

2

ALGORITMOS

2.1 Introducción

El objetivo principal de la materia es el de enseñar a resolver problemas mediante una computadora. Un programador de computadoras antes de nada es un resolutor de problemas.

Por lo que para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático.

Por esto la materia tiene como nombre Metodología de la programación.

Antes de comenzar con el tema vamos a considerar el significado de la palabra **ALGORITMO** esta palabra se deriva de la traducción al latín de la palabra árabe **ALKHÔWARÎZMI**, nombre de un matemático y astrónomo árabe que escribió un tratado sobre la manipulación de números y ecuaciones en el siglo IX, titulado **KITAB AL-JABR W'ALMUGALABA**, la palabra álgebra se derivó por su semejanza sonora de **AL-JABR**.

Etimológicamente la palabra problema deriva del griego **PROBALLEIN** y significa “*algo lanzado hacia delante*”. Un problema es un asunto o conjunto de cuestiones que se plantean para ser resueltas, la naturaleza de los problemas varia con el ámbito o con el contexto donde están planteados: así existen problemas matemáticos, físicos, filosóficos, etc.,

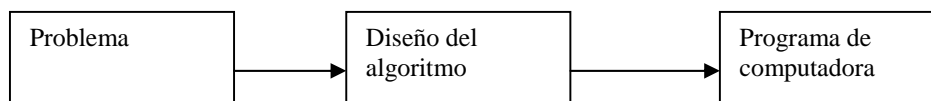


Fig.2 .1 Esquema del desarrollo de algoritmos

2.2 Definición

“Un Algoritmo es una secuencia de operaciones detalladas y no ambiguas, que al ejecutarse paso a paso, conducen a la solución de un problema”. En otras palabras es un conjunto de reglas para resolver una cierta clase de problema.

“Algoritmo es un conjunto de instrucciones que especifican la secuencia de operaciones a realizar, en orden, para resolver un sistema específico o clase de problema”.

“Un Algoritmo es la aplicación de pasos lógicos, secuenciales y metódicamente aplicados para dar solución a un problema en cuestión.” En otras palabras un algoritmo es una formula para resolver problemas.

“En otras palabras un algoritmo es una formula para la solución de un problema.”

“Todo problema se puede describir por medio de un algoritmo “

“ Todo algoritmo es independiente del lenguaje”

2.3 Características De Los Algoritmos.

Las propiedades de un algoritmo son las siguientes:

- a) El algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- b) El algoritmo debe ser definido, si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- c) El algoritmo debe ser finito, si se sigue un algoritmo se debe terminar en algún momento; o sea debe tener un número finito de pasos.

El algoritmo debe ser planteado como un sistema de información.

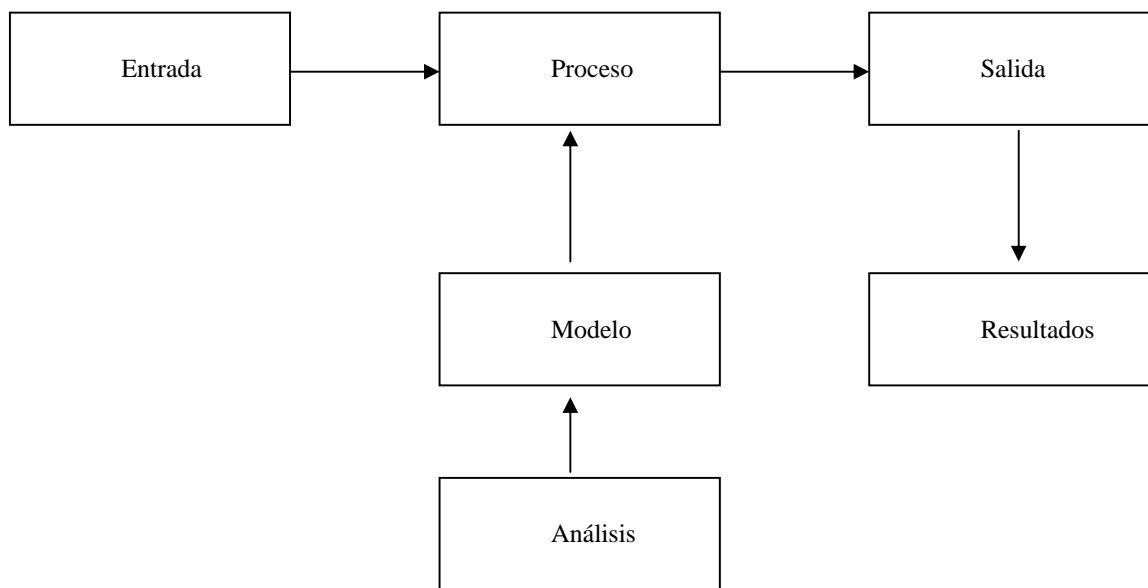


Fig. 2.2 Representación de un algoritmo como un sistema de Información

2.4 Clasificación De Los Algoritmos

Se puede clasificar tomando en cuenta dos aspectos.

- Secuenciales
- Condicionales
- Repetitivos

2.5 Tipos De Algoritmos

Cualitativos: Son aquellos en los que se describen los pasos utilizando palabras.

Cuantitativos: Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

2.6 Lenguajes Algorítmicos

Es una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso.

2.7 Tipos De Lenguajes Algorítmicos

- Gráficos: Es la representación gráfica de las operaciones que realiza un algoritmo (diagrama de flujo).
- No Gráficos: Representa en forma descriptiva las operaciones que debe realizar un algoritmo (pseudocódigo).

Un algoritmo puede ser expresado de las siguientes formas.

- a) Lenguaje Natural : el uso de términos del lenguaje natural, es una forma de representar un algoritmo.
- b) Lenguaje Simbólico: es otra forma de representación de un algoritmo, que además permite una introducción a la programación estructural.
- c) Lenguaje Gráfico : es una forma de escribir una secuencia de pasos en forma de diagrama, en la practica se denomina Diagramas de Flujo.

