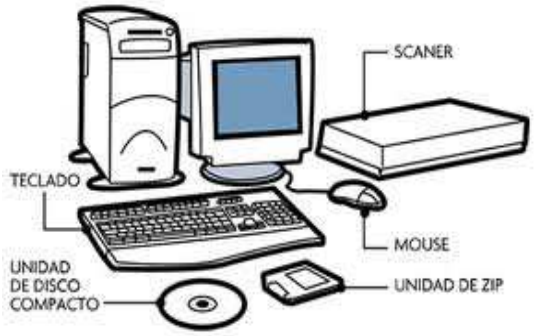
Capítulo 2

ORGANIZACIÓN BÁSICA DEL COMPUTADOR

2.1 INTRODUCCIÓN

El computador, habiendo sido diseñado para el procesamiento de datos, su organización es similar a la de cualquier otro proceso. Indiferente de lo que se desee procesar, siempre se tendrán tres elementos importantes, la materia prima, la transformación que es el proceso en sí, y el producto final, es decir la materia prima transformada en un nuevo producto. Así, el computador está conformado por dispositivos de entrada, unidad central de procesamiento, dispositivos de salida y adicionalmente memoria externa o dispositivos de almacenamiento.

Estos dispositivos son: teclado, ratón, escáner, micrófono, entre muchos otros, todos ellos permiten entrar datos al sistema. Los datos son transformados en señales eléctricas y almacenadas en la memoria central, donde permanecerán disponibles para ser procesados o almacenados en medios de almacenamiento permanente.



2.2 UNIDAD CENTRAL DE PROCESAMIENTO

Comúnmente se la conoce como CPU, que significa Central Processing Unit, ésta es quizá la parte más importante del computador, ya que en ella se encuentra la unidad de control y la unidad aritmético-lógica, las cuales en constante interacción con la memoria principal (también conocida como memoria interna) permiten manipular y procesar la información, y controlar los demás dispositivos de la unidad computacional.

2.3 DISPOSITIVOS DE ENTRADA

2.3.1 Memoria externa

También se la conoce como memoria auxiliar, ésta es la encargada de brindar seguridad a la información almacenada, por cuanto guarda los datos de manera permanente e independiente de que el computador esté en funcionamiento, a diferencia de la memoria interna (RAM) que solo mantiene la información mientras el equipo esté encendido. Los dispositivos de almacenamiento son discos y cintas principalmente, los discos pueden ser flexibles, duros u ópticos.

2.3.2 Disco Magnético

Es una superficie plana circular, puede ser plástica o metálica, recubierta con oxido de hierro. La superficie recubierta es magnetizada formando puntos microscópicos, cada uno de los cuales actúa como un pequeño imán permanente. Según la polarización de los puntos la señal puede indicar falso o verdadero, 0 o 1.

Existen dos tipos de discos magnéticos:

- Discos flexibles
- Discos duros

2.3.3 Discos flexibles

Estaban constituidos por una lámina magnética, recubierta por un plástico que la protege. Aunque existían de diferentes tipos, los más frecuentes fueron los discos 3 ½ pulgadas que almacenaban 1.44 megabytes. Estos fueron muy útiles por ser pequeños y fáciles de portar sin embargo su capacidad de almacenamiento era pequeña y su velocidad de acceso baja.

Los discos flexibles tenían la desventaja de dañarse con facilidad, por ello era importante tener en cuenta algunos cuidados, tales como:

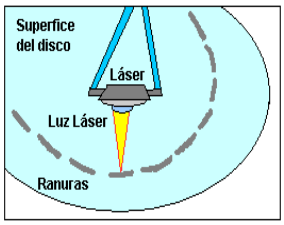
- × No doblarlos, ni arquearlos
- × No presionarlos
- × No acercarlos a campos magnéticos

2.3.4 Discos duros

A diferencia de los discos flexibles, estos están hechos generalmente de aluminio, giran a una velocidad 10 veces mayor y su capacidad de almacenamiento es muy grande (40 gigabytes). Un disco duro es un paquete herméticamente cerrado, conformado por varios discos o placas, sus respectivas cabezas de lectura/escritura y la unidad de disco. El disco duro constituye el medio de almacenamiento más importante de un computador ya que se encuentra normalmente dentro del ordenador.

2.3.5 CD-ROM

CD-ROM de solo lectura (sigla s del inglés ***C**ompact **D**isc - **R**ead **O**nly **M**emory*). Estos discos forman parte de la nueva tecnología para el almacenamiento de información. Esta tecnología consiste en almacenar la información en forma de pozos y planos microscópicos que se forman en la superficie del disco. Un haz de un pequeño láser en el reproductor de CD-ROM ilumina la superficie y refleja la información almacenada. Un disco compacto de datos, en la actualidad, almacena 650 y 700 megabytes de información.



2.4 DISPOSITIVOS DE SALIDA

Permiten presentar los resultados del procesamiento de datos, son el medio por el cual el computador presenta información a los usuarios. Los más comunes son la pantalla y la impresora.

2.4.1 Pantalla o monitor

Exhibe las imágenes que elabora de acuerdo con el programa o proceso que se esté ejecutando, puede ser videos, gráficos, fotografías o texto. Es la salida por defecto donde se presentan los mensajes generados por el computador, como errores, solicitud de datos, etc.

Hay dos grandes clasificaciones de los monitores: los monocromáticos que presentan la información en gama de grises y lo policromáticos o monitores a color que pueden utilizar desde 16 colores hasta colores reales. Los monocromáticos son cada vez menos usados, sin embargo aun quedan muchos de este tipo en el mercado.

En los monitores de color existen dos tipos, los VGA y los SVGA (superVGA). Estas características determinan la cantidad de colores que pueden reproducir y la resolución o nitidez.

Toda pantalla está formada por puntos de luz llamados pixeles que se iluminan para dar forma a las imágenes y a los caracteres. Cuantos más pixeles tenga una pantalla mejor es su resolución, por eso se habla de pantallas de 640 x 480, de 600x800 y de 1280 x 1024, siendo las últimas las de mayor nitidez.

La distancia existente entre los puntos se conoce como dot pitch (tamaño del punto) y es inversamente proporcional a la resolución de la pantalla, entre menor sea la distancia entre puntos, mejor es la imagen. En el mercado se escucha ofertas de equipos con pantalla superVGA punto 28, esto significa que la pantalla es de tipo SPVGA y que la distancia entre puntos es de 0.28 mm.

2.4.2 Impresora

Fija sobre el papel la información que se tiene en pantalla, en archivo o el resultado de un proceso. La impresión puede ser en negro o en colores según el tipo de impresora que se tenga.

Hay tres grupos de impresoras: las de matriz de puntos, de burbuja y laser. Las primeras son las más antiguas, son ruidosas y lentas, pero muy resistentes y económicas. Se llaman de matriz de puntos porque forman los caracteres mediante puntos marcados por los pines del cabezote. Hasta hace poco eran muy económicas, pero en la actualidad, algunas series, son mucho más costosas que las impresoras de otros tipos.

Las impresoras de burbuja, también se llaman de inyección de tinta, estas son silenciosas e imprimen hasta cinco páginas por minuto, la calidad de impresión es muy buena, el costo de la impresora es moderado, sin embargo el costo de la impresión es alto. No son recomendables para trabajo pesado.
Las impresoras láser trabajan como una fotocopiadora y producen imágenes de óptima calidad, tienen un bajo nivel de ruido y son las más rápidas, las impresoras son costosas pero la impresión es económica. Son recomendables para trabajos gráficos profesionales.

2.5 ASPECTOS DEL SOFTWARE

2.5.1 ¿Qué es el software?

El software es un conjunto de instrucciones que le indican a la computadora lo que tiene que hacer. Los paquetes de software son programas que ya han sido probados y modificados para cubrir las necesidades del usuario, hay dos tipos de software de paquetes: el software de sistemas y el software de aplicaciones.

El ***software de sistema*** es lo que se necesita para hacer que la computadora funcione. En este caso es el sistema operativo, conjunto de programas cuyo objetivo es facilitar el uso del computador y conseguir que los recursos se usen de manera eficiente, además administra y asigna los recursos del sistema. El ***software de aplicaciones*** son los programas que controlan y optimización la operación de la máquina, establecen una relación básica y fundamental entre el usuario y el computador, **hacen que el usuario pueda usar en forma cómoda y amigable complejos sistemas hardware**, actúan como intermediario entre el usuario y el hardware.

El software es el nexo de unión entre el hardware y el hombre[24]**.** El computador, por sí solo, no puede comunicarse con el hombre y viceversa, ya que lo separa la barrera del lenguaje. El software trata de acortar esa barrera, estableciendo procedimientos de comunicación entre el hombre y la máquina; es decir, **el software obra como un intermediario entre el hardware y el hombre**.

Si bien el software, es un conjunto de programas, entonces: **¿Qué es un programa?** Un programa es una serie de instrucciones que pueden ser interpretadas por un computador, obteniendo un resultado predeterminado por el ser humano.

2.5.2 Lenguajes y tipos de lenguajes de programación

- **Definición de Lenguaje**

Lenguaje es el empleo de notaciones, señales y vocales (voz, palabras) para expresar ideas, comunicarse, y establecer relaciones entre los seres humanos. Un lenguaje no sólo consta de “palabras”, sino también de su pronunciación y los métodos para combinar las palabras en frases y oraciones; los lenguajes se forman mediante combinaciones de palabras definidas en un diccionario terminológico previamente establecido. Las combinaciones posibles deben respetar un conjunto de reglas sintácticas establecidas, a ello se le conoce con el nombre de ***Sintaxis***. Además, las palabras deben tener determinado sentido, deben ser comprendidas por un grupo humano en un contexto dado, a ello se le denomina ***Semántica***.

Aunque existen muchas clasificaciones, en general se puede distinguir entre dos clases de lenguajes: los **lenguajes naturales** (inglés, alemán, español, etc.) y los **lenguajes artificiales** o formales matemático, lógico, computacional, etc.).Tanto el lenguaje natural como el lenguaje artificial son humanos. El primero es natural porque se aprende (o adquiere) inconsciente e involuntariamente. Ningún bebé decide aprender o no la lengua que hablan sus padres, y ningún padre sienta a su hijo y le enseña las reglas sintácticas de su lengua. Las personas hablan y se entienden, pero generalmente no se cuestionan las reglas que utilizan al hablar. Por otra parte, los lenguajes artificiales sí se aprenden de manera voluntaria y conscientemente. Un ejemplo de lenguaje artificial son los **lenguajes de programación** utilizados para desarrollar programas informáticos.

[24] http://www.bloginformatico.com/concepto-y-tipos-de-software.php

❖ Lenguaje de programación

Un **Lenguaje de Programación** es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones, el cual es utilizado para controlar el comportamiento físico y lógico de una maquina. Un aspecto importante son los ***programas***, que es un conjunto de órdenes o instrucciones que resuelven un problema específico basado en un Lenguaje de Programación. Existen varias clasificaciones para los lenguajes de programación.

Los lenguajes de programación se pueden clasificar atendiendo a varios criterios, los principales son:

- Según el nivel de abstracción: lenguajes de bajo nivel y ensambladores (cercanos a la máquina), lenguajes de alto nivel (cercanos al hombre).
- Según la forma de traducción - ejecución: compiladores, intérpretes.
- Según su campo de aplicación. Aplicaciones científicas, aplicaciones de tratamientos de textos, aplicaciones de inteligencia artificial, aplicaciones de programación de sistemas.
- Según el estilo de programación: imperativos, declarativos,

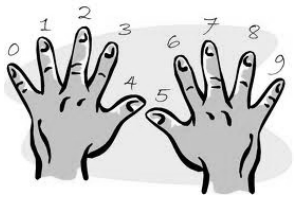
2.6 AUTOEVALUACIÓN

1. ¿A qué se debe el acelerado desarrollo de los computadores y el uso generalizado?
2. ¿Cuál considera es la característica más importante del computador?
3. ¿De qué manera el computador puede ayudarle en sus actividades diarias?
4. Si en este momento tuviera que adquirir una impresora, ¿de qué tipo la compraría? ¿Por qué?
5. De las categorías del software estudiadas ¿cuál le parece más importante?
6. ¿Qué sistemas operativos conoce? ¿Cuál le parece mejor?
7. Enumere los tipos de software de aplicación que existen y en qué consisten cada uno de ellos.

Capítulo 3

SISTEMAS DE NUMERACIÓN

3.1 INTRODUCCION



Los *sistemas de numeración* son las distintas formas de representar la información numérica. Se nombran haciendo referencia a la base, que representa el número de dígitos diferentes para representar todos los números.

Como concepto podemos definir el siguiente "*conjunto de símbolos y reglas que permiten construir todos los números validos en el sistema".*

El sistema habitual de numeración para las personas es el Decimal, cuya base es diez y corresponde a los distintos dedos de la mano, mientras que el método habitualmente utilizado por los sistemas electrónicos digitales es el Binario, que utiliza únicamente dos cifras para representar la información: el ***0*** y el ***1***.

Otros sistemas como el Octal (base 8) y el Hexadecimal (base 16) son utilizados en las computadoras.

Un *sistema de numeración* puede representarse como N = S + R donde:

- **N** → Sistema de numeración considerado
- **S** → Símbolos permitidos en el sistema
- **R** → Son la reglas de generación que nos indican que números son validos y cuales son no – validos en el sistema.

	Binario	Octal	Decimal	**Hexadecimal**
Base	2	8	10	**16**
Dígitos o símbolos permitidos en el sistema	0,1	0,1,2,3,4,5,6,7	0,1,2,3,4,5,6,7,8,9	**0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F**
Notación	**X_b ó X_2**	**X_o ó X_8**	**X_d ó X_{10}**	**X_h ó X_{16}**

3.2 SISTEMA BINARIO

Características:

- Su base es 2
- Emplea dos caracteres 0 y 1 (estos reciben el nombre de bits)
- Cada cifra se denomina bit

El sistema binario utiliza sólo dos signos:

0 1

Estos son mucho más fáciles de representar en el interior de una computadora, donde estas dos cifras se pueden asociar perfectamente a los dos posibles estados que pueden adoptar los circuitos o componentes electrónicos: apagado y encendido.

La *presencia de una corriente eléctrica* = 1 (encendido)

La *ausencia* = 0 (apagado). [25]

A estas dos cifras se le conocen como *bits.*

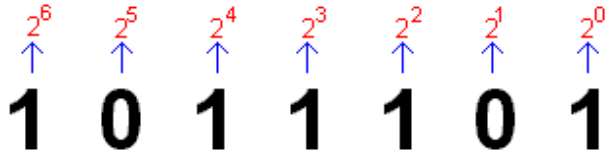
Bit cero = 0

Bit uno = 1

Se denomina "Binario" a todo aquello que tiene dos partes, dos aspectos, etc. Muchas cosas en los sistemas digitales son binarias: Los impulsos eléctricos que circulan en los circuitos son de baja o de alta tensión, los interruptores biestables están encendidos o apagados, abiertos o cerrados, etc.

Tanto las palabras, números y dibujos se traducen en el ordenador en secuencias de 1 y 0. De hecho toda letra, cifra o símbolo gráfico es codificado en una secuencia de 0 y 1. Si, por ejemplo, nuestro nombre tiene cinco letras, la representación para el ordenador constara de cinco bytes. La palabra bit deriva de las dos palabras inglesas "**bi**nary digi**t**" cifra binaria, y designa a las dos cifras 0 y 1, que se utilizan en el sistema binario. Un bit es también, la porción más pequeña de información representable mediante un número, e indica si una cosa es verdadera o falsa, alta o baja, negra o blanca, etc.

Un **byte** es generalmente una secuencia de 8 bits. Ocho ceros y unos se pueden ordenar de 256 maneras diferentes ya que cada bit tiene un valor de posición diferente, donde el bit numero 1 le corresponderá un valor de posición de 2^0(1), el siguiente bit tendrá un valor de 2^1(2), el siguiente 2^2(4), el siguiente 2^3(8), el siguiente 2^4(16), el siguiente un valor de 2^5(32), y así sucesivamente hasta llegar la última posición, o ultimo bit, en este caso el numero 8, que también es llamado el MSB (Bit Más Significativo) y el LSB (Bit Menos Significativo) correspondiente a la primera posición o bit numero 1. Ejemplo:



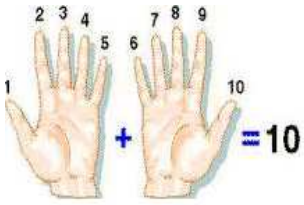
[25] http://herramientasofimaticas-flaquis9216.blogspot.com/2011/06/numeros-binarios.html

MSB LSB

0 1 1 0 1 0 1 1

Los circuitos digitales internos que componen las computadoras utilizan el sistema de numeración binario para la interpretación de la información y codificación de la misma.

3.3 SISTEMA DECIMAL



El sistema decimal de numeración que usamos en la vida diaria es de difícil empleo en las computadoras, ya que para representar los números y trabajar con ellos son necesarios diez símbolos:

0 1 2 3 4 5 6 7 8 9

Cuando en una numeración se usan diez símbolos diversos, a ésta se la denomina numeración decimal o base 10. El valor de cada cifra es el producto de la misma por una potencia a 10 (la base), cuyo exponente es igual a la posición 0, las decenas la 1 y así sucesivamente.

El sistema decimal está formado por diez símbolos, llamados números arábigos. También es llamado sistema de base 10. Usando los diez símbolos separadamente 0, 1, 2, 3, ..., 9 nos permite representar el valor de los números en unidades individuales, pero para representar más de nueve números es necesario combinarlos. Cuando usamos símbolos en combinación, el valor de cada uno de ellos depende de su posición con respecto al punto decimal, designando así un símbolo para las unidades, otro para las decenas, otro para las centenas, otro para los millares (de miles, no de millón), en adelante.

El símbolo correspondiente a las unidades asume la posición más izquierda antes del punto decimal. Esta designación de posición determina que la potencia del número se corresponde con la distancia en que está del punto decimal, y es por ello que la primera posición se llama UNIDAD ($10^0 = 1$). Matemáticamente esto puede ser representado como:

unidad = 10^0 decena = 10^1 centena = 10^2

Ejemplo: El valor en combinación de los símbolos 234 es determinado por la suma de los valores correspondientes a cada posición:

2×10^2 + 3×10^1 + 4×10^0

Que equivale a:

2×100 + 3×10 + 4×1

Efectuando las multiplicaciones esto da:

200 + 30 + 4

Cuya suma da como resultado: 234

La posición derecha del punto decimal es representada por número enteros pero negativos comenzando desde -1 para la primera posición. Matemáticamente las tres primeras posiciones a la derecha del punto decimal se expresan como:
décimas 10^{-1} centésimas 10^{-2} milésimas 10^{-3}

Ejemplo: el valor 18.947 equivale a:

= $1x10^1 + 8x10^0 + 9x10^{-1} + 4x10^{-2} + 7x10^{-3}$
= $1x10 + 8x1 + 9x0.1 + 4x0.01 + 7x0.001$
= $10 + 8 + 0.9 + 0.04 + 0.007$

Para representar un número base diez es posible colocar su valor seguido de la base en sub-índice (18.97410) o bien seguido de la letra d entre paréntesis: 645(d).

3.4 SISTEMA OCTAL

Características:

- Su base es 8.
- Consta de 8 símbolos desde el 0 al 7.
- Muy poco utilizado en las computadoras.
- Se utiliza por conveniencia con los operadores del sistema.

La facilidad con que se pueden convertir entre el sistema octal y el binario hace que el sistema octal sea atractivo como un medio "taquigráfico" de expresión de números binarios grandes. Cuando trabajamos con una gran cantidad de números binarios de muchos bits, es más adecuado y eficaz escribirlos en octal y no en binarios. Sin embargo, recordemos los circuitos y sistemas digitales trabajan eléctricamente en binario, usamos el sistema octal solo por conveniencia con los operadores del sistema.

3.5 SISTEMA HEXADECIMAL

Características:

- Su base es 16.
- Consta de 16 donde desde el 0 al 9 son números y del 10 al 15 son letras.

Los símbolos de este sistema se encuentran distribuidos en la siguiente forma:

Hexadecimal	Decimal
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

3.6 TEOREMA FUNDAMENTAL DE NUMERACION

El valor en el sistema decimal de una cantidad expresada en otro sistema cualquiera de numeración viene dado por la fórmula:

$$X_3 * B^3 + X_2 * B^2 + X_1 * B^1 + X_0 * B^0 + X_{-1} * B^{-1} + X_{-2} * B^{-2} + X_{-3} * B^{-3}$$

Donde:

X → es un digito

B → base del sistema de numeración (SN)

Ejemplo:

10011_b = $1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$

= $1*16 + 0*8 + 0*4 + 1*2 + 1*1$

= $16 + 2 + 1$

= 19_d (dígitos)

$A5E1_h$ = $A * 16^3 + 5 * 16^2 + E * 16^1 + 1 * 16^0$

= $10 * 16^3 + 5 * 16^2 + 14 * 16^1 + 1 * 16^0$

= $40960 + 1280 + 224 + 1$

= 42465_d (dígitos)

1741_o = $1 * 8^3 + 7 * 8^2 + 4 * 8^1 + 1 * 8^0$

= $1 * 512 + 7 * 64 + 4 * 8 + 1 * 1$

= $512 + 448 + 32 + 1$

= 993_d (dígitos)

3.7 CONVERSIÓN ENTRE SISTEMAS

3.7.1 binario a decimal:

Regla: Se toma la cantidad binaria y se suman las potencias de dos (2) correspondientes a los pesos de las posiciones de todos sus dígitos en los que hay un 1.

Ejemplo:

111_b = $1 * 2^2 + 1 * 2^1 + 1 * 2^0$

= 1*4 1*2 + 1*1

= 4 + 2 +1

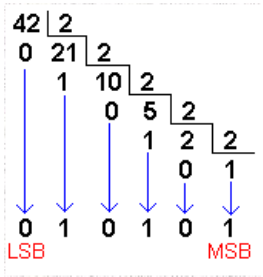
= 7_d (dígitos)

3.7.2 decimal a binario:

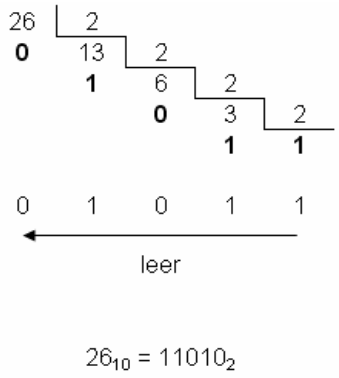
Regla (a): Se toma la cantidad dada y se divide sucesivamente entre 2. Los restos obtenidos en cada división (0,1) forman la cantidad binaria pedida, leído desde el último cociente al primer resto.

Ejemplo:

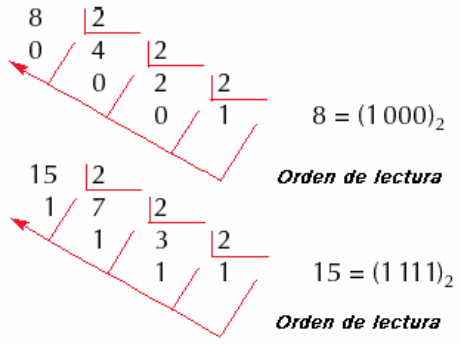
1.



2.



3.



4.

Existe otra manera de hacerlo y es dividir el cociente 1 entre 2, escribimos 0 como cociente, posteriormente multiplicamos 2 por 0 (que es cero) y ese resultado se lo restamos al último residuo que teníamos (que será 1) y tendremos como residuo 1. De esta forma comenzaremos la cuenta para obtener el valor binario desde el último residuo obtenido (que es siempre 1, excepto en el caso del número 0) hasta el primero. Podemos utilizar cualquiera de los dos métodos y ambos son correctos y presentan el último resultado, tal como veremos en los ejemplos a continuación.

3.7.3 octal a binario

Cada cifra se sustituirá por su equivalente binario. Por lo que tendremos en cuenta la siguiente tabla:

Carácter octal	Nº binario
0	000
1	001
2	010

3	011
4	100
5	101
6	110
7	111

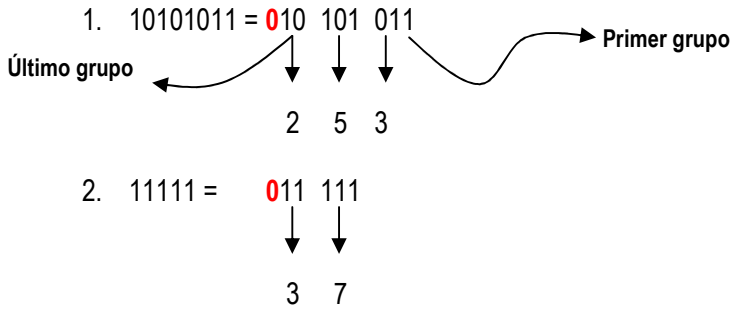
Ejemplo:

1. $374_{ob} = 11111100_{b}$

2. $626_{ob} = 110010110_{b}$

3.7.4 binario a octal

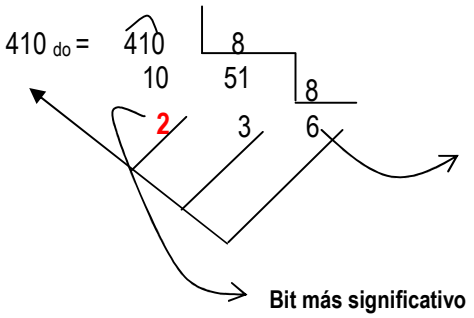
Se realiza de modo contrario a la anterior conversión. Los bits enteros o fraccionarios son agrupados en grupos de 3 comenzando por la derecha, luego cada grupo se convierte a su equivalente octal. Si no se consiguen todos los grupos de tres (3) se añadirán los ceros que sean necesarios al último grupo.



3.7.5 decimal a octal

Se realiza de modo similar a la conversión de decimal a binario (se divide entre **2** sucesivamente) pero ahora dividiendo entre **8**.

Ejemplo:

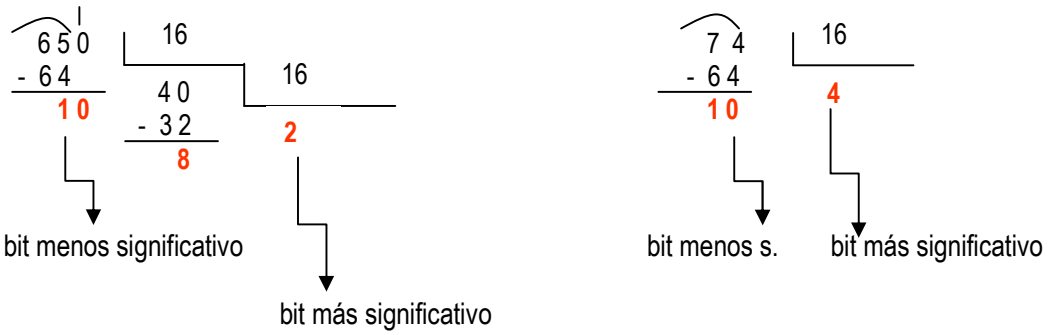


$410_{d} = 632_{o}$

3.7.6 decimal a hexadecimal

Se realiza de modo similar a la conversión decimal a binario (se divide entre ….) pero dividiendo entre 16 (base Hexadecimal).

Ejemplo 1: 650 de decimal a hexadecimal



650 decimal es equivalente a 28A hexadecimal

3.7.7 *binario a hexadecimal*

Los valores binarios y decimales correspondientes a las cifras hexadecimales se muestran en la siguiente tabla.

HEXADECIMAL	BINARIO
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Ejemplo:

1. **00** 10 1001 = 29 $_{bh}$

   2 9

   Se añaden dos ceros al último grupo para completar el grupo de 4 bits.

2. 1110 0001 1001 = E19 $_{bh}$

   E 1 9

3.8 CLASE PRÁCTICA

1. Convertir a decimal los siguientes números binarios:
   a) 1101101
   b) 101001
   c) 0101,11
   d) 101110

2. Dado los siguientes números decimales convertirlos a binarios:
   a) 150
   b) 54
   c) 228
   d) 1500

3. Convertir al sistema decimal las siguientes expresiones utilizando el teorema fundamental de numeración:
   a) 1000011_{b}
   b) $176A_{h}$
   c) 562_{o}
   d) $1AB_{h}$

4. Dada las siguientes expresiones identifique el error (si este existe) y justifique que símbolos son permitidos en el sistema correspondiente:
   a) 1011_{b}
   b) 108_{o}
   c) $012FG_{h}$
   d) 120_{h}
   e) 1015_{d}
   f) $10011F_{b}$

3.9 OPERACIONES ARITMETICAS CON NUMEROS BINARIOS

Las operaciones básicas con números binarios son:

- suma
- resta
- multiplicación
- división

Estas operaciones se realizan similar que en el sistema decimal

Vamos a tener en cuenta la tabla de Operaciones Aritméticas básicas con números binarios:

		SUMA	RESTA	MULTIPLICACION	DIVISION
A	**B**	**A + B**	**A - B**	**A * B**	**A / B**
0	0	0	0	0	Indeterminado
1	0	1	1	0	∞
0	1	1	1 y adeudo 1	0	0
1	1	0 y acarreo 1	0	1	1

3.9.1 Suma binaria:

Semejante a la suma en el sistema decimal con la diferencia que se manejan solo dos dígitos (0 y 1) y cuando el resultado excede de los símbolos utilizados se agrega el exceso (*acarreo*) a la suma parcial siguiente hacia la izquierda.

Ejemplo:

1. sumar 101101 + 10101 (es decir $45_d + 21_d = 66_d$)

```
  1 0 1 1 0 1
+   1 0 1 0 1
-------------
  1 0 0 0 0 1 0
```

Comprobarlo con el sistema decimal

2. sumar 11101 + 111 (es decir $29_d + 7_d = 36_d$)

```
  1 1 1 0 1
+     1 1 1
-----------
  1 0 0 1 0 0
```

Comprobarlo con el sistema decimal

3.9.2 *Resta binaria:*

La resta binaria es similar a la resta en el sistema decimal con la diferencia que solo se utilizan dos dígitos (0,1). Si el sustraendo es mayor que el minuendo se sustrae o presta una unidad del digito mas a la izquierda en el minuendo (si existe y vale **1**) convirtiéndose este último en cero.

