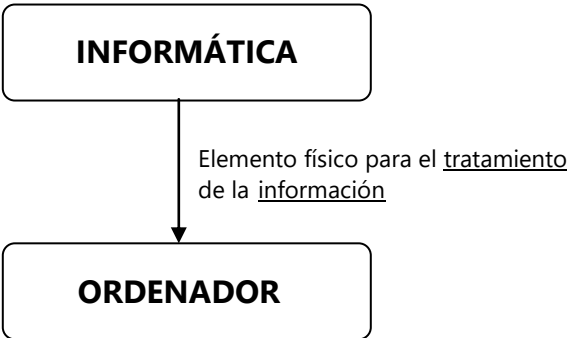
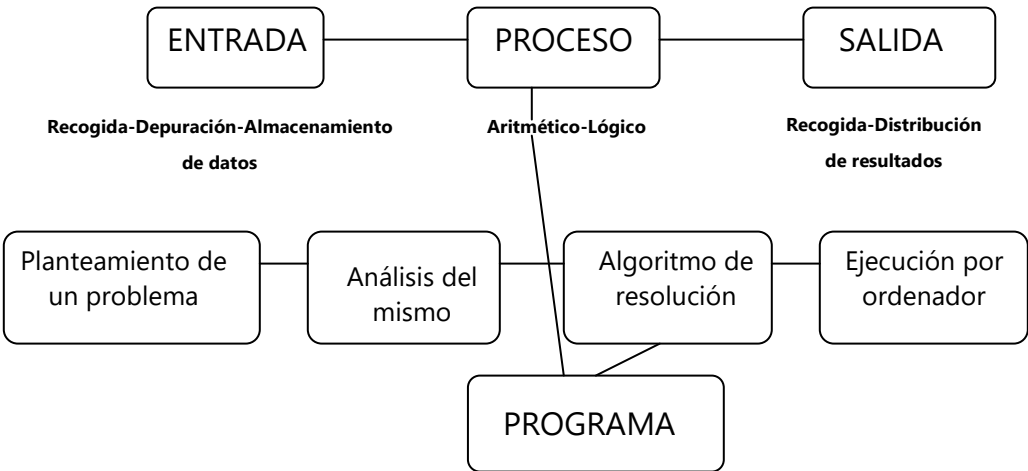


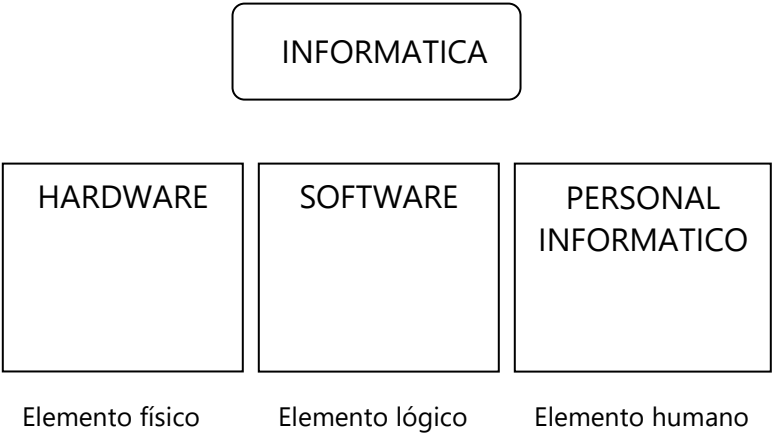
Ciencia que estudia el tratamiento automático y racional de la información.



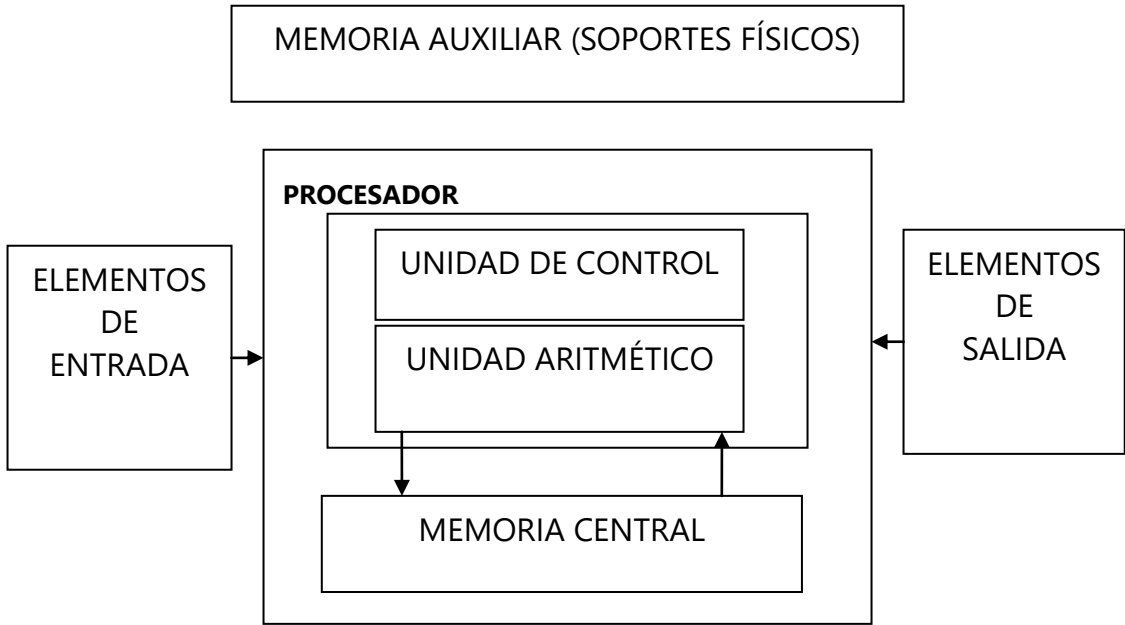
Maquina compuesta de elementos físicos, en su mayoría de origen electrónico capaz de realizar una gran variedad de trabajos a gran velocidad y con gran precisión, siempre que se le den las instrucciones adecuadas.



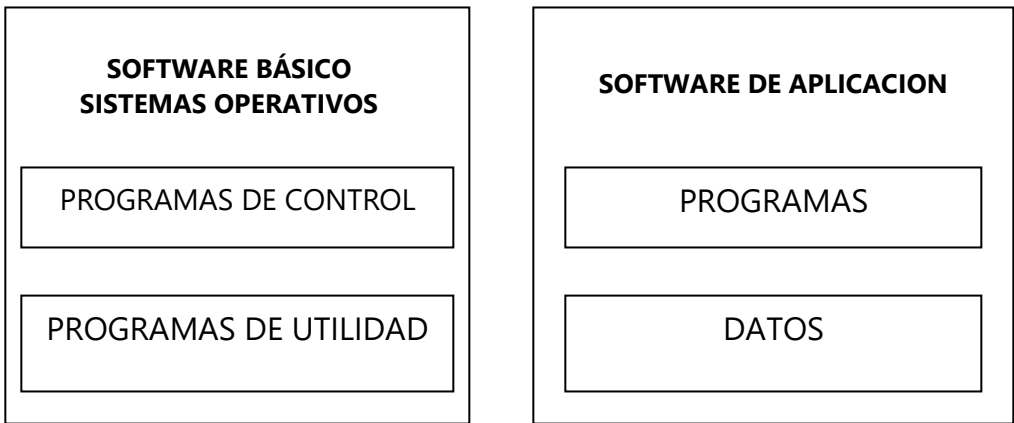
Conjunto de ordenes que se dan al ordenador para realizar un proceso determinado. Al conjunto de varios programas se denomina APLICACIÓN INFORMATICA



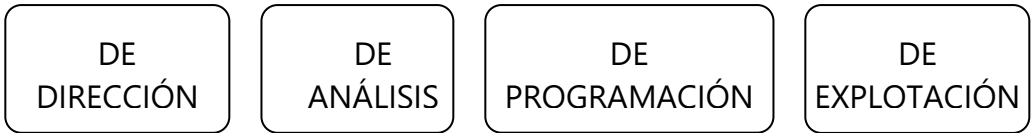
ELEMENTO FISICO (HARDWARE)



ELEMENTO LÓGICO (SOFTWARE)



ELEMENTO HUMANO (PERSONAL INFORMATICO)



PROGRAMA

Es la expresión de un algoritmo en un lenguaje de programación entendible por el ordenador.

TIPOS LENGUAJES DE PROGRAMACION

BAJO NIVEL: Lenguaje maquina

- Entendible directamente por el ordenador.
- Las instrucciones están compuestas por series de bits (1, 0).
- Es muy complicado de usar para las personas.

INTERMEDIOS: Lenguaje ensamblador

- Primer intento de sustituir el lenguaje maquina
- Las instrucciones están compuesta por símbolos nemotécnicos
- Cada modelo de ordenador tiene un lenguaje ensamblador propio
- El programador ha de conocer el hardware del equipo
- Hay que traducir las instrucciones a lenguaje maquina:
 - ensamblador

ALTO NIVEL:

- Facilitan la escritura de los programa y su entendimiento
- Tienen cierta similitud con el lenguaje humano
- Logran la independencia de la maquina, podemos utilizar un mismo programa en diferentes equipos
- Hay que traducir las instrucciones a lenguaje maquina
 - compilador: traduce un programa completo y analiza los errores sintácticos.
 - interpretar traduce, analiza los errores sintácticos y ejecuta Línea a Línea.

ERRORES

SINTACTICOS: Errores derivados de la sintaxis de la instrucción LOGICOS:

tral del ordenador, siendo los mas conocidos el disco o la cinta magnetica. tambien denominada streamer

Errores derivados de la lógica de funcionamiento

Introduccion a la programacio

Periféricos de comunicación con otro sistema físico u ordenador



Figura 1.4. Periféricos de

Son aquellos periféricos encargados de establecer y facilitar el trasiego o intercambio de información entre dos o mas ordenadores, o bien, entre un ordenador y otro sistema físico.

Un claro ejemplo de este tipo de periféricos sería un MODEM, encargado de convertir **señales digitales** (eléctricas) en **señales analógicas** (acústicas) y viceversa con el objetivo de facilitar el trasiego de información procedente de un sistema digital (ordenador) a través de un medio analógico (línea telefónica). Por tanto un MODEM puede ser considerado como un conversor de señales gracias al cual podemos poner en comunicación dos ordenadores remotos utilizando como medio de transmisión la línea (telefónica).



Figura 1.5. Periféricos de comunicación con otro sistema u ordenador físico.

La palabra **MODEM**, procede de las palabras **modulación** (proceso por el cual un tren de datos genera una señal analógica compatible con el medio o línea de transmisión) y **de modulación** (proceso inverso a la modulación y que consiste en reconstruir a partir de la señal recibida por la línea. el tren de datos que lo origina).

1.1.1. Conceptos y definiciones

- **Datos:** Son aquellos elementos considerados como unidades de tratamiento dentro de un sistema de procesamiento de datos.

Datos pendientes de procesar o elaborar, y **datos de salida**, que son aquellos resultados obtenidos una vez elaborados los datos iniciales. Al conjunto de los datos se le denomina **información**.

Programas: Son conjuntos de órdenes (instalaciones y sentencias) diseñadas y creadas a través del razonamiento lógico y almacenadas en ficheros de texto respetando la sintaxis de un determinado lenguaje de programación. Estos conjuntos de órdenes se transmiten al ordenador para la realización y ejecución de tareas concretas.

Aplicación informática: También recibe nombre de **paquete informático** y es la unión conjunta de uno o más programas enlazados relacionados entre sí. Junto con la documentación generada durante el proceso de desarrollo de dicha aplicación.