Una receta de un plato de cocina se puede expresar en español, inglés o francés pero cualquiera sea el lenguaje los pasos para la elaboración del plato se realizarán sin importar el cocinero.

2.8 Las Tecnicas De Diseño De Algoritmos

El diseño de la mayoría de los algoritmos requiere creatividad y conocimientos de la metodología de programación, esto significa que debe desarrollar una lógica computacional a través de la resolución de programas.

Y principalmente en diseño de un algoritmo debe realizarse apartir de un análisis del problema.

El acto de diseñar un algoritmo puede considerarse como una tarea que difícilmente podrá ser del todo automatizada, todo problema algoítmico es un reto para su diseñador. Algunos

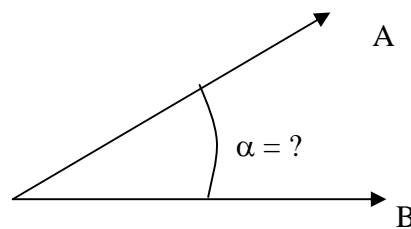
resultan inmediatos de resolver, otros son bastante complejos. la investigación en esta área ha permitido descubrir un conjunto de métodos y esquemas de diseño hacia los cuales puede orientarse la realización de muchos algoritmos. Idear un algoritmo continua siendo una labor bastante creativa donde los conocimientos y las experiencias del propio diseñador tiene un papel fundamental.

2.8.1 Divide y vencerás

Consiste en descomponer un problema en subproblemas, resolver independientemente los subproblemas para luego combinar sus soluciones y obtener una solución del problema original.

Esta técnica puede ser utilizada con éxito en problemas como multiplicación de matrices, ordenación de vectores, etc.

Ejemplo. Encontrar el ángulo entre dos vectores $A(x,y,z)$ y $B(x,y,x)$.



Determinar la ecuación del ángulo entre dos vectores. del producto escalar se tiene

$$\vec{A} * \vec{B} = |A| * |B| * \text{COS} (\alpha)$$

$$\text{COS } \alpha = \frac{\vec{A} * \vec{B}}{|A| * |B|}$$

Determinar las operaciones intermedias

$$A = |A| = \sqrt{Ax^2 + Ay^2 + Az^2}$$

$$B = |B| = \sqrt{Bx^2 + By^2 + Bz^2}$$

$$N = \vec{A} * \vec{B} = Ax * Bx + Ay * By + Az * Bz$$

$$D = A * B$$

Determinar la solución general.

$$ANG = \cos^{-1}(N / D) = ARCCOS (N / D)$$

2.8.2 Método voraz

Este método trata de producir algún tipo de mejor resultado a partir de un conjunto de opciones candidatas. Par ello se va procediendo paso a paso realizándose la mejor elección (utilizando un función objetivo que respeta un conjunto de restricciones) de entre las posibles. Puede emplearse en problemas de optimización, como el conocido de la mochila, en la búsqueda de caminos mínimos sobre grafos, la planificación en la ejecución de programas en un computador, etc.

Ejemplo Dar cambio utilizando el menor número de billetes.

```
Conjunto de billetes = {"doscientos", "cien", "cincuenta", "veinte", "diez", "cinco", "dos", "uno"};
Conjunto de cantidades = {200, 100, 50, 20, 10, 5, 2, 1};
Cambiar (dato, posición)
    si (dato es distinto de 0) entonces
        cuantos = dato DIV cantidades[posición];
        si (cuantos es distinto de 0) entonces
            imprimir ("hay ", cuantos, " billetes de ", billetes [ posición]);
        fin_si
        Cambiar (dato MOD cantidades[posición], posición+1);
    Fin_si
Fin_cambiar
inicio
    cambiar(1388,0);
fin
```

El estilo y calidad de los algoritmos van fuertemente unidos, ante la pregunta “Cuales son las características de un buen algoritmo?” la respuesta identifica los factores de calidad de los algoritmos.

- Corrección: el algoritmo debe funcionar.
- Eficiencia : el algoritmo no debe desaprovechar recursos.
- Claridad : el algoritmo debe estar bien documentado.

2.8.3 Top Down

También conocida como de arriba-abajo (diseño descendente) y consiste en establecer una serie de niveles de mayor a menor complejidad (arriba-abajo) que den solución al

problema. Consiste en efectuar una relación entre las etapas de la estructuración de forma que una etapa jerárquica y su inmediato inferior se relacionen mediante entradas y salidas de información.

Este diseño consiste en una serie de descomposiciones sucesivas del problema inicial, que recibe el refinamiento progresivo del repertorio de instrucciones que van a formar parte del programa.

La utilización de la técnica de diseño Top-Down tiene los siguientes objetivos básicos:

Simplificación del problema y de los subprogramas de cada descomposición.

Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.

El programa final queda estructurado en forma de bloque o módulos lo que hace mas sencilla su lectura y mantenimiento.