Ejemplo:

1. restar 11101 - 111 (es decir 29_d - 21_d = 22_d)

```
  1 1 1 0 1
-   1 1 1
-----------
  1 0 1 1 0
```

2. restar 101101 - 10101 (es decir 45_d - 21_d = 24_d)

```
  1 0 1 1 0 1
-   1 0 1 0 1
-------------
    0 1 1 0 0 0
```

3. restar 10000 - 01111 (es decir 16_d - 15_d = 1_d)

```
  1 0 0 0 0
- 0 1 1 1 1
-----------
  0 0 0 0 1
```

4. restar 1101010 - 1010111 (es decir 45_d - 21_d = 24_d)

```
  1 1 0 1 0 1 0
- 1 0 1 0 1 1 1
---------------
  0 0 1 0 0 1 1
```

5. restar 100 - 1 (es decir 3_d - 2_d = 1_d)

```
  1 0 0
-   1
-------
  0 1 1
```

6. restar 1010 - 111 (es decir 10_d - 7_d = 3_d)

```
  1 0 1 0
-   1 1 1
-----------
  0 0 1 1
```

3.9.3 *Multiplicación binaria:*

Se realiza de la misma manera que la multiplicación con números decimales a diferencia que se manejan solo dos dígitos desplazándose cada producto parcial una posición a la izquierda y luego se suman dichos productos.

Ejemplo:

1. multiplicar 1100 * 1011 (es decir $12_d * 11_d = 132_d$)

```
   1 1 0 0
 * 1 0 1 1
 ---------
   1 1 0 0
 1 1 0 0
 0 0 0 0
+1 1 0 0
 ---------
 1 0 0 0 0 1 0 0
```

2. multiplicar 1011 * 101 (es decir $11_d * 5_d = 55_d$)

```
   1 0 1 1
 x   1 0 1
 ---------
   1 0 1 1
   0 0 0 0
+1 0 1 1
 ---------
 1 1 0 1 1 1
```

3. multiplicar 1111 * 11 (es decir $15_d * 3_d = 45_d$)

```
   1 1 1 1
 x     1 1
 ---------
   1 1 1 1
+1 1 1 1
 ---------
 1 0 1 1 0 1
```

3.9.4 División binaria:

Similar a la división con números decimales. Si el número tomado del dividendo es mayor o igual que el divisor se pone **1** en el cociente y el divisor se resta al número considerado del dividendo. Si el dividendo es menor que el divisor o cero se pone **0** en el cociente y se resta cero al número considerado del dividendo.

Ejemplo:

1. dividir 110 / 11 (es decir $6_d / 3_d = 2_d$)

```
 11 0 | 11
 -----
  0   10
```

2. dividir 101010 / 110 (es decir 42_d / 6_d = 7_d)

```
 101010 | 110
-  110    111
---------
  1 0 0 1
 - 1 1 0
-----------
     1 1 0
   - 1 1 0
-----------
         0
```

3. dividir 11000 / 10 (es decir 24_d / 2_d = 12_d)

```
 11000 | 10
- 10     1100
---------
  10
- 10
-----------
   000
```

3.10 CLASE PRÁCTICA

1. Convierta a binario los siguientes números decimales:

a) 215	b) 63	c) 200	d) 33	e) 69
f) 77	g) 642	h) 111	i) 141	j) 88
k) 47	l) 514	m) 217	n) 365	ñ) 300

2. Convierta a decimal los siguientes números binarios:

a) $(10011)_2$	b) $(101100)_2$	c) $(11110111)_2$	d) $(1000000)_2$	e) $(11001100011)_2$
f) $(111011110)_2$	g) $(10001)_2$	h) $(1001000)_2$	i) $(110010011)_2$	j) $(11111111)_2$
k) $(10101010)_2$	l) $(1110111011)_2$	m) $(10000)_2$	n) $(1111101)_2$	ñ) $(11110000)_2$

3. Convierta a octal los siguientes números binarios:

a) $(111011)_2$	b) $(1111100)_2$	c) $(10010100)_2$	d) $(1001001100)_2$	e) $(1101010010)_2$
f) $(11001)_2$	g) $(10111010011)_2$	h) $(1011011001)_2$	i) $(100001011)_2$	j) $(1010110)_2$
k) $(111011010)_2$	l) $(1011011101)_2$	m) $(1000000010)_2$	n) $(11111111111)_2$	ñ) $(10110110010)_2$

4. Convierta a hexadecimal los siguientes números binarios:

a) $(101111011101)_2$	b) $(101101100101001)_2$	c) $(100010100000100)_2$	d) $(1011100111)_2$
e) $(101000)_2$	f) $(1101011011101111)_2$	g) $(1011011011)_2$	h) $(1111011110111)_2$
i) $(10100101101)_2$	j) $(110001100101)_2$	k) $(1000000101)_2$	l) $(101101110)_2$

5. Convierta a octal los siguientes hexadecimales:

a) $(DE4)_{16}$	b) $(3A7)_{16}$	c) $(1F2E)_{16}$	d) $(9A2B8)_{16}$
e) $(7531)_{16}$	f) $(1F2E)_{16}$	g) $(DD07)_{16}$	h) $(36B9)_{16}$
i) $(F2CCE)_{16}$	j) $(642)_{16}$	k) $(C495)_{16}$	l) $(5A08)_{16}$
m) $(D8539)_{16}$	n) $(FF)_{16}$	ñ) $(EAE)_{16}$	o) $(5070)_{16}$

6. Convierta a hexadecimal los siguientes octales:

a) $(3463)_8$	b) $(1035)_8$	c) $(3257)_8$	d) $(7147)_8$
e) $(12204)_8$	f) $(71)_8$	g) $(63714)_8$	h) $(7362)_8$
i) $(47667)_8$	j) $(201037)_8$	k) $(5555)_8$	l) $(77777)_8$
m) $(3571)_8$	n) $(646202)_8$	ñ) $(43057)_8$	o) $(44)_8$

RESPUESTAS A LOS EJERCICIOS ANTERIORES

1. Números decimales convertidos a binarios

a) $(11010111)_2$	b) $(111111)_2$	c) $(11001000)_2$	d) $(100001)_2$	e) $(1000101)_2$
f) $(1001101)_2$	g) $(1010000010)_2$	h) $(1101111)_2$	i) $(10001101)_2$	j) $(1011000)_2$
k) $(101111)_2$	l) $(1000000010)_2$	m) $(11011001)_2$	n) $(101101101)_2$	ñ) $(100101100)_2$

2. Números decimales convertidos de binarios

a) 19	b) 44	c) 247	d) 64	e) 1635
f) 478	g) 17	h) 72	i) 403	j) 255
k) 170	l) 955	m) 16	n) 125	ñ) 240

3. Números octales convertidos de binarios

a) $(73)_8$	b) $(174)_8$	c) $(224)_8$	d) $(1114)_8$	e) $(1522)_8$
f) $(31)_8$	g) $(2723)_8$	h) $(1331)_8$	i) $(413)_8$	j) $(126)_8$
k) $(732)_8$	l) $(1335)_8$	m) $(1002)_8$	n) $(3777)_8$	ñ) $(2662)_8$

4. Números hexadecimales convertidos de binarios

a) $(BDD)_{16}$	b) $(5B29)_{16}$	c) $(4504)_{16}$	d) $(2E7)_{16}$
e) $(28)_{16}$	f) $(D6EF)_{16}$	g) $(2DB)_{16}$	h) $(1EF7)_{16}$
i) $(52D)_{16}$	j) $(C65)_{16}$	k) $(205)_{16}$	l) $(16E)_{16}$

5. Números octales convertidos de hexadecimales

a) $(6744)_8$	b) $(1647)_8$	c) $(17456)_8$	d) $(2321270)_8$
e) $(72461)_8$	f) $(17456)_8$	g) $(156407)_8$	h) $(33271)_8$
i) $(3626316)_8$	j) $(3102)_8$	k) $(142225)_8$	l) $(55010)_8$
m) $(3302471)_8$	n) $(377)_8$	ñ) $(7256)_8$	o) $(50160)_8$

6. Números hexadecimales convertidos de octales

a) $(733)_{16}$	b) $(21D)_{16}$	c) $(6AF)_{16}$	d) $(E67)_{16}$
e) $(1484)_{16}$	f) $(39)_{16}$	g) $(67CC)_{16}$	h) $(EF2)_{16}$
i) $(4FB7)_{16}$	j) $(1021F)_{16}$	k) $(B6D)_{16}$	l) $(7FFF)_{16}$
m) $(779)_{16}$	n) $(34C82)_{16}$	ñ) $(462F)_{16}$	o) $(24)_{16}$

Capítulo 4

ALGORITMOS

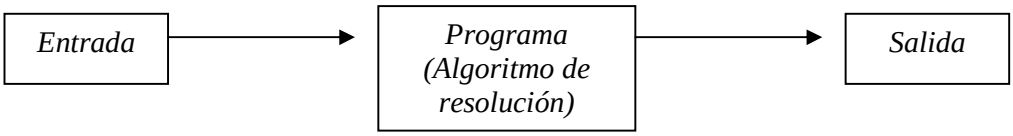
4.1 CONCEPTO DE PROGRAMA

Un programa de computadora es un conjunto de instrucciones que producirá la solución de problema. Por tanto el desarrollo de un programa requiere las siguientes fases:

1. Definición y análisis del problema.
2. Diseño de algoritmos:
   - Diagrama de flujo
   - Diagrama N-S
   - Pseudocódigo
3. Codificación del programa (*en base al diseño del algoritmo*)
4. Depuración y verificación del programa

4.2 COMPONENTES DE UN PROGRAMA / ALGORITMO

Entrada y salida de los datos y definición del algoritmo de solución son especificaciones que debe contener un programa.



Bloques de un programa

- Las entradas, procederán de un dispositivo de entrada – teclado, disco, …. – El proceso de introducir la información de entrada – datos – en la memoria de la computadora se denomina *entrada de datos,* operación de *lectura* o acción de **leer**.

- Las salidas de datos se deben presentar en dispositivos periféricos de salida: pantalla, impresora, discos, etc. La operación de salida de *datos* se conoce también como *escritura* o acción de **escribir**.

4.3 FASES PARA LA CONSTRUCCION DE UN PROGRAMA

4.3.1 Definición del problema

La identificación del problema es una fase muy importante en la metodología, pues de ella depende el desarrollo posterior en busca de la solución. Un problema bien delimitado es una gran ayuda para que el proceso general avance bien; un problema mal definido provocará desvíos conceptuales que serán difíciles de remediar posteriormente.

4.3.2 Análisis del problema

El propósito del análisis de un problema, es ayudar al programador para llegar a una cierta comprensión de la naturaleza del problema. El problema debe estar bien definido si se desea llegar a una solución satisfactoria. Para poder definir con precisión el problema se requiere que las especificaciones de entrada y salida sean descritas con detalle.

Tomar en cuenta:
¿Qué información debe proporcionar la resolución del problema?
¿Qué datos se necesitan para resolver el problema?

4.3.3 Diseño y desarrollo del algoritmo

Desarrollo del algoritmo que dará solución al problema establecido a través de herramientas básicas como son los 3 tipos de algoritmos: pseudocódigo, diagrama de flujo, diagrama N-S(Nassi Scheiderman).

4.3.4 Pruebas de escritorio

Seguimiento manual de los pasos descritos en el algoritmo. Se hace con valores reales para comprobar la buena ejecución del algoritmo y garantizar que el problema se resuelva.

4.3.5 Codificación

Aquí se selecciona un lenguaje de programación para digitar el pseudocódigo, haciendo uso de la sintaxis y estructura gramatical del lenguaje seleccionado.

4.3.6 Compilación o interpretación del programa

El software elegido convierte las instrucciones escritas en el lenguaje a las comprendidas por el computador.

4.3.7 Ejecución

El programa es ejecutado por la máquina para llegar a los resultados esperados.

4.3.8 Evaluación de los resultados

Evaluación de los resultados finales con el fin de verificar si estos son correctos.

4.4 DATOS, TIPOS DE DATOS

El objetivo fundamental de una computadora es el manejo de la información (datos). Un *dato* que es la representación simbólica (numérica, alfabética, algorítmica etc.) de un objeto. Existen dos tipos de clases de datos: *simples* (sin estructura) y *compuestos (estructurados).*

- Tipos de datos simples
  - numéricos (*integer, real*)
  - lógicos (boolean / true - false)
  - carácter (char, string)

4.4.1 Datos numéricos

Es el conjunto de los valores numéricos. Estos se representan de dos formas:
Enteros: es un subconjunto finito de los números enteros. Los enteros son números completos y no tienen parte fraccionaria, pueden ser positivos o negativos.

Ejemplo:

```
 8        7
-5        0
```

Reales: es un subconjunto de los números reales. Los números reales siempre tienen un punto decimal.

Ejemplo:

```
 0.08     723.1
-5.12     0.10
```

4.4.2 Datos lógicos (booleanos)

El tipo lógico se denomina *booleano* y es el que sólo puede tomar dos valores: ***cierto o verdadero*** *(true)* y ***falso*** *(false).*

4.4.3 Datos tipo carácter y tipo cadena

El tipo *carácter* es el conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato de tipo carácter contiene un solo caracter.

La mayoría de las computadoras reconocen los siguientes caracteres alfabéticos y numéricos.

- Caracteres alfabéticos (A,B,C,D, . . .,Z) (a,b,c,d, . . .,z)
- Caracteres numéricos (1,2,3, . . . 9,0)
- Caracteres especiales (+,-,*,/,^, . , ; , > ,<, $)

Una cadena de *caracteres* es una sucesión de caracteres delimitados por una comilla (apóstrofo) o dobles comillas, según el tipo de lenguaje de programación.

Ejemplo:

"Introducción a la ingeniería en computación"
"Bienvenidos alumnos . . . "

4.5 OPERADORES

Operador	*Significado*	*Ejemplo*
Relacionales		
>	Mayor que	3>2
<	Menor que	'ABC'<'abc'
=	Igual que	4=3
<=	Menor o igual que	'a'<='b'
>=	Mayor o igual que	4>=5
Logicos		
& ó Y	Conjunción (y).	(7>4) & (2=1) //falso
| ó O	Disyunción (o).	(1=1 | 2=1) //verdadero
~ ó NO	Negación (no).	~(2<5) //falso
Algebraicos		
+	Suma	total <- cant1 + cant2
-	Resta	stock <- disp - venta
*	Multiplicación	area <- base * altura
/	División	porc <- 100 * parte / total
^	Potenciación	sup <- 3.41 * radio ^ 2
% ó MOD	Módulo (resto de la división entera)	resto <- num MOD div

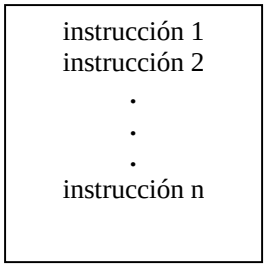
4.6 INSTRUCCIONES Y TIPOS DE INSTRUCCIONES

El proceso de diseño del algoritmo o posteriormente de codificación del programa consiste en definir las acciones o instrucciones que resolverán el problema.

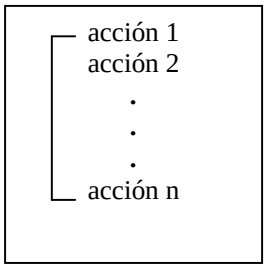
Las *acciones* o *instrucciones* se deben escribir y posteriormente almacenar en memoria en el mismo orden en que han de ejecutarse, es decir, *en secuencia*.

Un programa puede ser lineal o no lineal

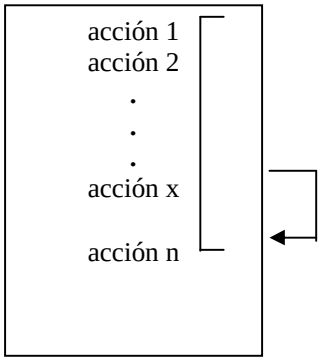
Es *lineal* cuando las instrucciones se ejecutan secuencialmente, sin bifurcaciones, decisión ni comparación:



En el caso del algoritmo las instrucciones se suelen conocer como *acciones,* se tendría:



Es *no lineal* cuando se interrumpe la secuencia mediante instrucciones de bifurcación:



Las instrucciones - acciones – que se pueden implementar de modo general en un algoritmo y que esencialmente soportan todos los lenguajes son las siguientes:

1. Instrucciones de inicio / fin
2. Instrucciones de asignación
3. Instrucciones de lectura
4. Instrucciones de escritura
5. Instrucciones de bifurcación

Instrucciones y tipos de instrucciones

Tipo de instrucción	Pseudocódigo ingles	Pseudocódigo español
comienzo de proceso	**begin**	**inicio**
fin de proceso	**end**	**fin**
entrada (lectura)	**read**	**leer**
salida (escritura)	**write**	**escribir**
asignación	**A ←3**	**A ←3**

4.6.1 Instrucciones de Asignación:

La instrucción de asignación es el modo de darle valores a una variable. La operación de asignación se representa con el símbolo u operador ← . La operación se conoce como *instrucción* o *sentencia* de asignación cuando se refiere a un lenguaje de programación.

Formato general:

nombre de la variable ← expresión

Ejemplo:

A ← 5 ---------- Significa que a la variable A se la ha asignado el valor de 5

Nota: La acción de asignación es *destructiva* ya que el valor que tuviera la variable antes de la asignación se pierde y se reemplaza por el nuevo valor. Así en la secuencia de operaciones

A ← 25
A ← 134
A ← 5

Cuando estas se ejecutan, el valor ultimo que tomará A será 5 (los valores 25 y134 han desaparecido).

DEFINA:

a) ¿Cuál será el valor de la variable AUX al ejecutarse la instrucción 5?
   1. A ← 10
   2. B ← 20
   3. AUX ← A
   4. A ← B
   5. B ← AUX

b) ¿Cuál es el significado de N ← N + 5 si N tiene el valor actual de 2?

N ← N + 5

4.6.2 Instrucción de lectura y escritura (entrada / salida de datos):

Las instrucciones de entrada permiten leer determinados valores y asignarlos a determinadas variables. Esta entrada se conoce como operación de **lectura**.

La operación de salida se denomina **escritura.**
En la escritura de algoritmos las acciones de lectura y escritura se representan por los formatos siguientes:

Leer (*lista de variables de entrada*) **Escribir** (*lista de expresiones de salida*)

Ejemplo:

Leer (A, B, C)
Representa la lectura de tres valores de entrada que se asignan a las variables A, B y C.

Escribir (***"Bienvenido a la Programación Estructurada"***)
Visualiza en la pantalla – o escribe en el dispositivo de salida – el mensaje "Bienvenido a la programación Estructurada".

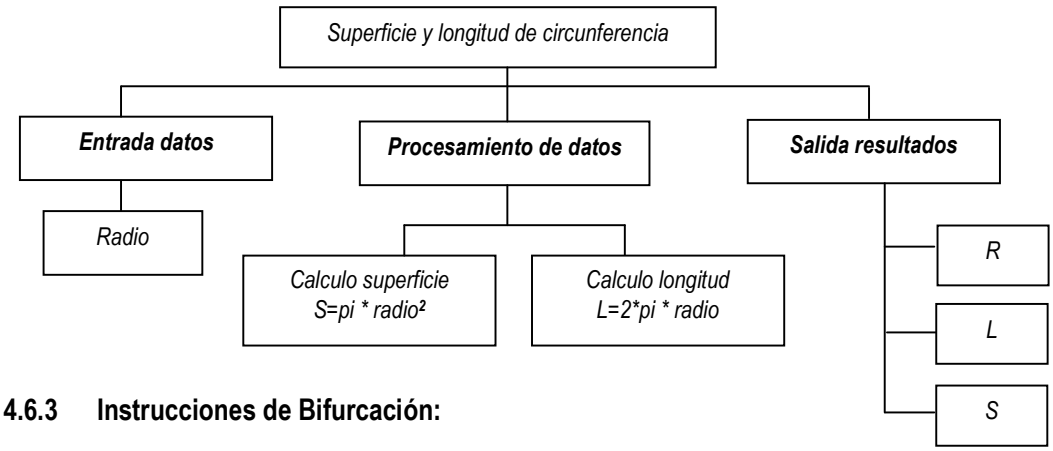
Ejemplo:

Leer el radio de un círculo y calcular e imprimir su superficie y la longitud de la circunferencia.

Las entradas de datos en este problema se concentran en el radio del círculo. Dado que el radio puede tomar cualquier valor dentro del rango de los números reales, el tipo de datos radio debe ser real.

Las salidas serán dos variables: superficie y circunferencia, que también serán de tipo real.

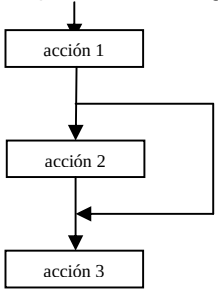
Entradas: radio del circulo(variable RADIO).
Salidas: superficie del circulo(variable Area).
Circunferencia del circulo(variable Circunferencia).
Variables: Radio, Área y circunferencia (tipo real).



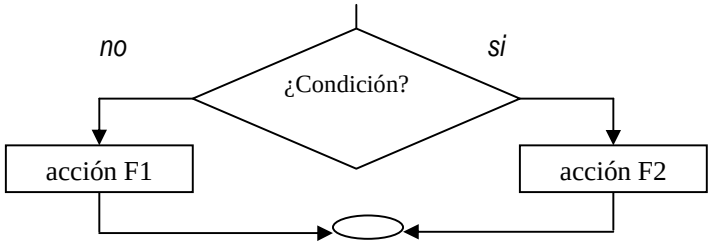
4.6.3 Instrucciones de Bifurcación:

El desarrollo lineal de un programa se interrumpe cuando se ejecuta una bifurcación. Las bifurcaciones en el flujo de un programa pueden realizarse de un modo incondicional o condicional.

- *Bifurcación incondicional:* la bifurcación se realiza siempre que el flujo del programa pase por la instrucción sin necesidad del cumplimiento de ninguna condición.



- *Bifurcación condicional:* la bifurcación depende del cumplimiento de una determinada condición. Si se cumple la condición, el flujo sigue ejecutando la acción F2. Si no se cumple, se ejecuta la acción F1.



4.7 HERRAMIENTAS DE PROGRAMACION

4.7.1 Algoritmo

La palabra algoritmo se deriva de la traducción al latín de la palabra árabe ***alkhowarizmi***, nombre de un matemático y astrónomo árabe que escribió un tratado sobre la manipulación de números y ecuaciones del siglo IX.

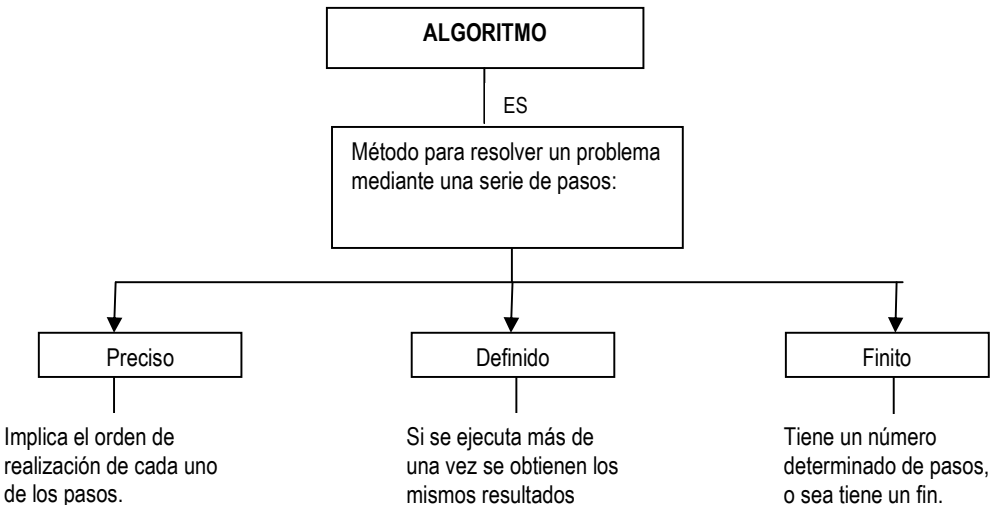
Definición 1: un algoritmo es una serie de pasos organizados que describe el proceso que se desea seguir, para dar solución a un problema especifico.

Definición 2: un algoritmo se puede definir como una secuencia finita de instrucciones cada una de las cuales tiene significado claro y puede ser efectuada con una cantidad finita de esfuerzo en una longitud de tiempo también finito.

4.7.2 Características de los algoritmos

Las características más relevantes de los algoritmos son:

- Preciso: Implica el orden de realización de cada uno de los pasos.
- Definido: Si se ejecuta más de una vez se obtienen los mismos resultados
- Finito: Tiene un número determinado de pasos, o sea tiene un fin.



Existen tres estructuras de control básicas ó primitivas y combinándolas se puede escribir cualquier algoritmo. Estas estructuras primitivas son: ***secuencial***, ***condicional o bifurcación*** y ***bucle o ciclo***.

4.7.3 Tipos de algoritmos

Pseudocódigo:

Es una herramienta utilizada para el diseño de programas. Permite al programador expresar su pensamiento de una forma clara, utilizado su lenguaje natural y mostrando el orden de ejecución de la sentencia del programa sin ninguna ambigüedad.

El pseudocódigo no está diseñado como lenguaje compatible. Tiene pocas reglas sintácticas y ofrece al usuario la libertad de expresar sus pensamientos en su lenguaje natural en lugar de hacerlo en un leguaje particular de programación. Pasar del programa escrito en pseudocódigo al programa escrito en un lenguaje cualquiera de programación resulta sencillo.

Reglas para escribir Pseudocódigo

1. Utilizar palabras como DO, WHILE, END-DO, IF, ELSE, END-IF (o sus correspondientes en español).
2. Seguir reglas de sangrado para mostrar la dependencia dentro de cada uno de los segmentos que componen el diseño del programa.
3. Desglosar en segmento la solución diseñada, cada uno de los cuales resuelve una función lo más clara y concreta posible, tal como aconseja la programación estructurada.

Dentro de cada estructura se encuentran presente las 3 estructuras básicas:

Pseudocódigo Secuencial

En este caso las acciones se deben realizar en *secuencia* se escribe una a continuación de otras.

Ejemplo: Calcular el importe de un determinado artículo, teniendo como entrada de datos la cantidad y el precio del mismo.

```
{Calculo de importe (costo a pagar)}
Inicio
   Var: cantidad, precio
   Leer cantidad, precio
   importe ← cantidad * precio
   Imprimir importe
Fin
```

Pseudocódigo Condicional

Si utilizamos un condicional, es para indicar que según el resultado *cierto o falso* de una expresión hay que realizar una acción u otra de las dos especificada.

Ejemplo:

```
if expresión then
  acción 1
else
  acción 2
end if
```

Ejemplo:

Calcule el importe de la venta de un determinado artículo, sabiendo que al comprar entre 1 y 15 unidades del mismo artículo el descuento es de un 10% y si la compra es superior a 15 unidades el descuento es de un 15%.

```
{Calcular importe}
Inicio
Var: cantidad, precio
Leer  cantidad, precio
importe← cantidad * precio
        si (cantidad >= 1) y (cantidad <= 15)
                importe ← importe – (importe * 0.10)
        sino
                importe ← importe – (importe * 0.15)
Imprimir precio, cantidad, importe
Fin
```

Pseudocódigo Repetitivo

Se utiliza cuando se desea que un conjunto de instrucciones se realicen (o se repitan) un número finito (que tenga un fin) de veces. En este caso existen 3 tipos de ciclos o bucles: mientras, para/hasta, hacer-mientras.

Ejemplo: en este ejemplo utilizaremos el uso del mientras.

```
Mientras (condición)  {iniciación de la condición}
<Sentencias repetitivas>
Progresión de la condición
Fin - Mientras
```

El funcionamiento de la condición anterior es la siguiente:

1. Se evalúa la condición.
2. Si el resultado es cierto (o verdadero), se ejecuta el bloque de sentencia repetitiva. Si el resultado es falso no se ejecuta y se continúa en la sentencia siguiente a **Fin - Mientras**.

Ejemplo:

Diseñe un algoritmo que calcule el valor medio de la altura de los alumnos de un curso determinado. Finalizar la entrada de datos con un valor 0 para la variable altura.

Análisis del problema: se debe de establecer una variable sumadora llamada *suma* la cual almacenara la suma de las alturas de todos los alumnos, también declararemos una variable contadora llamada cant la cuál contará la cantidad de alumnos al que se les está leyendo sus alturas.

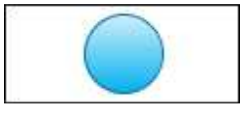
```
Inicio
Var: suma, cant, altura, num, media
suma ← 0 //inicialización de variable
cant ← 0  //inicialización de variable
Leer altura //se lee la altura del primer alumno
Hacer
    suma ← suma + altura
    num ← num + 1
    leer altura  //se lee la altura del siguiente alumno

mientras (altura <> 0)  //se evalúa que la altura del alumno sea distinto de cero
fin- hacer
media ← suma / num //se calcula la media
Imprimir media
Fin
```

Diagrama de flujo

Un diagrama de flujo es una representación gráfica del flujo lógico de datos que se utilizará como solución al problema, generalmente de una determinada parte del programa. Esto quiere decir que los diagramas se dibujan antes de escribir el programa en la computadora, para asegurar un desarrollo lógico.

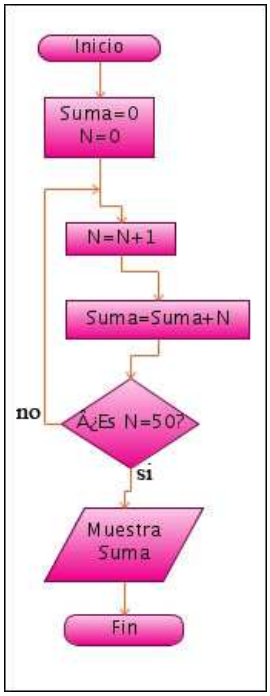
Reglas para dibujar diagramas de flujo.

Los Diagramas de flujo se dibujan generalmente usando algunos símbolos estándares; sin embargo, algunos símbolos especiales pueden también ser desarrollados cuando sean requeridos. Algunos símbolos estándares, que se requieren con frecuencia para diagramar programas de computadora se muestran a continuación:

	Inicio o fin del programa
	Pasos, procesos o líneas de instrucción de programa de computo
	Entrada y salida de datos
	Toma de decisiones y Ramificación
	Conector para unir el flujo a otra parte del diagrama

Ejemplo de diagrama de flujo

Diagrama de flujo que encuentra la suma de los primeros 50 números naturales

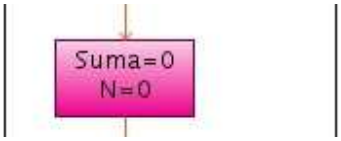


Descripción del algoritmo anterior

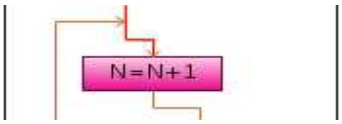
Suma, es la variable a la que se le va agregando el valor de cada número natural, es una variable acumuladora o sumadora, ambas variables tienen que tener un valor inicial. **N**, es el contador. Éste recorrerá los números hasta llegar al 50.