Sistema: Podemos definir un sistema como un conjunto de elementos relacionados entre sí la consecución de un determinado fin.

Sistema informático: Un sistema informático es un conjunto de elementos que permiten procesar información por medio de equipos informáticos (ordenadores) y cuya finalidad es la de obtener nueva información a partir de la ya existente y elaborada. Todo sistema informático queda dividido de forma global en cuatro capas o niveles generales que son:

1. El hardware.

2. Software.

- *El sistema operativo.*
- *Programas de aplicación.*

3. Recursos humanos, que son aquellas personas encargadas del desarrollo, implantación, explotación y mantenimiento de un sistema informático.



Los datos pueden ser básicamente de dos tipos. Denominados **datos de entrada**, que son aquellos

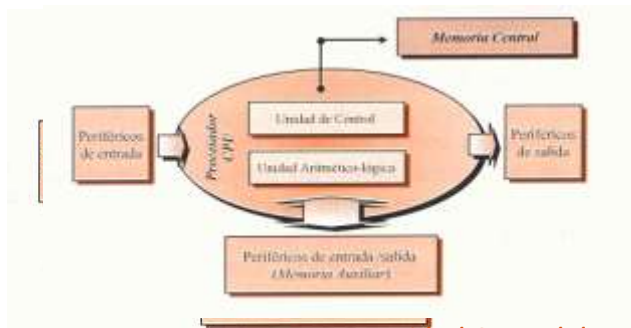
Figura 1.6. Estructura de un sistema informático.

Existen otros dos elementos fundamentales dentro de un sistema informático y que son los **datos**, definidos anteriormente, y los **protocolos** y **procedimientos** que se deben seguir para el Correcto uso o explotación del sistema.

Hardware: Se denomina hardware a la parte física de un sistema informático, por ejemplo

Disco duro, un monitor, una tarjeta grafica o de comunicaciones, cables, axial como cualquier dispositivo electrónico y elemento físico que conformen un ordenador.

Los principales elementos que constituyen el hardware son:



1,7. Principales elementos del hardware. Unidad Central de Proceso

Unidad Central de proceso (CPU- central process unit)

Es el verdadero cerebro de la maquina y encargado de controlar, coordinar y realizar todas aquellas opera-Siones de un sistema informático.

Cuando hablamos de un **microprocesador**, hablamos de una unidad central de proceso integrada en un circuito (chip) de muy alta escala de integración y constituido por dos elementos principales:

1. Unidad de Control (CU, Control Unit):

Es la parte encargada de detectar por medio de señales eléctricas el estado de cada uno de los elementos conectados al ordenador y gobernar las unidades de entrada, salida y entrada/salida, además de interpretar y ejecutar las instrucciones.

2. Unidad Aritmético Lógica (ALU, Aritmética-Lógica Unit):

Es la parte del procesador encargada de realizar todas aquellas operaciones de tipo aritmético y tipo lógico.

Desde el punto de vista de la estructura o la



Figura 1.8. Micro procesador Pentium II.

arquitectura hay ciertas características en común que mantienen todos los microprocesadores y que

son las que se des-criben a continuación:

1. Tamaho en bits:

Desde el nacimiento del primer microprocesador hasta los más actuales, se clasifican segun el tamaño en bits, también denominado **longitud** o **ancho de palabra**. Por ejemplo los microprocesadores 8088 y 8086 de Intel una longitud de palabra de 16 bits mientras que en los 80386 era de **32** bits. En la actualidad se alcanzan longitudes de palabra de hasta 64 bits.

2. Bus de datos:

Los distintos componentes de la CPU se conectan entre si a traves de líneas denominadas **buses**, que son circuitos de origen electrónico. En este caso, los buses de datos son las líneas o caminos utilizados por el microprocesador para el intercambio de datos e instrucciones con la memoria y los controladores de E/S. El tamaño del bus de datos coincide con el tamaño en bits o longitud de palabra del microprocesador.

3. Bus de direcciones:

Son los encargados de transmitir las direcciones de memoria y las direcciones de los elementos a el conectados. Según el humero de bits de este bus se podrán direccionar mas o menos cantidad de memoria, por ejemplo, los antiguos ordenadores con un microprocesador Intel 8088/8086 disponían de un bus de direcciones de 20 bits, por lo que solo podían direccionar 1.024 Kb de memoria, De ello podemos deducir que un bus de direcciones mas ancho, permite direccionar memorias mayores.

4. Bus de control:

Lleva la secuenciación de los procesos de cálculo y las comunicaciones. Las señales de reloj, las señales de interrupción, señales de E/S o las señales de control de los buses, son algunas de las señales del bus de control.

5. Registros internos:

Son considerados registros internos el **contador del programa** (que contiene la siguiente instrucción del programa), el **registro de instrucción**, el **acumulador** (son los registros asociados a la Unidad Aritmético Lógica y a las

operaciones de E/S), el **registro de estado** (contienen indicadores que mar-can el estado de funcionamiento del microprocesador), **registros de propósito general** (utilizados para almacenar datos temporalmente o contener una dirección y se caracterizan por no tener asigna-da una tarea específica), **registro índice** (se utiliza para guardar la dirección de un operando cuando se utiliza direccionamiento indexado) y el **registro puntero de pila** (su funcionalidad es la de almacenar temporalmente datos, direcciones de retorno y registros, pues es utilizada durante la llamada a subrutinas e interrupciones).

Memoria

Se puede diferenciar principalmente entre dos tipos de memoria:

1. Memoria Central (Central Emory):

También se conoce comúnmente con el nombre de memoria principal (Main Emory) y es la zona del sistema donde se almacenan los programas que se ponen en ejecución junto con los datos que queremos procesar. La información almacenada en esta memoria tiene un periodo de vida limitado, ya que una vez que el ordenador se apaga, la información en ella contenida se pierde.



Figura 1.9. Memoria RAM (memoria

2. Memoria Auxiliar (Auxiliary Emory):

Son dispositivos de almacenamiento masivo de información. Los datos y programas pueden quedar almacenados en este tipo de memoria de forma permanente, dando opción al usuario a recuperarlos en próximas sesiones de Trabajo.

•Software:

El software es la parte lógica o funcional de un sistema informático, es decir, la parte que hace funcionar al conjunto de elementos de origen electrónico de un ordenador, permitiendo y coordinando cada uno de los componentes físicos.

El software se divide principalmente en dos ele-



Figura 1.10. Discos duros (HD).

mentos, que son el **software básico** o **sistema operativo** (que trataremos en profundidad en el siguiente punto) y **software de aplicación**, que es

Se consideran como software de aplicación los programas de comunicaciones, los programas de ofimática (procesadores de texto, hojas de

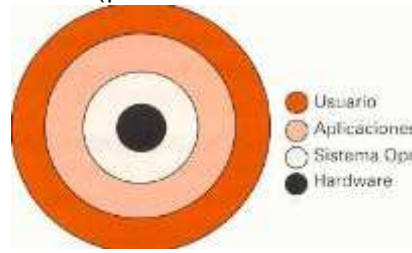


Figura 1.11. Estructura de un sistema

calculos, bases de datos, programas de presentación gráfica, etc.), programas de gestión (de clientes, facturación, nóminas, etc.), etc.

aquel constituido por una serie de programas y datos.

El software de aplicación es desarrollado con un objetivo muy concreto, que es el de proporcionar al usuario la posibilidad de realizar una serie de tareas de forma fácil y sencilla con la ayuda del ordenador.

1.2. ¿Que es un sistema operativo?

Es un conjunto de programas básicos encargado de hacer posible el manejo del ordenador y utilización de programas de aplicación, actuando como intermedia entre el usuario y el sistema (hardware), es decir, no es más que una interfase entre ordenador y usuario facilitando a este un entorno en el que pueda ejecutar programas de aplicación y controlar los dispositivos periféricos conectados al ordenador,

Un sistema operativo se encarga de la **gestión del procesador** (CPU), de la **memoria principal** (que es el lugar donde se almacenan el conjunto de datos o instrucciones que deberán ser ejecutadas para el tratamiento de la información), de los **dispositivos**

de entrada y sal (que son aquellos que sirven para comunicar al prosador con el mundo exterior y viceversa) y la memoria **auxiliar** (que son dispositivos de almacenamiento donde el usuario puede dejar o guardar grandes volúmenes de información de forma permanente).

En definitiva se puede decir que la misión de un lema operativo es la de controlar el flujo de información entre dispositivos y facilitar la interacción interactividad del usuario con la máquina de Manh cómoda, axial como la asignación de tareas y coordinación del funcionamiento interno del ordenador eficaz, eficiente y segura.

1.2.1. Tipos de sistemas operativos

Atendiendo al número de usuarios que pueden atendidos simultáneamente, el número de tareas o programas que se pueden ejecutar al mismo tiempo, número de procesadores soportados y el tiempo de respuesta, los sistemas operativos se pueden clasificar en

Sistemas monoprogramados

Este tipo de sistemas operativos se caracteriza por permitir la ejecución de un solo programa cada vez, por lo que no concederá la ejecución de otro hasta que no analiza el anterior.

El programa que se desea poner en ejecución es cargado en memoria y permanece en ella hasta que finaliza, adueñándose de la totalidad de los recursos del sistema, ya que en este tipo de sistemas operativos no se da opción a que un segundo proceso participe de los mismos.

Sistemas multiprogramados

También reciben el nombre de **sistemas multitarea**, y se caracterizan básicamente por ser sistemas que aprovechan los tiempos de inactividad o tiempos muertos de la **CPU** permitiendo la ejecución de varios programas simultáneamente, rentabilizando así la utilización y el trabajo del procesador.

El proceso es sencillo, se cargan en memoria varios programas y se divide el tiempo de proceso que la CPU dedicará a cada uno, lo que permite la ejecución alternativa o simultánea de todos ellos. Esto es lo que se denomina **concurrency de procesos**.

Sistemas multiproceso

Los sistemas operativos capaces de funcionar bajo ordenadores cuya arquitectura soporta dos procesadores reciben el nombre de sistemas multiproceso.

Los ordenadores capaces de soportar este tipo de arquitectura, dotan al sistema de mayor velocidad de proceso y seguridad, ya que aseguran en todo momento el continuo funcionamiento del ordenador en caso de fallo o mal funcionamiento de cualquiera de los procesadores, este hecho recibe el nombre de **proceso paralelo**.

Sistemas de tiempo compartido

Es lo que se conoce como **multiprogramación interactiva**, es decir, que permiten la ejecución simultánea de varios programas junto con la interactividad del usuario, pudiendo este realizar peticiones al sistema que serán atendidas inmediatamente.

En este tipo de sistemas, los usuarios cada vez que se conectan al ordenador abren una sesión (que es el periodo de tiempo transcurrido desde que el usuario se conecta hasta que se despiden) que crean un proceso capaz de atender a ese usuario facilitándole la comunicación con el sistema operativo.

Sistemas de tiempo real

El **tiempo de respuesta** es el periodo de tiempo transcurrido desde que se realiza una petición o solicitando al sistema hasta que este responde. Si ese periodo de tiempo de respuesta es muy breve (entre 1 milisegundo y 1 segundo) hablamos de **tiempo real**.

Estos sistemas también son considerados sistemas multiprogramados e interactivos, caracterizados por su rápida reacción y manejar información que debe ser continuamente actualizada según los cambios producidos en tiempo real, por lo que requieren grandes restricciones en el tiempo de respuesta. Son muy utilizados en grandes sistemas capaces de enlazar en tiempo real, puntos muy distantes.

Sistemas monousuario

Son sistemas muy simples que solo permiten el acceso a un usuario cada vez, por lo que no se requiere ningún tipo de restricción o control en la gestión de los usuarios conectados. Este tipo de sistemas pueden basarse tanto en la monoprogramación como en la multiprogramación y suelen ser principalmente ordenadores personales.