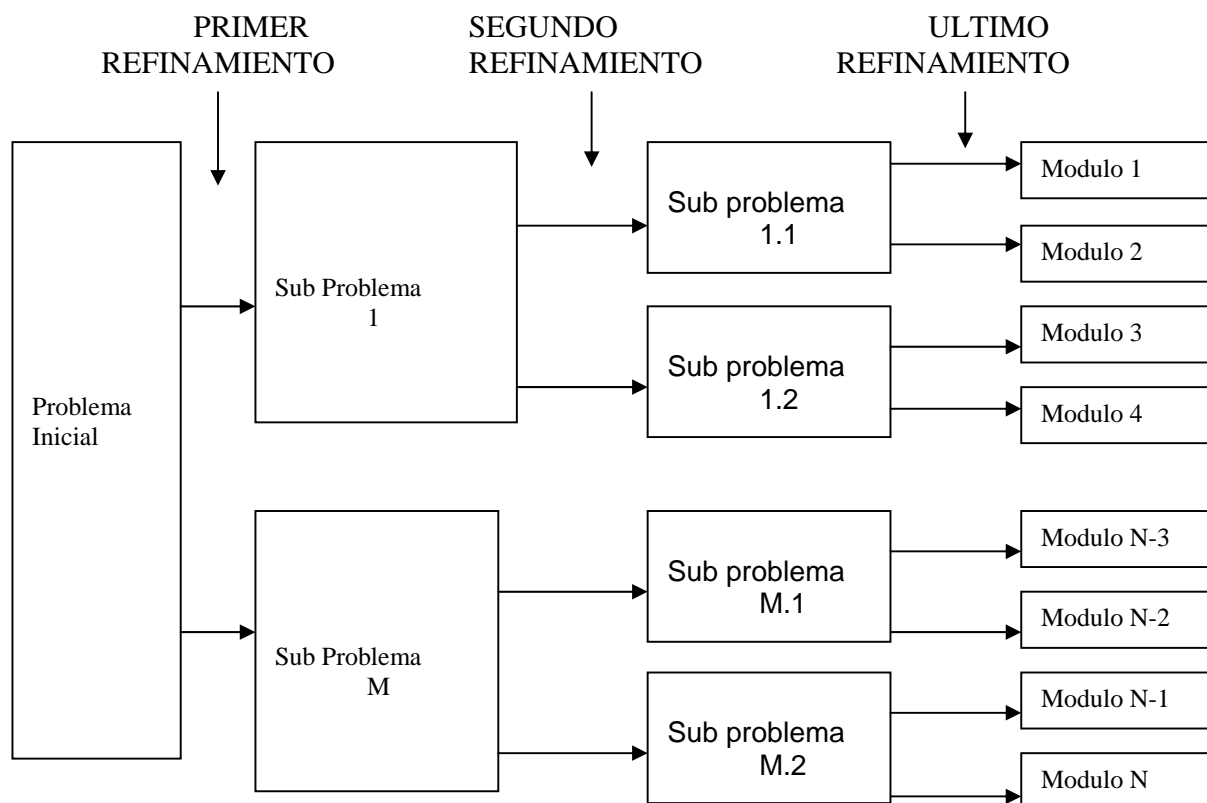


Fig. 2.3 Diseño Top Down

2.8.4 Bottom Up

El diseño ascendente se refiere a la identificación de aquellos procesos que necesitan computarizarse con forme vayan apareciendo, su análisis como sistema y su codificación, o bien, la adquisición de paquetes de software para satisfacer el problema inmediato.

Cuando la programación se realiza internamente y haciendo un enfoque ascendente, es difícil llegar a integrar los subsistemas al grado tal de que el desempeño global, sea fluido. Los problemas de integración entre los subsistemas son sumamente costosos y muchos de ellos no se solucionan hasta que la programación alcanza la fecha límite para la integración total del sistema. En esta fecha, ya se cuenta con muy poco tiempo, presupuesto o paciencia de los usuarios, como para corregir aquellas delicadas interfaces, que en un principio, se ignoran.

Aunque cada subsistema parece ofrecer lo que se requiere, cuando se contempla al sistema como una entidad global, adolece de ciertas limitaciones por haber tomado un enfoque ascendente. Uno de ellos es la duplicación de esfuerzos para acceder el software y más aun al introducir los datos. Otro es, que se introducen al sistema muchos datos carentes de valor. Un tercero y tal vez el más serio inconveniente del enfoque ascendente, es que los objetivos globales de la organización no fueron considerados y en consecuencia no se satisfacen.

2.9 Metodología Para La Solución De Problemas Por Medio De Computadora

La principal razón para que las personas aprendan lenguajes de programación es utilizar la computadora como una herramienta para la resolución de problemas. Dos fases pueden ser identificadas en el proceso de resolución de problemas.

- Fase de resolución del problema.
- Fase de implementación (realización) en un lenguaje de programación.