- El primer bloque indica el inicio del Diagrama de flujo.

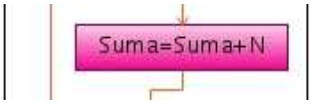
- El segundo bloque, es un **Símbolo de procesos**. En este bloque se asume que las variables **suma** y **N** han sido declaradas previamente y las inicializa en 0 para comenzar el conteo y la suma de valores.



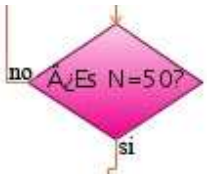
- El tercer bloque, es también un **Símbolo de procesos** En éste paso se incrementa en 1 la variable **N** (N = N + 1). Por lo que, en la primera pasada **N** valdrá 1, ya que estaba inicializada en 0.



- El cuarto bloque es exactamente lo mismo que el anterior pero en éste, ya se le agrega el valor de **N** a la variable que contendrá la suma (En el primer caso contendrá 1, ya que **N = 1**).



- El quinto bloque es un **Símbolo de Toma de** decisiones. Lo que hay dentro del bloque es una pregunta que se le hace a los valores que actualmente influyen en el proceso **¿Es N=50?**, Obviamente la respuesta es **no**, ya que **N** todavía es 1, por lo que el flujo de nuestro programa se dirigirá hacía la parte en donde se observa la palabra no o sea al *Tercer Bloque*, éste le sumará 1 (N=N+1) y vuelve a llegar a éste bloque, donde preguntará **¿Es N=50?**... ¡No!, todavía es 2. Ha pues, regresa al Tercer bloque y vuelve hacer lo mismo. Y así hasta llegar a 50, obteniendo así la suma de los primeros 50 primeros números naturales.



- Por último indicamos que el resultado será mostrado en la impresora (Este lo puedes cambiarlo por el display para mostrar datos).

- Fin del programa (o diagrama)

Diagrama N-S o de chapin

En este diagrama 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. Veamos a continuación las estructuras básicas de control que se aplican para este diagrama:

4.8 ESTRUCTURAS DE CONTROL BÁSICAS

1. Estructura secuencial: es aquella en la que una acción (instrucción) sigue a otra secuencia. La salida de una es la entrada de la otra y así sucesivamente hasta llegar al final.

Titulo del algoritmo
Inicio
Instrucción 1
Instrucción 2
Instrucción 3
. . .
Fin

Ejemplo: Diseñe un algoritmo NS que calcule el salario neto de una persona a través de la lectura de el nombre, horas trabajadas, el precio de la hora.

Ejemplo:

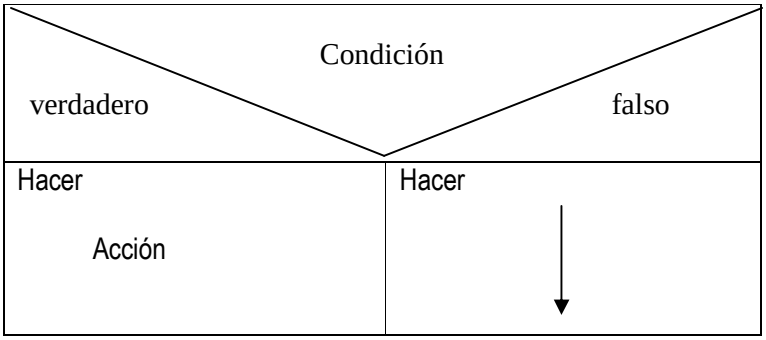
Calculo del salario neto
Inicio
Leer nombre, horas_trab, precio_hora
Salario_bruto = horas_trab * precio
Tasas = 0.10 * Salario_bruto
Salario_neto = Salario_bruto - Tasas
Escribir (nombre, horas, Salario_bruto, Salario_neto)
Fin

2. Estructuras selectivas: se utilizan para tomar decisiones lógicas. En estas estructuras se evalúa una condición y en función del resultado de la misma se realiza una acción u otra.

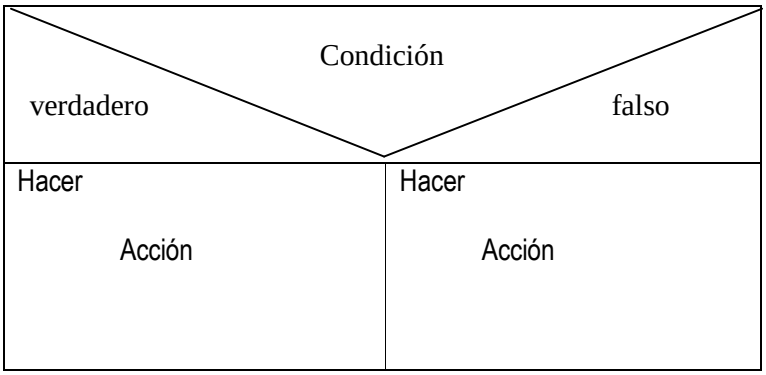
Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con la palabra en pseudocódigo (**si – entonces, si_no**) con una figura geométrica en forma de triangulo en el interior de la caja rectangular.

Estas pueden ser:

a) Simples (si - entonces)

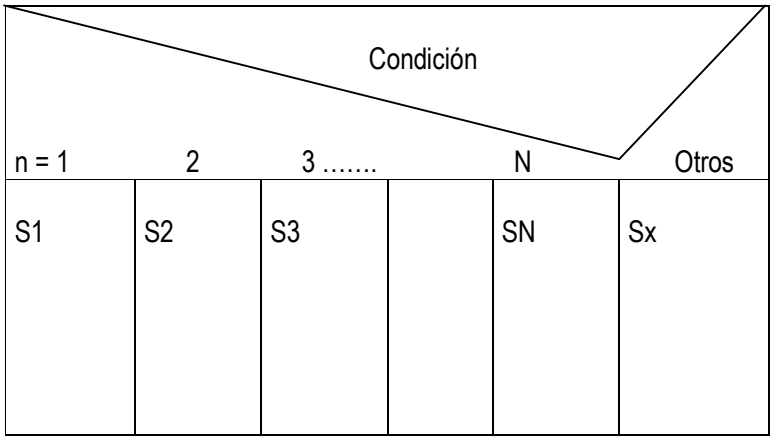


b) Dobles (si – entonces, si_no)

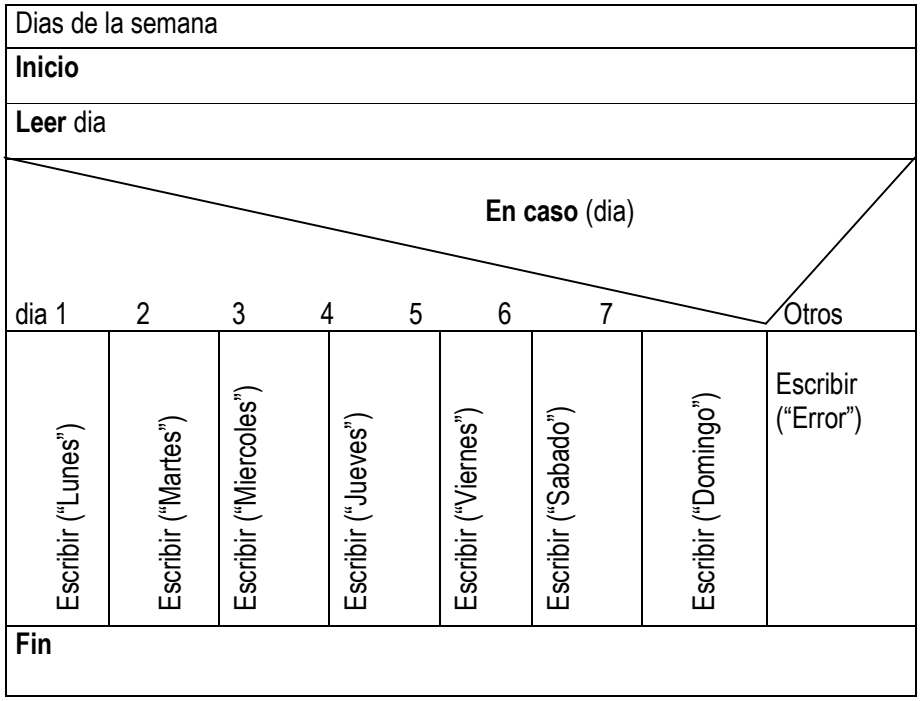


c) Múltiples (según_sea, caso de)

Evalúa una expresión que podrá tomar n valores distintos: 1, 2, 3, 4, 5, . . . **n**. según la condición elegida se realizará una de las **n** acciones.



Ejemplo: se desea diseñar un algoritmo que escriba los nombres de los días de la semana en función de una variable **dia** introducida por el teclado, que representa su posición dentro de la semana. Nota: los días de la semana son 7, **dia** tomará los valores del 1, ….7, en caso que **dia** tome un valor diferente a estos producirá un mensaje de ERROR.



3. Estructuras repetitivas: son la construcción y uso de bucles (o ciclos), un bucle es un conjunto de instrucciones que se repiten un numero o indeterminado de veces hasta que se cumple una determinada condición específica.

Las tres estructuras repetitivas más importantes son:

1. Estructura repetitiva **mientras** (while): es aquella que el cuerpo del bucle se repite mientras se cumple una determinada condición. Se utiliza normalmente cuando no se sabe exactamente cuántas veces se repetirá el ciclo (bucle).

Numeros enteros
Inicio
Leer numero
Mientras numero > 0 contador = contador +1 **Leer** número **Fin_Mientras**
Escribir ("Números enteros", contador)
Fin

2. Estructura repetitiva **para** (for): es aquella en el que el número de iteraciones del bucle se conoce de antemano. Utiliza el contador que cuenta el número de iteraciones fijadas y se termina cuando llega al valor final. Si la condición es verdadera se ejecutarán las acciones, si es falsa se saldrá del ciclo.

Su pseudocódigo es:

for (vc = vi ; vi ; vc < vf ; vc = vc + 1)

<Acciones >

Fin_for

Donde

vc : variable contadora
vc : valor inicial
vf : valor final

Diagrama N – S, Estructura Para

Para vc = vi **hasta** vf [incremento / decremento <valor> **Hacer**

<Acciones>

fin - para

3. Estructura Repetitiva **Hacer - Mientras** (do-while)
   La condición se sitúa al final del bucle y las intrusiones interiores al bucle se repetirán hasta que se cumplan la condición, si es verdadera se regresará al bucle, si es falsa saldrá del bucle.

   Pseudocódigo

   do

   <Acciones >

   while (condición)

   Ejemplo: Imprimir las treinta primeras potencias de 4, es decir 4 elevado a 1, 4 elevado a 2, 4 elevado a 3, etc.

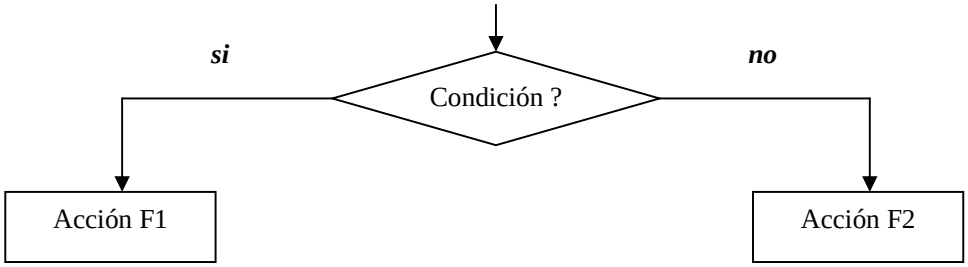
Potencias
Inicio
N ←1
Hacer Imprimir (4 n) N ←n +1 **Mientras** (n<=30)
Fin

4.9 ELEMENTOS BASICOS DE UN PROGRAMA

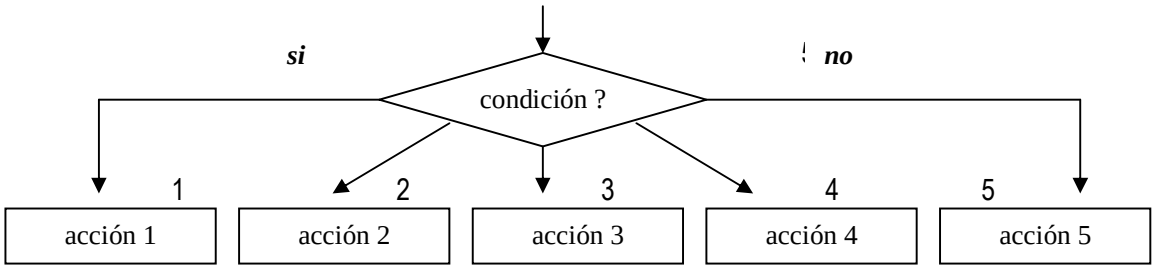
4.9.1 Decisión o Selección

Evalúa una condición y, en función del resultado de esa condición, se bifurcará a un determinado punto.

❖ Instrucción alternativa con dos posibles caminos (representación en diagrama de flujo):

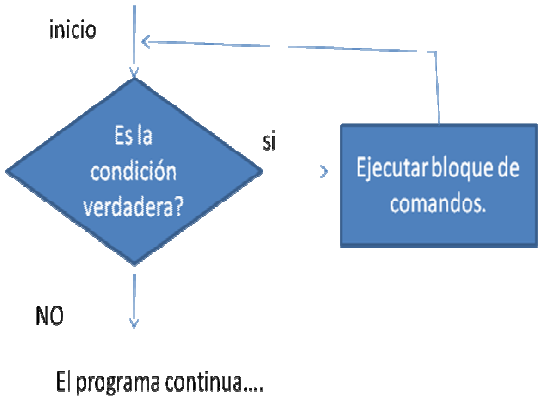


❖ Instrucción alternativa con múltiples caminos (según_sea, caso de / diagrama de flujo):



4.9.2 Bucles

Los bucles son segmentos de un algoritmo o programa, cuyas instrucciones se repiten un número determinado de veces mientras se cumple una determinada *condición* (existe o es verdadera la condición). La condición será el mecanismo el cual determinará las veces que se repetirán las instrucciones esta puede ser verdadera o falsa y se comprueba una vez a cada *paso* o *iteración* del bucle.



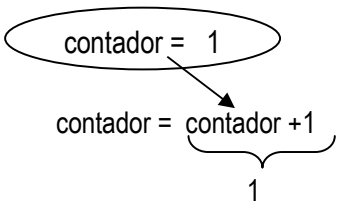
Una forma de controlar un bucle es mediante un contador*:*

4.9.3 Contador

Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante en cada iteración. En todo contador es necesario que exista una instrucción que inicializa la variable que va a tener la función de contador y esta variable puede ser cualquier identificador.

Ejemplo 1: Contador que incrementa de uno en uno

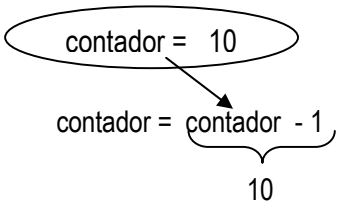
La variable *contador* va a tener la función de ser el contador.



// Inicialización de la variable contador con el valor 1

// La variable contador incrementa a 2 por la suma

Ejemplo 2: Contador que decrementa de uno en uno

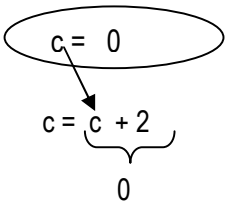


// Inicialización de la variable contador con el valor 10

// La variable contador decrementa a 9 por la resta

Nota:
Un contador puede ser positivo (incrementos, uno a uno) o negativo (decrementos, uno a uno). Un contador también puede incrementar o decrementar de dos en dos, tres en tres, cuatro en cuatro y así sucesivamente, dependiendo de la tarea que esté realizando el bucle.

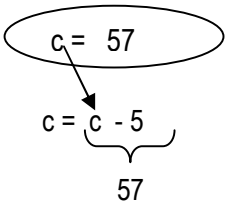
Ejemplo 3: Contador que incrementa de dos en dos



// Inicialización de la variable c con el valor 0

// La variable c incrementa a 2 por la suma

Ejemplo 4: Contador que decrementa de cinco en cinco

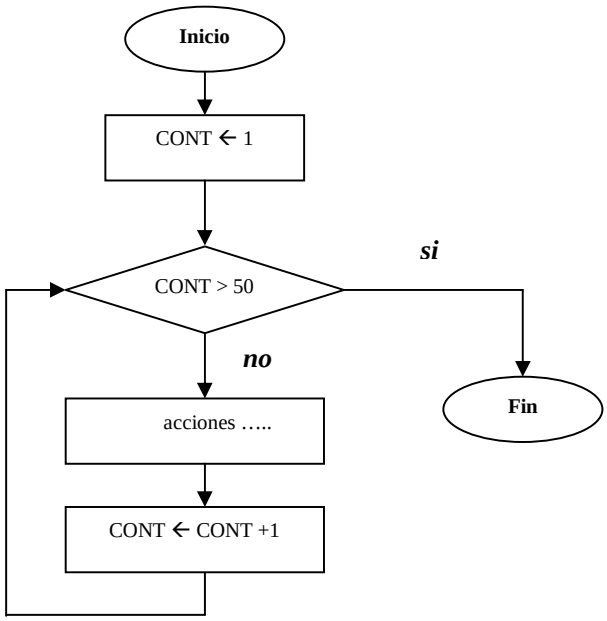


// Inicialización de la variable c con el valor 57

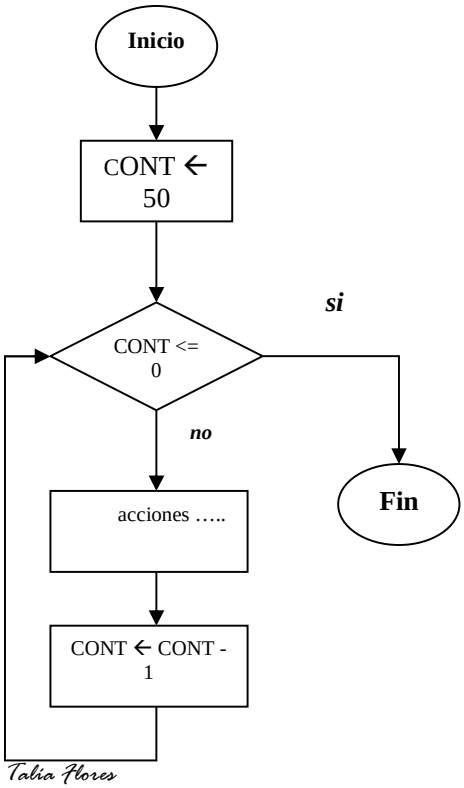
// La variable c decrementa a 52 por la resta

El siguiente ejemplo representa un diagrama de flujo para un algoritmo que se desea repetir 50 veces; el contador cuenta desde 1 hasta 50 y deja de contar hasta que la variable CONT toma el valor de 51 y se termina el bucle. El contador se representa en este ejemplo con la variable CONT, la instrucción que representa a un contador es la asignación:
CONT ← CONT + 1.

El contador puede ser positivo (incrementos, uno en uno):



El contador puede ser negativo (decrementos, uno en uno):



4.9.4 Acumulador / totalizador o sumador

Variable cuya misión es almacenar cantidades variables resultantes de sumas sucesivas. Esta variable trabaja dentro de los bucles y realiza la misma función que un contador, con la diferencia de que el incremento o decremento de cada suma es variable en lugar de constante, como el caso del contador.

La forma de representar a un acumulador es:

```
SUMA ← SUMA + N
```

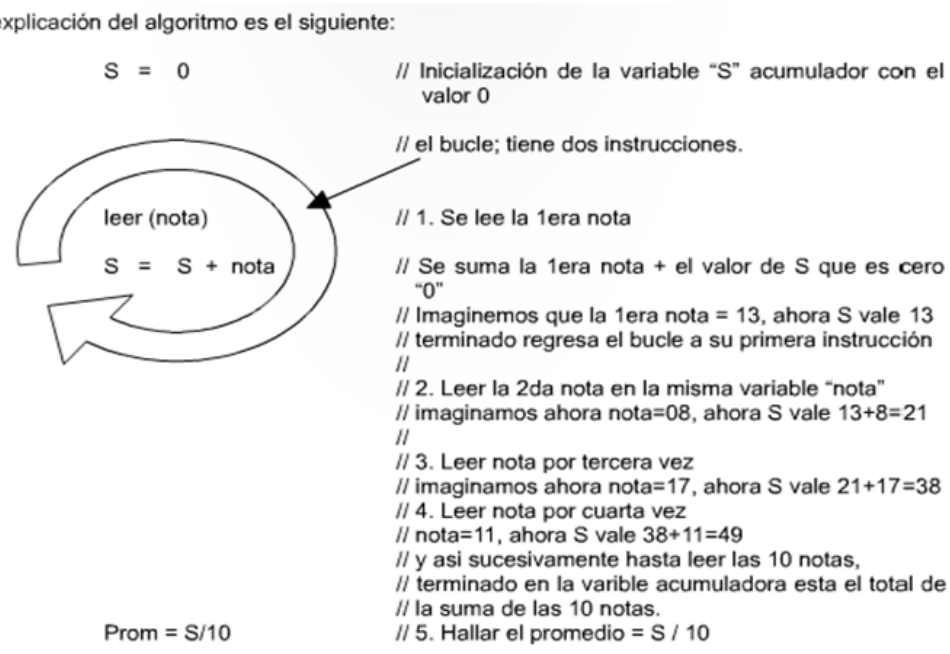
En donde N es una variable y no una constante.

Nota:
En todo acumulador es necesario que exista una instrucción que inicializa la variable que va a tener la función de acumulador y esta variable puede ser cualquier identificador.

Ejemplo 1: Calcular el promedio de notas de 10 alumnos. Primero debemos utilizar un acumulador para ir sumando las 10 notas (cantidades variables) y luego dividir este valor entre diez. La variable **S** va a tener la función de ser el acumulador

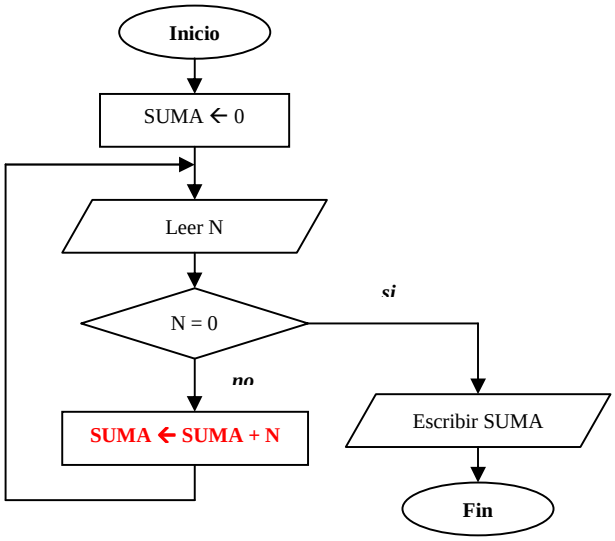
```
S ← 0              // Inicialización de la variable contador con el valor 1
Leer (nota)
S ← S + nota
Prom ← S /10
```

La explicación del algoritmo es el siguiente:



Esto es la forma como trabaja un acumulador.

El siguiente algoritmo realiza la suma de N números, la cual es almacenada en una variable llamada SUMA, el proceso se repetirá hasta que el numero ingresado sea diferente de cero. Si no cumple la condición finalizará e imprimirá la suma total de los números.



4.10 SENTENCIAS DE BUCLES O LAZOS REPETITIVOS

Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que puedan ejecutar una cantidad específica de veces. Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable (estar en función de algún dato dentro del programa). Los bucles se clasifican en:

- PARA / DESDE (FOR)
- MIENTRAS (WHILE)
- HACER - MIENTRAS (DO WHILE)

4.10.1 Estructura Repetitiva Desde / Para

En muchas ocasiones se conoce de antemano el número de veces que se desean ejecutar las acciones o instrucciones de un bucle. En estos casos en los que el numero de iteraciones es fijo, se debe usar la estructura ***desde/para***. Por ejemplo, ingresar 10 notas, 100 números, etc.

Representación en Diagrama de Flujo de la estructura ***desde/para***.

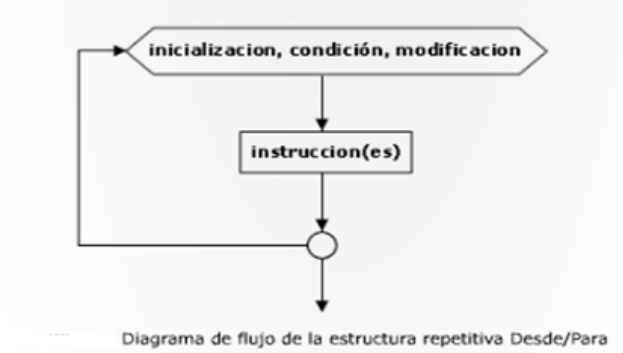
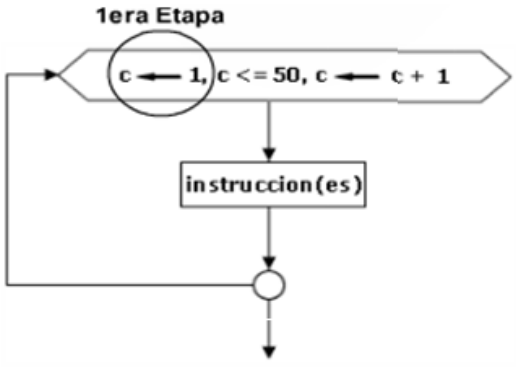


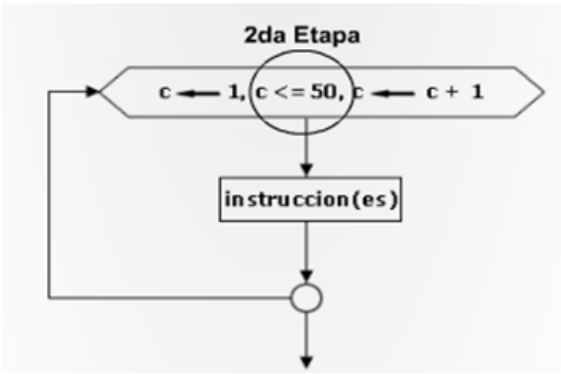
Diagrama de flujo de la estructura repetitiva Desde/Para

Ejemplo: Queremos que se repita 50 veces el bloque de instrucciones.

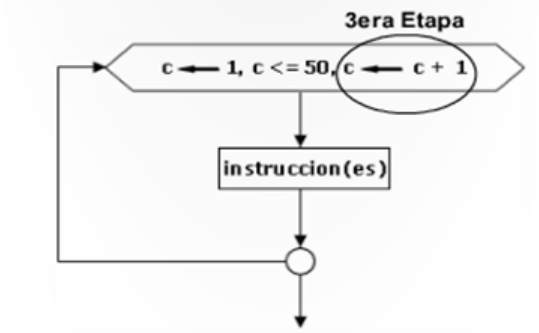
1ra etapa (c=1): cuando el ciclo comienza antes de dar la primera repetición, la variable de inicialización toma el valor indicado en el bloque de inicialización.



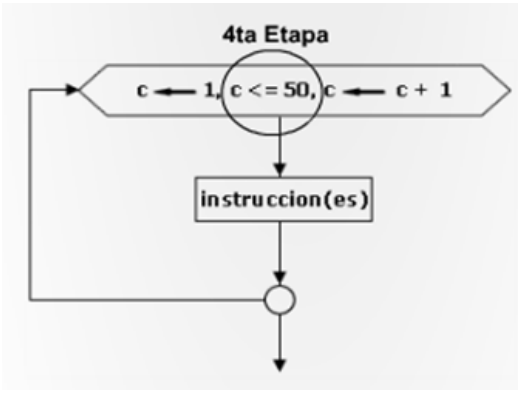
2da etapa (c<=50): inmediatamente se verifica en forma automática, si la condición es verdadera. En caso de serlo se ejecuta el bloque de instrucciones del ciclo, es decir, si el bloque tuviera 5 instrucciones esas se realizan una por una.



3ra etapa (c=c+1): al finalizar de ejecutarse el bloque de instrucciones del bucle, la ejecución de la estructura repetitiva se regresa a la sección de modificación. Se incrementa en una unidad el contador.



4ta etapa (c<=50): seguidamente, se vuelve a comprobar la condición si es verdadera, y si lo es, vuelve a realizar las instrucciones del ciclo, así prosigue ejecutándose todo el bucle hasta que la condición es falsa y sale del bucle.



Representación en Diagrama de Chapin de la estructura ***desde/para***.

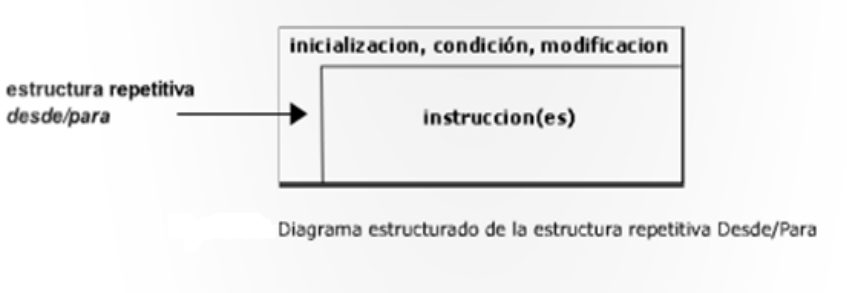
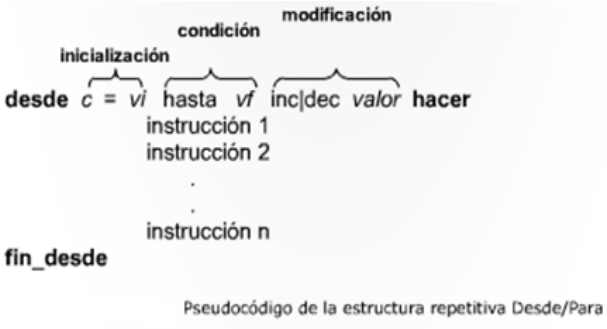


Diagrama estructurado de la estructura repetitiva Desde/Para

Representación en Pseudocódigo de la estructura ***desde/para***.



Pseudocódigo de la estructura repetitiva Desde/Para

Nota:
vi: valor inicial
vf: valor final
inc: incrementa
dec: decrementa

Ejemplo 1:

Realizar un algoritmo que permita hallar la suma de los 10 primeros números enteros positivos y su promedio.