Sistemas multiusuario

Son sistemas que a su vez se basan en sistemas multiprogramados, permitiendo el acceso de varios usuarios simultáneamente.

Los usuarios tienen la posibilidad de ejecutar varios programas al mismo tiempo, lo que permite rentabilizar al máximo el rendimiento del procesador.

1.3. Tipos de lenguajes

Todo lenguaje de programación debe adaptarse a reglas previamente establecidas; por

Coll podemos decir que un **lenguaje de programación** es una notación o conjunto de símbolos y caracteres combinados entre si de acuerdo con una sintaxis ya definida para posibilitar la transmisión de instrucciones a la CPU (Unidad Central de Proceso). Dichos símbolos y caracteres son traducidos a un conjunto de señales eléctricas representadas en código binario (0 y 1). La razón de convertir esos símbolos y caracteres a ceros y unos se debe a que el microprocesador (cerebro del ordenador) solo entiende un lenguaje, que es el código binario o código máquina, también denominado **instrucciones máquina**,

Los lenguajes de programación se clasifican en dos grandes grupos, **lenguajes de bajo nivel** y **lenguajes de alto nivel**

1.3.1. Lenguajes de bajo nivel

Son aquellos que por sus características se encuentran mas próximos a la arquitectura de la maquina. Englobándose en este grupo el **lenguaje maquina** y el **lenguaje ensamblador**.

Lenguaje maquina

Todos aquellos problemas a los que se pretende dar una solución informática se plantean en el ámbito de expresión de algún lenguaje natural y para que dicha solución {mediante ordenes o instrucciones} pueda ser entendida por un ordenador, debe ser traducida a un lenguaje denominado **lenguaje maquina**.

El lenguaje maquina se caracteriza por:

- a) Ser considerado el primer lenguaje de programación.
- b) Ser el unico lenguaje inteligible directamente por un ordenador.
- c) Basarse en la combinación de dos únicos símbolos el cero y el uno denominados Bd (binary digit).
- d) Ser propio de un determinado procesador, es decir, que cada procesador tiene su propio y particular lenguaje maquina que no podrá ser entendido por cualquier otro.

Lenguaje ensamblador

El **lenguaje ensamblador** surge como sustituto del lenguaje maquina \ esta ligado en el uso de nemotécnicos (palabras abreviadas procedentes del ingles formadas por letras y números). La programación en lenguaje ensamblador precisa de un amplio conocimiento sobre la constitución, estructura y funcionamiento interno de un ordenador, así como un hábil manejo de los códigos y sistemas de numeración. Especialmente el binario y el hexadecimal.

Los programas desarrollados en ensamblador se caracterizan por:

- a) Ejecutarse mas rápidamente que si hubieran sido desarrollados en un lenguaje DC alto nivel.
- b) Ocupar mucho menos espacio en memoria.
- c) Facilitar el trabajo a la hora de desarrollar programas que controlen periféricos o dispositivos de E/S, simulen movimiento. Generen sonido. etc.
- d) Aportar mayor velocidad de operación y que el código fuente sea ensamblado directamente a lenguaje maquina.
- e) Generar programas más largos que los desarrollados en lenguajes de alto nivel.
- f) No ser transportables; es decir, un programa escrito para un microprocesador concreto no funcionara con un microprocesador diferente.
- g) Ser el lenguaje de programación más difícil escribir y depurar, lo que dificulta la verificación corrección y modificación de los mismos.

| Instrucciones en ensamblador | descripción |
|------------------------------|--|
| MOVE.L #35,D2 | Copia el numero 00000035 en hexadecimal |
| MOVE.B (A0)+,D1 | Copia en D1 el contenido de la posición de memoria direccionada por A0 incrementando el contenido de A0 en uno |
| ADD.L D1,D2 | Suma el contenido del registro D1 al contenido del registro D2 guardando el resultado en el registro D2 |

1.3.2. Lenguajes de alto nivel

Son aquellos lenguajes que por sus característica: encuentran mas próximos al usuario o programador, lo que son

lenguajes dirigidos a solucionar problemas mediante el manejo y tratamiento de estructuras de datos. A su vez, son abordadas por acciones como estas. Se consideran como tales el resto del lenguaje de programación como por ejemplo COBOL, Pascal, Visual Basic, Visual C++, etc.

Una de las características más importantes de los lenguajes es que, a diferencia de los lenguajes de 1º nivel, son independientes de la arquitectura del ordenador utilizado como soporte. Lo que implica que los programas desarrollados en lenguajes de alto nivel pueden ser ejecutados sobre ordenadores con distinto microprocesador. Este hecho hace que el programador no necesite amplios conocimientos sobre el funcionamiento interno del ordenador para el que está programando. Por otro lado, cabe destacar una mayor facilidad en el desarrollo, depuración y mantenimiento de los programas frente a los desarrollados con lenguajes de 1º nivel.

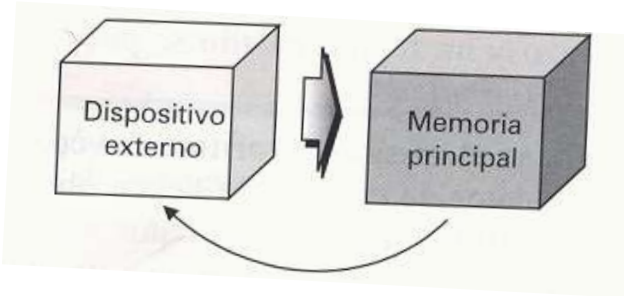
Como inconveniente destacable, es la necesidad de traducir los programas escritos en un lenguaje de nivel alto a un lenguaje de programación tan arcaico y motivado como el lenguaje máquina para que pueda ser interpretado y ejecutado por la Unidad

Central de eso, lo que significa disponer necesariamente de Traductor (para más información ver punto Ensambladores).

4.1. Partes de un programa

Todo programa está constituido por un conjunto de órdenes o instrucciones capaces de manipular un conjunto de datos. Estas órdenes o instrucciones pueden ser divididas en tres grandes bloques claramente diferenciados, correspondientes cada uno de ellos a una parte del diseño de un programa:

- Entrada de datos.
- Proceso o algoritmo.
- Salida de datos o resultados.



Salida de datos o resultados

Este bloque está formado por todas aquellas instrucciones que toman los resultados depositados en memoria principal una vez procesados los datos de entrada, enviándolos seguidamente a un dispositivo o periférico externo.

Figura 4.1. Partes de un programa.

Un algoritmo puede ser considerado como una caja negra encargada de procesar unos datos de entrada y generar unos resultados o datos de salida.

Entrada de datos

En este bloque se engloban todas aquellas instrucciones que toman datos de un dispositivo o periférico externo,



Figura 4.2. Entrada

depositándolos posteriormente en memoria central o principal para poder ser procesados.

Proceso o algoritmo

Engloba todas aquellas instrucciones encargadas de procesar la información o aquellos datos pendientes de elaborar y que previamente habían sido depositados en memoria principal para su posterior tratamiento. Finalmente, todos los resultados obtenidos en el tratamiento de

dicha información son depositados nuevamente en memoria principal, quedando de esta manera disponibles.

4.2. Algoritmos

4.2.1. Concepto

Un algoritmo se puede definir como la descripción abstracta de todas las acciones u operaciones que debe realizar un ordenador de forma clara y detallada, así como el orden en el que estas deberán ejecutarse junto con la descripción de todos aquellos datos que deberán ser manipulados por dichas acciones y que nos conduzcan a la solución del problema facilitando así su posterior traducción al lenguaje de programación correspondiente. El diseño de todo algoritmo debe reflejar las tres partes de un programa (vistas en el punto 4.1) y que son la entrada, el proceso y la salida.

Es importante tener en cuenta que todo algoritmo debe ser totalmente independiente del lenguaje de programación que será utilizado, es decir, que el algoritmo diseñado deberá permitir su traducción a cualquier lenguaje de programación con independencia del ordenador en el que se vaya a ejecutar dicho programa habitualmente.

La dificultad a la hora de conseguir una solución a un problema concreto reside en la fase de diseño, no en la traducción del algoritmo a un lenguaje de programación determinado. Por ello, se debe dar mayor importancia y prestar más atención al desarrollo del algoritmo que a la propia codificación, pues el conseguir un buen diseño nos facilitará totalmente su traducción.

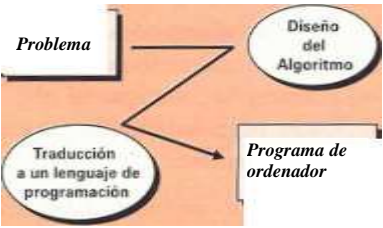
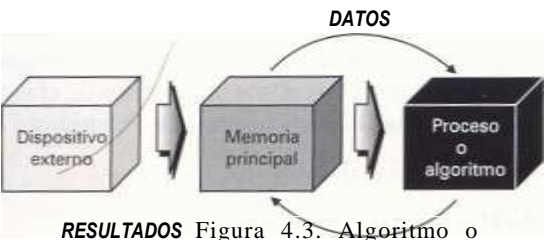


Figura 4.5. Estructura de elaboración de

Las características que debe cumplir el diseño de todo algoritmo son las siguientes:

- a) Debe ser **conciso** y **de/afado**, es decir, debe reflejar con el máximo detalle el orden de ejecución de cada acción u operación que vaya a realizar el ordenador.
 - I>) Todo algoritmo se caracteriza por tener un comienzo y un final. Por ello se puede decir que es **finito** o **limitado**.
 - c) Al aplicar el mismo algoritmo // veces con los mismos datos de entrada, se deben obtener siempre los mismos resultados o datos de salida. Por ello se puede decir que es **exacto** o **preciso**.
 - d) Un algoritmo nunca debe ser rígido en su diseño, debiendo mantener esta cualidad o característica de **flexibilidad** en sus representaciones gráficas, permitiendo y facilitando así las futuras modificaciones o actualizaciones del diseño realizado.
 - e) Debe ser lo más **claro** y **sencillo** posible para facilitar su entendimiento y comprensión por parte del personal informático.
- Existen normas en las que basarse, dadas por distintas organizaciones, como la I.S.O. (International Standard Organization), A.N.S.I. (American National Standard Institute), etc.

- c) Debe ser **intuitivo**, es decir, lo más claro y sencillo posible para facilitar su entendimiento y comprensión por parte del personal informático.
- d) Un algoritmo nunca debe ser rígido en su diseño, debiendo mantener esta cualidad o característica de **flexibilidad** en sus representaciones gráficas, permitiendo y facilitando así las futuras modificaciones o actualizaciones del diseño realizado.

Los diagramas de flujo se pueden clasificar en dos grandes grupos:

- a) **Organigramas**.
- b) **Ordinogramas**.

Una de las principales diferencias entre ambos, es que pertenecen a distintas fases o etapas de la resolución de un programa. Mientras que los organigramas corresponden a la fase de análisis, los ordinogramas corresponden a la fase de diseño.

ALGORITMO

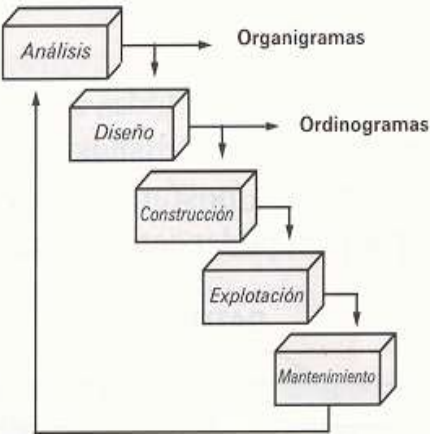


Figura 4.6. Características de un algoritmo.