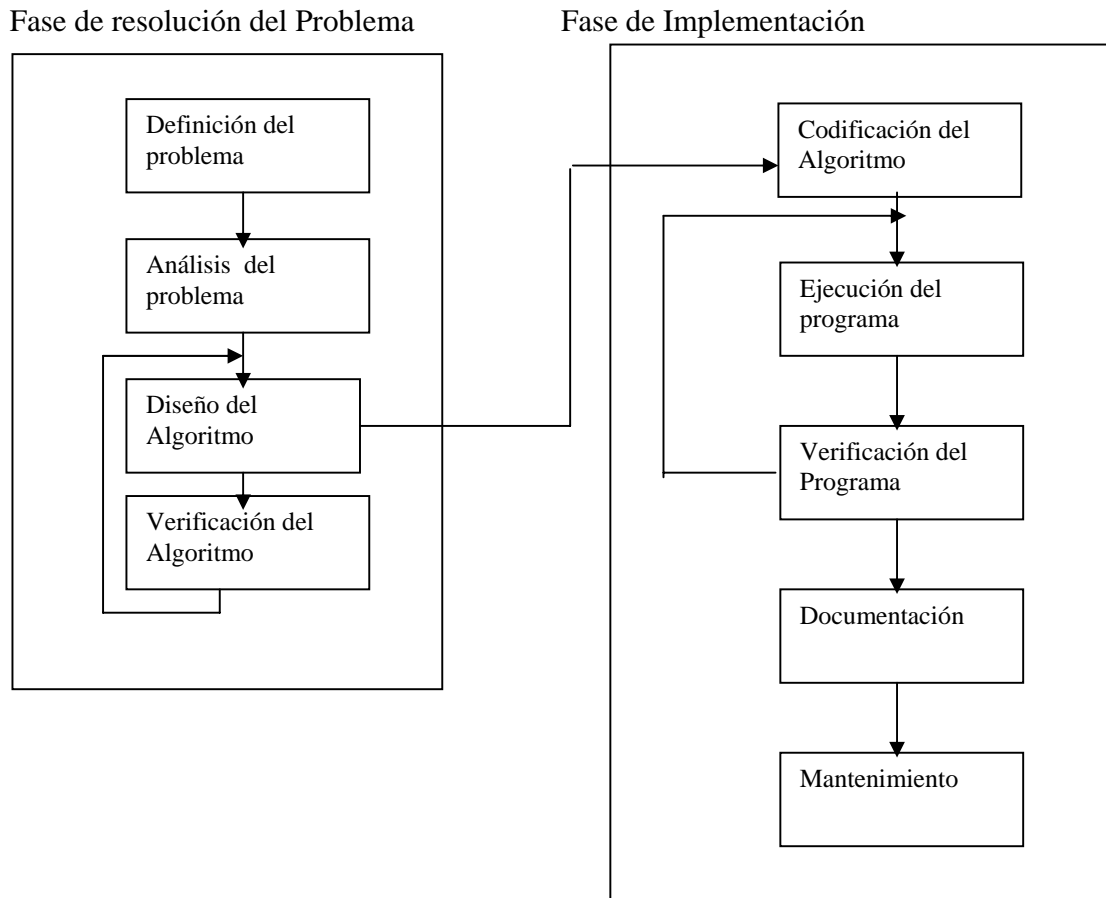


Fig. 2.4 Solución de problemas empleando Computadoras

2.9.1 Definición del Problema

Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa.

Tomar en cuenta que la solución del problema debe estar en función de lo que el problema requiera y no en función de lo que el programador quiera.

Es importante que se conozca lo que se desea que realice la computadora; mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa.

2.9.2 Análisis del Problema

No existe un método general para la resolución de problemas, la resolución de un problema es un proceso creativo donde el conocimiento, la habilidad y la experiencia tiene un papel importante, el proceder de manera sistemática ayuda a resolver un problema.

Al comenzar a abordar un problema hay que tener en cuenta que para la mayoría de ellos hay muchas maneras de resolver y pueden existir muchas soluciones, se plantean algunos criterios o estrategias generales que se deben tomar en cuenta, las cuales son útiles en el análisis del problema.

- Usar toda la información útil (no superficial) disponible en el enunciado del problema.
- Hacer explícita las reglas y datos que aparezcan implícitos (en muchos problemas numéricos se pueden utilizar reglas de la aritmética o álgebra)
- Profundizar en el problema considerado (emplear algún tipo de notación).
- Dividir el problema complejo en subproblemas más simples. que se pueden resolver independientemente y después combinar sus soluciones.
- Otra forma de abordar el problema consiste en trabajar “hacia atrás” es decir partir de la solución e intentar llegar al estado inicial.

El propósito del análisis del problema es ayudar al programador para llegar a una cierta comprensión de la naturaleza del problema. Una buena definición del problema junto con una descripción detallada de las especificaciones de entrada y de salida, son los requisitos más importantes para llegar a una solución eficaz.

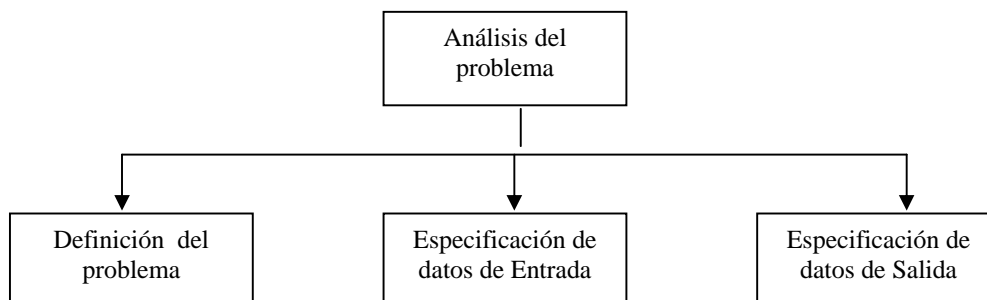


Fig. 2.5 Análisis del Problema

Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:

- Los datos de entrada.
- Cual es la información que se desea producir (salida)
- Los métodos y fórmulas que se necesitan para procesar los datos.

Una recomendación muy practica es el que nos pongamos en el lugar de la computadora y analicemos que es lo que necesitamos que nos ordenen y en que secuencia para producir los resultados esperados.

Ejercicio.- Leer el radio de circulo y calcular e imprimir su superficie.

Definición del problema.

Calcular la superficie de una circunferencia

Análisis del problema

La entrada a este problema es el radio de la circunferencia y de tipo real.

La salida de este problema es la superficie que también es de tipo real.

2.9.3 Diseño del Algoritmo

Un computador no tiene capacidad para solucionar problemas más que cuando se le proporciona los sucesivos pasos a realizar. Estos pasos indican las instrucciones a ejecutar por la máquina y se denomina algoritmo.

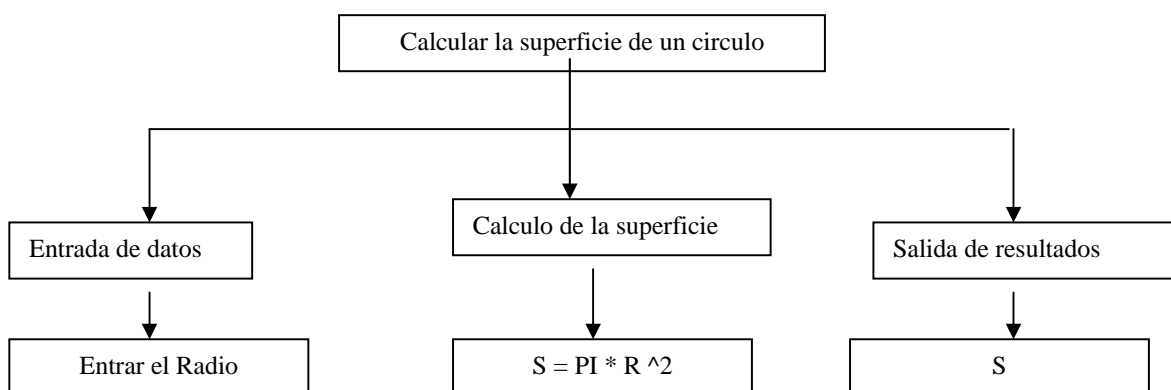
Las características de un buen algoritmo son:

- Debe tener un punto particular de inicio.
- Debe ser definido, no debe permitir dobles interpretaciones.
- Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.
- Debe ser finito en tamaño y tiempo de ejecución.