DIAGRAMA DE FLUJO

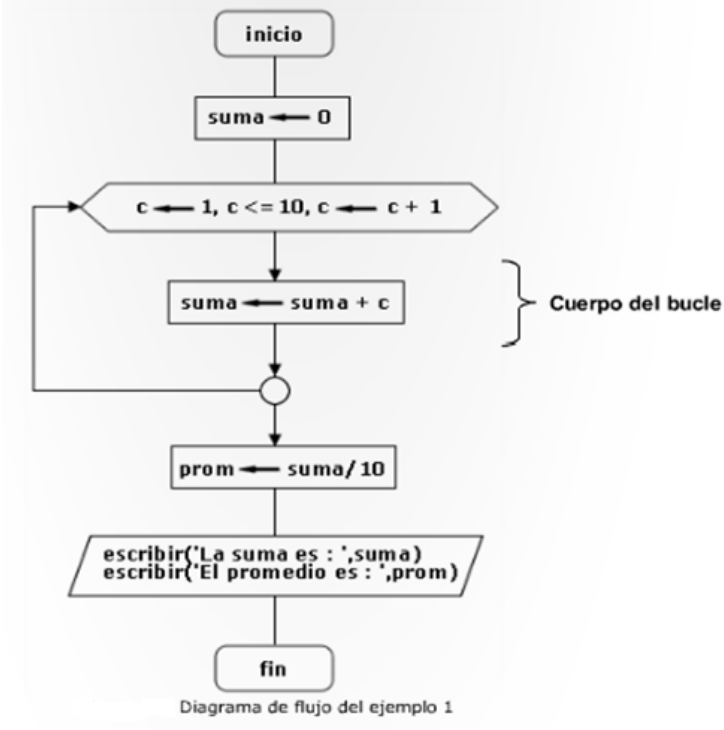


Diagrama de flujo del ejemplo 1

DIAGRAMA DE CHAPIN

Ejemplo1	
inicio	c, suma = 0, prom
c = 1, c <= 10, c = c + 1	
suma = suma + c	} Cuerpo del bucle
prom = suma/10	
escribir ('La suma es : ',suma)	
escribir ('El promedio es : ',prom)	
fin	

Diagrama de Chapin del ejemplo 1

PSEUDOCODIGO

```
algoritmo Ejemplo1
var
        entero: c , suma
        real: prom
inicio
        suma = 0
        desde  c  = 1  hasta  10  inc  1  hacer
                suma = suma + c                    } Cuerpo del bucle
        fin_desde
        prom = suma/10
        escribir('La suma es : ',suma)
        escribir('El promedio es : ',prom)
fin
```

Pseudocódigo del ejemplo 1

Ejemplo 2:

Realizar un algoritmo que permita hallar la suma y promedio de 20 números enteros ingresados por teclado.

DIAGRAMA DE FLUJO

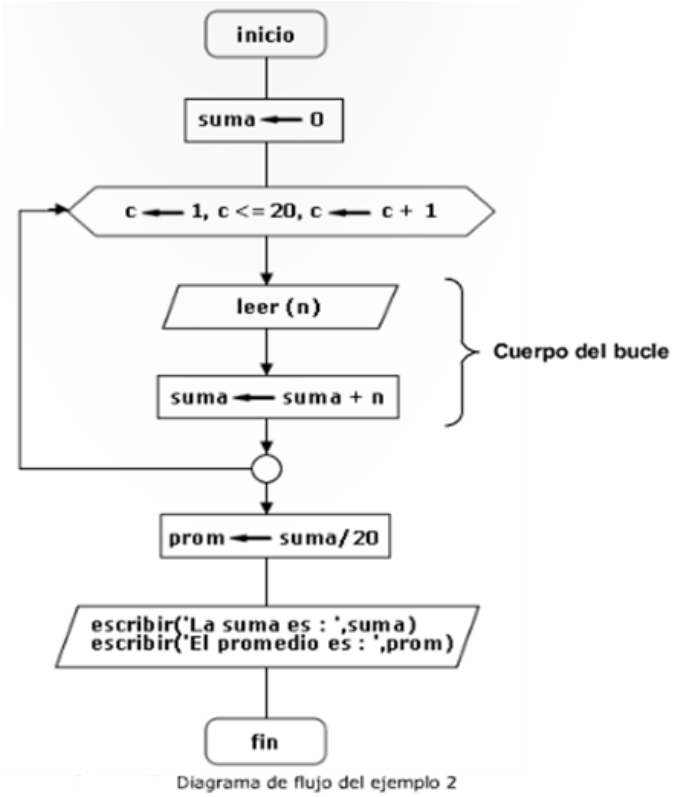


Diagrama de flujo del ejemplo 2

DIAGRAMA DE CHAPIN

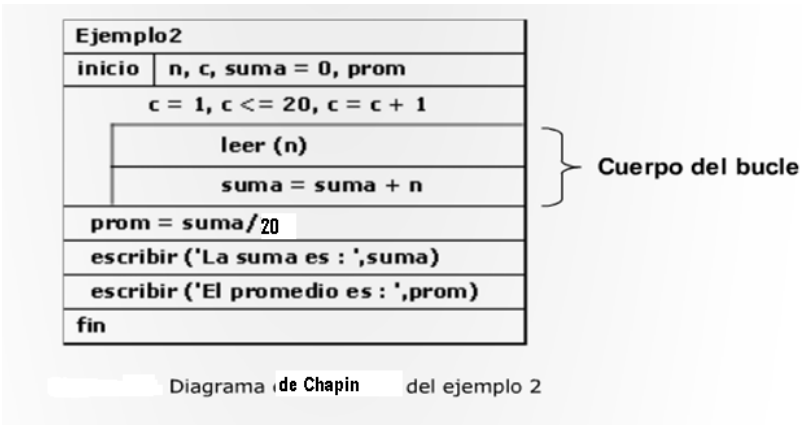


Diagrama de Chapin del ejemplo 2

PSEUDOCODIGO

```
algoritmo Ejemplo2
var
        entero: n, c, suma
        real: prom
inicio
        suma = 0
        desde c = 1 hasta 20 inc 1
                leer(n)
                suma = suma + n
        fin_desde
        prom = suma/20
        escribir('La suma es : ',suma)
        escribir('El promedio es : ',prom)
fin
```

Cuerpo del bucle

Pseudocódigo del ejemplo 2

4.10.2 Estructura Repetitiva Mientras

El proceso de la estructura repetitiva *Mientras* es el siguiente: para que ingrese al cuerpo del bucle tiene que evaluarse una condición, si ésta es verdadera se ingresa y se realizan todas las instrucciones que están dentro del cuerpo del bucle; terminada la última instrucción se vuelve a comprobar la condición; se seguirá realizando el bucle mientras la condición siga siendo *verdadera* y si en un momento es *falsa* sale del bucle.

Es decir, la estructura repetitiva *mientras* es aquella en que el cuerpo del bucle se repite mientras se cumpla una determinada condición, termina cuando ya no se cumple.

Si al querer ingresar al cuerpo del bucle por primera vez y la condición es *falsa*, no ingresa al bucle, ninguna acción se realiza y el algoritmo prosigue en la siguiente instrucción fuera del bucle.

Representación en Diagrama de Flujo de la estructura ***mientras***.

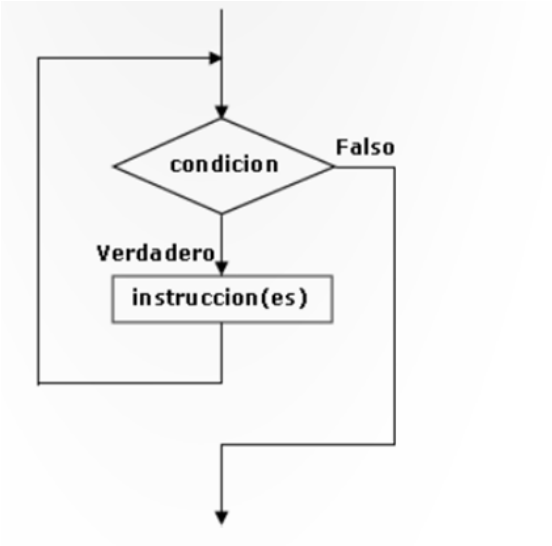


Diagrama de Flujo de la estructura repetitiva Mientras

Representación en Diagrama de Chapin de la estructura ***mientras***

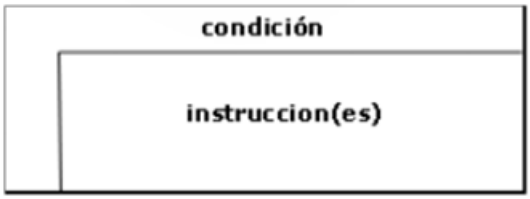


Diagrama **de chapin** de la estructura repetitiva Mientras

Representación en Pseudocódigo de la estructura ***mientras***

```
mientras (condición) hacer
        instrucción 1
        instrucción 2
            .
            .
        instrucción n
fin_mientras
```

Pseudocodigo de la estructura repetitiva mientras

Ejemplo 1:

Escribir un algoritmo que lea las 40 notas finales del curso de Introducción a la Ingeniería en Computación e informe cuantos alumnos han aprobado y cuantos reprobaron. Dato:(Nota >=60 Aprobado).

DIAGRAMA DE FLUJO

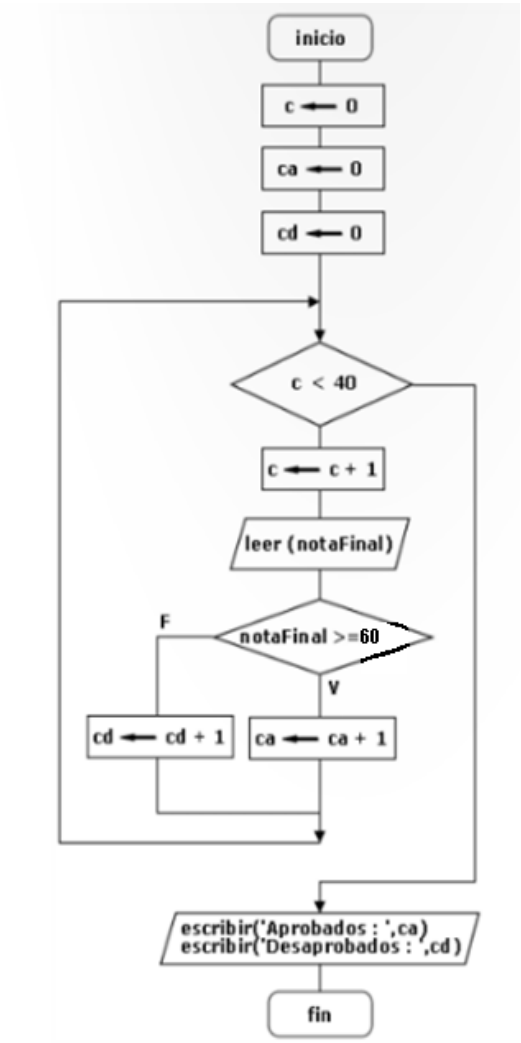


Diagrama de Flujo del ejemplo

DIAGRAMA DE CHAPIN

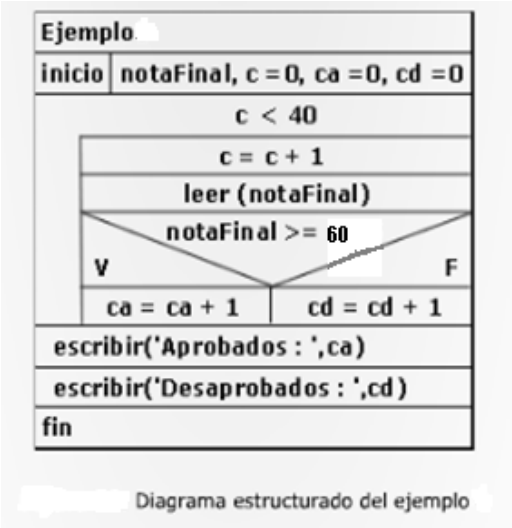


Diagrama estructurado del ejemplo

PSEUDOCODIGO

```
algoritmo Ejemplo
var
        entero: c, ca, cd
        real: notaFinal
inicio
        c = 0
        ca = 0
        cd = 0
        mientras ( c < 40 ) hacer
                c = c + 1
                leer (notaFinal)
                si  (notaFinal >= 10.5)  entonces
                        ca = ca + 1
                sino
                        cd = cd + 1
                fin_si
        fin_mientras
        escribir('Aprobados : ',ca)
        escribir('Desaprobados : ',cd)
fin
```

Pseudocódigo del ejemplo

4.10.3 Estructura Repetitiva Hacer - Mientras

Existen muchas situaciones en las que se desea que un bucle se ejecute al menos una vez antes de comprobar la condición de repetición, para ello se puede utilizar la estructura repetitiva *Hacer – Mientras*. Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutara el bloque repetitivo.

Cuando una instrucción *Hacer – Mientras* se ejecuta, lo primero que sucede es la ejecución del bucle (todas las instrucciones) y a continuación se evalúa la expresión booleana de la condición. Si se evalúa como *verdadera* se seguirá repitiendo el bucle hasta que la condición sea *falsa*.

Dentro del bucle existirá una instrucción que en un cierto momento hará que la condición sea *falsa*.

Representación en Diagrama de Flujo de la estructura ***mientras***.

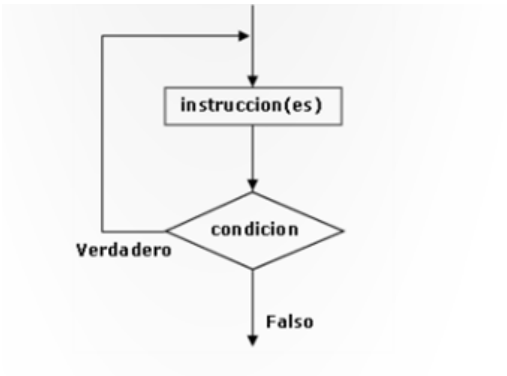


Diagrama de flujo de la estructura repetitiva Hacer-Mientras

Representación en Diagrama de Chapin de la estructura ***mientras***

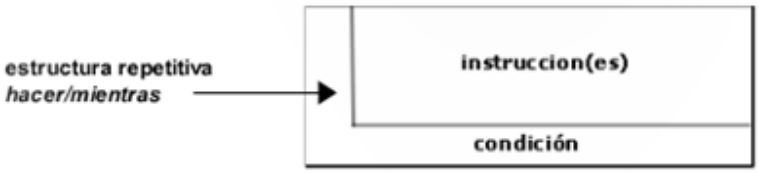


Diagrama **de chapin** de la estructura repetitiva Hacer-Mientras

Representación en Pseudocódigo de la estructura ***mientras***

```
hacer
        instrucción 1
        instrucción 2
            .
            .
        instrucción n
mientras (condición)
```

Pseudocódigo de la estructura repetitiva Hacer-Mientras

Ejemplo 1:

En un curso de programación existen 40 alumnos, de los cuales se tiene la nota final del mismo. Se pide realizar un algoritmo que permita hallar el promedio general del curso.

SOLUCION:

DIAGRAMA DE FLUJO

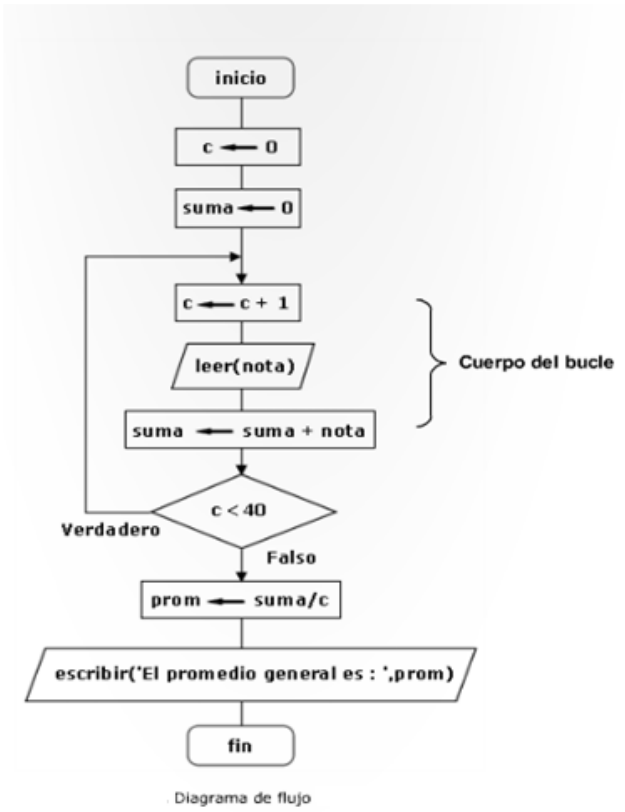


Diagrama de flujo

DIAGRAMA DE CHAPIN

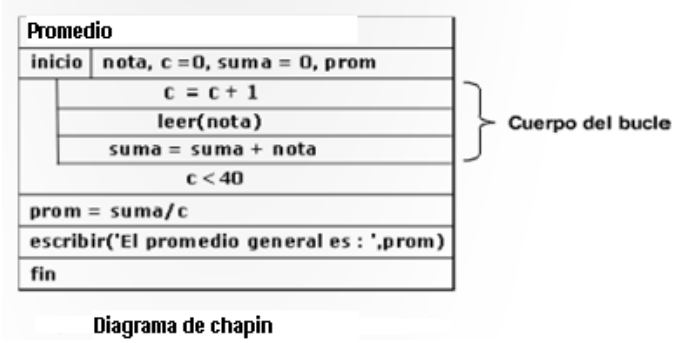


Diagrama de chapin

PSEUDOCODIGO

```
algoritmo promedio
var
        entero: nota, c, suma
        real: prom
inicio
        c = 0
        suma = 0
        hacer
                c = c + 1
                leer(nota)
                suma = suma + nota
        mientras (c < 40)
        prom = suma/c
        escribir('El promedio general es : ',prom)
fin
```

Cuerpo del bucle

Pseudocódigo

Ejemplo 2:

Se desea contabilizar cuantos números son positivos, negativos, y neutros del ingreso continuo de números enteros. Este proceso se repite hasta que el usuario pulse la tecla ´**N**´ al mensaje en pantalla "¿Desea continuar [s/n]?".

SOLUCION:

Para la solución del problema se va a necesitar de tres contadores, que realizaran la acción de contar los números positivos, negativos y neutros que se ingresan por teclado, de una serie de números.

Para ello, utilizaremos tres variables:

- cc: contador de ceros
- cp: contador de positivos
- cn: contador de negativos

Utilizaremos estructuras selectivas anidadas, con las que haremos la comprobación de cuáles números son iguales a cero, positivos y negativos. De acuerdo al resultado de la condición, los contadores (cc, cp y cn) incrementaran su valor.

Terminadas las condiciones se preguntará al usuario *¿Desea continuar [s/n]?*(en caso de 's' podrá seguir ingresando mas números), en este caso se utilizara una variable tipo carácter llamada **opc**. El cual almacenará el carácter 's'ó 'n'. De acuerdo a la respuesta seguiremos o finalizaremos el bucle.

DIAGRAMA DE FLUJO

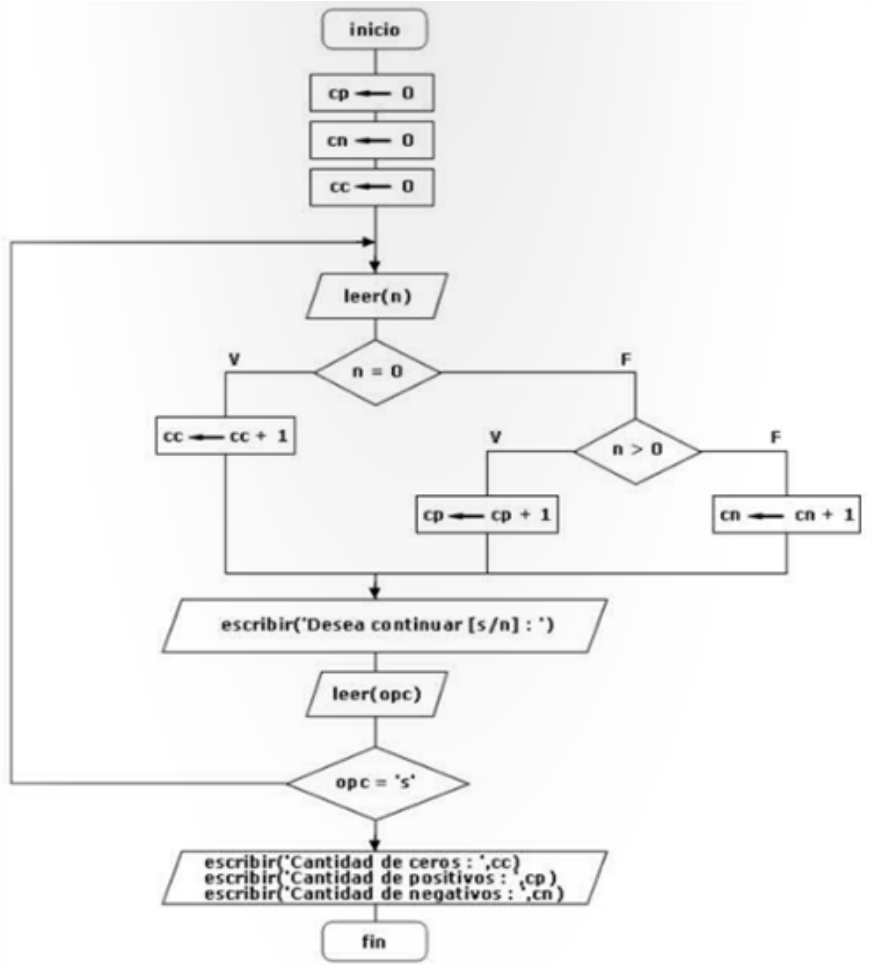


Diagrama de flujo

DIAGRAMA DE CHAPIN

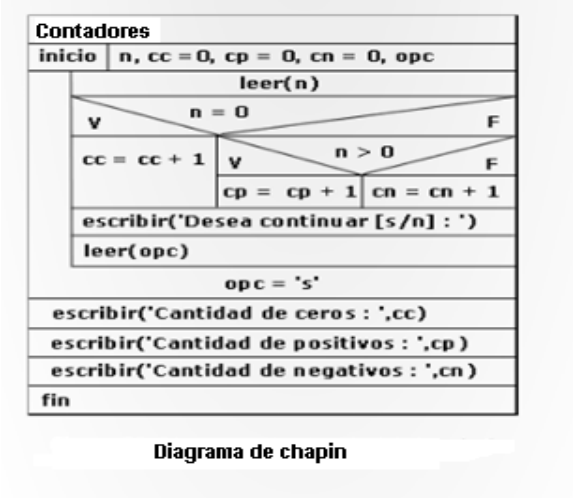


Diagrama de chapin

PSEUDOCODIGO

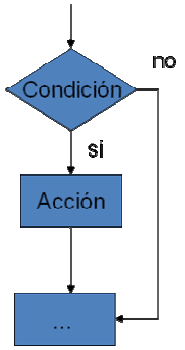
```
Algoritmo Contadores
var
        entero: n, cc, cp, cn
        caracter: opc
inicio
        cc = 0
        cp = 0
        cn = 0
        hacer
                leer (n)
                si (n=0) entonces
                        cc = cc + 1
                sino
                        si (n>0) entonces
                                cp = cp + 1
                        sino
                                cn = cn + 1
                        fin_si
                fin_si
                escribir('Desea continuar : [s/n] :')
                escribir(opc)
        mientras (opc = 's')
        escribir('Cantidad de ceros : ',cc)
        escribir('Cantidad de positivos : ',cp)
        escribir('Cantidad de ceros : ',cn)
fin
```

Pseudocódigo

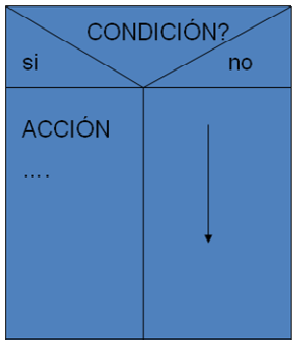
4.11 RESUMEN DE LA UNIDAD

- REPRESENTACION DE CONDICIONAL (IF)

Diagrama de Flujo	Pseudocodigo	Diagramas N/S

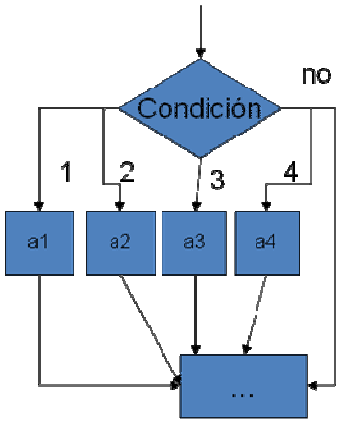



```
SI <CONDICIÓN> ENTONCES
        ACCIÓN
        ....
SINO
        ....
FIN_SI
```

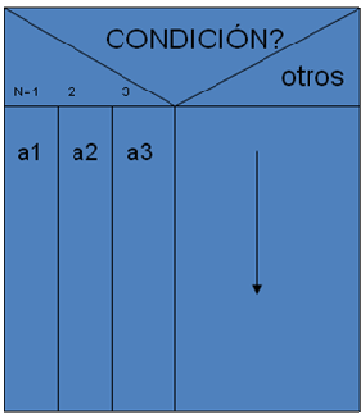


- REPRESENTACION DE ALTERNATIVA MULTIPLES (CASE / CASO DE)

Diagrama de Flujo	Pseudocodigo	Diagramas N/S

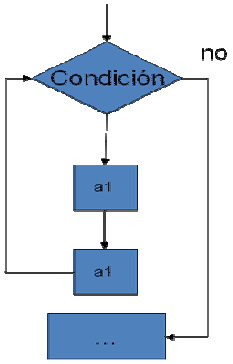



```
CASE <CONDICIÓN>
        V1: a1
        V2: a2
        V3: a3
        V4: a4
        ELSE
                OTRAS
END_CASE
```

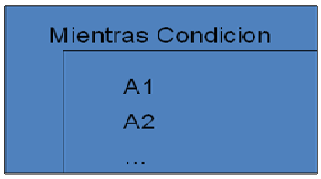


REPRESENTACION DE BUCLE (MIENTRAS / WHILE)

Diagrama de Flujo	Pseudocodigo	Diagramas N/S

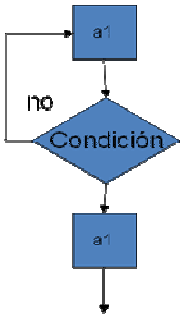



```
MIENTRAS <CONDICIÓN> HAGA
        A1
        A2

FIN MIENTRAS
```

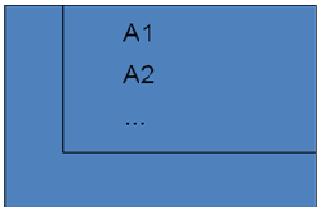


REPRESENTACION DE BUCLE (HACER - MIENTRAS / DO-WHILE)

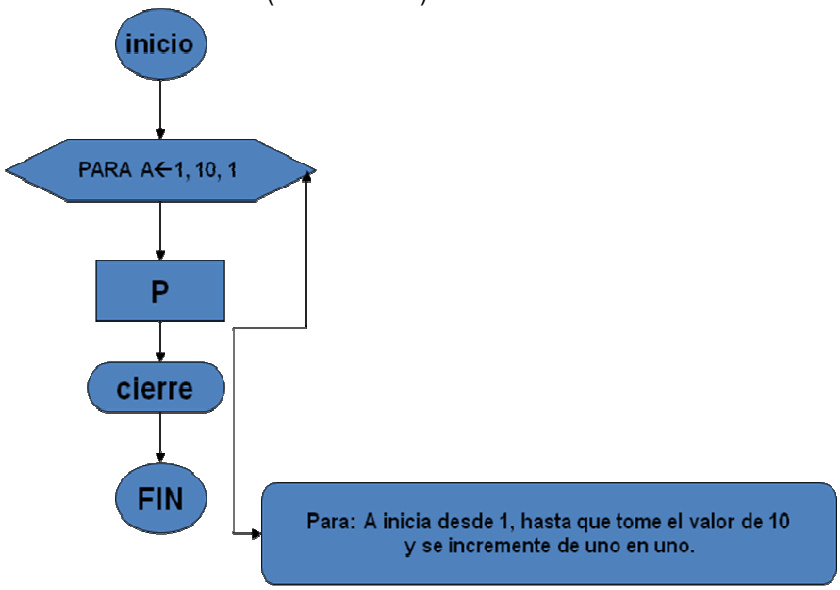
Diagrama de Flujo	Pseudocodigo	Diagramas N/S




```
HAGA
        A1
        A2
        ...
HASTAQUE CONDICION
```



REPRESENTACION DE BUCLE (PARA / FOR)



4.12 EJEMPLOS SOBRE ALGORITMOS

EJEMPLO 1

```
Algoritmo: GRADOS A RADIANES.
/*Este algoritmo convierte de grados a radianes. Ejemplo de una Estructura Secuencial.
Constante PI 3.1416 */
PRINCIPAL
inicio
    flotante GRAD, RAD ;
    leer (GRAD);
    RAD ← GRAD * PI /180;
  /* El símbolo * indica multiplicación*/
   imprimir (RAD) ;
fin
```

EJEMPLO 2

```
Algoritmo: SEGUNDO GRADO
/* Calcula las raíces reales de la ecuación de segundo grado: aX² + bX + c = 0. Ejemplo de Estructura de
Bifurcación Condicional. */
PRINCIPAL
inicio
    flotante   a, b, c, d, X1, X2, RE, IM ;
    imprimir  ("DAME LOS VALORES DE a, b, c") ;
    leer (a, b, c) ;
    d ← (b *  b) – 4 * a * c ;
      si (d > 0)
        inicio
         /* La función sqrt calcula la raíz cuadrada */
            X1 ← (-b + sqrt (d)) / (2 * a) ;
            X2 ← (-b - sqrt (d))  / (2 * a) ;
             imprimir ("LAS RAICES SON:") ;
             imprimir (X1 , X2) ;
         fin
        sino
            inicio
         /* La función fabs calcula el valor absoluto y la función sqrt calcula la raíz cuadrada */
              d ← fabs (d) ;
              RE ← -b / (2 * a) ;
              IM ← sqrt (d) / (2 * a) ;
              imprimir (" LAS RAICES SON IMAGINARIAS Y SON:") ;
              imprimir (RE, IM) ;
          fin
fin
```

EJEMPLO 3

```
Algoritmo: POTENCIA
/* Calcula el número X elevado a la n.Ejemplo de Estructura Ciclo Mientras. */
PRINCIPAL
inicio
    flotante X, pot, n, k ;
    imprimir ("Dame el valor de la base y del exponente") ;
    leer (X, n) ;
    k ← 0 ;
   pot ← 1 ;
   mientras  (k < n)
      inicio
         pot ← pot  *  X ;
         k ← k + 1 ;
      fin
   imprimir ("El resultado es" pot) ;
fin
```

EJEMPLO 4

```
Algoritmo: ÁREA MÁXIMA
/* Calcula cuando el área de un círculo es mayor a cierto valor dado. Despliega el valor del radio para el cual
todavía es menor y el área ocupada. Ejemplo de estructura Ciclo Hacer_Mientras. */
   constante pi  3.1416
PRINCIPAL
inicio
    flotante    rad, inc, amax, área ;
    imprimir ("Radio Inicial") ;
    leer (rad) ;
    imprimir ("Incremento del Radio") ;
    leer (inc) ;
    imprimir ("Area Máxima") ;
    leer (amax) ;
       hacer
          inicio
             area ← pi*rad*rad ;
             rad ← rad+inc ;
          fin
       mientras (amax > área)
             rad ← rad-inc ;
             area ← pi*rad*rad ;
             imprimir ("Para el radio", rad) ;
             imprimir ("se tiene un área de ", área) ;
             imprimir ("que es menor el área máxima de ", amax) ;
```

```
fin
```

EJEMPLO 5

```
Algoritmo: PROMEDIOS
/* Para un grupo con N alumnos se calcula el promedio de sus calificaciones, teniendo cada alumno tres calificaciones diferentes.
Ejemplo de Estructura Ciclo Desde. */
PRINCIPAL
inicio
    flotante  N, i, j, SUM CALIF, PROM ;
    imprimir ("DAME EL NUMERO DE ALUMNOS") ;
    leer (N) ;
    desde (i ← 1; i < = N; i ← i +1)
        inicio
          SUM ← 0 ;
          desde (j ← 1;  j < = 3; j ← j + 1)
            inicio
               imprimir ("DAME LA CALIFICACION", j) ;
               leer (CALIF) ;
               SUM ← SUM  + CALIF ;
            fin
          PROM ← SUM  /  3 ;
          imprimir  ("EL PROMEDIO DEL ALUMNO ES", PROM) ;
        fin
fin
```

4.13 CLASE PRÁCTICA

Conteste:

1. ¿Qué entiende por expresiones condicionales?
2. La diferencia que encuentra en una bifurcación y una condición.
3. ¿Cuál es la diferencia entre un contador y variable acumulador?
4. Diferencia entre la instrucción mientras (while) y hacer-mientras (do-while).
5. Cuando utilizar la instrucción para (for).