4.2.2. Diagramas de flujo

Para el diseño de algoritmos se utilizan técnicas de representación. Una de estas técnicas son los denominados

diagramas de flujo, que se definen como la representación gráfica que mediante el uso de símbolos estándar conectados o unidos mediante líneas de flujo, muestran la secuencia lógica de las operaciones o acciones que debe realizar un ordenador, así como la corriente o flujo de datos en la resolución de un problema.

Todo diagrama de flujo debe cumplir unas características de diseño, que son;

- a) Debe ser **independiente** del lenguaje de programación elegido o utilizado para su posterior codificación, permitiendo así que el algoritmo pueda ser codificado indistintamente en cualquier lenguaje de programación.
- b) Debe ser un diseño **normalizado** para facilitar el intercambio de documentación entre el personal informático (programadores y analistas). Para ello

Organigramas

También denominados **diagramas de flujo de sistemas** o **diagramas de flujo de configuración**. Son representaciones gráficas del flujo de datos e información entre los

Figura 4.7. Diagramas de flujo.

periféricos o soportes físicos (de entrada/salida) que maneja un programa.

Todo organigrama debe reflejar:

- a) Las distintas áreas o programas en los que se divide la solución del problema, así como el nombre de cada uno de ellos.
- b) Las entradas y salidas de cada área indicando los soportes que serán utilizados para el almacenamiento tanto de los datos pendientes de elaborar o procesar como de los resultados obtenidos.
- c) El flujo de los datos.

Todo ello debe proporcionar:

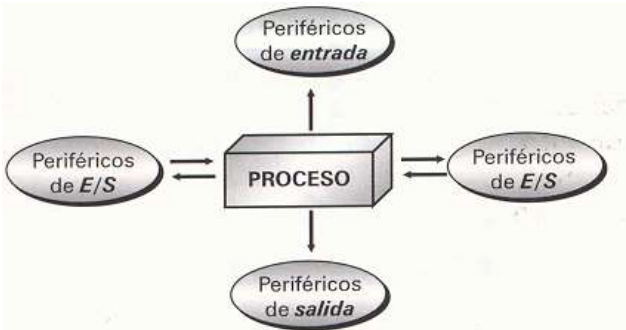
- a) Una visión global de la solución del problema.
- b) Una fácil realización de futuras correcciones.

c) Un control de todas las posibles soluciones.

Los organigramas deben respetar las siguientes reglas de representación:

- a) En la parte central del diseño, debe figurar el símbolo de proceso.
- b) En la parte superior del diseño y siempre por encima del símbolo de proceso deben figurar los soportes de entrada.
- e) En la parte inferior del diseño y siempre por debajo del símbolo de proceso deben figurar los soportes de salida.
- d) A izquierda y derecha del diseño, y por tanto a ambos lados del símbolo de proceso, figurarán los soportes que son tanto de entrada como de salida.

:



La simbología utilizada en la construcción de organigramas es la siguiente:

| Símbolo | Denominación | Tipo de dispositivo |
|---------|-------------------|---------------------|
| | TECLADO | Entrada |
| | SOPORTE MAGNÉTICO | Entrada |
| | PANTALLA/CRT | Salida |
| | IMPRESORA | Salida |
| | TARJETA PERFORADA | Entrada/Salida |
| | CINTA DE PAPEL | Entrada/Salida |
| | DISCO MAGNÉTICO | Entrada/Salida |
| | DISCO MAGNÉTICO | Entrada/Salida |
| | CINTA MAGNÉTICA | Entrada/Salida |

| Símbolo | Función |
|---------|--|
| | Proceso u operación. |
| | Clasificación u ordenación de datos en un fichero. |
| | Fusión o mezcla de dos o más ficheros en uno sólo. |
| | Partición o extracción de datos de un fichero. |
| | Manipulación de uno o varios ficheros (intercalación). |

b) símbolo de proceso

| Símbolo | Denominación | Tipo de dispositivo |
|---------|-------------------|---------------------|
| | CINTA ENCAPSULADA | Entrada/Salida |
| | DISCO FLEXIBLE | Entrada/Salida |
| | TAMBOR MAGNÉTICO | Entrada/Salida |

c) líneas de flujos de datos

| Símbolo | Función |
|---------|--|
| | Dirección del proceso o flujo de datos. |
| | Líneas de teleproceso (transmisión de datos). |
| | Línea conectora. Permite la unión entre unidades o elementos de información. |

Organigramas

También denominados *diagramas de flujo de sistemas* o *diagramas de flujo de configuración*. Son representaciones gráficas del flujo de datos e información entre los periféricos o soportes físicos (de entrada/salida) que maneja un programa.

Todo organigrama debe reflejar:

- a) Las distintas áreas o programas en los que se divide la solución del problema, así como el nombre de cada uno de ellos.
- b) Las entradas y salidas de cada área indicando los soportes que serán utilizados para el almacenamiento tanto de los datos pendientes de elaborar o procesar como de los resultados obtenidos.

c) El flujo de los datos.

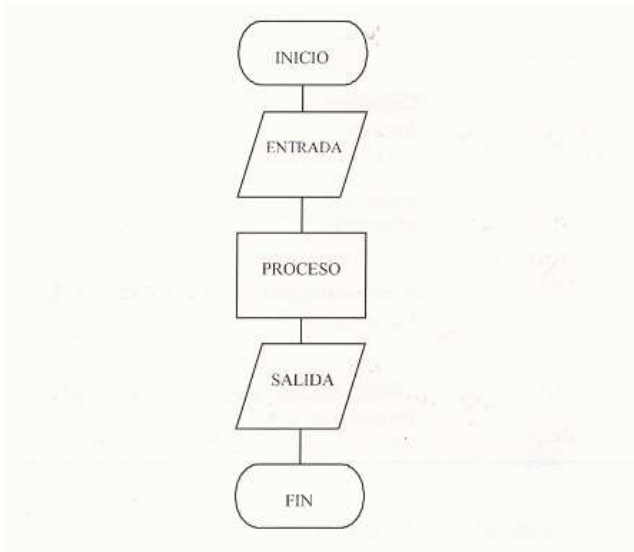
Todo ello debe proporcionar:

- a) Una visión global de la solución del problema.
- b) Una fácil realización de futuras correcciones.

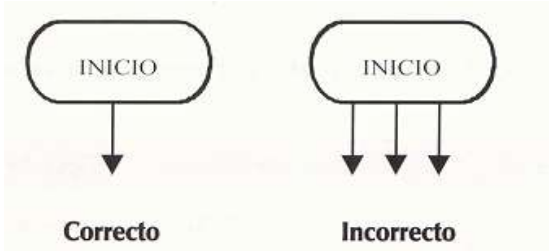
asi en ia mcddida en la que sea posible su entendimiento y comprcnsion, procurando llinilar al maxirao el uso de comentarios aelaratorios.

El diseno de todo ordinograma debe reflejar;

- a) Un principio o inicio que marca el comienzo de ejecucion del programa y que viene determinado por la palabra "INICIO".
- b) La secuencia de operaciones, lo mas detallada posible y siguiendo siempre el orden en el que se deberan ejecutar (de arriba-abajo y de izquierda-derecha).
- c) Un fin que marca la finalizacion de ejecucion del programa y que viene determinado por la palabra



e) A un simbolo de inicio de proceso no llega ninguna

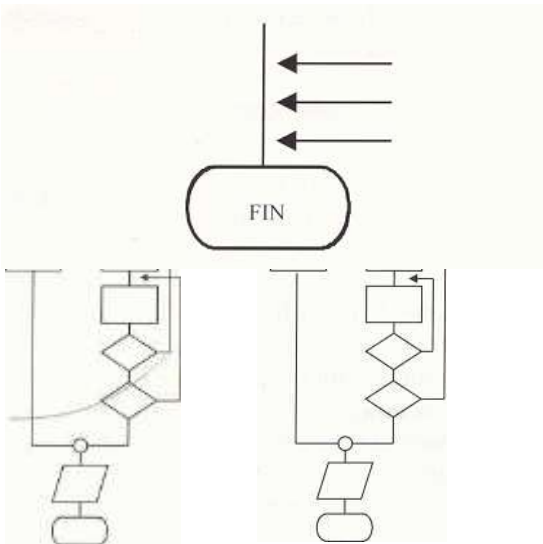


linca de conexion o flujo y de el solo puede partir una linea de conexion.

Figura 4.9 Estructura general de un ordinograma.

Las reglas que hay que seguir para la confeccion de un ordinograma son las siguientes:

- a) Todos los simbolos utilizados en el diseno deben estar conectados por medio de lineas de conexion o lineas de flujo de datos.
- b) El diseno debe realizarse con la maxima claridad dc arriba-abajo y dc izquircda-dcrccha.

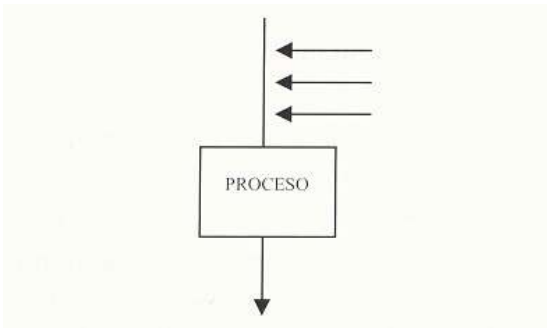


Disefio

Diseno

"FIN".

c)A un simbolo de proceso pueden llegarle varias lineas de conexion o flujo, pero de el solo puede salir una.



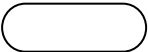
d) A un simbolo dc decision pueden llegarle varias lineas de conexion o flujo de datos, pero de el solo puede salir una linca de entre las dos posibi-lidades cxistentes (verdadero o falso).

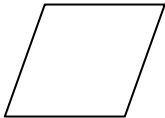

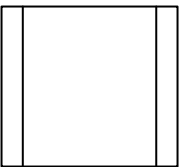
c) Queda terminantemente prohibido el eruce de lineas de conexion, pues ello nos indica que el ordinograma no esta correetamente disenado.

f) A un simbolo de final de proceso o ejecucion de programa pueden llegar muchas lineas de conexion, pero de el no puede partir ninguna.