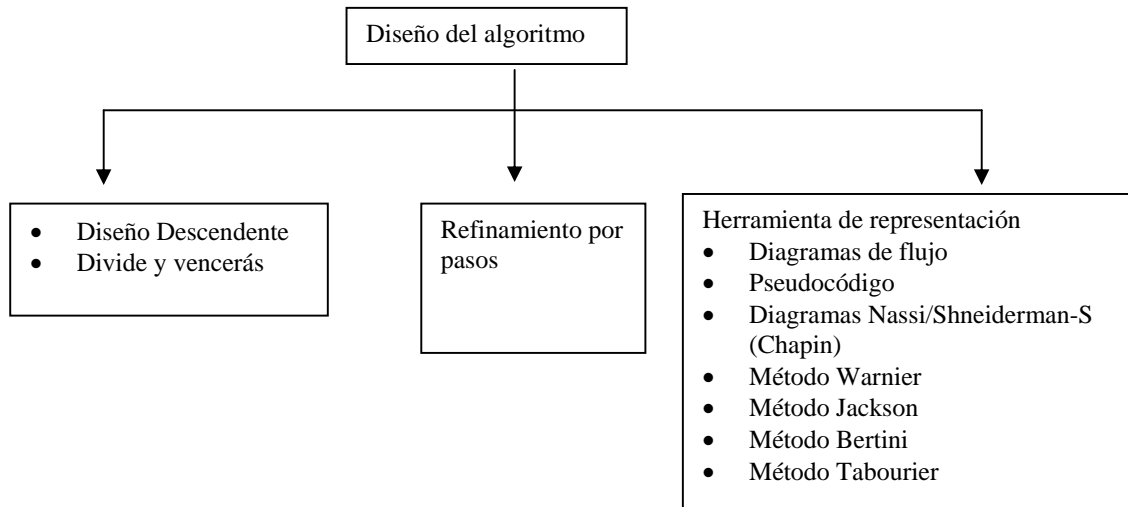
Se pueden utilizar cualquier técnica de diseño de algoritmos, diseño descendente, divide y vencerás.

Normalmente los pasos diseñados en un primer esbozo del algoritmo son incompletos e indican solo unos pocos pasos, tras esta primera descripción estos se amplían en una descripción más detallada con pasos específicos este proceso se denomina refinamiento del algoritmo.

Ejemplo problema calculo de la superficie de una circunferencia.



En el diseño de un algoritmo tomamos en cuenta el siguiente desarrollo.



2.9.4 Verificación del algoritmo

Una vez que se ha terminado de escribir un algoritmo es necesario comprobar que realiza las tareas para las que ha sido diseñado y produce el resultado correcto y esperado.

El modo más normal de comprobar un algoritmo es mediante la ejecución manual (prueba de escritorio), usando datos significativos que abarquen todo el posible rango de valores y anotando en una hoja de papel las modificaciones que se producen en las diferentes fases hasta la obtención de los resultados,

2.9.5 Fase de Implementación

Una vez que el algoritmo está diseñado, representado gráficamente mediante una herramienta y verificado se debe pasar a la fase de codificación, traducir el algoritmo a un determinado lenguaje de programación que deberá ser completada con la ejecución y verificación de resultado en el computador.

2.9.6 Codificación

La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por la computadora, la serie de instrucciones detalladas se le conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

2.9.7 Prueba y Depuración

Los errores humanos dentro de la programación de computadoras son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama depuración.

La depuración o prueba resulta una tarea tan creativa como el mismo desarrollo de la solución, por ello se debe considerar con el mismo interés y entusiasmo.

Resulta conveniente observar los siguientes principios al realizar una depuración, ya que de este trabajo depende el éxito de nuestra solución.

2.9.8 Documentación

Es la guía o comunicación escrita en sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas.

A menudo un programa escrito por una persona, es usado por otra. Por ello la documentación sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones (mantenimiento).

La documentación se divide en tres partes:

- Documentación Interna
- Documentación Externa
- Manual del Usuario

Documentación Interna: Son los comentarios o mensaje que se añaden al código fuente para hacer más claro el entendimiento de un proceso.

Documentación Externa: Se define en un documento escrito los siguientes puntos:

- Descripción del Problema
- Nombre del Autor
- Algoritmo (diagrama de flujo o pseudocódigo)
- Diccionario de Datos
- Código Fuente (programa)

Manual del Usuario: Describe paso a paso la manera como funciona el programa, con el fin de que el usuario obtenga el resultado deseado.

2.9.9 Mantenimiento

Se lleva a cabo después de terminado el programa, cuando se detecta que es necesario hacer algún cambio, ajuste o complementación al programa para que siga trabajando de manera correcta. Para poder realizar este trabajo se requiere que el programa este correctamente documentado.

2.10 Tecnicas Para La Formulacion De Algoritmos

Para representar un algoritmo se debe utilizar algún método que permita independizar dicho algoritmo del lenguaje de programación elegido. Ello permitirá que un algoritmo pueda ser codificado inmediatamente en cualquier lenguaje.

Las herramientas utilizadas comúnmente para diseñar algoritmos son:

- Pseudocódigo
- Diagrama de Flujo.
- Diagramas Nassi/Shneiderman-S (Chapin)
- Método Warnier
- Método Jackson
- Método Bertini
- Método Tabourier

2.10.1 Pseudocódigo

Mezcla de lenguaje de programación y español (o inglés o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa. En esencia, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El pseudocódigo utiliza palabras que indican el proceso a realizar.

El inicio de un algoritmo en pseudocódigo comienza con la palabra Inicio y termina con la palabra fin.

Las líneas que están entre llaves ({ }) se denomina comentario.