Razone:

1. Realizar un algoritmo que contenga todos los pasos para preparar un sándwich.

2. Construir un algoritmo para efectuar una llamada telefónica a un amigo cuyo número se desconoce pero viene en la guía telefónica.

Escriba los siguientes algoritmos en pseudocódigo, diagrama de flujo, y N-S:

1. Se desea calcular independiente la suma de los números pares e impares comprendidos entre 1 y 200.
2. Leer una serie de números distintos de cero. Obtener el número mayor, como resultado se debe visualizar el número mayor y un mensaje de indicación de número negativo, caso de que se haya leído un número negativo.
3. Calcular la media de cincuenta números e imprimir su resultado.
4. Sumar diez números introducidos por el teclado.
5. Desarrolle un algoritmo que permita leer dos valores distintos, determinar cuál de los dos valores es el mayor y escribirlo.
6. Desarrolle un algoritmo que suma dos números.
7. Desarrolle un algoritmo que permita leer tres valores y almacenarlos en las variables A, B y C, respectivamente. El algoritmo debe imprimir cual es el mayor y cuál es el menor. Recuerde constatar que los tres valores introducidos por el teclado sean valores distintos. Presente un mensaje de alerta en caso de que se detecte la introducción de valores iguales.
8. Desarrolle un algoritmo que lea cuatro números diferentes y a continuación imprima el mayor de los cuatro números introducidos y también el menor de ellos.
9. Desarrolle un algoritmo que realice la sumatoria de los números enteros comprendidos entre el 1 y el 10, es decir, 1 + 2 + 3 + …. + 10.
10. Realizar un algoritmo que permita leer 10 números positivos y negativos, e imprima solamente los números positivos.
11. Realizar un algoritmo que permita leer 20 números e imprimir cuantos son positivos, negativos y neutros. Nota: neutro se le conoce al cero “0”.

Capítulo 5

PSEint

5.1 INTRODUCCIÓN

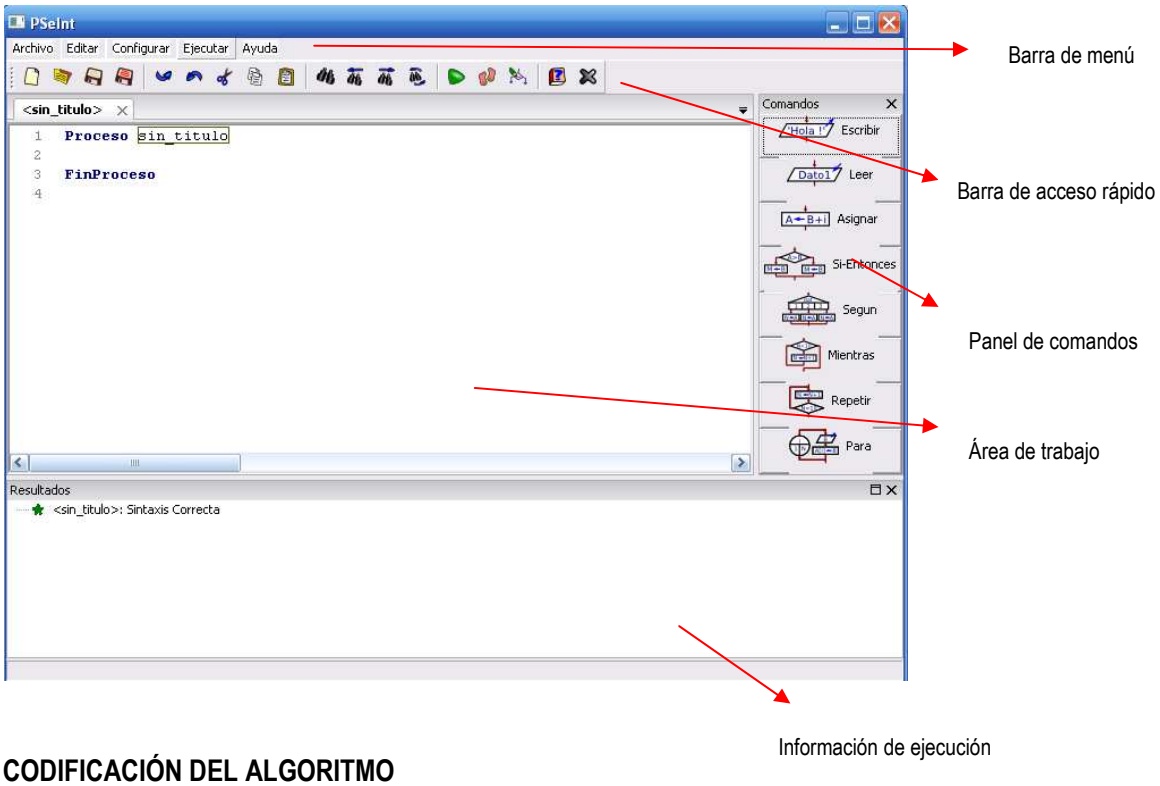
PseInt (*Pseudo-Intérprete*) es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés), es un editor e intérprete de programas escritos en pseudocódigo. Su interfaz gráfica permite crear, almacenar, ejecutar y corregir fácilmente programas en Pseudocódigo.

La sencillez del lenguaje Pseudocódigo lo hacen ideal para la enseñanza de la programación. Permite escribir programas con instrucciones condicionales (Si-Entonces-Sino, Según) y ciclos (Mientras, Hasta Que, Para), y también usar valores numéricos (números decimales), lógicos, caracteres y arreglos. También provee funciones de entrada/salida y algunas funciones matemáticas.

5.2 UTILIZACION DEL ENTORNO

- **Elementos de la Ventana**

- Barra de Título
- Menú de Opciones
- Barra de Acceso Rápido
- Área de Trabajo
- Área de Información de Ejecución
- Barras de scroll



5.3 CODIFICACIÓN DEL ALGORITMO

Explore la herramienta dando click en los diferentes botones del panel de comando, observe el efecto en el área de trabajo, una vez que se haya familiarizado un poco con la herramienta intente adaptar el código mostrado en el pseudocódigo del problema anterior en el PseInt:

Las variables en PSEint:

Tipo de dato	Descripción	Ejemplo
entero	Tipo de dato asociado a cantidades enteras. No poseen parte decimal. Ejemplo: 5, 6, -15, 199,...	Numero de vacas, edad.
real	Tipo de dato asociado a cantidades con parte decimal. Por ejemplo: 0.06, -3.4, 2.16, 1000.345,...	Estatura, peso, volumen.
lógicos	Se refiere a aquellos datos que pueden tomar solo dos posibles valores falso (F) o verdadero (T)	
alfanuméricos	Asociado a aquellos datos que contienen caracteres alfanuméricos (letras, número, signos de puntuación, etc).	Nombre, cedula, telefono

5.3.1 Instrucciones de asignación:

Las instrucciones de asignaciones permiten escribir sobre una variable el valor de una expresión:

variable = expresión

Donde, una expresión es una combinación de valores, variables y operadores, los siguientes son algunos ejemplos de expresiones:

a = 5
*b =c*d+(c-f)*m*
z=(x+y)/(w+s)
s=(a/b)^3

5.3.2 Operadores aritméticos:

Operador	Significado
^	Potenciación
+	Suma
-	Resta
*	Multiplicación
/	División

5.3.3 Instrucciones de entrada y salida

- **Instrucciones de entrada**: Permite tomar uno o más datos de un medio externo (comúnmente el teclado) y asignarlos a una o más variables, su representación en pseudocódigo es:

  LEA(var1, var2, …, varN)

- **Instrucciones de salida**: Permite mostrar de variables y constante en un medio externo (comúnmente la pantalla). En pseudocódigo la instrucción asociada a la salida tiene la siguiente forma:

  ESCRIBA(var1,var2, …, varN)

Ejemplo 1:
Codifique un algoritmo que solicite el nombre y devuelva como salida el mensaje: ***Hola nombre_ingresado***. Por ejemplo, si el usuario digita Ramón, el mensaje desplegado será: ***Hola Ramón***.

Solución:
La codificación en Pseudocódigo del algoritmo se muestra a continuación:

```
Algoritmo(nombre)
        Variables:
            alfanumerica: nom
        INICIO
            ESCRIBA("Digite el nombre")
            LEA(nom)
            ESCRIBA("Hola ",nom)
        FIN_INICIO
Fin
```

Ejemplo 2:
Realizar un algoritmo que calcule el perímetro y el área de un rectángulo dada la base y la altura del mismo.

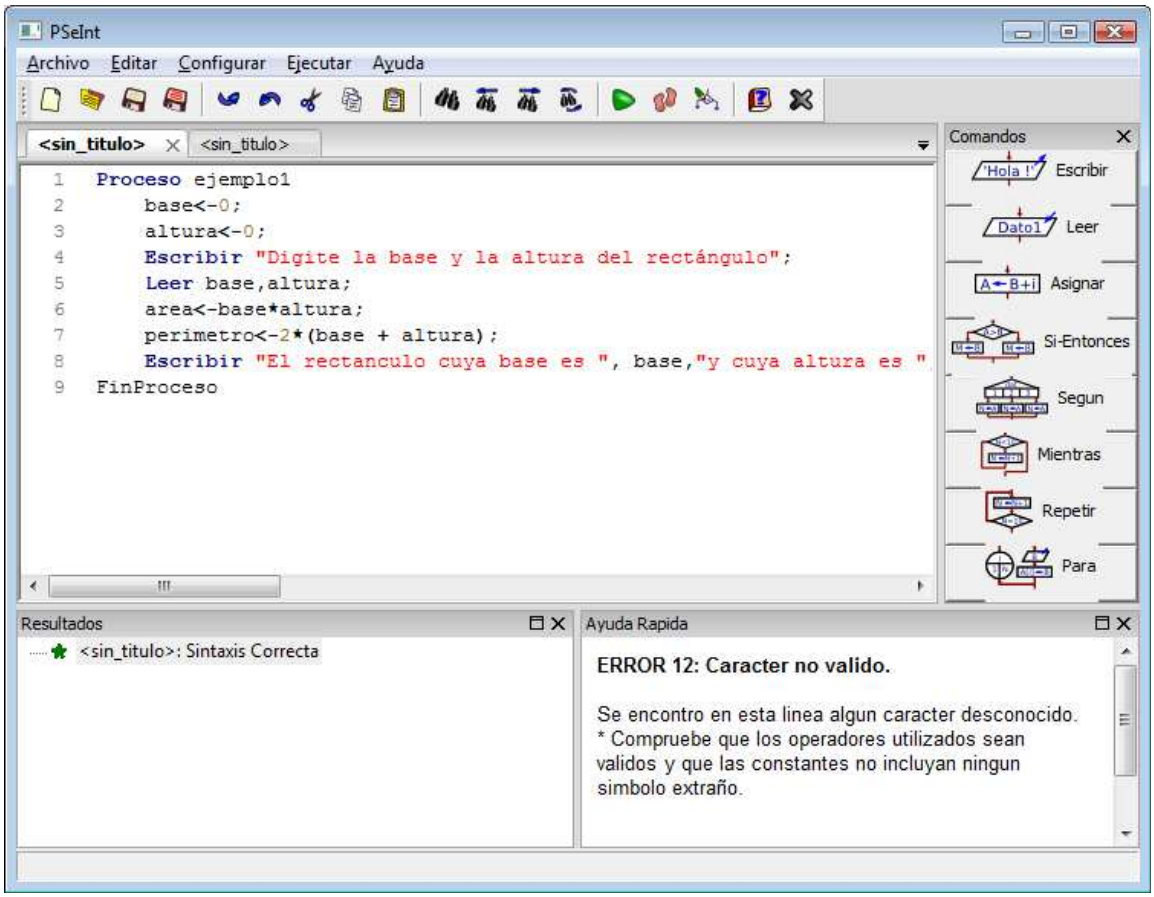
Solución en Pseudocodigo

Algoritmo(ejemplo2)
Variables:
real: area, perímetro, base = 0, altura = 0
INICIO
Escriba ("Digite la base y la altura del rectángulo")
Lea (base,altura)
area = base*altura
perímetro = 2*(base + altura)
Escriba ("El rectángulo cuya base es ", base, **"y cuya altura es "**, altura, **" tiene un perímetro de "**,perímetro, **" y una altura de** ", altura)
FIN_INICIO
Fin(ejemplo2)

Ya que se tiene el pseudocódigo del programa codificado utilizaremos PSeint para probarlo, sin embargo en el momento de codificar el anterior programa en PSeInt debemos tener en cuenta que el pseudocódigo manejado en PSeInt es un poco diferente, la siguiente tabla muestra esto en detalle:

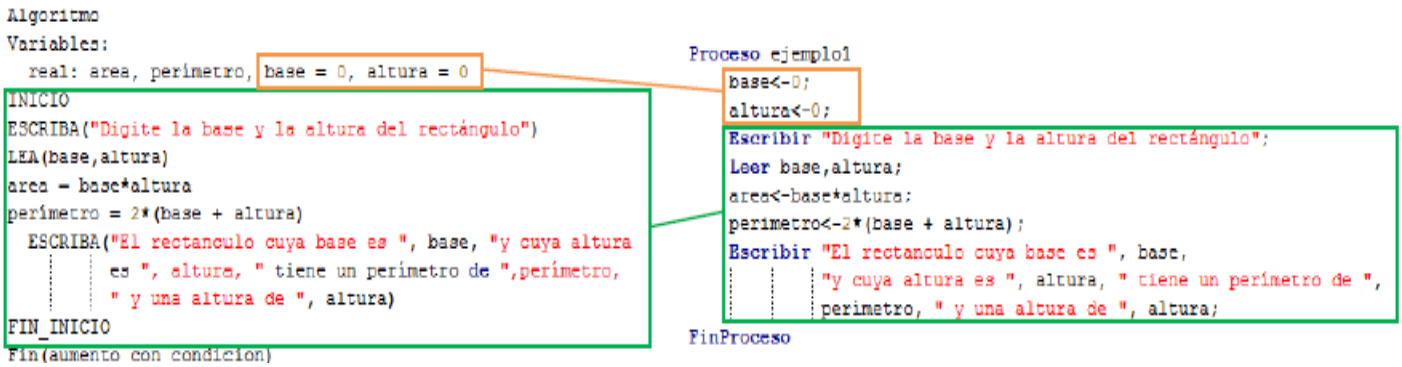
Instrucción	Pseudocódigo propio	Pseudocódigo PSeint	Observaciones
Asignación	c = 2*a*(b + c)	c <- 2*a*(b + c);	La asignación en PSeInt no es con igual (=) sino con flecha (<-) y al final va punto y coma (;).
Entrada	LEA(a,b,c)	Leer a,b,c;	La instrucción de entrada en PSeInt se llama ***Leer*** no hace uso de paréntesis y termina con signo de punto y coma (;).
Salida	ESCRIBA("Hola ",nombre)	Escribir "Hola",nombre;	La instrucción de entrada en PSeInt se llama ***Escribir*** no hace uso de paréntesis y termina con signo de punto y coma (;). Al igual que en el ESCRIBA usado por convesion la parte del mensaje que no cambia (que es constante) va entre comillas (""), y la parte variable va sin comillas ("").

5.3.4 Codificación en PSEint:



Una vez codificado el pseudocódigo (ayudado de los botones del panel de comandos) en el área de trabajo guarde el archivo como ***ejemplo1*** En una ruta conocida.

La siguiente figura muestra una comparación entre el Pseudocódigo convención y el Pseudocódigo del Pseint:

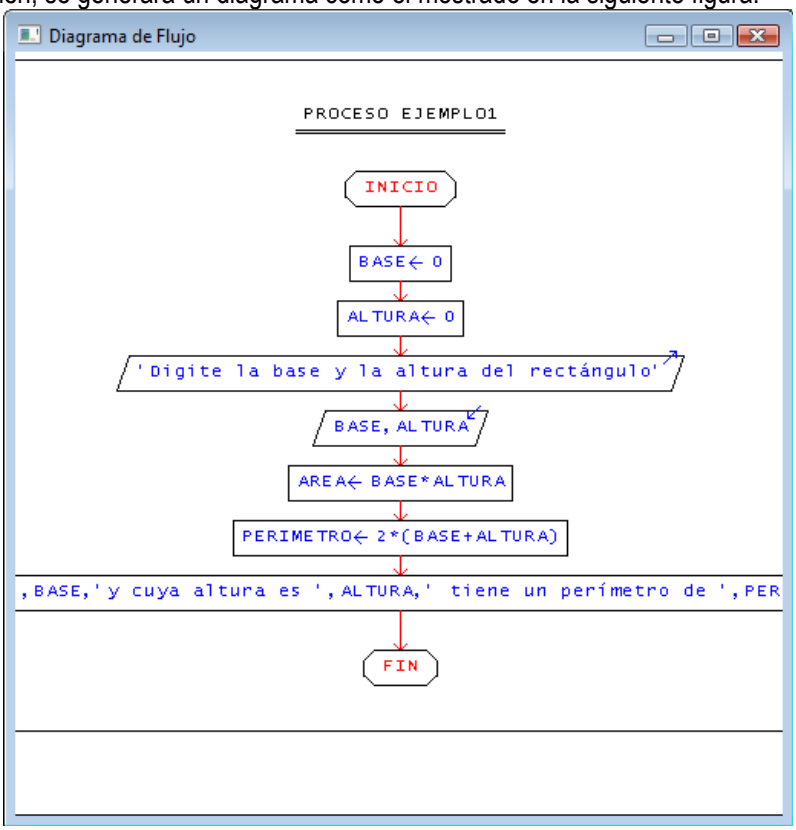


Note lo siguiente:

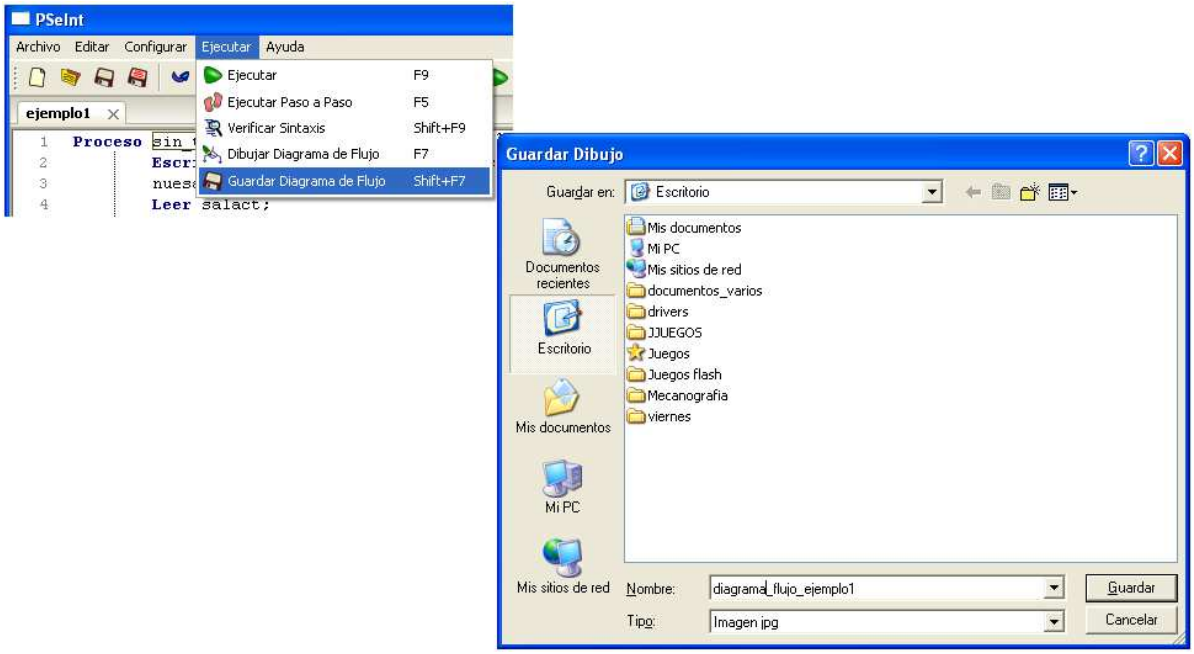
- En el PSeint no hay declaración de variables por lo tanto la codificación del algoritmo en PSeint empieza desde la parte de INICIO del algoritmo, sin embargo cuando una variable es inicializada, esto si debe ser tenido en cuenta en el Pseint antes de iniciar la codificación del programa desde el INICIO. Como se puede notar en el pseudocódigo, las variables base y altura están inicializadas en cero (base = 0, altura = 0), por ello antes de empezar la primera instrucción después del INICIO (***Escriba("Digite la base y la altura del rectángulo")***) es necesario codificar en PSeInt dicha inicialización por ello las líneas ***base ← 0;*** y ***altura ←0;*** antes del Escribir.
- El Pseudocódigo que se codifica es propiamente el que se encuentra entre las sentencias ***INICIO*** y ***FIN_INICIO*** (Ver parte resaltada en verde), salvo cuando se tiene que tener en cuenta la nota anterior.

Botón para obtener el diagrama de flujo

Si lo anterior está bien, se generara un diagrama como el mostrado en la siguiente figura:

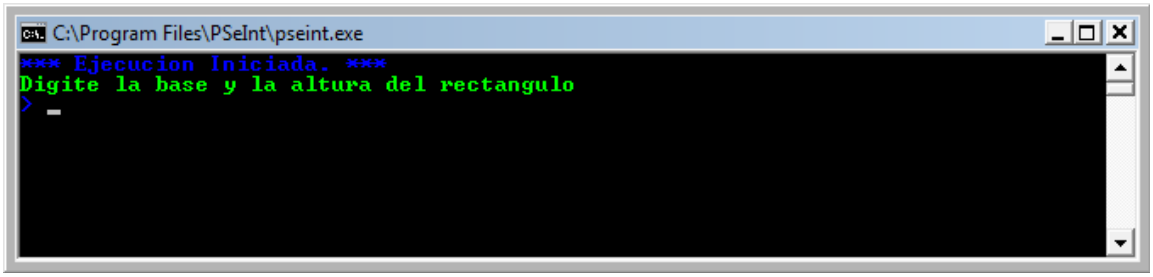


Guarde el diagrama de flujo anterior como una imagen ***jpg*** (puede serle útil después, por ejemplo para un informe).



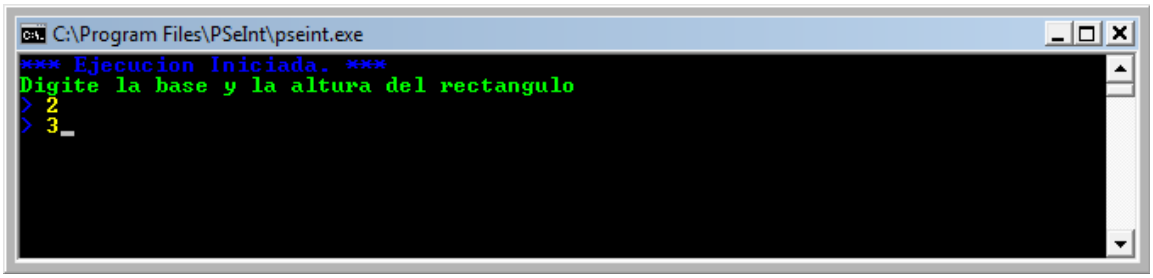
Ejecución del algoritmo
Una vez guardado el programa anterior, proceda a realizar la prueba del algoritmo presionando el botón ejecutar.

Deberá aparecer una ventana como la siguiente asociada al programa:

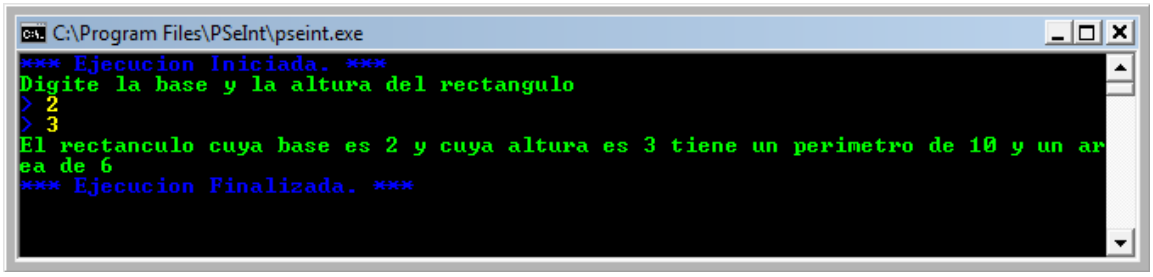


Lo anterior se debe a la instrucción **Escribir "Digite la base y la altura del rectángulo";**
El cursor se queda parpadeando, esperando a que sean introducidos los valores para la altura y la base, esto debido a la instrucción **Leer base,altura;**

Introduzca el valor de 2 como valor para la **base** y 3 como valor para la **altura**.



Note que cada vez que introduce un valor por teclado debe presionar **Enter**. Una vez que presione el **Enter** después de digitar el segundo valor aparece algo como lo siguiente:



Después de que aparece la ventana anterior si damos **Enter** esta se cierra. Intente nuevamente ejecutar el algoritmo pero esta vez de 6 como valor para la base y 7 como valor para la altura.

Ejercicio de refuerzo

Con el fin de obtener un poco de familiaridad con el Pseint, se muestra a continuación el pseudocódigo del ejemplo 1 y su codificación en PSeInt. Codifique dicha codificación en el PSeInt, genere el diagrama de flujos y ejecute el programa.

```
Algoritmo(nombre)
    Variables:
        alfanumerica: nom
    INICIO
        ESCRIBA("Digite el nombre")
        LEA(nom)
        ESCRIBA("Hola ",nom)
    FIN_INICIO
Fin(sumar)
```

```
Proceso nombre
    Escribir "Digite el nombre";
    Leer nom;
    Escribir "Hola",nom;
FinProceso
```

Note de la figura anterior que la codificación inicia desde el INICIO del pseudocódigo de convención. De la declaración de variables (alfanumérica: nom) no se tuvo en cuenta nada pues no hay inicialización de variable alguna.

Capítulo 6

PROGRAMACIÓN ESTRUCTURADA

6.1 INTRODUCCIÓN

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente.

El lenguaje de programación permite al programador[26] especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, tal como sucede con el lenguaje Léxico.

6.2 TIPOS DE LENGUAJES

Lenguajes de alto y bajo nivel:

Lenguaje de bajo nivel: Es el tipo de lenguaje que cualquier computadora es capaz de entender. Se dice que los programas escritos en forma de ceros y unos están en lenguaje de máquina, porque esa es la versión del programa que la computadora realmente lee y sigue.
Lenguajes de alto nivel: Son lenguajes de programación que se asemejan a las lenguas humanas usando palabras y frases fáciles de entender.

6.3 HISTORIA DEL LENGUAJE C

El lenguajes C nació en los Laboratorios Bell de AT&T y ha sido estrechamente asociado con el Sistema Operativo UNIX, ya que su desarrollo se realizó en este sistema y debido a que tanto UNIX como el propio compilador de C y la casi totalidad de los programas y herramientas de UNIX, fueron escritos en C. Su eficacia y claridad han hecho que el lenguaje ensamblador apenas haya sido utilizado en UNIX.

Este lenguaje está inspirado en el lenguaje B escrito por Ken Thompson en 1970 con intención de recodificar el UNIX, que en la fase de arranque estaba escrito en ensamblador, en vista a su portabilidad a otras máquinas. B era un lenguaje evolucionado e independiente de la máquina, inspirado en el lenguaje BCPL concedido por Martin Richard en 1967.

[26] Un **programador** es aquella persona que escribe, depura y mantiene el código fuente de un programa informático, es decir, del conjunto de instrucciones que ejecuta el hardware de una computadora para realizar una tarea determinada.

En 1972, Dennis Ritchie, toma el relevo y modifica el lenguaje B, creando el lenguaje C y reescribiendo el UNIX en dicho lenguaje. La novedad que proporcionó el lenguaje C sobre el B fue el diseño de tipos y estructuras de datos.