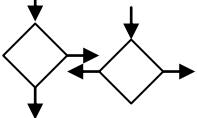

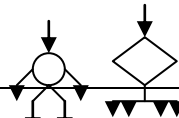
La simbologia utilizada en la construction de ordino-gramas es la sigui cntc:

a) *Simbolos de operacion o proceso:*

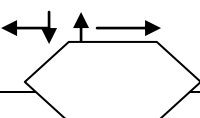

| Simbolo | Funcion |
|---|---|
|  | Terminal (marca el inicio, final o una para-da necesaria realizada en la ejecucion del programa). |

| Simbolo | Funcion |
|---|--|
|  | Operacion de E/S en general (utilizada para mostrar la introduccion de datos desde un periferico a la memoria del ordenador y la salida de resultados desde la memoria del ordenador a un periferico). |
|  | Proceso u operacion en general (utilizado para mostrar cualquier tipo de operacion durante el proceso de elaboracion de los datos depositados en la memoria). |
|  | Subprograma o subrutina (utilizado para realizar una llamada a un subprograma o proceso, es decir, un modulo indepen- diente cuyo objetivo es realizar una tarea y devolver el control de ejecucion del pro- grama al modulo principal). |


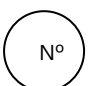
b) Simbolos de decision:

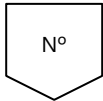
| Simbolo | Funcion |
|---|--|
|  | Decision de dos salidas (indica operacio- nes logicas o comparativas seleccionando en funcion del resultado entre dos cami- nos alternativos que se pueden seguir). |
|  | Decision multiple con "n" salidas (indica el camino que se puede seguir entre varias posibilidades segun el resultado de la ope- racion logica o comparacion establecida). |
|  | Bucle definido, empleado para modificar una instruccion o bloque de instrucciones que a su vez producen una alteracion o modificacion en el comportamiento del programa |

c) Lineas deflujo:

| Simbolo | Funcion |
|--|--|
|  | Flechas indicadoras de la direccion del flujo de datos. |
|  | Linea conectora, tambien llamada linea de flujo de datos (permite la conexon entre los diferentes simbolos utilizados en el diseno). |

d) Simbolos de connexion:

| Simbolo | Funcion |
|---|--|
|  | Conector (este simbolo es utilizado para el reagrupamiento de lineas de flujo). |
|  | Conector de lineas de flujo en la misma pagina (utilizado para enlazar dos partes 'cualesquiera del diseno a traves de un conector de salida y un conector |

| | |
|---|--|
| | de entrada). |
|  | Conector de lineas de flujo en distintas pagi- nas (utilizado para enlazar dos partes cua- lesquiera del diseio a traves de un conector de salida y un conector de entrada). |

c) Simbolo de comentarios:

| Simbolo | Funcion |
|---------|---------|
|---------|---------|

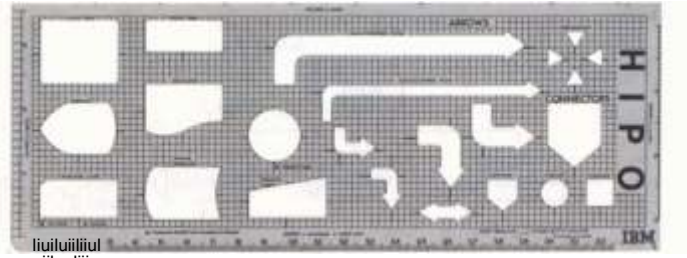
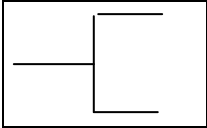


Figura 4.10. Plantilla.

| | |
|--|--|
|  | Permite escribir comentarios a lo largo del diseno realizado |
|--|--|

4.2.3. Pseudocodigo

• Pseudocodigo o notation pseudocodificada: Se

puede definir como el lenguaje intermedio entre el lenguaje natural y el lenguaje de programacion seleccionado. Esta notation se cncuentra sujeta a unas determinadas reglas que nos permiten y faci- litan el diseno de algoritmos. La notation pseudocodificada surge como metodo para la representa- cion de instrucciones en una metodologia estructurada y nacio como un lenguaje similar al ingles, que utilizaba palabras reservadas de este idioma (start, end, stop, while, repeat, for, if, if- then-else, etc.) y que posteriormente se fue adap- tando a otros lenguajes de lengua hispana.

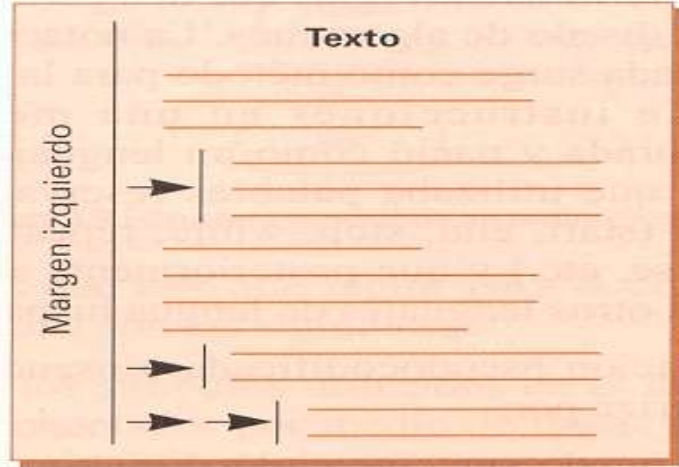
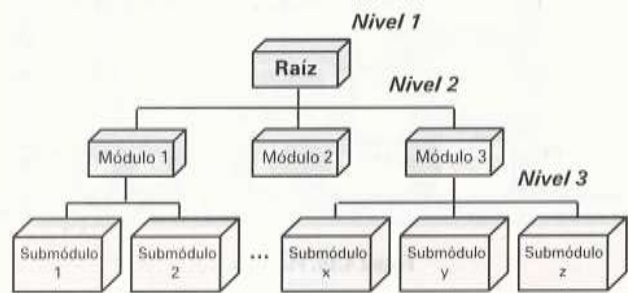
La notation pseudocodificada o pseudocodigo se caracteriza por:

- a)No puede ser ejecutado directamente por un ordenador, por lo que tampoco es considerado como un lenguaje de programacion propia- mente dicho.
- b)Ser una forma de representation muy sencilla de aprender y utilizar.

- c) Permitir el diseño y desarrollo de algoritmos totalmente independientes del lenguaje de programación posteriormente utilizado en la fase de traducción o codificación.
- d) Facilitar el paso del algoritmo al correspondiente lenguaje de programación.
- e) Permitir una gran flexibilidad en el diseño del algoritmo a la hora de expresar acciones concretas.
- f) Facilitar la realización de futuras correcciones o actualizaciones gracias a que no es un sistema de representación rígido.

estructura de un proband

vas, es decir, lo que se conoce comunmente como disen o diseendente o **Top down** y que consiste en la descomposicion sucesiva del problema en niveles o subproblemas mas pcquenos, lo que nos permite la simplification del problema general.



La escritura o disen o de un algoritmo mediante el uso de esta herramienta, exige la **"identacion"** o **"sangria"** del texto en el margen izquierdo de las dife- rentes lineas, lo que facilita el entendimiento y comprension del disen o realizado

Figura 4.12. Sangría o indentación del texto.

Toda notation pseudocodificada debe permitir la description de:

a) Instrucciones primitivas (entrada, salida y asigna- cion).

a) Instrucciones de proceso o calculo.

b) Instrucciones de control.

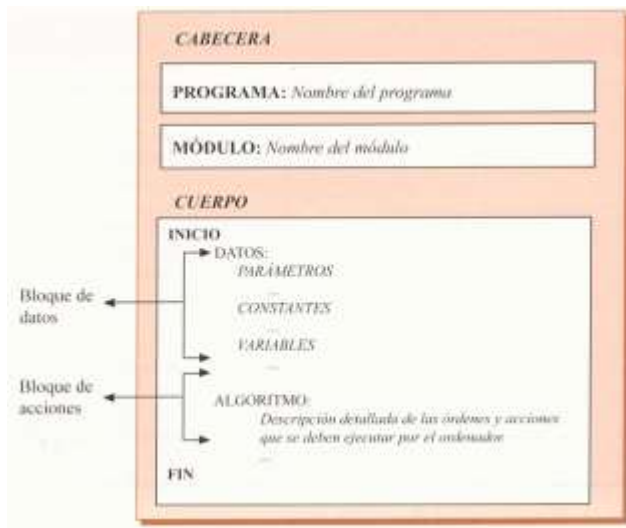
c) Instrucciones compuestas.

b) La description de todos aquellos elementos de trabajo y estructuras de datos que se vayan a manipular en el programa (variables, constantes, tablas, ficheros, etc.).

Todo algoritmo representado en notation pseudocodificada debera reflejar las siguientes partes:

- Cabecera:** Es el area o bloque informativo donde quedara reflejado el nombre del algoritmo y el nombre del programa al que pertenece dicho disen o, debiendosc especificar el primero en el apartado denominado **Modulo** y el segundo en el apartado denominado **Programa**.

Cuerpo: Se denomina asi al resto del disen o, el cual queda dividido en otros dos bloques denomi- nados **Bloque de datos**, que es el lugar donde deberan qucdar descritos todos los elementos de trabajo necesarios en la ejecucion del programa, entendiend o como tal



parametros, constantes y variables de todo tipo y **Bloque**

de acciones que es la zona en la que se deben describir con la maxima claridad y detalle todas aquellas acciones que el ordenador debera realizar durante la ejecu- cion del programa cuando estas sean convertidas en instrucciones ejecutables

Ejemplo:

Programa que calcula el area de un rectangulo dada su base y su altura

| Notacion Pseudocodificada | |
|--|---------------|
| PROGRAMA: Area del reciangulo | |
| MODULO: | |
| Principal | |
| INICIO | |
| DATOS: | |
| VARIABLE | |
| S | |
| Area | Numerico Real |
| Base | Numerico Real |
| Altura | Numerico Real |
| ALGORITMO | |
| : | |
| Leer | |
| Base | |
| Leer | |
| Altura | |
| Area = Altura | |
| Base | |
| Escribir "El valor del Area es:". | |
| Area | |
| FIN | |

```
CODIGO FUENTE
main()
{
    float Area;
    float Base;
    float
    Altura;

    scanf("%f",&Base); scanf("%f",&Altura); Area
    = Base * Altura;
    printf("El valor de Area es: %f",Area);
}
```

de comentarios lo haremos anteponiendo dos asteriscos al comentario propiamente dicho.

** Linea de comentario |

Los dos cuadros anteriores muestran un ejemplo comparativo del calculo del area de un rectangulo. El primer cuadro muestra el algoritmo representado en notation pseudocoficada y el segundo cuadro su correspondiente traduccion al lenguaje dc programacion seleccionado (en este caso C).

Comentarios

Son utilizados a lo largo del diseno realizado para aclarar o facilitar su comprension. Normalmente los comentarios son reservados al propio codigo fuente del programa una vez traducido el algoritmo discnado al lenguaje de programacion seleccionado, donde se hacen verdaderamente imprescindibles para la futura comprension del programa en la denominada fase dc mantenimiento.

Los comentarios, no afectan directamente a la compilation de un programa y suelen ser utilizados sobre todo para:

- a)Aclarar el cometido o funcion de una variable definida.
- b)Explicar el objetivo de una instruction de control (alternativa o repetitiva).
- c)Aclarar zonas del programa donde se realizan cal- culos y operaciones complejas.
- d)Comentar llamadas realizadas a subprogramas o funciones.

De la misma forma que su uso es ncccsario e impres- cindible, es recomendable no hacer uso indiscriminada- mente de el los para comentar puntos del programa obvios y carentes de dificultad.

Existen varias notacioncs para la representation de comentarios segun el lenguaje de programacion utilizado, pero para el diseno de algoritmos mediante notation pseudocodificada, el uso

4.2.4. Tablas de decision

Las tablas de decision son herramientas para el diseno de algoritmos que muestran las acciones que se deben ejecutar en nuestro programa cuando se cumplan ciertas condiciones. Una tabla de decision presenta cuatro bloques o apartados:

| | |
|-----------------------|------------------------|
| Matriz de Condiciones | Entrada de Condiciones |
| Matriz de Acciones | Entrada de Acciones |

- **Matriz de condiciones:** Contiene todas las condiciones del problema que se plantea.
- **Matriz de acciones:** Refleja todas las acciones a realizar.
- **Entrada de condiciones:** Muestra las situaciones que se pueden presentar.
- **Entrada de acciones:** Indica las acciones a efectuar.

Cada combination de entrada de condiciones con su correspondiente entrada de acciones recibe el nombre de **regla de decision**.

En una tabla de decision existiran tantas reglas de decision como entradas condiciones/acciones diferentes. Teniendo en cuenta que el numero de reglas de decision debe cubrir todas las posibilidades sin repeticiones ni omisiones.

Reglas de construccion

1. A una condition de entrada solo le corresponde una decision.
2. A un condition de salida le pueden corresponder varias condiciones de entrada.
3. El numero de reglas de decision es 2^n siendo n el numero de condiciones que se pueden dar.
4. La entrada de condiciones se representan con una X (indiferencia), S (afirmativo), N (negativo).
5. La entrada de acciones se representa con una X si se realiza y con un guion (-) en caso contrario.