Un ejemplo aclaratorio es el siguiente. Calcular el área de un cuadrado.

```

Inicio
  Leer (lado)
  A ← lado * lado
  Imprimir( A)
Fin
  
```

2.10.1.1 Acciones simples

Las acciones simples, también denominadas instrucciones primitivas, son aquellas que el procesador ejecuta de forma inmediata.

- **Asignación** Almacena en una variable el resultado de evaluar una expresión

Variable ← expresión

- **Entrada** Toma un dato del dispositivo de entrada

Leer (Variable)

- **Salida** Devuelve un dato al dispositivo de salida

Imprimir (variable)

2.10.1.2 Sentencias de control

También se llaman sentencias estructuradas y controlan el flujo de ejecución de otras instrucciones.

- **Secuencia.** Se ejecutan instrucciones de I1,I2,...,In en el mismo orden en el que aparece

I1,I2,...,I3

- **Alternativa.** En esta instrucción la condición es booleana

Si condición entonces
 I1,I2,...,In

Fin_si

Si condición
 entonces I1,I2,...,In
 Sino J1, J2,...,Jn

Fin_si

| Opción | Expresión | de |
|--------|-----------|--------------|
| V1 | Hacer | I1,I2,...,In |
| V2 | Hacer | J1,J2,...,In |
| V3 | Hacer | K1,K2,...,Kn |
| ... | | |
| VN | Hacer | L1,L2,...,Ln |
| Otro | Hacer | M1,M2,...,Mn |

Fin_opción

- **Repetición o bucles.** En un bucle hay una o varias acciones que se han de repetir y una condición que determina el número de veces que se repiten las instrucciones.

Mientras

Mientras condición hacer
I1,I2,...,In
Fin_mientras

Repetir

Repetir
I1,I2,...,In
Hasta condición

Para

Para Variable de Valinc a ValFin hacer
I1,I2,...,In
Fin_para

2.10.2 Diagrama De Flujo

Un diagrama de flujo es la representación gráfica de un algoritmo. También se puede decir que es la representación detallada en forma gráfica de como deben realizarse los pasos en la computadora para producir resultados.

Esta representación gráfica se da cuando varios símbolos (que indican diferentes procesos en la computadora), se relacionan entre si mediante líneas que indican el orden en que se deben ejecutar los procesos.





2.10.2.1 Características


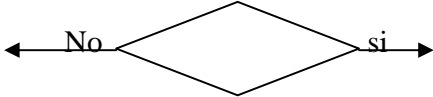
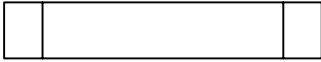


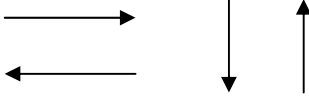
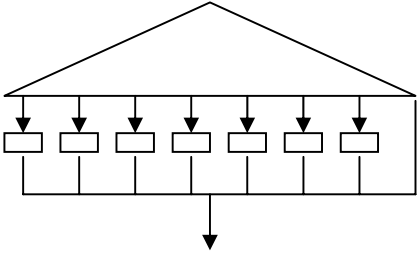
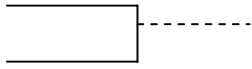
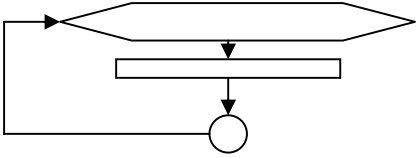
Toda representación gráfica, de cualquier tipo sea, debe cumplir las siguientes cualidades.

- Sencillez. Un método gráfico de diseño de algoritmo debe permitir la construcción de estos de manera fácil y sencilla
- Claridad. Cuando un algoritmo es representado por un método gráfico necesita ser interpretado por otra persona distinta de la que lo diseñó, debe estar lo suficientemente claro para su un fácil reconocimiento de todos los elementos.
- Normalización. Tanto los diseñadores de programas como los usuarios que necesitan la documentación de estos deben utilizar las mismas normas de documentación.
- Flexibilidad. Todo método gráfico de representación debe permitir, sin grandes dificultades, posteriores modificaciones de algunas partes de un algoritmo y la inserción de alguna nueva.

2.10.2.2 Descripción de los bloques utilizados

Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI).

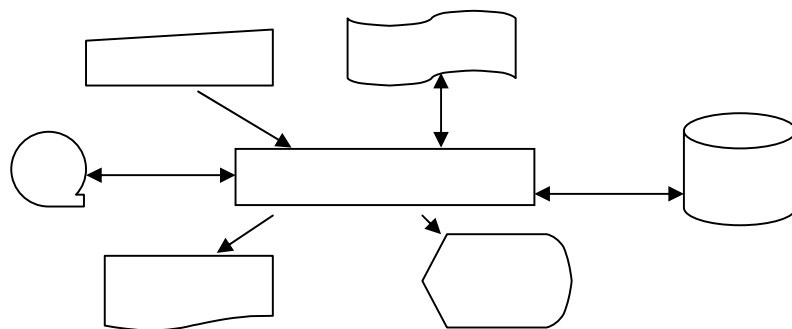
| | |
|---|---|
| Terminal. Indica el inicio y el final de nuestro diagrama de flujo. |  |
| Indica la entrada y salida de datos. |  |
| Indican la entrada de datos |  |
| Indican salida de Datos |  |

| | |
|--|--|
| <p>Símbolo de proceso y nos indica la asignación de un valor en la memoria y/o la ejecución de una operación aritmética.</p> |  |
| <p>Símbolo de decisión indica la realización de una comparación de valores.</p> |  |
| <p>Subprogramas</p> |  |
| <p>Conector dentro de pagina. Representa la continuidad del diagrama dentro de la misma pagina.</p> |  |
| <p>Conector de pagina. Representa la continuidad del diagrama en otra pagina.</p> |  |
| <p>Líneas de flujo o dirección. Indican la secuencia en que se realizan las operaciones.</p> |  |
| <p>Símbolo de decisión , con opciones múltiples</p> |  |
| <p>comentarios</p> |  |
| <p>Ciclo repetitivo para</p> |  |