C como lenguaje de propósito general posee las siguientes características:

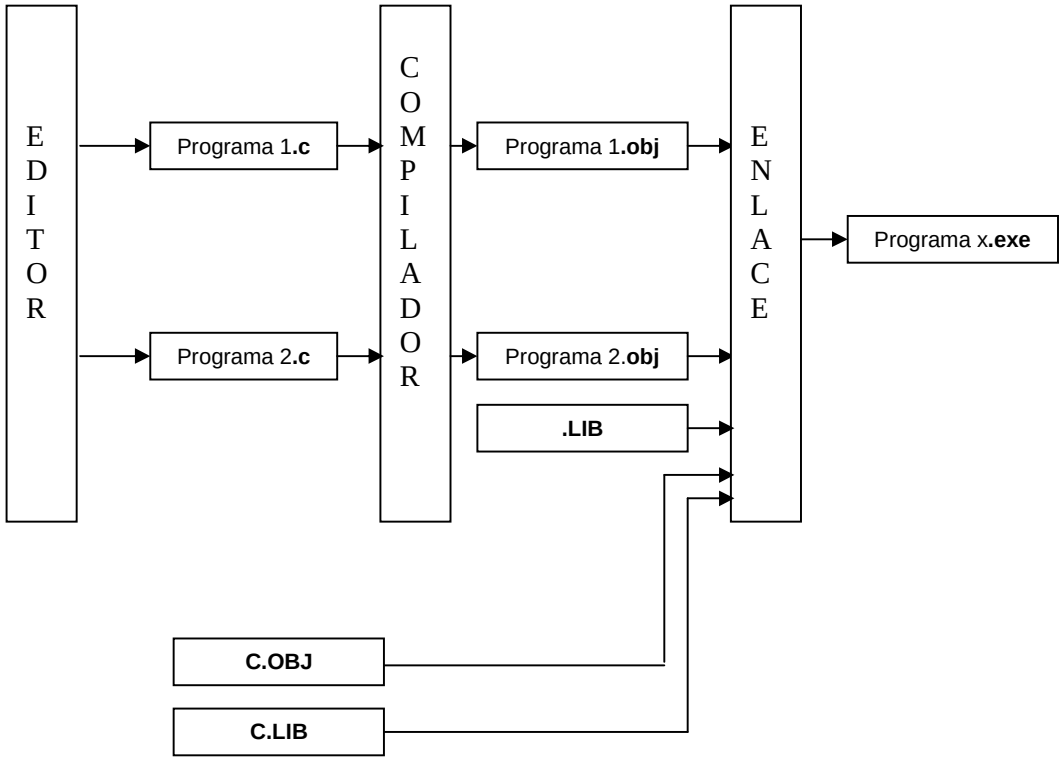
- **Programación estructurada**
- Economía en las expresiones

x = x + 1; en C se traduce en x++

- Abundancia en operadores y tipos de datos
- Codificación en alto y bajo nivel simultáneamente
- Utilización natural de las funciones primitivas del sistema
- No está orientado a ningún área especial
- Facilidad de aprendizaje

Permite lo que es la programación estructurada: división del programa en partes, módulos, bloques, su ventaja es evitar la repetición en software.

6.4 REALIZACIÓN DE UN PROGRAMA EN C:



6.5 TIPOS DE DATOS BÁSICOS:

C diferencia entre mayúsculas y minúsculas

Tipo de dato **char**:

Es usado para almacenar un valor correspondiente a un carácter del código ASCII, almacena un dato en el rango de valores de – 128 a 127 y tiene un tamaño equivalente a un byte.

Tipo de dato **int**:

Es usado para almacenar un número sin punto decimal. Almacena un dato de tipo entero en el rango -32768 a 32767.

Tipo de dato **float, double**:

Son usados para almacenar un nº real, un número real en simple precisión no tiene más de 7 dígitos significativos. Un nº en doble precisión no tiene más de 15 dígitos significativos.

Tipo de dato **enum**:

Define un tipo enumerado, no es más que una lista de valores representados por identificadores que pueden ser tomados por una variable de ese tipo.

Tipo adicional:

void: se utiliza para declarar funciones que no retornan ningún valor si void aparece entre paréntesis a continuación del nombre de una función no es interpretado como un tipo, en este caso indica que la función no acepta argumentos.

Tipos de datos	Descripción	Memoria
int	entero	2 bytes
char	caracter	1 byte
float	flotante	4 bytes
double	flotante de doble precisión	8 bytes

6.5.1 Constantes

Es un valor que no cambia durante la ejecución del programa. Una constante en C puede ser un número, un carácter o una cadena de caracteres.

Ejemplo:

420 → numero
`x´ → caracter

"constantes en C" → cadena de caracteres

6.5.2 Identificadores:

Son nombres dados a constantes, variables, tipos, funciones y etiquetas de un programa. Un identificador consta de 1 ó más caracteres (letras, dígitos, y el carácter subraya) siendo el primer carácter una letra o el carácter subraya, las letras pueden ser mayúsculas o minúsculas y se consideran como caracteres diferentes.

Ejemplo:

cont
suma_total
cuenta1
Inicio

** como regla no podemos utilizar identificar a constantes, variables , tipos, funciones y etiquetas con las palabras reservadas del lenguaje.

Sintaxis de declaración

[clase] tipo identificador [, identificador 2, . . .]

Clase: puede ser una de las siguientes:

auto, register, static o extern. La clase en una variable determina si esta tiene carácter global o local.

Tipo: indica el tipo de variable: char, int, float, double, etc. Y puede ser cualquiera de los tipos predefinidos en C o un dato definido por el usuario.

Identificador o nombre de la variable: cumple las reglas anteriores:

Ejemplo:

int valor;
char cadena;
float promedio;
static int control;

6.5.3 Palabras Reservadas

Son identificadores pre – definidos que tienen un significado especial para el compilador de C. un identificador definido por el usuario no puede tener el mismo nombre que una palabra reservada.

auto	**double**	**int**	**struct**
break	**else**	**long**	**switch**
case	**enum**	**register**	**typedef**
char	**extern**	**return**	**union**

const	**float**	**short**	**unsigned**
continue	**for**	**signed**	**void**
default	**goto**	**sizeof**	**volatile**
do	**if**	**static**	**while**

6.5.4 Comentarios:

Es una secuencia de caracteres encerrada entre los símbolos **/*** ***/** y puede aparecer en cualquier lugar donde sea permisible en espacio en blanco.

6.5.5 Variables

A diferencia de las constantes puede cambiar su valor a lo largo de la ejecución de un programa, cada variable en un programa debe de declararse antes de ser usada. La declaración consiste en enunciar el nombre de la variable y asociarle un tipo. El tipo determina los valores que puede tomar la variable así como las operaciones que pueden realizarse con ella.

Las variables pueden ser *locales* y *globales.*

Las variables locales son aquellas que se definen en el interior de una función y son visibles solo en esa función específica. Ejemplo:

```
#include <stdio.h>
void main(void)
{
  int a,b,c;  /*variables locales*/
 printf("Ingrese los valores a las variables: ")
scanf("%d %d",&a,&b);
c=a+b;
printf("La suma es: %d ",c);
getch();

}
```

Las variables globales son aquellas que se declaran fuera de la función y por defecto son visibles a cualquier función, incluyendo el main. Ejemplo:

```
#include <stdio.h>
  int a,b,c; /*variables globales*/

void main(void)
{

printf("Ingrese los valores a las variables: ")
scanf("%d %d",&a,&b);
c=a+b;
printf("La suma es: %d ",c);
getch();

 }
```

6.5.6 Contador

Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante en cada repetición. La forma en que se construye un contador es la siguiente:

```
int contador = 1; //variable con valor inicial de 1

contador = contador+1;
```

6.5.7 Acumulador (totalizador o sumador)

Un acumulador o totalizador es una variable cuya función es almacenar cantidades resultantes de operaciones sucesivas. Realiza la misma función que un contador con la diferencia de que el incremento o decremento es variable en lugar de constante.

```
int acumulador = 0;

acumulador = acumulador + valor;
```

es lo mismo que:

```
acumulador += valor;
```

6.5.8 Banderas

Una bandera, también denominada interruptor o conmutador es una variable que puede tomar uno de dos valores (verdadero o falso) a lo largo de la ejecución del programa y permite comunicar información de una parte a otra del mismo.

```
int primo;

primo = 0;

primo = 1;
```

6.5.9 Expresiones

Una expresión puede ser una constante, una variable, una función o una combinación de constantes, variables y funciones unidas por operadores.

Ejemplo:

```
a = x + y;
w = FALSE;
b = sqrt (25);   /* sqrt es propia de C y calcula la raíz cuadrada de un número */
x = b * w;
```

Ejercicios:

```
int x, n ,i;
i = 2;
n =10;
x = 5;
x++;
x = --n;
i += 2;
x *= n-3;
```

6.6 OPERADORES ARITMÉTICOS:

+	Los operadores pueden ser enteros o reales
-	
*****	
/	
%	Modulo lo resto de una división entera (residuo), sus operandos deben ser enteros

6.6.1 Operadores de relación:

< Menor que
> Mayor que
<= Menor o igual que
>= Mayor o igual que
== Igual que
!= Distinto que

6.6.2 Operadores lógicos:

&& Operador **AND**. Da 1 si ambos operadores son distintos de cero. Si uno de ellos es cero el resultado es el valor lógico 0. Si el primer operando es igual a cero, el segundo operando no es evaluado.

|| Operador **OR**. Es 0 si ambos operandos son 0. si uno de los operandos tiene un valor distinto de 0 el resultado es 1. Si el primer operando es distinto de cero, el segundo operando no es evaluado.

! Operador **NOT**. El resultado es cero si el operando tiene un valor distinto de cero, y 1 en caso contrario.

Ejemplo:

```
int  p = 10, q = 0;
p && q      da como resultado 0.
p  ||  q    da como resultado 1.
! p         da como resultado 0.
```

6.6.3 Operadores de asignación:

++	Incremento
--	Decremento
=	Asignación simple
***=**	Multiplicación mas asignación
/=	División más asignación
%=	Modulo más asignación
+=	Suma más asignación
-=	Resta más asignación
&=	Operación AND sobre bits + asignación
|=	Operación OR sobre bits + asignación
^=	Operación XOR sobre bits + asignación

6.6.4 Operadores unitarios *(Lógicos para el manejo de bits)*:

&	Operación AND a nivel de bits
|	Operación OR a nivel de bits
^	Operación XOR a nivel de bits

6.7 PRIORIDAD Y ORDEN DE EVALUACIÓN:

Categoría del Operador	Operadores	Asociatividad
	( ), [ ], .	Izquierda a derecha
Operadores monarios	-, ++ , -- , !, sizeof(tipo)	Derecha a izquierda
Mult, div, y resto aritméticos	* , /, %	Izquierda a derecha
Suma y resta aritmética	+, -	Izquierda a derecha
Operadores relacionales	<, <=, >, >=	Izquierda a derecha
Operador de igualdad	==, !=	Izquierda a derecha
y lógica	&&	Izquierda a derecha
o lógica	||	Izquierda a derecha
Operador condicional	? :	Derecha a izquierda
	=, *=, /=, %=, +=, -=	Derecha a izquierda

Ejemplo:

1) x = 4 * 5 / 8 + 2;
2) x = 2+4 * 5 / 8;

6.8 CONVERSIONES DE TIPO:

1. Los operandos que intervienen en una determinada operación son convertidos al tipo del operando de precisión más alta.
2. La aritmética de punto flotante se realiza en doble precisión. Cualquier operando de tipo float es convertido al tipo double.
3. en una asignación el valor de la parte derecha es convertido al tipo de valor de la parte izquierda de acuerdo con las siguientes reglas:

   - los caracteres se convierten a enteros.
   - Los enteros se convierten a caracteres preservando los bits de menor peso, esto es desechando los bits de mayor peso en exceso.
   - Los reales son convertidos a enteros truncando la parte fraccionaria. Un double pasa a float redondeando y perdiendo precisión, si el valor double no puede ser representado exactamente como float.

Ejemplo:

1)

```
int car = 10, acum;
float un = 2.5, ac2;
char ca2, ca = 'a';
ca2 = car + car;
ac2 = car + un;
acum = car + un;
```

2)

```
char c;
int a, b;
float w, x, y, z, d;

c = 7;
a = 2;
b = 3;
d = 1.5;
w = a / b + c;
x = a * b + d;
y = b + d / a;
z = c * d + b;
```

6.9 CLASE PRÁCTICA

I) Determine el error en las siguientes sentencias:

```
a, b, c, int;
float a = 3.7;
char k, enum;
w = 5 + a;
a + w * 1 = c;
enum ++;
k == 8;
contador = k / a;
double float 4mitipo;
void contador;
```

II) Declarar 3 variables de tipo entero:

- inicializarlas con el valor de -1
- asígnele a la primera el valor de 0
- asígnele a la segunda el valor de la 1ra * 100
- asígnele a la tercera el valor de la 2da entre 45
- incremente el valor de la primera en 10
- decrementa la segunda en 1
- asígnele a la primera la negación de la segunda

III) Cual será el resultado de las siguientes expresiones:

```
int a, b;
float r, s, b;
char tramp;
a = 20;
b = 5;
r = 8 + a;
tramp = r +b;
tramp *= 10;
```

IV) Supóngase que i es una variable entera cuyo valor es 7, f una variable tipo flota cuyo valor es 5.5, y c una variable de carácter que representa el carácter 'w'. Determine el valor de cada una de las siguientes expresiones lógicas complejas que se muestran a continuación:

Expresión	Interpretación	Valor
(i >= 6) && (c == 'w')	_____	_____
(i >=6) || (c == 119)	_____	_____
(f < 11) && (i > 100)	_____	_____
(c != 'p') || ((i + f) <= 10)	_____	_____

6.10 IMPLEMENTACION DE UN PROGRAMA EN C

Programa en C: Un programa fuente C es una colección de cualquier numero de directrices para el compilador, declaraciones, definiciones, expresiones, sentencias y funciones.

Todo programa en C debe contener una función nombrada *main();* donde el programa comience a ejecutarse. Las llaves { } que incluyen el cuerpo de esta función principal, definen el principio y el final del programa.

6.10.1 Estructura de un Programa en C

/* Directrices para el procesador */
/* Declaraciones de las funciones */
/* Función principal (main) */
/* Llave de apertura del bloque principal */
/* Declaración de las variables locales */
/* Declaraciones e inicializaciones */
/* Sentencias */
/* Llave de cierre del bloque principal */
/* Definiciones de funciones */

6.10.2 Directriz

Las directrices para el pre–procesador son utilizadas para hacer programas fuentes fácil, para cambiar y compilar en diferentes situaciones.

Una directriz va precedida por el símbolo **#** (numeral) e indica una acción específica a ejecutar, pueden aparecer en cualquier parte del archivo fuente, pero solo se aplican al resto del programa.

- Directriz **#define**

Le indica al compilador que toda ocurrencia en el programa del identificador debe de ser sustituida por su valor, es usada para asociar identificadores con palabras claves, constantes, sentencias y expresiones.

Sintaxis:

#define *identificador* *valor*

Ej.:

```
#define   CAJAS      4
#define   PI         3.1416
#define   CADENA     “Desbordamiento de Pila\n”
```

Nota: Las constantes se suelen escribir en mayúsculas, solo se puede definir una constante por fila.

- Directriz **#include**

Sintaxis:

```
#include < archivo.extensión >
#include  "archivo.extensión "
```

Le dice al compilador que incluya el fichero especificado, en el programa fuente. Esto es necesario porque estos ficheros aportan las funciones prototipo de las funciones de la Librería estándar que utilizamos en nuestros programas.

6.10.3 Sentencia

Controlan el flujo u orden de ejecución de un programa en C, una sentencia en C consta de una palabra reservada, de expresiones o de llamadas a funciones, toda sentencia en C termina con un punto y coma (;).

Ej.:

```
i = 0;
a = a+5;
c = a/2;
```

- **Sentencias compuestas:**

Una sentencia compuesta o bloque es una colección de sentencias incluidas entre llaves { }.

Ej.:

```
int x =0;
 do
  {
    x++;
    printf("Valor de x: %d", x);
  } while (x < 5)
```

- **Sentencia de Asignación**

Una sentencia de asignación tiene la forma:

variable *operador de asignación* *expresión*

variable = expresión

Ej.:

```
total = 0;
x = x+1;
y = a + abs(x);
```

Las sentencias de asignación son aritméticas, esto significa que la expresión de la derecha es evaluada para ser asignada el resultado a la variable especificada en la parte izquierda.

Esto no es válido: a + abs(x) = y;

```
/* Las siguientes líneas de código calcula e imprime el producto de dos números */

#include<stdio.h>
#include<conio.h>
#define MENSAJE “Mi primer programa en C \n”

void main(void)

{                       /*llave de apertura del bloque main*/
    int a,b;            /*declaración de variables enteras */
    float res;          /*declaración de variable tipo float */
    clrscr();           /*limpiar pantalla*/
    printf(MENSAJE);
    a=100;              /*sentencia de asignación */
    b=5;                /*sentencia de asignación */
    res=a*b;
    printf("El Resultado es: %f", res);      /*imprime valor */
    getch();      /*pausa*/
}     /* llave de cierre del bloque main */
```

6.10.4 Instrucciones de Entrada y Salida

El lenguaje C va acompañado de una colección de funciones de biblioteca que incluye ciertos números de funciones de entrada / salida: printf, scanf, getchar, putchar, gets y puts, estas funciones permiten la transferencia de información entre la computadora y los dispositivos de entrada y salida estándar (ej, un teclado y un monitor).

1. Funciones de entrada y salida con formato;

- **printf** → Sintaxis: printf(formato, argumento [, argumento 2 …]);

Escribe con formato una serie de caracteres o un valor en el archivo de salida estándar stdout (típicamente un monitor). Esta función permite visualizar los datos introducidos en la computadora.

Formato: Específica cómo va ser la salida, siempre inicia con el carácter **%**, seguido por el símbolo correspondiente al tipo de dato a imprimir.

Argumento: Representa el valor o valores a escribir.

Ejemplo:

```
printf("%d", a);
printf("%f", b);
printf("%s", cad);
printf("%c", car);
printf("%d %f", a,b);
printf("Presione cualquier tecla para continuar \n");  /*Lo que se encuentra dentro de las comillas se imprime literalmente*/
```

%d → Formato de impresión para enteros.
%f → Formato de impresión para reales.
%s → Formato de impresión para cadenas.
%c → Formato de impresión para caracteres.

- **scanf** → Sintaxis: scanf(formato, argumento [, argumento 2 …]);

Lee datos desde la entrada estándar stdin (típicamente un teclado) y las interpreta de acuerdo con el formato indicado, almacenándolos en los argumentos especificados, cada argumento debe ser un apuntador a una variable cuyo tipo se debe corresponder con el tipo especificado en el formato.

Formato: Interpreta cada dato de la entrada, formado por caracteres en blanco, tabuladores, saltos de líneas, caracteres ordinarios y especificaciones de formato, el formato es leído de izquierda a derecha.

Argumento: Es un puntero a una variable que se quiere leer.
Ejemplo:

```
scanf("%d", &a);
scanf("%f", &b);
scanf("%d %d", &a, &x);
.
.
printf("Digite el valor de n: \n");
scanf("%d", &n);
```

Representa la dirección de memoria de la variable **a** donde & (ampersand) es un operador monario, llamado *operador de dirección,* que proporciona la dirección del operando.

- **getchar( )** → Sintaxis: variable=getchar();

Lee un carácter de la entrada estándar (típicamente un teclado)y avanza la posición de lectura al siguiente carácter a leer, getchar devuelve el carácter leído, donde *variable* tuvo que haber sido declarada previamente.

Ej:

```
char c;
. . . . . .
c = getchar (); /*lee un carácter del dispositivo de entrada estándar y lo asigna a c */
```

- **putchar( )** → Sintaxis: putchar(variable);

La función putchar, así como getchar es parte de la biblioteca de E/S estándar. Escribe un carácter en la salida estándar (típicamente un monitor) en la posición actual y avanza a la siguiente posición de escritura. El carácter estará representado por una variable tipo carácter.

Ej:

```
char c;
. . . . . .
putchar (c); /*imprime un carácter al dispositivo de salida estándar y lo asigna a c */
```

- **getch( )**

Lee un carácter del teclado sin visualizarlo. Espera que el usuario introduzca un carácter por el teclado.

- **getche( )**

Lee un caracter desde el teclado visualizándolo en pantalla.

Nota: ambas funciones son utilizadas para lectura de caracteres.

Ej:

```
printf("Presione una tecla para continuar: ");
getch();
```

- **gets( )** → Sintaxis: gets(cadena);

Lee una cadena de caracteres (línea de texto) desde el teclado.

- **puts( )** → Sintaxis: puts(cadena);

Manda a imprimir una cadena de caracteres (línea de texto) en la pantalla.

Ejemplo: El siguiente programa lee dos variables del teclado y luego calcula e imprime su promedio.

```
#include<stdio.h>
#include<conio.h>
void main(void)
  {
        int a,b,c;
        float acum;
        clrscr();
        printf("Comenzando a programar en C \n");
        printf("Digite el valor de a: \n");
        scanf("%d", &a);
        printf("Digite el valor de b: \n");
        scanf("%d", &b);
        c=a+b;
        acum=c/2;
        printf("El valor promedio es: %.2f", acum);
        getch();    }
```

6.11 ESTRUCTURAS DE CONTROL

Introducción:

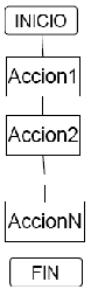
El procesador de una computadora solo es capaz de ejecutar las sentencias de un programa una a una. El orden en el cual el procesador ejecuta estas sentencias se llama *"flujo de control"*, y se dice que, a medida que el programa se va ejecutando, el control va pasando de una sentencia a otra. C incluye diversas instrucciones de control: Estructura secuencial, donde el control pasa de una sentencia a la siguiente; la estructura condicional donde se plantea una bifurcación y el control puede seguir por un camino u otro; y la estructura repetitiva, donde un conjunto de sentencias se repetirán un cierto número de veces.

6.11.1 Estructura secuencial

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.

```
INICIO
  Accion1
  Accion2
    .
    .
  AccionN
FIN
```

En Diagrama de flujo se realiza así:



Ejemplo:
El siguiente programa evalúa la expresión: resultado = x + w /2*w; x,w son variables que se leen desde el teclado.

```
#include<stdio.h>
void main(void)
{
 float resultado,x,w;/*se escriben dos valores separados por la barra espaciadora*/
 printf("ingrese el valor de x , también de w \n");
 scanf("%f %f",&x,&w); /*se leen dos valores */
 resultado=(x+w)/(2*w); /*Se evalúa la expresión y el resultado es asignado a la variable resultado*/
 printf("%f",resultado);/*se imprime el valor de la variable resultado*/

}
```

6.11.2 Estructuras selectivas

Las estructuras selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelan denominar también estructuras de decisión o alternativas.

En las estructuras selectivas se evalúa una condición y en función del resultado la misma se realiza una opción u otra. Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con palabras if, else. Las estructuras selectivas o alternativas pueden ser:

- Simples (if)
- Dobles (if -else)
- Múltiples (switch)

1. **Instrucción IF:** permite a un programa tomar una decisión, basándose en la verdad o falsedad de la instrucción ***if***, si se cumple la condición, es decir si la condición es *verdadera* se ejecuta la (s) instrucción (es) en el cuerpo de la instrucción ***if***. Si la condición no se cumple, es decir la condición es *falsa*, no se ejecutara la instrucción en el cuerpo de la estructura***.***

   Las condiciones en instrucciones ***if*** se forman utilizando *operadores de igualdad* y *de relación:*

Operadores de Relación

<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que

Operadores de Igualdad

==	Igual que
!=	Distinto que

Sintaxis:

```
if (expresión)
        sentencia 1;

[ else
        sentencia 2; ]
```

Expresión: debe ser una expresión numérica relacional o lógica, el resultado que se obtiene al evaluar la expresión es verdadero o falso.

Sentencia1/2: representa una sentencia simple o compuesta cada una de ellas deberá estar separada por un **;**

Forma de evaluación: si el resultado de la expresión es verdadero se ejecutará lo indicado por la sentencia 1, si el resultado de la expresión es falso y la cláusula **else** ha sido omitida, la sentencia 1 se ignora, en cualquier caso la ejecución continúa con la siguiente sentencia ejecutable.

Ejemplo:

```
I)      int x = 10, b = 4;
        if (x > b)
```

```
        printf("El mayor es %d", x);
```

II)
```
int car, x=0;
char resp;
printf("Continuar 1 / Finalizar 2");
car = getch();
if (car == 1)
 {
      resp = 'y';
      printf("Nuevo Juego");
 }
x++;
```

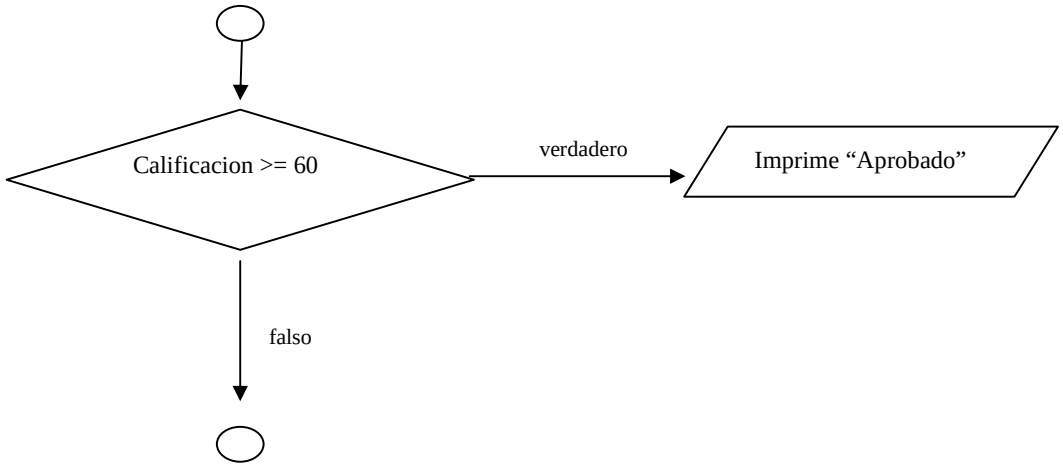
Ejemplo:

Suponga que la calificación del estudiante mínima para aprobar en un examen es 60:

La instrucción **if** se puede escribir en C de la siguiente forma:

```
if(calificacion >= 60)
    printf("Aprobado\n");
```

Diagrama de Flujo:



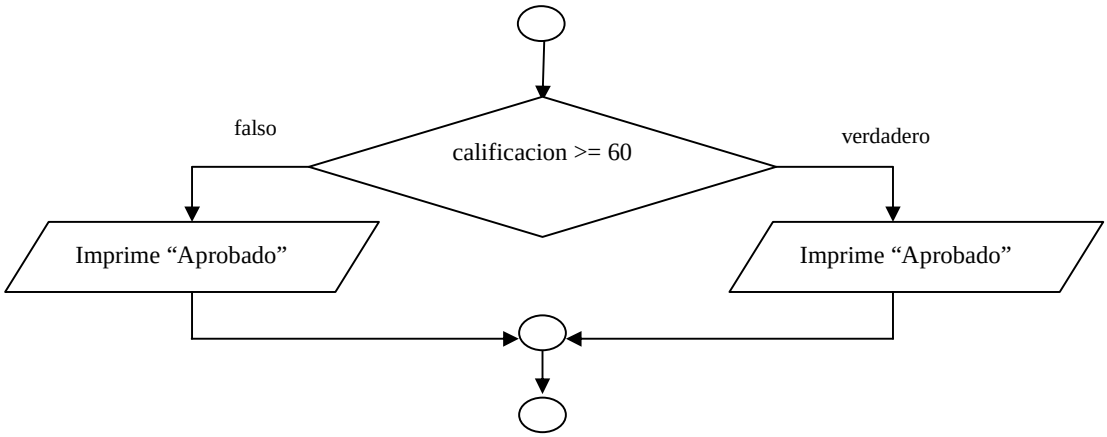
Otro Ejemplo: El siguiente ejemplo utiliza instrucciones ***if*** para comparar dos números introducidos por el usuario. Si se satisface la condición en cualquiera de las instrucciones ***if*** se ejecutará la instrucción **printf** asociada con ese **i*f***

```
/*uso de instrucciones if*/

#include<stdio.h>
void main (void) /*esta función inicia la ejecución del programa*/
{
  int num1;
  int num2;
  clrscr();
  printf("Introduzca dos enteros, y le dire\n");
  printf("las relaciones que satisfacen: ");
  scanf("%d %d",&num1,&num2);

  if(num1 == num2)
    printf("%d es igual que %d\n",num1,num2);

  if(num1 != num2)
    printf("%d no es igual que %d\n",num1,num2);

  if(num1 < num2)
    printf("%d es menor que %d\n",num1,num2);

  if(num1 > num2)
    printf("%d es mayor que %d\n",num1,num2);

  if(num1 < num2)
    printf("%d es menor que %d\n",num1,num2);

  if(num1 > num2)
    printf("%d es mayor que %d\n",num1,num2);

  if(num1 <= num2)
    printf("%d es menor o igual que %d\n",num1,num2);

  if(num1 >= num2)
    printf("%d es mayor o igual que %d\n",num1,num2);
  getch();
}
```

2. **Instrucción IF …..ELSE:** la instrucción **if . . . else** permite al programador especificar que se realizaran acciones diferentes cuando la condición sea verdadera y cuando la condición sea falsa. Por ejemplo la instrucción en pseudocódigo

```
if  calificacion >= 60)

    imprime “Aprobado”
else
    imprime “Reprobado”
```

En Diagrama de Flujo:



3. **Instrucción IF . . . ELSE anidadas:** una sentencia **if** es anidada cuando la sentencia de la rama verdadera o la rama falsa es a su vez un sentencia **if**. Una sentencia **if** anidada se puede utilizar para implementar decisiones con varias alternativas o multi-alternativas.

   Ejemplo: **Calcular el mayor de tres números:**

```
void main(void)
{
 int a,b,c,mayor;
 clrscr();
 printf("Introduzca tres enteros: ");
 scanf("%d %d %d ",&a,&b,&c);

 if(a > b)
    if(a > c)
       mayor = a;
    else mayor = c;

 else
    if(b > c)
       mayor = b;
    else
        mayor = c;

 printf("El mayor es %d  ",mayor);
 getch();

}
```

4. **Operador Ternario?:** Los operadores incluidos en esta clase tiene tres operandos. Estos operadores suelen ser infijos (es decir, es frecuente que sean escritos entre los dos operandos sobre los que actúan).

Sintaxis:

Operando 1 **?** Operando 2 **:** Operando 3 ;

Ejemplo:

mayor = (a >b) ? a : b

Nota: El operador condicional (a > b) evalúa la condición que se le pasa como primer operando y dependiendo de que se cumpla o no devuelve el valor de uno de sus operandos. Si la condición es cierta devuelve el valor del primer operando *a* en caso contrario devuelve el segundo operando *b*.