6. La lista de condiciones pueden especificarse en cualquier orden.

- 7.Cada rcgla dc decision o columna equivale a un camino en un diagrama dc flujo.
- 8.Las tablas de decision se leen sicmpre de izquier- da a derecha y las reglas de arriba abajo.

Ejemplo:

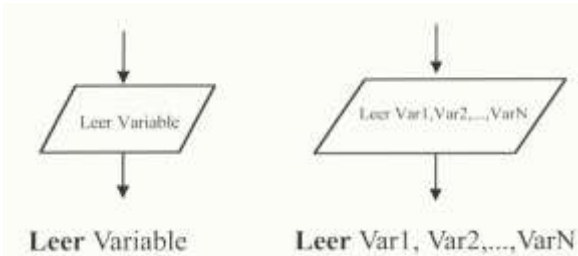
Una empresa se dedica a la distribution de piezas mecanicas para la reparation de vehiculos. Los pedidos recibidos, asi como la emision de facturas a clicntes, estan sujetas a las siguientes situaciones:

- Los pedidos realizados pueden ser de Madrid Capital o de la periferia.
- Indcpndientcmnte de la procedencia del pedido, se emitira una factura sabiendo que si el pedido proccdc de la capital y este es superior a 225.000 pesetas se debe liacer un 7% de descuento.
- Para aquellos pedidos procedentes de la periferia dc Madrid:
 - Se imprimiran ctiquetas con las direcciones de las empresas cliente.
 - Los portes corrqn a cuenta del cliente, siendo estos cargados cn factura.

| | | | | |
|---------------------------------|---|---|---|---|
| Envio procedente de la capital. | S | s | N | N |
| Pedido superior a 225.000 ptas. | N | s | S | N |
| Impresion de etiquetas. | - | . | X | X |
| Calculo del precio del pedido. | X | X | X | X |
| Aplicar descuento (7%). | - | X | X | - |
| Anadir portes a factura. | - | - | X | X |

4.3.2. Instrucciones de definicion de datos

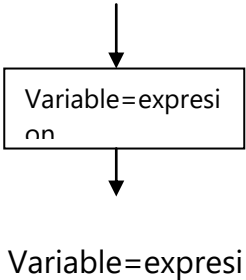
Son aquellas instrucciones utilizadas para informal- al procesador del espacio que debe reservar cn memoria con la finalidad de almacenar un dato mediante el uso de variables simples o estructuras de datos mas comple- jas como, por ejemplo, tablas.



4.3. Tipos de instrucciones

4.3.1. Concepto

Una instruction puede ser considerada como un hecho o suceso dc duration limitada que genera unos cambios previstos en la ejecucion de un programa, por lo que debe ser una action previamente estudiada y definida.



La definition consiste en indicar un nombre a traves del cual haremos referenda al dato y un tipo a traves del cual informaremos al procesador de las caracteristicas y espacio que debera reservar en memoria.

4.3.3. Instrucciones primitivas

Se consideran como tal las instrucciones de asignacion y las instrucciones de entrada/salida.

Instrucciones de entrada

Son aquellas instrucciones encargadas de recoger el dato de un periferico o dispositivo de entrada (por ejemplo el teclado, un raton, un escaner, etc.) y seguidamente almacenarlo en memoria en una variable previamente definida, para la cual se ha reservado suficiente espacio en memoria.

En el supuesto de leer varios valores consecutivos con la intencion de almacenarlos en variables diferentes, lo indicaremos situando uno a continuacion del otro separados por comas.

Instrucciones de asignacion

Son aquellas instrucciones cuyo cometido es almacenar un dato o valor simple obtenido como resultado al evaluar una expresion en una variable previamente definida y declarada.

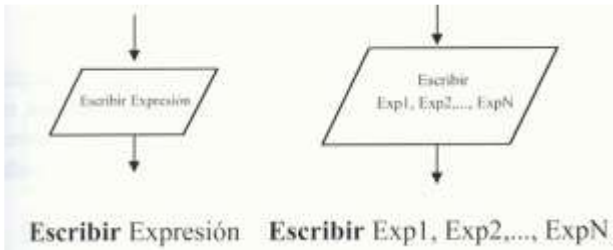
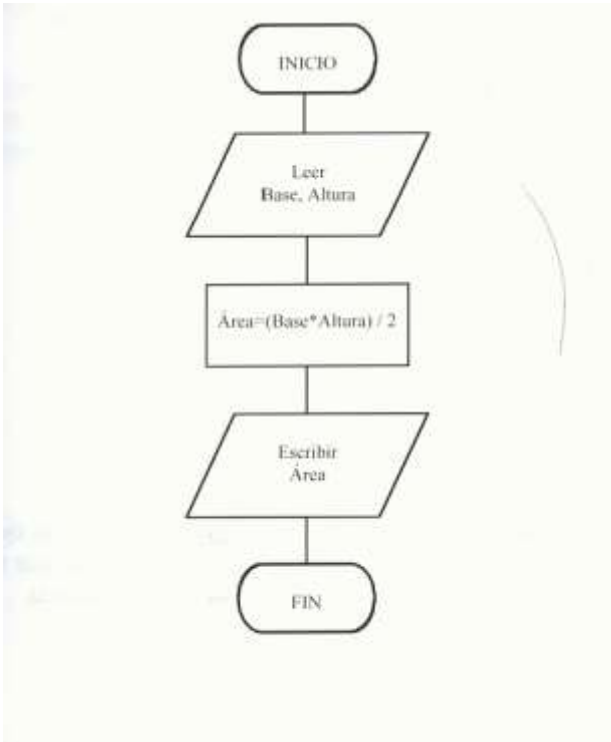
En toda instruction de asignacion es conveniente tener en cuenta las siguientes recomendaciones:

- a)** El tipo de la variable sobre la que se va a realizar la asignacion debera coincidir con el tipo de dato del valor obtenido por la expresion que aparece en la parte derecha de la instruction de asignacion.

b) En aquellos casos en los que variable y expresion no tengan el mismo tipo de dato, a veccs, existe la posibilidad dc que el tipo de dato del valor obte- nido por la expresion que aparece en la parte derecha de la instruction de asignacion sea con- vertido al tipo de dato de la variable que aparece en la parte izquierda dc la misma.

En ocasiones este tipo de conversiones es realiza- do automaticamente por algunos compiladores

Instrucciones de salida
Son aquellas instrucciones encargadas de recoger los datos procedentes dc variables o los resultados obteni- dos de expresiones cvaluadas y depositados en un periférico o dispositivo de salida (por ejcmpl,



la panta- 11a, una impresora, un plotter, etc).

En el supuesto de escribir varios valores o resultados de forma consecutiva, lo indicaremos situando uno a continuation del otro y separados por comas de la misma forma que ocurría con las instrucciones de entrada.

Ejcmpl:

Disenar un algoritmo que calcule el area de un triangulo utilizando como metodos de representation el ordinograma y la notation pseudocodificada.

PROGRAM A: A rea del triangulo

MODULO: Principal

INICIO

DATOS:

VARIABLES

Area Numerico Real

Base Numerico Real

Altura Numerico

Real **ALGORITMO:**

Leer Base,

Altura Area =

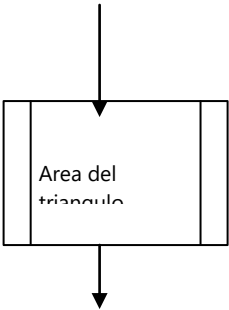
(Base*Altura) / 2

Escribir Area

FIN

4.3.4. Instrucciones compuestas

Son aquellas instrucciones que no pueden ser ejecutadas directamente por el procesador, y estan constituidas por un bloque de acciones agrupadas en subrutinas, subprogramas, funciones o modulos.



4.3.5. Instrucciones de control

Son utilizadas para controlar la secuencia de ejecucion de un programa asi como, determinados bloques de instrucciones.

Instrucciones de salto

Son aquellas instrucciones que alteran o rompen la secuencia normal de ejecucion de un programa perdiendo toda posibilidad de retornar el control de ejecucion del programa al punto de llamada. El uso de este tipo de instrucciones debe quedar restringido en una programacion estructurada.

Formato: **GOTO Etiqueta**

INICIO

DATOS:

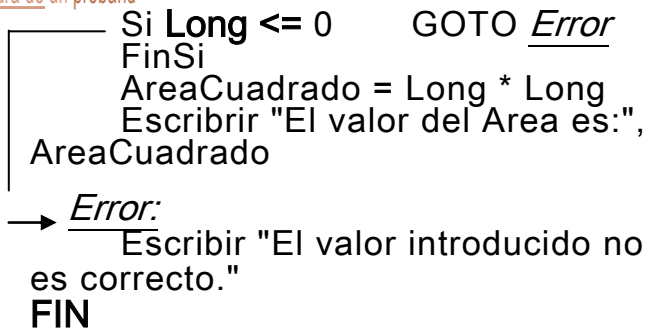
VARIABLES Long

AreaCuadrado

ALGORITMO

Leer long

Numerico
Real
Numerico
Real



INICIO
DATOS:
VARIABLES Long Numerico Real
AreaCuadrado Numerico Real

ALGORITMO:
Inicio: Leer Long
AreaCuadrado = Long * Long Escribir "El
valor del Area es:", AreaCuadrado

GOTO Inicio

FIN ***Salto incondicional***

Ejemplo:

Algoritmo que lee dos valores numericos y calcula primero la suma, despues la resla, a

a) *Instrucciones de salto conditional*

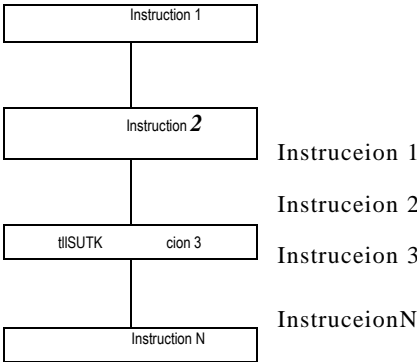
Son aquellas instrucciones que alteran la secuen- cia de ejecucion de un programa solo y exclusi- vamente en el caso de que la condition especi- ficada sea cierta.

b) *Instrucciones de salto incondicional*

Alteran la secuencia normal de ejecucion de un programa siempre, pues no van acompanadas de una condition que limite en determinadas ocasio- nes la realization del salto a otra parte del pro- grama.

Instrucciones secuenciales

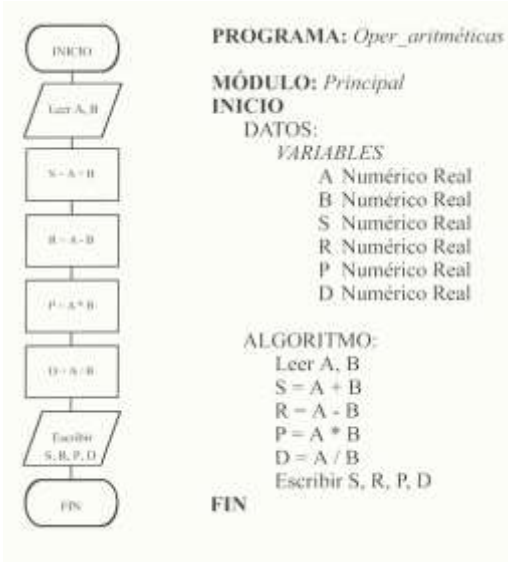
El orden de ejecucion de las instrucciones de un programa es secuencial, es decir, que las instrucciones se ejecutan de arriba-abajo y de izquierda-derccha una detras de otra respetando siempre el orden inicialmen- te establecido entre ellas, por ello, las estructuras secuenciales se caracterizan por respetar esta misma cualidad.