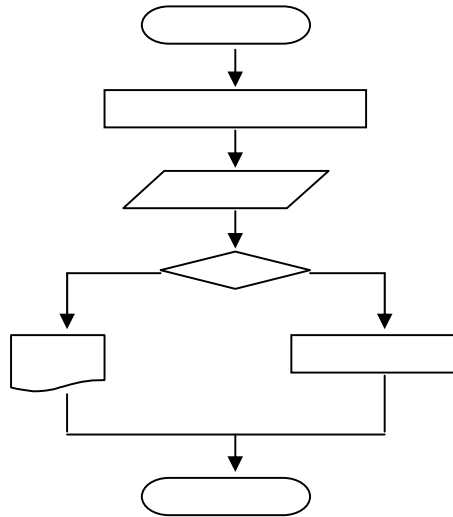
| | |
|----------------------------------|--|
| <p>Ciclo repetitivo Mientras</p> | |
| <p>Ciclo repetitivo Repetir</p> | |

Diferencia entre organigrama y ordinograma (D.F.)

Organigrama → diagramas de flujo del sistema.



Ordinogramas → diagramas de flujo del programa



2.10.2.3 Recomendaciones para el diseño de Diagramas de Flujo

Se deben usar solamente líneas de flujo horizontales y/o verticales.

Se debe evitar el cruce de líneas utilizando los conectores.

Se deben usar conectores solo cuando sea necesario.

No deben quedar líneas de flujo sin conectar.

Se deben trazar los símbolos de manera que se puedan leer de arriba hacia abajo y de izquierda a derecha.

Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.

2.10.2.4 Formato de todo tipo de Diagramas de Flujo

Todo diagrama de flujo, en general, consta de los siguientes pasos.

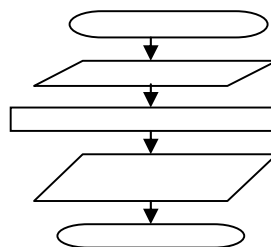


Fig. 2.6 Esquema de un diagrama de flujo

2.10.2.5 Ventajas De Utilizar Un Pseudocodigo A Un Diagrama De Flujo

Ocupa menos espacio en una hoja de papel

Permite representar en forma fácil operaciones repetitivas complejas

Es muy fácil pasar de pseudocodigo a un programa en algún lenguaje de programación.

Si se siguen las reglas se puede observar claramente los niveles que tiene cada operación.

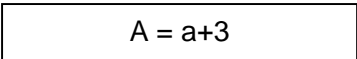
2.10.3 Diagramas Estructurados (Nassi-Schneiderman)

El diagrama estructurado N-S también conocido como diagrama de chapin es como un diagrama de flujo en el que se omiten las flechas de unión y las cajas son contiguas. Las acciones sucesivas se pueden escribir en cajas sucesivas y como en los diagramas de flujo, se pueden escribir diferentes acciones en una caja. Un algoritmo se representa en la sig. forma:

2.10.3.1 Acciones simples

Las acciones simples, también denominadas instrucciones primitivas, son aquellas que el procesador ejecuta de forma inmediata.

- **Asignación** Almacena en una variable el resultado de evaluar una expresión



A = a+3

- **Entrada** Toma un dato del dispositivo de entrada



Leer A,C

- **Salida** Devuelve un dato al dispositivo de salida



Escribir a,b

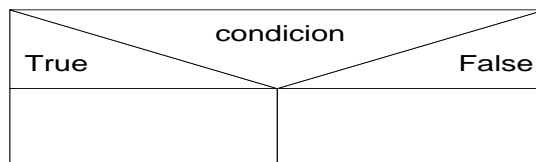
2.10.3.2 Sentencias de control

También se llaman sentencias estructuradas y controlan el flujo de ejecución de otras instrucciones.

- **Secuencia.** Se ejecutan instrucciones de I1,I2,...,In en el mismo orden en el que aparece



- **Alternativa.** En esta instrucción la condición es booleana



| Condicion | | | | | | | |
|-----------|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | | | | | | | |

- **Repetición o bucles.** En un bucle hay una o varias acciones que se han de repetir y una condición que determina el número de veces que se repiten las instrucciones.

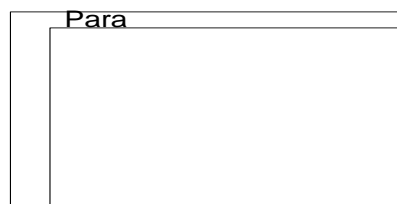
Mientras



Repetir



Para



2.11 Otras Herramientas De La Programación Estructurada

También existen otros métodos de representar los algoritmos y son los siguientes.

2.11.1 Método Warnier

El método se basa en el empleo de llaves de distintos tamaños que se relacionan entre sí.

La representación del algoritmo se basa en los siguientes puntos.

- Un programa se representa por un solo diagrama en la cual se engloban todas las operaciones, estas operaciones están colocadas a la derecha de la llave y en la parte izquierda se encuentra el nombre.
- En la parte superior de la llave principal se coloca inicio
- En la parte inferior de la llave principal se coloca fin

2.11.1.2 Acciones simples

Las acciones simples, también denominadas instrucciones primitivas, son aquellas que el procesador ejecuta de forma inmediata.

- **Asignación** Almacena en una variable el resultado de evaluar una expresión

Variable ← expresión

- **Entrada** Toma un dato del dispositivo de entrada

Leer (Variable)

- **Salida** Devuelve un dato al dispositivo de salida

Imprimir (variable)

2.11.1.3 Sentencias de control

También se llaman sentencias estructuradas y controlan el flujo de ejecución de otras instrucciones.