5. Operador switch

Cuando el número de posibles alternativas que hay que considerar es muy elevado, puede resultar muy tedioso y complejo escribir **if**´s anidados. El lenguaje C ofrece la estructura **switch** la cual permite ejecutar una de varias acciones en función del valor de la expresión.
La sentencia **switch** es especialmente útil cuando la selección se basa en el valor de una variable simple o de una expresión simple denominada *expresión de control o selector.* El valor de esta expresión puede ser de tipo *int* o *char*, pero no de tipo double

Sintaxis:

```
switch (expresión)
 {
  [declaraciones]
   case CONSTANTE 1:
        [sentencia 1;]

   case CONSTANTE 2:
        [sentencia 2;]
    .
    .
   .
   default:
[sentencia n;]

}
```

Expresión: es una constante entera, una constante de caracteres o una expresión constante.

Sentencia1,2,n: pueden ser expresiones simples o compuestas.

Forma de Evaluación: la sentencia *switch* evalúa la expresión entre paréntesis ( ) y compara su valor con las constantes de cada *case*, en el orden en la que aparecen, si el resultado de esta evaluación coincide con alguno de los valores incluidos en las etiquetas *case* se ejecutan las sentencias que aparecen a continuación de los dos puntos y todas las que aparecen tras ellas o hasta una sentencia que transfiera el control fuera del cuerpo (por ej.: break); si el valor no coincide se pasa a comparar con la siguiente etiqueta

case, si no existe un valor igual al valor de la expresión entonces se ejecutan las sentencias a continuación del default si este ha sido especificado.

6. Sentencia break

Finaliza la ejecución de una sentencia do, while, for, switch en la que aparece.

Ejemplo1: el siguiente programa lee dos enteros (int) y un operador (char) el programa evaluará que tipo de operador es el elegido y realizará la acción (sumar, restar, multiplicar, dividir).

```
#include<stdio.h>
void main(void)
{
   int a,b,resul;
   char operador;
   printf("Digite dos operadores:");
   scanf("%d %d ",&a,&b);
   printf("Digite el operador (+, -, *, /)");
   scanf("%c ",&operador);
switch(operador)
  {
      case '+':
           resul=a+b;
           printf("%d",resul);
           break;
     case '-':
           resul =a-b;
          printf("%d", resul);
           break;
    case '*':
           resul =a*b;
           printf("%d", resul);
           break;
    case '/':
           resul =a/b;
           printf("%d", resul);
           break;
   default:
         printf("\nOperador no valido");
  } /*Cierre del switch*/
} /*cierre del main*/
```

Ejemplo 2: Calcular el importe a pagar por varios vehículos al circular por una autopista. El vehículo puede ser bicicleta, moto, coche o un camión. El importe se calculara según los siguientes datos:

1.) Un importe fijo de C$5.00 para bicicletas
2.) Las motos y los coches pagaran C$5.00 por km.
3.) Los camiones pagaran C$5.00 por km más C$10.00 por toneladas. */

```
#include<stdio.h>
#include<conio.h>
void main()
{       int opc;
        float imp, km, tan, ton;
        clrscr();
        gotoxy(38,5); printf("VEHICULOS");
        gotoxy(29,8); printf("1.Bicicletas");
        gotoxy(29,10); printf("2.Motos y Coches");
        gotoxy(29,12); printf("3.Camión");
        gotoxy(29,14); printf("4.Salir");
        gotoxy(29,17); printf("Elija su Opción ==   ");
        scanf("%d", &opc);
        clrscr();
          switch(opc)
          {
                case 1: imp=5;
                        printf("Importe=%.2f", imp);
                        break;
                case 2: printf("Introducir Cantidad de Kilómetros:  ");
                        scanf("%f", &km);
                        imp=5*km;
                        printf("Importe=%.2f", imp);
                        break;
                case 3: printf("Introducir Cantidad de Kilómetros y Toneladas:   ");
                        scanf("%f %f", &km, &ton);
                        imp=(5*km)+(10*ton);
                        printf("Importe=%.2f", imp);
                        break;
                case 4: break;
                default:
                        printf("Opcion incorrecta");
                        break;
          }  /*cierre del switch*/
      getch();
}/*cierre del main*/
```

Tarea:

1. Una distribuidor de motocicletas tiene una promoción de fin de semana que consiste en lo siguiente: las motos marca HONDA tienen un descuento del 5%, la marca YAMAHA del 8% y las SUZUKI el 10%, las otras marcas el 2%. Calcular el valor del descuento y el valor a pagar por la motocicleta.

6.11.3 Estructuras repetitivas o cíclicas

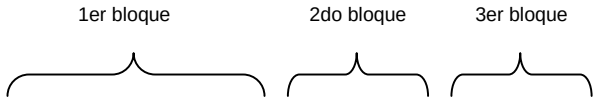
Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan ciclos o bucles y se denomina **iteración** al hecho de repetir la ejecución de una secuencia de acciones. Entre las estructuras repetitivas se encuentran:

- desde (for)
- mientras (while)
- hacer - mientras (do - while)

1. ciclo for:

Ejecuta una sentencia simple o compuesta repetidamente un nº de veces conocidas

Sintaxis:




```
for ([v1 = ve1 [,v2 = ve2 . . .]] ; [condición] ;[incremento])
    sentencia ;
```

1er bloque: Representan las variables que serán inicializadas.

Condición (2do bloque): es una expresión booleana (operadores unidos por operadores relacionales y / o lógicos).

Incremento (3er bloque): es una expresión cuyo valor evoluciona en el sentido de que se dé la condición para finalizar la ejecución de la sentencia for.

Sentencia: cualquier sentencia simple o compuesta.

Forma de ejecución del for:

1. Se inicializan las variables
2. Se evalúa la expresión (Condición)

   2.1 si el resultado es distinto de cero (**verdadero**), se ejecuta la sentencia, se evalúa la expresión que da lugar a la progresión de la condición y se vuelve al punto 2.

   2.2 si el resultado de 2 es cero (**falso**), la ejecución de la sentencia *for* se da por finalizada y se continúa en la siguiente sentencia del programa.

Ejemplos:

```
/* Este ej. Imprime los números del 1 al 100*/
for (i =1; i<=100; i++)
    printf("%d", i);
```

```
/* Imprime los múltiplos de 7 que hay entre 7 y 112*/
for (k =7; k<=112; k+=7)
    printf("%d", k);

/* Imprime los múltiplos de 7 que hay entre 7 y 112*/
for (k =7; k<=112; k+=7)
    printf("%d", k);

/* Imprime los números del 9 al 1*/
for (a =9; a>=1; a--)
printf("%d", a);

/* Bucle infinito*/
 int n=100;
 for(; ;)
   printf("%d\n",n);
```

La terminación de este bucle o ciclo infinito se realizará con **break** o con **ctrl+c**.

2. ciclo while

Ejecuta una serie de instrucciones mientras una condición permanezca como verdadera en el momento en que la condición se convierte en falsa el ciclo termina.

Sintaxis:

```
. ..
while(condición)
{
grupo cierto de instrucciones;
instrucción(es) para salir del ciclo;
};
```

condición: cualquier expresión numérica, relacional o lógica.
instrucción: cualquier sentencia simple o compuesta.

Ejemplo:

```
int n = 0;
while (n < o)
{
    printf("Entradas para n: \n");
    scanf("%d",&n);
}
```

3. ciclo do - while

Funciona similar a la instrucción while, la diferencia básica con el ciclo while es que la prueba de condición es hecha al finalizar el ciclo, es decir las instrucciones se ejecutan *cuando al menos una vez* porque primero ejecuta las instrucciones y al final evalúa la condición.

Sintaxis:
do {
grupo cierto de instrucción(es);
instrucción(es) de rompimiento de ciclo;
} while (condición);

Ejemplo 1:

```
#include <stdio.h>
void main(void)
{   /* Visualizar los números del 0 al 9.*/
int digito=0;
do
printf("%d ",digito++); /*valor actual de digito luego establece un post incremento*/
while (digito<=9);
}
```

Visualizará: 0 1 2 3 4 5 6 7 8 9

EJEMPLOS:

1. Haga un programa en C que lea e imprima **n** números hasta encontrar el primer número negativo.

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
      int num;
      clrscr();
      printf("Digite el numero:");
      scanf("%d",&num);
        while (num>=0)
         {
            printf("Digite otro numero:");
            scanf("%d",&num);
          }/*fin del while*/

     printf("El ultimo numero fue negativo, Adiós !!");
     getch();
}/*fin del main */
```

2. Escriba un programa en C que lea 4 operadores aritméticos (+, -, *, /) y dos enteros, el programa deberá realizar la operación especificada por cada operador sobre los dos enteros leídos, si el operador leído no es ninguno de los especificados se enviará un mensaje a pantalla indicando que el operador es desconocido.

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
      int n1,n2,resul;
       char oper;
       clrscr();
       printf("Digite el primer número:");
       scanf("%d",&n1);
       printf("Digite el segundo numero:");
       scanf("%d",&n2);
       printf("Digite el Operador:");
       oper=getche();
switch(oper)
 {
      case '+':
         resul=n1+n2;
          printf("\nResultado: %d",resul);
          break;

      case '-':
           resul=n1-n2;
           printf("\nResultado: %d",resul);
           break;

     case '*':
         resul=n1*n2;
         printf("\nResultado: %d",resul);
         break;

      case '/':
        resul=n1/n2;
        printf("\nResultado: %d",resul);
        break;

  default:
      printf("\n Operador no valido");

      } /*fin del switch*/
      getch();

} /*fin del main*/
```

3. Defina un programa en C que lea las notas correspondientes a las tres asignaturas de un grupo de 50 alumnos, que calcule e imprima el promedio más alto obtenido por el mejor alumno del grupo.

```
#include<stdio.h>
#include<conio.h>
void main(void)
```

```
{
      int not1,not2,not3,cont,mejor;
      float prom,prom_mayor=0;
      clrscr();
       for(cont=1;cont<=2;cont++)
              {
                     printf("\nAlumno Nº %d",cont);
                     printf("\nNota 1: ");
                     scanf("%d",&not1);
                     printf("\nNota 2: ");
                     scanf("%d",&not2);
                     printf("\nNota 3: ");
                     scanf("%d",&not3);
                     prom=(not1+not2+not3)/3;

                     if(prom>prom_mayor) /*evalúa que promedio es el mayor*/
                      prom_mayor=prom;
                       mejor=cont;

              }/*fin del for*/
      printf("\n\nEl Promedio mayor fue: %.2f del alumno N° %d",prom_mayor,mejor);

     getch();
}/*fin del main*/
```

4. Escriba un programa en C que imprima los números: 0,-2, -4, -6, …….-100

```
#include<stdio.h>
#include<conio.h>
void main(void)

{
      int num;
      clrscr();
        for(num=0;num>=-100;num--)
          printf("\t%d",num);

     getch();
}
```

5. Construir un programa en C que escriba los nombres de los días de la semana, en función de una variable **día** que será leída a través del teclado, en donde dicha variable puede tomar los siguientes valores:

```
switch (dia)
{
   case 1:
   printf(“Lunes\n”);
   break;
   case 2:
```

```
        printf("Martes\n");
          break;
          ……
          default:
            printf("No corresponde a ningún día de la Semana\n");  }
```

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
 int dia;
 clrscr();
 printf("Digite el día Por favor:");
 scanf("%d",&dia);

   switch (dia)
      {
       case 1:
                  printf("\nLunes");
                  break;
       case 2:
                  printf("\nMartes\n");
                  break;
       case 3:
                  printf("\nMiercoles");
                  break;
       case 4:
                  printf("\nJueves");
                  break;
       case 5:
                  printf("\nViernes");
                  break;
       case 6:
                 printf("\nSabado");
                 break;
       case 7:
                 printf("\nDomingo");
                 break;

      defaulf:
                 printf("No corresponde a ningún día de la Semana\n");

      } /*CIERRE DEL SWITCH*/
   getch();

} /*CIERRE DEL MAIN*/
```

6. Defina un programa que sume los cuadrados de n = 1, 2, 3, ….., mientras n sea menor o igual a 30 y la suma sea menor que 20,000. El programa imprime el valor de n, n^2 y la suma total.

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
       int n=1,cuadrado,sum_cuad=0;
       clrscr();
         while((n<=30) && (sum_cuad<20000))
           {
                cuadrado=n*n;
                sum_cuad=sum_cuad+cuadrado;
                printf("\nNumero: %d, cuadrado: %d",n,cuadrado);
                n++;
           }/*fin del while*/
                 printf("\n\tSuma total de los cuadrados: %d",sum_cuad);
      getch();
}/*fin del main */
```

7. Hacer un programa que lea la edad y el sexo de 25 trabajadores, el programa imprime la cantidad de hombres y de mujeres.

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
  int edad, ch=0,cm=0,i;
  char sexo;
  clrscr();

 for(i=0;i<5;i++)
    {
      printf("\nDigite la edad:");
      scanf("%d",&edad);
      printf("\nDigite el Sexo M o F:");
      scanf("%c",&sexo);
        if(se=='M'|| se=='m')
             ch++;
        else if(se=='F'|| se=='f')
         cm++;

  }/*fin del for*/
   printf("\n Cantidad de hombres: %d",ch);
   printf("\n\nCantidad de mujeres: %d",cm);
   getch();
}/*fin del main*/
```

8. Sumar los números n = 3, 6, 9 …….., hasta que la suma sea mayor que 1000. Imprimir los valores finales de n y suma.

```
#include<stdio.h>
#include<conio.h>
void main(void)
 {
      int a=3,suma=0;


       clrscr();
        while(suma<100)
        {
                printf("\nValor actual de suma: %d",suma);
                printf("\nValor de a: %d",a);
                suma=suma+a;
                a+=3;
          }/*fin del while*/
      getch();
}/*fin del main*/
```

6.12 CLASE PRÁCTICA

Programas secuenciales:

1. Calcular el número de pulsaciones que una persona debe tener por cada 10 segundos de ejercicio, si la formula es:

   Numero pulsaciones = (220 - edad)/10

2. Una persona desea saber cuál es su saldo bancario si el banco le paga el 2% por concepto de intereses.

3. Determinar el área de un triangulo. Sabiendo que la formula está dada por $A = \frac{b \cdot h}{2}$

4. Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.

5. Determine la edad de una persona dado el año actual y el año en que esta nace.

6. Determine el porcentaje de hombres y mujeres que hay en un curso. La suma de ambas cantidades representan el 100%.

   Nota: Para determinar el % de hombres y mujeres en el curso utilizar la formula ***h *100/h+m*** lo mismo se hará con las mujeres.

7. Calcular el nuevo salario de un obrero si obtuvo un incremento del 25% sobre su salario anterior.

8. En un hospital existen tres áreas: Ginecología, Pediatría, Traumatología. El presupuesto anual del hospital se reparte conforme a la sig. Tabla:

Área	**Porcentaje del presupuesto**
Ginecología	40%
Traumatología	30%
Pediatría	30%

   Obtener la cantidad de dinero que recibirá cada área, para cualquier monto presupuestal.

Programas selectivos:

1. Determinar si un número es mayor o menor que 0.

2. Un estudiante desea saber cuál es su nota final en *Introducción a la ingeniería en computación.* En el semestre hay tres prácticas de laboratorios y dos parciales que sumaran un 100%, sabiendo que una nota mayor o igual a 60 pasa la materia. Escriba un programa indicando si dicho alumno aprueba o no la materia y con cuanto la aprueba.

3. Determinar si una letra es mayúscula o minúscula teniendo en cuenta que si un número es mayor o igual que 97 y menor o igual que 122 la letra será minúscula y si es mayor o igual que 65 y menor o igual que 90 será mayúscula. (Véase en anexo código ASCII).

4. Determine el valor absoluto de un número.

5. Calcular el número de pulsaciones que debe tener una persona por cada 10 segundos de ejercicio aeróbico; la fórmula que se aplica cuando el sexo es femenino es:

   Número pulsaciones = (220 - edad)/10

   Y si el sexo es masculino:

   Número pulsaciones = (210 - edad)/10

Programas repetitivos:

6. Escriba un programa que imprima en pantalla un triangulo formado con los números del 1 al 10. Se imprimirán 10 líneas como la siguiente:

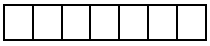
   1
   12
   123
   1234
   .
   .
   12345678910

Capítulo 7

ARREGLOS

7.1 DEFINICION

Los arreglos son también conocidos como *lista* o *tabla.* Es una estructura homogénea compuesta por varios elementos todos del mismo tipo y almacenados consecutivamente en memoria, es decir un arreglo (array) es un conjunto de variables del mismo tipo, que tienen el mismo nombre y se diferencian en el índice.



Esquema de representación

7.2 ARREGLOS UNIDIMENSIONALES

A los arreglos de una dimensión se les llama también ***listas***. Cada componente puede ser accedido directamente por el nombre de la variable arreglo seguido de un índice encerrado entre corchete **[ ]**. Cada *índice* debe ser expresado como un entero no negativo.

La declaración de un arreglo de una dimensión, se hace de la forma:

*tipo identificador **[**tamaño**];***

tipo: indica el tipo de datos a almacenar (void no, puede ser otro tipo).
identificador: variable que da nombre al arreglo.
tamaño: constante que representa la cantidad de elementos.

El siguiente ejemplo representa un arreglo de 10 elementos llamado **x**

- **int** *x[10];*

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	x[9]
10	15	20	25	1	12	5	30	8	22

Este ejemplo declara un arreglo denominado *x* con 10 elementos (del 0 al 9), cada uno de ellos de tipo ***int***.

- **char** *nom[10];*

nom[0]	nom[1]	nom[2]	nom[3]	nom[4]	nom[5]	nom[6]	nom[7]	nom[8]	nom[9]
j	a	c	q	u	e	l	i	n	e

Este ejemplo declara un arreglo denominado *nom* con 10 elementos (del 0 al 9), cada uno de ellos de tipo ***char***.

Nótese que en Los ejemplos los **índices son enteros consecutivos** y que el primer subíndice vale **0**.

Nota: en C, todos los arreglos usan cero como índice para el primer elemento.

7.2.1 Inicialización de arreglos unidimensionales

Al igual que las variables (int n = 30;) a los arreglos podemos inicializarlos al declararlos:

- Inicialización Explícita:

  int lista [13] = { 15, 18, 20, 23, 22, 24, 22, 25, 26, 25, 24, 22,21};

- Podemos omitir el tamaño del arreglo en la inicialización explícita:

  int lista_2[ ]= {1,8,3,2};

Practica 1:

- ✓ Declare e inicialice un arreglo tipo entero llamado **num** con una dimensión de 20 elementos (inicialización explícita).
- ✓ Declare una variable suma (tipo entero), resta (tipo entero), mult (tipo entero), div (tipo float).
- ✓ Asigne a la variable suma el resultado de la suma del elemento 3 y 8 del arreglo **num**.
- ✓ Asigne a la variable resta el resultado de la resta del elemento 18 y 0 del arreglo **num**.
- ✓ Asigne a la variable mult el resultado de la multiplicación del elemento 15 y 10 del arreglo **num**.
- ✓ Asigne a la variable div el resultado de la división de la variable suma y el elemento 7 del arreglo **num**.
- ✓ Imprima los valores del arreglo utilizando el ciclo **for.**
- ✓ Imprima los valores de las variables suma, resta, mult, div.

7.2.2 Ejemplos de arreglos unidimensionales

Programa 1:

Realizar un programa que lea las notas correspondientes a 50 alumnos de un determinado curso, las almacene en un arreglo y de cómo resultado la nota media correspondiente del curso.

```
/********************* Nota media de 50 alumnos ************************/

#include<stdio.h>
#include<conio.h>

  void main(void)
  {
     int alumnos[50],cont,suma=0;
     float not_media;
```

```
    clrscr();              /*limpiar pantalla*/

     for(cont=0;cont<50;cont++)

      {
        printf("Digite la nota del alumno numero %d:",cont+1);
        scanf("%d",&alumnos[cont]);
       cont_suma+=alumnos[cont];
      }/*fin del for*/

    not_media=cont_suma/50;
    printf("\n\nLa nota media es %.2f",not_media);
    getch();
}/*fin del main*/
```

Programa 2:

Realizar un programa que asigne datos a una matriz de una dimensión, el programa escribirá el numero mayor que se ha escrito.

```
    /******************** Número mayor en un arreglo ************************/

   #include<stdio.h>
   #include<conio.h>
   #define FILA 10
   void main(void)

   {
        int matriz[FILA],fil,f,num_mayor=0;
        clrscr();
        printf("Digite la cantidad de filas de la matriz:");
        scanf("%d",&fil);

        for(f=0;f<fil;f++)
                {
                  scanf("%d",&matriz[f]);

                  if(matriz[f]>=num_mayor)
                          num_mayor=matriz[f];
                 } /*Cierre del ciclo*/

                 printf("El numero mayor digitado fue: %d",num_mayor);
                 getch();
}/*fin del main*/
```

Programa 3:

Escribir un programa en C que permita calcular el cuadrado de los cien primeros números enteros y a continuación escribir una tabla que contenga dichos cien números cuadrados.

```
 /***** Cuadrados de los primeros cien números ****/

#include<stdio.h>
#include<conio.h>
#define TAM 100
void main(void)
{
      int cuadrados[TAM],i,num,cuad;
      clrscr();
     printf("Digite los 100 (cien) números:");

      for(i=0;i<TAM;i++)
          {
            scanf("%d",&num);   /*leo el numero*/
            cuad=num*num;     /*se calcula el cuadrado del numero*/
            cuadrados[i]=cuad; /*almaceno en el arreglo el cuadrado del numero*/

          } /*Cierre del ciclo*/

/*********CICLO QUE CONTROLA LA IMPRESIÓN DEL ARREGLO**********/

     for(i=0;i<TAM;i++)
          {
             printf("%d\n",cuadrados[i]); /*Imprime el contenido del arreglo*/

          } /*Cierre del ciclo*/

      getch();
}/*fin del main*/
```

Programa 4:

Escriba un programa que lea 10 enteros los cuales se almacenaran en un arreglo lineal. El programa sumara los números pares que fueron digitados, se imprimirá en pantalla la cantidad de números pares su suma y la cantidad de impares.

```
#include<stdio.h>
#include<conio.h>
#define T 5
void main(void)
{
  int a[T],par=0,impar=0,sumpar=0,i;
  clrscr();
```

```
 printf("\nHola Bienvenido al programa");
 printf("\n\n\n\n\nDigite los elementos del arreglo:\n");

 for(i=0;i<T;i++)
 {
          printf("\na[%d] = ",i);
          scanf("%d",&a[i]);
          if(a[i]%2==0)
          {
            par++;
            sumpar+=a[i];
          }/*cierre del if*/
          else impar++;

 } /*Cierre del for*/
 printf("\n\nSe digitaron %d números pares\n y %d números impares\n\n \tLa suma de los pares es:
%d",par,impar,sumpar);
 getch();

}/*fin del main*/
```

Programa 5:

Defina un programa que pida al usuario ingrese valores a dos arreglos unidimensionales, el programa sumara elemento a elemento y el resultado se almacenara en un tercer arreglo el cual se imprimirá en pantalla.

```
#include<stdio.h>
#include<conio.h>
void main (void)
{
  int tam,array1[10],array2[10],array3[10],i;
  clrscr();
  printf("Cuantos elementos para el arreglo??: ");
  scanf("%d",&tam);
  for(i=0;i<tam;i++)
   {
       printf("\nElemento %d del Arreglo 1: ",i+1);
       scanf("%d",&array1[i]);
       printf("Elemento %d del Arreglo 2: ",i+1);
       scanf("%d",&array2[i]);
       printf("\n");
   }/*cierre del for*/
   textcolor(YELLOW);
   cprintf("\n\nRESULTADO DE LA SUMA\n\n");
     for(i=0;i<tam;i++)
   {

       array3[i]=array1[i]+array2[i];
       textcolor(LIGHTBLUE); /*se le da color al texto en azul claro*/
```

```
        cprintf("%d ",array3[i]);

      }/*cierre del for*/
      getch();
}/*cierre del main*/
```

Programa 6:

Escriba un programa que lea un arreglo de N elementos. El programa imprimirá el elemento mayor que se ha digitado.

```
#include<stdio.h>
#include<conio.h>
#define N 10
void main(void)
{
  int lista[N],i=0,mayor,elem;
  clrscr();
  printf("Digite la cantidad de elementos que tendrá el arreglo: ");
  scanf("%d",&elem);
  printf("lista[0]: ");
      scanf("%d",&lista[0]);
      mayor=lista[0];

  for(i=1;i<elem;i++)
  {
      printf("lista[%d]:",i);
      scanf("%d",&lista[i]);
      /*mayor=lista[0];*/

      if(lista[i]>=mayor)
        mayor=lista[i];
  }/*cierre del for*/
  printf("El mayor de los números es: %d",mayor);
  getch();
  }/*cierre del main*/
```

Programa 7:

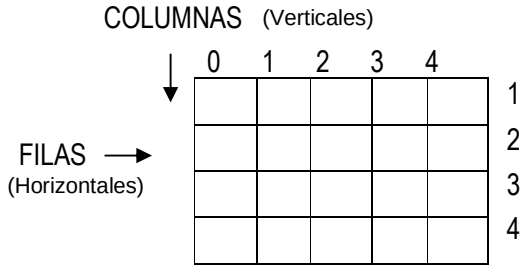
Que lea e imprima una serie de números enteros hasta q se digite el primer valor nulo (o sea hasta q se digite un 0). El programa cuenta la cantidad de números positivos y negativos e imprime el valor final de los contadores

```
#include <conio.h>
#include <stdio.h>

void main(void)
{
 int num,cpos=0,cneg=0;
```

```
       clrscr();

      do
       {
         printf("Escriba un numero: ");
         scanf("%d",&num);

        if(num>0)
          cpos++;

        else if(num<0)
          cneg++;

      }while(num!=0); /*cierre del do-while*/

      printf("\nSe digitaron \n\t%d números positivos ",cpos);
      printf("\n\t%d números negativos ",cneg);
       getch();
}/*fin del main*/
```

7.3 ARREGLOS BIDIMENSIONALES

Son arreglos de dos dimensiones, considerados también como una MATRIZ (términos matemáticos) o TABLA (términos financieros), por lo cual se manejan dos índices; el primer índice se refiere a las filas o renglón y el segundo a las columnas; gráficamente lo podemos entender así:



Sintaxis:

tipo identificador **[**filas**][**cols**]**;

Ejemplo:

- int tabla[4][2]; /*declara una tabla de 8 elementos (4 x 2) con 4 filas y 2 columnas*/
- int tabla2[5][5]; /*declara una tabla de 25 elementos (5 x 5) con 5 filas y 5 columnas*/
- int a[2][25]; /*declara una tabla de 50 elementos (2 x 25) con 2 filas y 25 columnas*/

7.3.1 Inicialización explícita en arreglos bidimensionales

- int tabla2[4][3] = {17,18,3,22,15,0,4,-8,3,5,2,1};

Nota: se puede omitir las filas pero no las columnas (es obligatorio indicar las columnas):

- int a[ ][ ]; → ERROR
- int a[ ][3]; → ERROR
- int a[ ][3] = {2,4,5,2,6,0}; → ESTO ES CORRECTO

7.3.2 Recorrido de los elementos de un arreglo bidimensional

El recorrido de los elementos de una matriz se realiza utilizando estructuras repetitivas, con las que se manejan los subíndices del arreglo (filas, columnas). En el caso de una matriz se necesita dos estructuras repetitivas anidadas, una que controla la fila y otra que controla las columnas.

- *Recorrido por filas*

Pseudocódigo:

```
desde i ←1 hasta fila hacer
   desde j ←1 hasta columna hacer
        escribir (matriz[i][j])
   fin_desde
fin_desde
```

- *Recorrido por columnas*

Pseudocódigo:

```
desde i ←1 hasta columna hacer
   desde j ←1 hasta fila hacer
        escribir (matriz[i][j])
   fin_desde
fin_desde
```

Ejemplos:

1) Rellenar una matriz de identidad de 4 x 4 elementos

Una matriz de identidad es aquella en la que la diagonal principal está llena de unos y el resto de los elementos son ceros. Como siempre que se quiere trabajar con matrices bidimensionales se debe de utilizar dos bucles anidados que en este caso serán de 1 hasta 4. Para ingresar los datos en la matriz identidad se tiene que comprobar:

1. que los índices de las filas y calumas son iguales
2. si esto es verdad ingresar en ese elemento el valor de 1
3. si es falso ingresar en ese elemento el valor de 0

```
algoritmo identidad
var
    array [1..4, 1..4] de enteros : matriz
    entero : i, j
inicio
     desde  i ←1 hasta 4 hacer
        desde  j ←1 hasta 4 hacer
         si i = j   entonces
           matriz[i,j] ← 1
         si_no
           matriz[i,j] ← 0
         fin_si
        fin_desde
     fin_desde
fin
```

Programa en C

```
#include<stdio.h>
#include<conio.h>
#define L 4
void main(void)
{
  int matriz[L][L],fila,col;
  clrscr();
  for(fila=0;fila<L;fila++)
            for(col=0;col<L;col++)
             if(fila==col)
               matriz[fila][col]=1;
             else
               matriz[fila][col]=0;
  /*impresion del arreglo*/
  gotoxy(25,5);
  printf("MATRIZ IDENTIDAD\n\n");
  for(fila=0;fila<L;fila++)
  {
            printf("\n\n\t\t");
            for(col=0;col<L;col++)
            {
              if(fila==col)
              {
              textcolor(YELLOW);
              cprintf("%d",matriz[fila][col]);
              }/*fin del if*/
              else
             {
              textcolor(CYAN);
              cprintf("%d",matriz[fila][col]);
              }/*fin del else*/
              printf("\t");
             }/*fin del for q controla las col*/
            }/*fin del for q controla las filas*/
            printf("\n\n\n");
            system("pause");
}/*fin del main*/
```

2) Similar al anterior con la diferencia de ser de 20 filas por 30 columnas

```
int tabla[20][30],i,j;
for(i=0;i<20;i++)       /*Controla las filas*/
 {
   for(j=0;j<30;j++)   /*Controla las columnas*/
   scanf("%d",&tabla[i][j]);  /*Lectura de los elementos de la tabla*/
 }
```

3) Realizar un programa que asigne datos a una matriz t de dos dimensiones y a continuación, escriba las sumas correspondientes a las filas de la matriz.