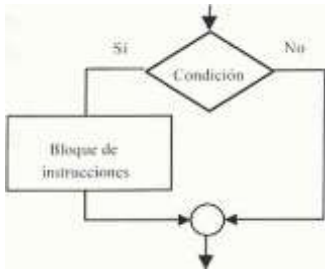
continuation el producto y seguidamente la division de ambos valores, cscribiendo finalmente el resultado obtenido en cada una de estas operaciones.

Instrucciones alternativas

Son aquellas que controlan la ejecucion o la no eje- cucion de una o mas instrucciones en funcion de que se curapla o no una condition previamente establecida

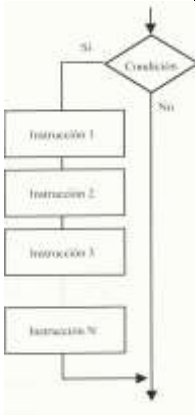


a) Alternativa simple



Si Condición
Instrucción 1
Instrucción 2
Instrucción N
Fin
Si Condición
Inst.1;
Inst.2;...; Inst.N
FinSi

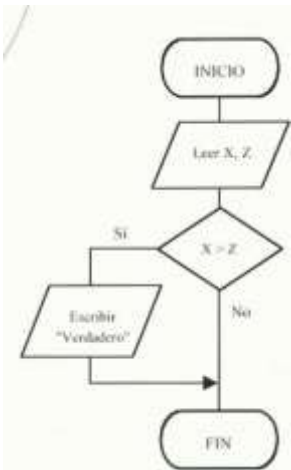
Otra posible forma de representación



Si Condición
Instrucción 1
Instrucción 2
Instrucción 3
Instrucción N
FinSi

Ejemplo:

Algoritmo que lee dos valores numéricos y los almacena en dos variables de nombre 'X' y 'Z', mostrando en aquellos



casos en los que 'X' es mayor que 'Z' un mensaje que diga "Verdadero".

PROGRAMA: Condición simple

MODULO: Principal INICIO

DATOS:

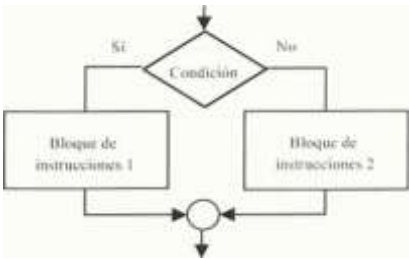
VARIABLES

X Numérico Entero Z
Numérico Entero

ALGORITMO: Leer

X, Z Si X > Z

b) Alternativa doble

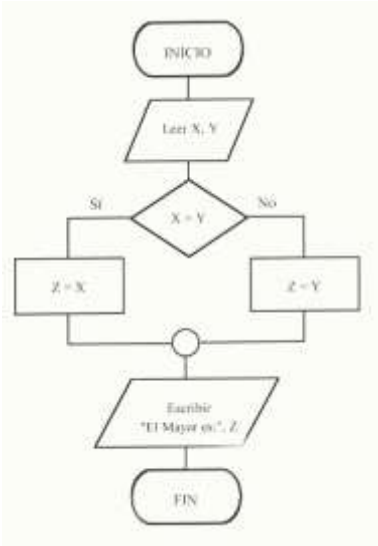


Escribir "Verdadero"
Fin

Si Condición Instrucción
↓
Instrucción A
Instrucción B
Instrucción N

Ejemplo:

Algoritmo que lee dos valores numéricos distintos "x" e "y" y determina cual es el mayor dejando el resultado en una tercera variable de nombre "z".



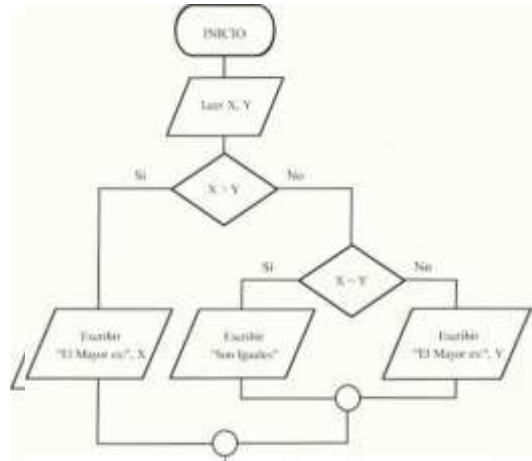
PROGRAMA: ConditionJoble

MODULO: Principal/INICIO

DATOS: VARIABLES

Z numerico real
X numerico real
Y numerico real

ALGO
RIT
MO
:
Lee
r X,
Y
Si
X > Y
Z =
X Si no
Z =
F
i
n



Si
Escribir "El Mayor es:", Z

FIN Ejemplo:

Algoritmo que lee dos valores numericos 'X' e 'Y', determina si son iguales y en caso de no serlo, indica cual

PROGRAMA: Condition anidada

MODULO: Principal/INICIO

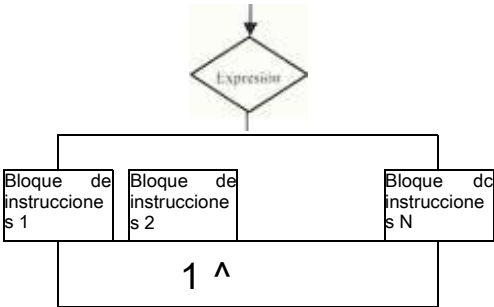
DATOS: VARIABLES
X Numerico Real Y

Numerico Real

ALGORITMO: Leer X, Y Si
X > Y

Escribir "El Mayor es:", X
Sino Si X = Y
Escribir "Son iguales" Sino
Escribir "El Mayor es:", Y
FinSi FinSi

c) Alternativa multiple



Segunvalor

Expresion

Valor 1:

Bloque de instrucciones 1

Valor2:

Bloque de instrucciones 2

ValorN:

Bloque de instrucciones N Otros
FinSegunvalor

Ejemplo:

Algoritmo que lee 'N' caracteres y contabiliza el numero de veces que se repite la letra a, e, i, o y u.

PROGRAMA: Contador vocales

MODULO: Principal/INICIO

DATOS:

VARIABLES Car N, C

Cont_a, Cont_e, Cont_i, Cont_o, Cont_u, ALGORITMO: Leer N

C = Cont_a = Cont_e =
Cont_i = Cont_o = Cont_u
= 0 Repetir Leer Car
Segun_valor Car
'a': Cont_a =

Cont_a + 1 'e':

Cont_e = Cont_e + 1

'i': Cont_i = Cont_i +

1 'o': Cont_o = Cont_o

+ 1 'u': Cont_u =

Cont_u + 1

FinSegun_valor C =

C + 1 Mientras C <

N

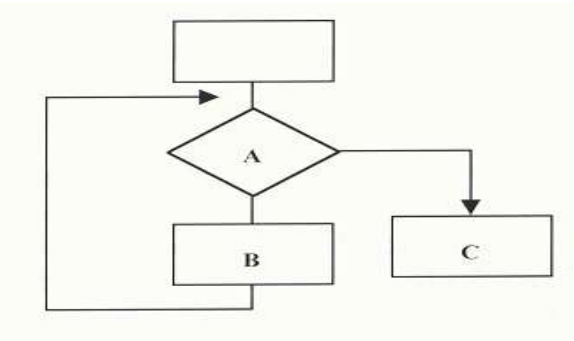
Escribir Cont_a, Cont_e, Cont_i, Cont_o, Cont_u

Instrucciones repetitivas

Son aquellas instrucciones que nos permiten variar o alterar la secuencia normal de ejecucion de un programa

haciendo posible que un grupo de acciones se ejecute mas de una vez de forma consecutiva. Este tipo de instrucciones tambien recibe el notnbre de bucles o lazos.

Todo bucle o instruccion repelitiva se caracteriza por estar constiluida por tres partes:



A. Condicion o Expresion conditional

B. Cuerpo. constituido por la inslruccion o bloque de instrucciones que se deberan ejecutar en caso de ser verdadera la expresion conditional establecida.

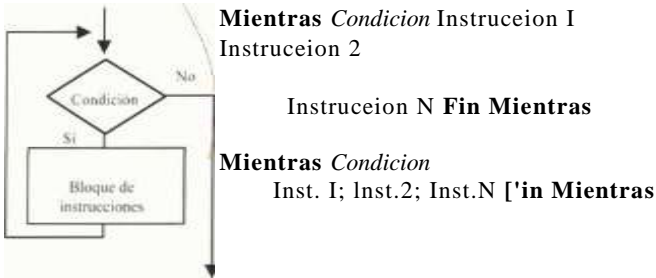
C. Salida o Final del bucle.

a) Estructura Mientras

La estructura **Mientras** se caracteriza porque su diseno permite repetir un bloque tie instrucciones de 0-*n* veces, es decir:

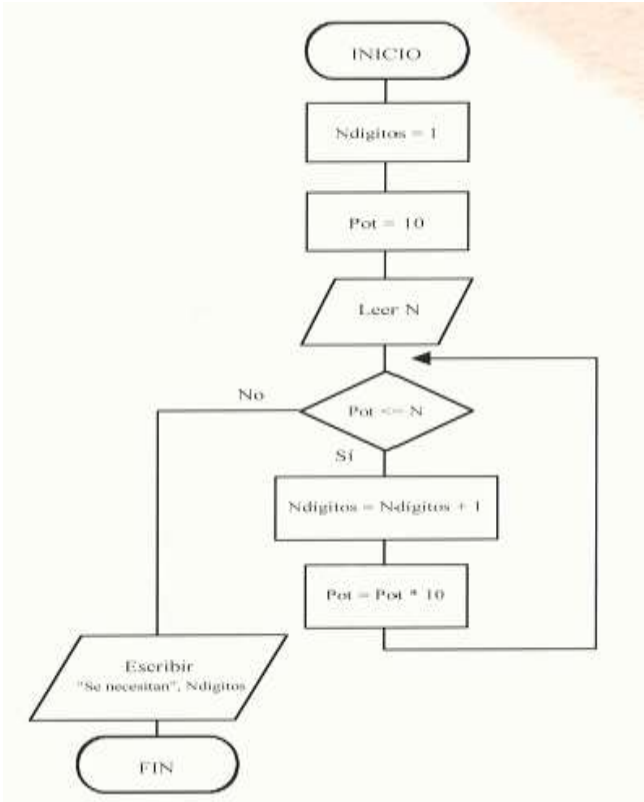
- En aquellos casos en los que la condition establecida sea verdadera, el numero de veces que se eje-culan dicho bloque de instrucciones sera una vez como minimo y *n* como maximo.

En aquellos casos en los que la condition establecida sea falsa dicho bloque de instrucciones no se ejecutara ninguna vez



Ejemplo:

Diseno del algoritmo correspondiente a un programa que lee un numero entero posilivo y determina el numero de digitos decimales neesarios para la representacion de dicho valor.



PROGRAMA: Cuentajugitos

MODULO:

Phncipcd

INICIO

DATOS:

VARIABLES N digitos Pot
Numerico Entero
Numerico Entero
Numerico Entero

ALGORITMO:

Ndigitos = 1 Pot = 10 Leer N
Ivlientras Pot <= N
Ndigitos =
Ndigitos + I Pot
= Pot * 10

FiiiMientras

Eseribir "Sc neccsilan ". Ndititos

KIN

El control del bucle normalmente se realiza titilizan-do un valor de la corriente de entrada de datos que indica el final de los mismos. Este valor clcgido no debe ser significative dentro de la corriente de datos.