```
/********************* Crear un arreglo bidimensional ************************/

#include<stdio.h>
#include<conio.h>
#define FILAS 5
#define COLS 4
void main(void)
{
      int t[FILAS][COLS],sumfilas[FILAS]={0};
      int filas,cols,f,c;
      clrscr();
      printf("\nNumero de filas de la matriz:");
      scanf("%d",&filas);
      printf("Numero de columnas de la matriz:");
      scanf("%d",&cols);

      /*Entradas de datos*/

       for(f=0;f<filas;f++)
          for(c=0;c<cols;c++)
            {
                 printf("\n Datos para la fila %d columna %d:",f,c);
                 scanf("%d",&t[f][c]);
                 sumfilas[f]+=t[f][c];

            } /*Cierre del segundo for*/

      /*Impresión del arreglo*/

       for(f=0;f<filas;f++)
         printf("\n\nSuma de la fila %d es %d\n",f,sumfilas[f]);

      getch();

} /*Cierre del main*/
```

7.4 FUNCIONES PARA MANIPULAR CADENAS DE CARACTERES

Cadenas

En C, una cadena es un arreglo de caracteres en el cual todos sus elementos son de tipo *char*, en donde la terminación de la cadena se debe indicar con nulo. Un nulo se especifica como '\0'. Por lo anterior, cuando se declare un arreglo de caracteres se debe considerar un carácter adicional a la cadena más larga que se vaya a guardar.

Se pueden hacer también inicializaciones de arreglos de caracteres en donde automáticamente C asigna el caracter nulo al final de la cadena, de la siguiente forma:

char cad[**tam**]=**"cadena"**;

Un array de caracteres puede ser inicializado asignándole un literal:

Por ejemplo:

char cadena[ ] = “computacion”;

Este ejemplo inicializa el array de caracteres *cadena* con 11 elementos (cadena[0] a cadena[10]) el undécimo elemento es el carácter nulo (\ **0**), con el cual **C** finaliza todas las cadenas de caracteres.

Otro ejemplo, el siguiente fragmento inicializa cadena con ``hola'':

char cadena[5]="hola";

El código anterior es equivalente a:

char cadena[5]={'h','o','l','a','\0'};

Importante: si se especifica el tamaño del array de caracteres y la cadena asignada es más larga que el tamaño especificado, se obtiene un **error** en el momento de la compilación.

Por ejemplo:

char cadena[3] = “abcd”;

Este ejemplo daría lugar a un mensaje de error, indicándonos que hemos excedido los límites del arreglo. Si la cadena asignada es más corta que el tamaño del arreglo de caracteres, el resto de los elementos del arreglo son inicializados a valor nulo (\ **0**).

- Antes de leer una cadena de caracteres, debe declararse el arreglo de tipo *char* que la va a contener.
-

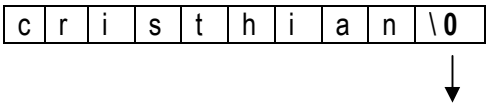
- La dimensión de este arreglo debe de corresponderse con el número de caracteres máximo que puede contener la cadena, mas uno correspondiente al carácter nulo de terminación.

Ejemplo: si queremos leer un nombre de 9 caracteres de longitud máxima, debemos declarar el array de la forma siguiente:

char *nombre*[10];

Para leer esta cadena de caracteres, podemos emplear la función *scanf ( )*. En este caso, la variable *nombre*, no necesita ser precedida por el operador **&**, porque *nombre* es una dirección, la dirección de comienzo del arreglo. Si se lee elemento a elemento, el operador **&** es necesario:

scanf ("%s", nombre);



Algunas funciones para la manipulación de cadenas:

Las declaraciones de las funciones para manipular cadenas de caracteres están contenidas en el fichero **string.h**. La sentencia:

#include<string.h>
Se requiere solamente para declaraciones de función.

- **strcat** (cadena1, cadena2);

  Esta función añade la **cadena2** a la **cadena1**, termina la cadena resultante con el carácter nulo (\0) y devuelve un apuntador a la **cadena1**, en donde cadena1 cadena2 deben ser arreglos de tipo char.

  char cadena1[15], cadena2[8];

  cadena1: | f | i | n | a | l | \0 |

  cadena2: | f | e | l | i | z | \0 |

  Añade cadena2 a cadena1 incluso el carácter nulo, el tamaño de cadena1 tiene que ser suficientemente grande para que pueda alcanzar la concatenación.

cadena1:

f	i	n	a	l	f	e	l	i	z	\ **0**

- **strcpy** (cadena1,cadena2)

Copia la **cadena2** incluyendo el carácter de terminación nulo en la **cadena1** y devuelve un puntero a **cadena1**.

char cadena1[ ]="final", cadena2[ ]="feliz";

cadena1:

f	e	l	i	z	\ **0**

→ Sobrescribe en el contenido que había anteriormente

cadena2:

f	e	l	i	z	\ **0**

- **strcmp**(cadena1, cadena2)

Compara la cadena1 con la cadena2 y devuelve un valor:

menor que cero (**< 0**) si la cadena1 es menor que la cadena2
igual a cero (**= 0**) si la cadena1 es igual a la cadena2
mayor que cero (**> 0**) si la cadena1 es mayor que la cadena2

Ejemplo:

char cadena1 [ ] = "sima", cadena2 [ ] = "tope";

```
x = strcmp(cad1, cad2);

if(x == 0)
    printf("Las cadenas son iguales\n");
else
    printf("Las cadenas son diferentes\n");
```

Nota: la igualdad se refiere a los mismos elementos:

< 0 ⇨ c1 < c2
== 0 ⇨ c1 == c2
> 0 ⇨ c1 > c2

- **strlen**(cadena1)

Esta función devuelve la longitud de una cadena no incluyendo el carácter de terminación nulo.

```
char cad1[ ] = "Finalmente";
char cad2[ ] = "Hora";
char cad3[ ] = "Sistemático";

x = strlen (cad1);
x = strlen (cad2);
x = strlen (cad3);
```

Tabla de funciones para cadenas de caracteres	
Función	**Descripción**
strcpy	Copia un string origen a un destino
strncpy	Copia hasta n caracteres de origen a destino
stpcpy	Copia uno string en otro
strdup	Copia un string dentro una locacion nuevamente creada
strstr	Busca la primera ocurrencia de un subcadena en otro string
strrchr	Busca la última ocurrencia de un caracter en un string
strchr	Busca un string por la primera ocurrencia de un caracter dado
strspn	Busca un string por un segmento que no contiene
strcspn	Busca un string por un segmento que no contiene
strpbrk	Busca un string1 la primera ocurrencia de cualquier caracter que esta string2
strtok	Busca s1 por el primera señal no contenida en s2
strcmp	Compara dos strings
stricmp	Compara dos strings sin caso sensitivo
strcmpi	Compara dos strings sin caso sensitivo
strcoll	Compara dos strings
strncmp	Compara porciones de dos strings
strnicmp	Compara porciones de dos strings
strncmpi	Comparas porciones de dos strings
strcat	Añade un string a otro
strlen	Calcula la longitud de un string
strncat	Añade un string a otro
strrev	Revierte todo caracteres en string(excepto el nulo)
strset	Pone todos caracteres en s a ch
strnset	Pone los primeros n caracteres de origen a destino

EJEMPLOS:

Programa 1:

Construya un programa que lea una frase (con espacios en blancos) e imprima cada una de las palabras en una línea diferente:

Entrada:

Introduzca una frase: arreglos en turbo c

Salida:
arreglos
en
turbo
c

```
#include <stdio.h>
#include<stdlib.h>
#include<conio.h>
void main(void)
 {
   char frase[100];
   int i = 0;
          system("cls");
   printf( "Escriba una frase: " );
   gets( frase );
          printf( "\nResultado:\n" );
   while ( frase[i]!='\0' ) {
      if ( frase[i]==' ' )
           printf( "\n" );
      else
           printf( "%c", frase[i] );
      i++;
   }
          getch();

}
```

Programa 2:

Escriba un programa que después de introducir una palabra convierta alternativamente las letras a mayúsculas y minúsculas:
Ejemplo:
Introduce una palabra: chocolate

Salida: ChOcoLaTe */

```
#include <stdio.h>
#include<stdlib.h>
#include<ctype.h>
void  main(void) {
   char palabra[100];
          int i = 0, j = 0;
          system("cls");

   printf( "Escribe una palabra: " );
```

```
    gets( palabra );
    printf( "Resultado:\n" );
    while ( palabra[i]!='\0' ) {
        if ( j==0 )
                printf( "%c", toupper( palabra[i] ) );
        else
                printf( "%c", tolower( palabra[i] ) );
        j = 1 - j;
        i++;
    }
    printf( "\n" );
    system( "pause" );
    return 0;
}
```

Programa 3:

Escriba un programa que solicite una cadena y un carácter a borrar. El programa recorrerá la cadena y borrará el elemento si este es encontrado en la cadena. El programa borra las ocurrencias de un carácter en toda una frase.

```
#include <conio.h>
#include <stdio.h>
void main(void)
{
 char a[100];
 char caracter[1];
 int i,j;
 clrscr();
 printf("Dame la cadena: ");
 gets(a);
 printf("\nDame el caracter a borrar: ");
 gets(caracter);
i=0;
while(a[i]!='\0')
 {
  if(caracter[0]==a[i])
  {
   j=i;
   while(a[j]!='\0')
   {
    a[j]=a[j+1];
    j++;
   }
   i--;
  }
  i++;
 }
 puts(a);
 getch();
 }
```

Programa 4:

Programa que solicita dos cadenas y calcula longitud, compara las cadenas, concatena las cadenas, copia una cadena a otra cadena e imprime en pantalla los resultados.

```
#include <stdio.h>
#include <string.h>
void main(void)
{       char str1[80],str2[80];
        int i;
        printf ("Introduce una cadena de caracteres: ");
        gets (str1);
        printf ("Introduce la segunda cadena de caracteres: ");
        gets (str2);
        /*longitud de las cadenas*/
        printf ("\n%s es de %d caracteres de largo\n",str1,strlen(str1));
        printf ("\n%s es de %d caracteres de largo\n",str2,strlen(str2));
        /*comparar cadenas*/
        i=strcmp(str1,str2);
        if (!i)       printf ("\nLas cadenas son iguales\n");
        else       if (i<0) printf ("\n%s es menor que %s",str1,str2);
        else printf ("\n%s es mayor que %s",str1,str2);
        /*concatenación de srt1 con str2*/
        if (strlen(str1)+strlen(str2)<80)
                {
                 strcat (str1,str2);
                 printf ("\n%s",str1);
                }
        /*copia str2 a str1*/
        strcpy (str1,str2);
        printf ("\n%s y %s",str1,str2);
}
```

7.5 CLASE PRÁCTICA

a) Explique que realiza el siguiente programa.

1.

```
#include <stdio.h>
void main (void){
int a[10], b[10], c[10], i;
for(i=0; i<=10; i++) {
        clrscr();
        printf("Ingrese numero de A: ");
        scanf("%d", &a[i]);
        printf("Ingrese numero de B: ");
        scanf("%d", &b[i]);
        if(a[i] < b[i])
        c[i] = a[i];
        else
        c[i] = b[i];
        }
clrscr();
printf("\t\t\tA\t\tB\t\tC\n\n");
for(i=0; i<=10; i++)
printf("\n\n\t\t\t%d\t\t%d\t\t%d", a[i], b[i], c[i]);
getch();
}
```

2.

```
#include <stdio.h>
#include <conio.h>
void main (void){
int aux, i, j, n[20];
for(i=0; i<20; i++){
        clrscr();
        printf("\nIngrese un numero entero positivo: ");
        scanf("%d", &n[i]);
        while(n[i]<0){
                clrscr();
                printf("\nIngrese el numero de nuevo: ");
                scanf("%d", &n[i]);}}
clrscr();
printf("Sus números son: ");
gotoxy(1,4);
for(i=0; i<=20; i++)
```

```
printf("*%d", n[i]);
gotoxy(1,8);
printf("Los números positivos en orden decreciente son:");
gotoxy(1,10);
for(i=0; i<19; i++)
        for(j=i+1; j<20; j++)
                if(n[i] < n[j])
                        {
                        aux=n[i];
                        n[i]=n[j];
                        n[j]=aux;
                        }
for(i=0; i<20; i++)
printf("*%d", n[i]);
getch();
}
```

b) Cadenas de caracteres

1. Haga un programa en C que lea e imprima una serie de números enteros hasta que se digite el primer valor nulo (o sea hasta que se digite un 0). El programa cuenta la cantidad de números positivos y negativos e imprime el valor final de los contadores.
2. Escriba un programa que asigne datos a una matriz unidimensional de **n** elementos y luego imprima el contenido de dicha matriz.
3. Realice un programa que asigne datos a una matriz de dos dimensiones y a continuación imprima la suma correspondiente a cada columna de la matriz.
4. Hacer un programa que lea una cadena de caracteres desde el teclado y un carácter adicional, el programa encontrará la posición de la primera ocurrencia del carácter de la cadena y lo imprimirá, en caso de no encontrarse se imprime un mensaje indicando que el carácter no pertenece a la cadena.

c) Arreglos lineales

1. Considere el segmento:

```
int arreglo1[4], arreglo2[4], i, j;
for (i = 0; i <= 3; i++)
{
  scanf (“%d”, &arreglo1[i]);
    for (j = i; j <= 3; j++)
      scanf (“%d”, &arreglo2[ j]);
}
```

y los datos de entrada: 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Después de ejecutarse el código ¿Cuál será el contenido de los arreglos?

a) arreglo1: 1 2 3 4
arreglo2: 5 6 7 8

b) arreglo1: 1 6 7 8
arreglo2: 2 3 4 5

c) arreglo1: 1 6 10 13
arreglo2: 2 7 12 14

d) arreglo1: 1 6 10 13
arreglo2: 2 7 11 14

e) arreglo1: 1 2 3 4
arreglo2: 12 13 14 15

f) faltan datos de entrada

5. Se tienen tres arreglos reales x, y y z. Elabore un programa que lea los datos de cada arreglo y calcule e imprima las sumatorias de todos sus elementos. Los arreglos tendrán la misma cantidad de elementos.
6. Calcular el promedio de 50 valores almacenados en un arreglo. Determinar además cuantos son mayores que el promedio, imprimir el promedio, el número de datos mayores que el promedio y una lista de valores mayores que el promedio.
7. Llenar dos vectores A y B de 5 elementos cada uno, sumar el elemento uno del vector A con el elemento uno del vector B y así sucesivamente hasta 5, almacenar el resultado en un vector C, e imprimir el vector resultante.
8. Llenar un arreglo de 20 elementos, imprimir la posición y el valor del elemento mayor almacenado en el vector. Suponga que todos los elementos del arreglo son diferentes (valide que todos los elementos sean diferentes).

d) Arreglos bidimensionales

1. Hacer un programa que llene una matriz de 10 * 10 y que almacene en la diagonal principal unos y en las demás posiciones ceros.

2. Hacer un programa que llene una matriz de 6 * 8 y que almacene toda la matriz en un vector. Imprimir el vector resultante.

3. Hacer un programa que llene una matriz de 8 * 8, que almacene la suma de los renglones (filas) y la suma de las columnas en un vector (2 vectores en total). Imprimir los vectores resultantes.
4. Hacer un programa que llene una matriz de 5 * 6 y que imprima cuantos de los números almacenados son ceros, cuántos son positivos y cuántos son negativos.
5. Escriba un programa que indique el número de la hilera cuya suma sea mayor que las demás hileras. Suponga que todas las hileras suman diferente cantidad.
6. Se tiene almacenado el vector M el cuál contiene la información sobre las calificaciones de la materia de INTRODUCCION A LA ING. EN Co. Diseñe un programa que imprima:

   - Cantidad de alumnos que aprobaron la materia.
   - Cantidad de alumnos que tienen derecho a examen de convocatoria.
   - El (o los) numero (s) de control de lo(s) alumno(s) que haya (n) obtenido la máxima calificación final.

Anexo

La librería estándar de C

Introducción

El estándar de C define sólo unas pocas palabras reservadas. Sólo con ellas no puede hacerse un programa ``normal'' en la vida real. El programador necesita una serie de funciones y herramientas **estándar**, que deben estar disponibles en cualquier entorno de programación de C / C++. A este conjunto de funciones se le llama **librería estándar**. Las funciones se declaran en ficheros de cabecera o **.h**. Las cabeceras contienen única y exclusivamente los prototipos de las funciones. El código o cuerpo de las funciones se incluye en **ficheros objeto** que son realmente la librería.

Principales ficheros de cabecera

Los principales ficheros de cabecera de C ``suelen ser'' los siguientes:

- ctype.h: Funciones útiles para la clasificación y el mapeado de códigos.
- errno.h: Funciones que permiten comprobar el valor almacenado en errno por algunas funciones de librerías.
- float.h: Funciones que establecen algunas propiedades de las representaciones de tipos real.
- limits.h: Funciones que establecen algunas propiedades de las representaciones de tipos enteros.
- math.h: Funciones que sirven para realizar operaciones matemáticas comunes sobre valores de tipo double.
- stdarg.h: Son declaraciones que permiten acceder a los argumentos adicionales sin nombre en una función que acepta un número variable de argumentos.
- stdio.h: Macros y funciones para realizar operaciones de entrada y salida sobre ficheros y flujos de datos.
- stdlib.h y a veces unistd.h: Declaraciones de una colección de funciones útiles y la definición de tipos y macros para usarlas. Entre ellas suele estar la función **malloc** que permite hacer peticiones de memoria dinámica al sistema.
- string.h: Declaración de una colección de funciones útiles para manejar cadenas y otros arrays de caracteres.
- time.h: Declaración de funciones para el manejo de fechas.

stdio.h

Funciones para el manejo de la Entrada/Salida

printf

int printf (const char *formato, ...);

Escribe texto formateado por el flujo stdout, según las especificaciones de ``formato'' y la lista de expresiones. Devuelve el número de caracteres escritos o un valor negativo en caso de error.

scanf

int scanf (const char *formato, ...);

Lee texto por el flujo stdin y lo almacena según las especificaciones de ``formato''. Devuelve el número de valores asignados o EOF si se produce error o se alcanza fin de fichero sin producirse lectura.

puts

int puts (const char *s);

Escribe los caracteres de la cadena ``s" por el flujo stdout. Escribe un carácter ``NL" en lugar del nulo de terminación. Devuelve un valor no negativo. En caso de error devuelve EOF.

Funciones para el manejo de ficheros

fopen

FILE *fopen(const char *nombre_fichero, const char *modo);

Abre el fichero de nombre ``nombre_fichero", lo asocia con un flujo de datos y devuelve un puntero al mismo. Si falla la llamada, devuelve un puntero nulo. Algunos de los caracteres iniciales de ``modo" son:

``r", para abrir un fichero de texto existente para su lectura

``w", para crear un fichero de texto o abrir y truncar uno existente, para su escritura

``a", para crear un fichero de texto o abrir uno existente, para su escritura. El indicador de posición se coloca al final del fichero antes de cada escritura

``r+", para abrir un fichero de texto existente para su lectura y escritura

fclose

int fclose(FILE *flujo);

Cierra el fichero asociado con ``flujo". Devuelve 0 en caso de éxito y EOF (end of file) en caso contrario.

fwrite

size_t fwrite(const void *buffer, size_t n, size_t c, FILE *flujo);

La rutina fwrite permite escribir c elementos de longitud n bytes almacenados en el buffer apuntado por ``flujo".

fread

size_t fread(const void *buffer, size_t n, size_t c, FILE *flujo);

La rutina fread permite leer c elementos de longitud n bytes del fichero apuntado por ``flujo" y los almacena en el buffer especificado.

fgetc

int fgetc(FILE *flujo);

Lee el siguiente carácter por ``flujo", avanza el indicador de posición y devuelve int. Devuelve EOF si pone a 1 el indicador de fin de fichero o el de error.

fgets

char *fgets(char *s, int n, FILE *flujo);

Lee caracteres por ``flujo'' y los almacena en elementos sucesivos del ``array'' que comienza en ``s'', continuando hasta que almacene ``n-1'' caracteres, almacene un carácter del nueva línea o ponga a 1 los indicadores de error o de fin de fichero. Si almacena un carácter, concluye almacenando un carácter nulo en el siguiente elemento del ``array''. Devuelve ``s'' si almacena algún carácter y no ha puesto a 1 el indicador de error; en caso contrario devuelve un puntero nulo.

fputc

int fputc(int c, FILE *flujo);

Escribe el carácter c por ``flujo'', avanza el indicador de posición del fichero y devuelve int c . En caso de error devuelve EOF.

fputs

int fputs(const char *s, FILE *flujo);

Escribe los caracteres de la cadena s por ``flujo''. No escribe el carácter nulo de terminación. En caso de éxito, devuelve un valor no negativo; en caso de error devuelve EOF.

fscanf

int fscanf(FILE *flujo, const char *formato, ...);

Lee texto y convierte a la representación interna según el formato especificado en formato. Devuelve el número de entradas emparejadas y asignadas, o EOF si no se almacenan valores antes de que se active el indicador de error o de fin de fichero.

fprintf

int fprintf(FILE *flujo, const char *formato, ...);

Genera texto formateado, bajo el control de formato y escribe los caracteres generados por flujo. Devuelve el número de caracteres generados o un valor negativo en caso de error.
A modo de resumen estas son las especificaciones de formato más comunes:

Formato	Descripción
%d	Entero con signo
%u	Entero sin signo
%c	Caracter
%s	Puntero a cadena de caracteres

fseek

int fseek(FILE *flujo, long desplazamiento, int origen);

La función fseek mueve el puntero de posición del fichero correspondiente al flujo de datos apuntado por ``flujo''. La nueva posición, medida en bytes, se obtiene añadiendo el número indicado por desplazamiento a la posición especificada por origen. La variable origen puede tomar tres valores:

- SEEK_SET: El puntero de posición apuntará al inicio del fichero más el desplazamiento
- SEEK_CUR: El puntero de posición apuntará a la posición actual del puntero de posición del fichero más el desplazamiento.
- SEEK_END: El puntero de posición apuntará al fin del fichero más el desplazamiento (deberá ser menor o igual que cero).

stdlib.h

Funciones para la conversión de tipos

abs

int abs(int i);

Devuelve el valor absoluto de i.

atof

double atof(const char *s);

Convierte los caracteres de la cadena s a la representación interna de tipo double y devuelve ese valor.

atoi

int atoi(const char *s);

Convierte los caracteres de la cadena s a la representación interna de tipo int y devuelve ese valor.

atol

long atol(const char *s);

Convierte los caracteres de la cadena s a la representación interna de tipo long y devuelve ese valor.

strtod

double strtod(const char *s, char **finptr);

Convierte los caracteres iniciales de la cadena s en la correspondiente representación interna de tipo double y devuelve ese valor. Si finptr no es un puntero nulo, la función almacena en él un puntero al resto de la cadena que no se ha convertido.

strtol

long strtol(const char *s, char **finptr);

Convierte los caracteres iniciales de la cadena s en la correspondiente representación interna de tipo long y devuelve ese valor. Si finptr no es un puntero nulo, la función almacena en él un puntero al resto de la cadena que no se ha convertido.

Funciones para el manejo de memoria

malloc

void *malloc(size_t longitud);

Asigna una dirección de memoria para un objeto de datos de tamaño longitud y devuelve esa dirección.

calloc

void *calloc(size_t nelem, size_t longitud);

Asigna una localización en memoria a un objeto de datos *array* que contiene *nelem* elementos de tamaño longitud, asignando ceros a todos los bytes del *array* y devuelve la dirección del primer elemento en caso de éxito; en caso contrario, devuelve un puntero nulo.

realloc

void *realloc(void *p, size_t longitud);

Cambia el tamaño de la memoria apuntada por p al que se indica con longitud. Asigna una dirección de memoria para un objeto de datos de tamaño longitud, copiando los valores almacenados en p. Devuelve la nueva dirección de memoria asignada.

free

void free(void *p);

Si p no es un puntero nulo, la función libera la memoria asignada al objeto de datos cuya dirección es p, en caso contrario, no hace nada. Se puede liberar la memoria asignada con calloc, malloc, realloc.

string.h

strcmp

int strcmp(const char *s1, const char *s2);
Compara los elementos de dos cadenas s1 y s2 hasta que encuentra elementos diferentes. Si todos son iguales, devuelve 0. Si el elemento diferente de s1 es mayor que el de s2, devuelve un valor mayor que cero; en caso contrario, devuelve un valor menor que cero.

strcpy

char *strcpy(char *s1, const char *s2);

Copia la cadena s2, incluyendo el nulo, en el *array* de elementos char que comienza en s1. Devuelve s1.

strdup

char *strdup(const char *s);

Devuelve un puntero a una nueva cadena de caracteres que es un duplicado de la cadena s. La memoria para esta cadena de caracteres se obtiene con la función malloc y se libera con la función free.

strlen

size_t strlen (const char *s);

Devuelve el número de caracteres de la cadena s, sin incluir el nulo de terminación.

strncmp

int strncmp(const char *s1, const char *s2, size_t n);

Compara los elementos de las cadenas s1 y s2 hasta que encuentra alguno diferente o hasta que se han comparado n elementos. Si todos los elementos son iguales, devuelve 0. Si el elemento diferente de s1 es mayor que el de s2, devuelve un número positivo. En caso contrario, devuelve un número negativo.

strncpy

char *strncpy(char *s1, const char *s2, size_t n);

Copia la cadena s2, sin incluir el nulo, en la cadena s1. Copia no más de n caracteres de s2. Entonces almacena, cero o más caracteres nulos si son necesarios para completar un total de n caracteres. Devuelve s1.

strndup

char *strndup(const char *s, size_t n);

Devuelve un puntero a una nueva cadena de caracteres que es un duplicado de la cadena s, solo copia los primeros n caracteres, incluyendo el nulo. La memoria para esta cadena de caracteres se obtiene con la función malloc y se libera con la función free.

math.h

ceil

double ceil(double x);

Valor entero más pequeño no menor que x.

cos

double cos(double x);

Coseno de x en radianes.

exp

double exp(double x);

Exponencial de x, e^x.

fabs

double fabs(double x);

Valor absoluto de x, $|x|$.

floor

double floor(double x);

Mayor valor entero menor que x.

log

double log(double x);

Devuelve el logaritmo natural de x.

log10

double log10(double x);

Devuelve el logaritmo en base 10 de x.

pow

double pow(double x, double y);

Devuelve x elevado a la potencia y, x^y.

sin

double sin(double x);

Devuelve el seno de x (en radianes).

sqrt

double sqrt(double x);

Devuelve la raíz cuadrada de x.

tan

double tan(double x);

Devuelve la tangente de x (en radianes).

ctype.h

islower

int islower (int c);

Devuelve un valor distinto de cero si c es cualquiera de las letras minúsculas [a-z] u otra minúscula local.

isupper

int isupper (int c);

Devuelve un valor distinto de cero si c es cualquiera de las letras mayúsculas [A-Z] u otra mayúscula local.

tolower

tolower (int c);

Devuelve la correspondiente letra minúscula si existe y si isupper(c) es distinto de cero; en caso contrario, devuelve c.

toupper

toupper (int c);

Devuelve la correspondiente letra mayúscula si existe y si islower(c) es distinto de cero; en caso contrario, devuelve c.

isdigit

isdigit() determina si el argumento es un dígito (0-9).

isalpha

isalpha() determina si el argumento es una letra (A-Z o a-z).

Bibliografía

1. **Fundamentos de programación.**
   Algoritmos y estructuras de datos
   Luis Joyanes Aguilar, segunda edición.

2. **Enciclopedia del lenguaje C.**
   Francisco Javier Ceballos

Historia de la ingeniería:

1. http://www.inti.gov.ar/cirsoc/pdf/historia_ingenieria/historia.pdf
2. http://pdf.rincondelvago.com/origen-de-la-ingenieria.html
3. http://members.fortunecity.es/orlandobr/Ing.htm
4. http://www.taringa.net/posts/info/13625510/Evolucion-_Los-mejores-microProcesadores_-_info-Completa_.html **(microprocesador)**
5. http://www.informatica-hoy.com.ar/hardware-pc-desktop/Generaciones-de-la-computadora.php

Dispositivos de entrada y salida:

1. http://www.reparарpc.info/2009/12/dispositivos-de-entrada-en-una-pc.html
2. http://blogdeconsultaentecnologia.blogspot.com/2009/08/partes-basicas-de-un-sistema_18.html
3. http://sulaycsb.blogspot.com/2010/05/dispositivos-de-salida.html
4. http://infoevolucion.blogspot.com/2009/01/los-discos-magneticos.html
5. http://www.informaticamoderna.com/Lectora_CD.htm
6.

Sistemas de numeración:

7. http://www.monografias.com/trabajos26/suma-binarios/suma-binarios.shtml#sumabinar
8. http://www.pablin.com.ar/computer/cursos/varios/basesnum.htm
9. http://www.cmelectronics.8m.com/sistemas_de_numeracion.html
10. http://www.pablin.com.ar/computer/cursos/varios/basesnum.htm

Algoritmos:

1. http://ing.unne.edu.ar/pub/informatica/Alg_diag.pdf
2. http://enriquebarrueto0.tripod.com/algoritmos/cap02.pdf
3. http://enriquebarrueto0.tripod.com/algoritmos/algor01.pdf
4. http://luda.uam.mx/curso1/Introduccion%20a%20la%20Programacion/ejemplos.htm
5. http://www.slideshare.net/diego_87/diagramas-de-flujosecuencias-bucles
6. http://www.slideshare.net/josedavidsia/ejercicios-de-algoritmos-4287605
7. http://s3.amazonaws.com/ppt-download/ejerciciosdeprogramacion-101022125103-phpapp02.pdf?Signature=t9dzyayfT16ERVowasdF7bBHzlo%3D&Expires=1296587517&AWSAccessKeyId=AKIAJLJT267DEGKZDHEQ

Lenguajes de programación:

1. http://es.wikipedia.org/wiki/Programador
2. http://www.iqcelaya.itc.mx/~vicente/Programacion/Lenguajes.pdf
3. http://upload.wikimedia.org/wikipedia/commons/8/89/G-Lenguajes_de_programacion.pdf

Estructuras de control:

1. http://www.desarrolloweb.com/articulos/2199.php
2. http://html.rincondelvago.com/estructuras-algoritmicas.html
3. http://www.programacionfacil.com/cpp:ciclo_do_while
4. http://www.mailxmail.com/curso-algoritmos-lenguaje-c/estructuras-secuenciales
5. http://personales.com/peru/lima/JpnCruz/Listado%20de%20EjerciciosArreglos.htm

PSEint:

1. http://pseint.sourceforge.net/manual.html#Introducci.C3.B3n

Otros:

1. http://es.wikipedia.org/wiki/Ariete
2. http://maxy.com.ar/~maxy/ejercicios.pdf

Ejemplos de cuadros sinópticos:

1. http://analisis.atompedia.com/es/sintesis/cuadro-sinoptico/cuadro-sinoptico-ejemplo
2. video: http://www.youtube.com/watch?v=4ZrZ1TOys8I&feature=related
3. definición: http://www3.unileon.es/dp/ado/ENRIQUE/Didactic/Mapas.htm#subir