Ejemplo:

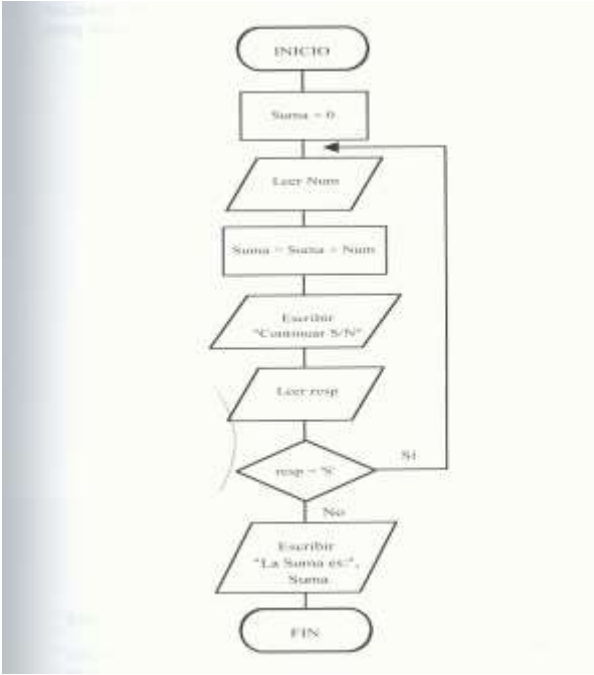
Disefio del algoritmo correspondiente a un programa que suma todos aquellos numeros leidos mientras no sean negativos.

Existen instrucciones repetitivas que utili/an estruc-turas **Hasta** y **Repetir-Hasta**, cuyo funcionamiento es muy similar a las estructuras **Mientras** y **Repetir-Mien**-ffiflspero modificando la condicion {en el ejemplo anterior bastaria con poner AST = NUMAST) y las salidas de la condicion (en el ejemplo anterior intercambiando la salida Si por la No y viceversa).

El control del bucle en una estructura **Repetir-Mien-Iras** se realiza normalmcnte utilizando la evaluation de la respuesta a un mensaje despues de haber realizado las instrucciones contnidas en el bucle por lo menos una vez.

Ejemplo:

Algoritmo que escribe la suma de una secuencia de pmeros enteros leidos del teclado finalizando la entrada de datos al evaluar la respuesta dada a un mensaje que diga "Continual- S/N", moslrado despues de realizar las operaeiones del bucle.



DATOS:
VARIABLES Suma Num resp

ALGORITM

O Suma =
0 Repetir
Leer Num
Suma Suma + Num Escribir
"Continual- S/N" Leer resp

Datos

Variables

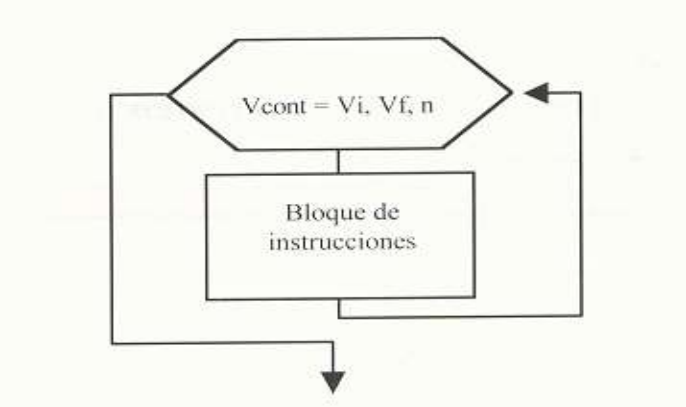
Suma Numerico entero
Num Numerico entero
Resp Numerico entero

ALGORITMO

Suma=0
Leer Num
Suma=Suma+Num
Leer resp
Mientras resp= "S"
Escribir "La Suma es" , Suma
FIN

c) Estructura Para

Este lipo de instrucciones repetitivas se caracteriza porque el numero dc veces que se repetira el bloque de instrucciones generalmente esta tljado de antemano



Para Vcont de Vi a Vf con inc=n

Instrucción 1

Instrucción 2

...

Instrucion n

FinPara

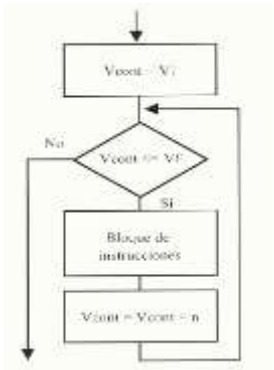
Para Vcont de Vi a Vf con Inc=n

Inst l; Inst2; ... ; InstN

FIN PARA

| | |
|-------|--|
| Vcont | Variable contador del bucle. |
| Vi | Valor inicial que toma Vcont (valor inicial a partir del cual comienza la ejecución del bucle). |
| Vf | Valor final para Vcont (es el valor final que se toma como referencia para la finalización del bucle). |
| n | Cantidad en que incrementa o decrementa (según sea el valor especificado positivo o negativo) la variable Vcont al final de cada vuelta de bucle. Por defecto el valor es siempre 1. |

La instrucción PARA es una forma compacta de representar un bucle MIENTRAS especifico, siendo la estructura equivalente a la anteriormente mostrada la presentada a continuación



Vcont= Vi
Mientras Vcont<=Vf
Instruccion1
Instruccion2
...
Instrucción
Vcont=Vcont+n
FinMientras

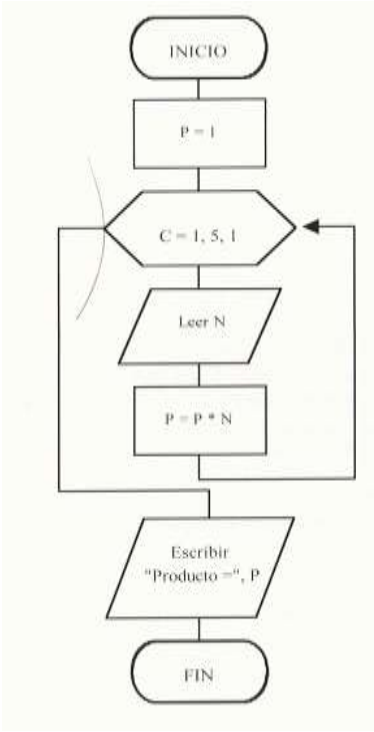
El control del bucle en una estructura *Para*, se realiza mediante el previo conocimiento del numero de veces que se van a efectuar las operaciones del bucle. Este numero de veces puede ser establecido por una constante o por una variable que almacena el valor introducido por teclado, o bien calculado en funcion de los valores '*initial*', '*final*' e '*incremento*'.

Ejemplo:

Número de veces = $(Vf - Vi) \setminus Inc + 1$

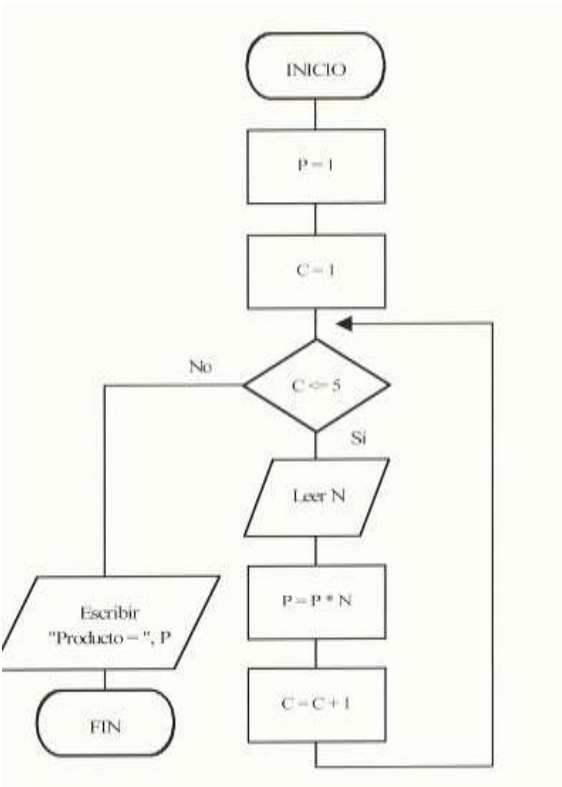
Algoritmo que lee cinco valores numericos y calcula su producto.

Utilizando una estructura *Para*, el diseno del algoritmo resultante seria el siguiente:



PROGRAMA:Producto
MODULO: Princial
INICIO
Datos:
Variables
P Numerico entero
C Numerico entero
N numerico entero
ALGORITMO
P=1
Para C de 1 a 5 Con Inc=1
Leer N
P=P*N
FinPara
Escribir “Producto”=P
FIN

Utilizando una estructura *Mientras*, el diseno del algoritmo seria el siguiente, donde podemos apreciar que ambas soluciones son totalmente equivalentes pero empleando estructuras de control diferentes.



Numerico Entero

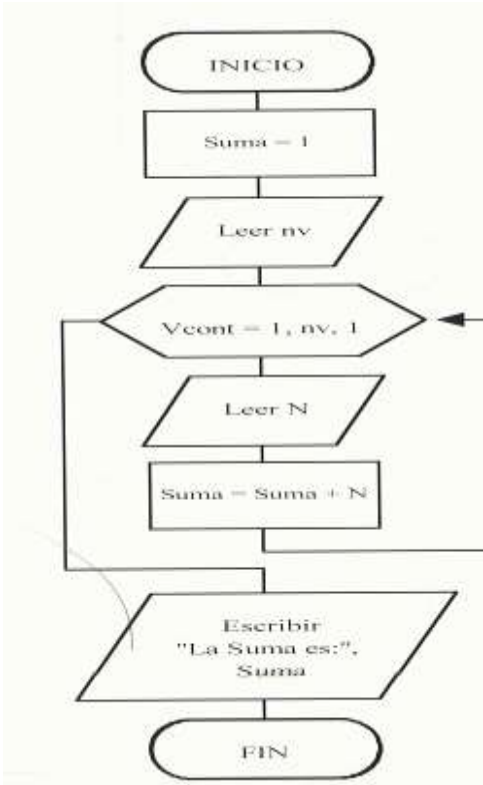
ALGORITMO:

```
P = I
C = I
  Mientras C <= 5
  Leer N
  P = P * N
  c= C + I
  Fin
  Mientras
  Escribir "Producto =",P
```

FIN

Ejemplo:

Algoritmo correspondiente a un programa que escribe la suma de una serie de numeros lefdos, siendo intro-ducido por Icclado el numero de valores que hay que leer.



PROGRAMA: Sumaji MODULO: Principal

INICIO

DATOS:

VARIABLES Suma

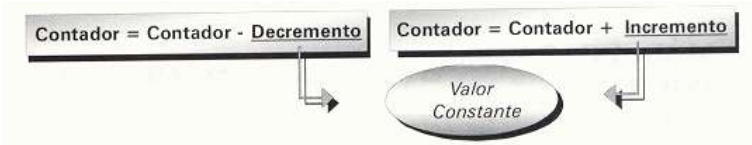
4.4. Contadores, acumuladores e interruptores

4.4.1. Contadores

Un contador no es mas que una variable destinada a contener un valor que se ira incrementando o decrementando en una cantidad fija y constante y que es almacenado en memoria principal.

Los contadores suelen utilizarse generalmente para el control de procesos repetitivos, es decir, su principal objetivo es contabilizar un conjunto de sucesos o acciones que se desean repetir en un programa mediante el uso de cstructuras de control repetitivas (Mientras, Repelir-Mientras y Para).

Todo contador debe tomar un valor inicial antes de ser utilizado.



Ejemplo:

Diseno del algoritmo dc un programa que lee M numeros y determina cuales son pares y posilivos.