- **Secuencia.** Se ejecutan instrucciones de I1,I2,...,In en el mismo orden en el que aparece

$$\left\{ \begin{array}{l} I1 \\ I2 \\ \dots \\ I3 \end{array} \right.$$

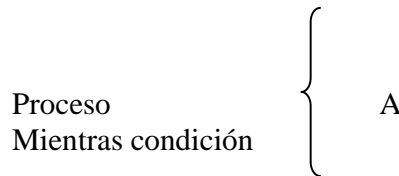
- **Alternativa.** En esta instrucción la condición es booleana

$$\text{Condición} \left\{ \begin{array}{l} \text{Si} \quad \{ I1 \\ \text{No} \quad \{ I2 \end{array} \right.$$

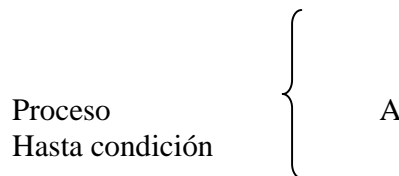
$$\left\{ \begin{array}{l} \text{Expresión} = V1 \quad \{ A \\ \text{Expresión} = V2 \quad \{ A \\ \text{Expresión} = V3 \quad \{ A \\ \text{Expresión} = \text{Otros} \quad \{ A \end{array} \right.$$

- **Repetición o bucles.** En un bucle hay una o varias acciones que se han de repetir y una condición que determina el número de veces que se repiten las instrucciones.

Mientras



Repetir



Para

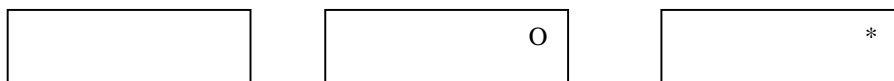


2.11.2 Método Jackson

Se trata de un método de representación de programa en forma de árbol denominado diagrama arborescente de Jackson, un diagrama de Jackson consta de :

- Definición detallada de los datos de entrada y salida incluyendo los archivos lógicos utilizados.
- Representación del proceso o algoritmo.

La simbología utilizada es la siguiente



La lectura del árbol se realiza en preorden

- Situar en la raíz (R)
- Recorrer el subárbol izquierdo (I)
- Recorrer el subárbol derecho (D)

2.11.2.1 Acciones simples

Las acciones simples, también denominadas instrucciones primitivas, son aquellas que el procesador ejecuta de forma inmediata.

- **Asignación** Almacena en una variable el resultado de evaluar una expresión

Variable ← expresión

- **Entrada** Toma un dato del dispositivo de entrada

Leer (Variable)

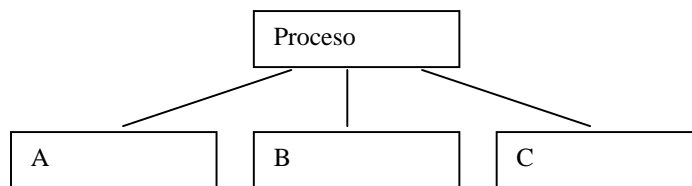
- **Salida** Devuelve un dato al dispositivo de salida

Imprimir (variable)

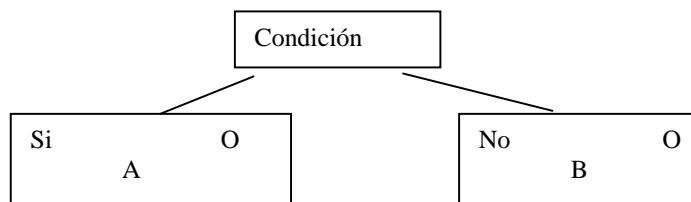
2.11.2.2 Sentencias de control

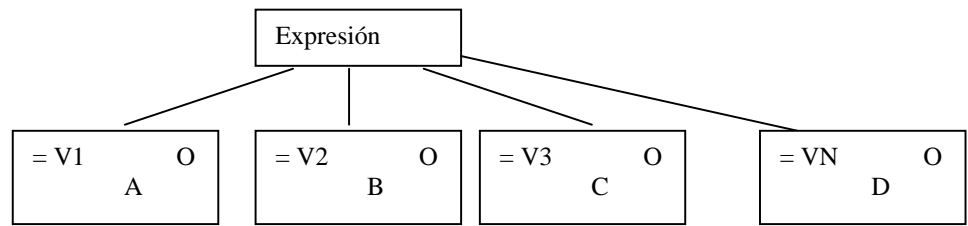
También se llaman sentencias estructuradas y controlan el flujo de ejecución de otras instrucciones.

- **Secuencia.** Se ejecutan instrucciones de I1,I2,...,In en el mismo orden en el que aparece

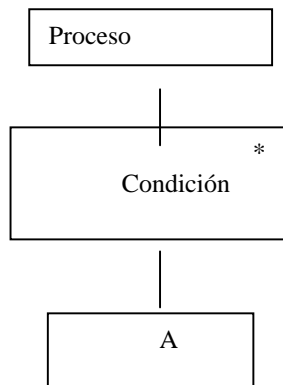


- **Alternativa.** En esta instrucción la condición es booleana
-





- **Repetición o bucles.** En un bucle hay una o varias acciones que se han de repetir y una condición que determina el número de veces que se repiten las instrucciones.



Mientras

En el bloque condición se escribe **mientras condición**

Repetir

En el bloque condición se escribe **hasta condición**

Para

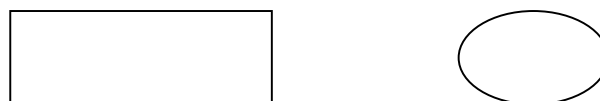
En el bloque condición se escribe **N veces**

2.11.3 Método Bertini

Al igual que Jackson, la representación de programas es en forma de árbol denominado diagrama arborescente de Bertini. Un diagrama de Bertini consta de.

- Definición detallada de los datos de entrada y salida.
- Representación del proceso o algoritmo.

La simbología utilizada es la siguiente.



La lectura del árbol se realiza en postorden

- Situarse en la raíz (R)
- Recorrer el subarbol Derecho (D)
- Recorrer el subarbol Izquierdo (I)

2.11.3.1 Acciones simples

Las acciones simples, también denominadas instrucciones primitivas, son aquellas que el procesador ejecuta de forma inmediata.

- **Asignación** Almacena en una variable el resultado de evaluar una expresión

Variable \leftarrow expresión

- **Entrada** Toma un dato del dispositivo de entrada

Leer (Variable)

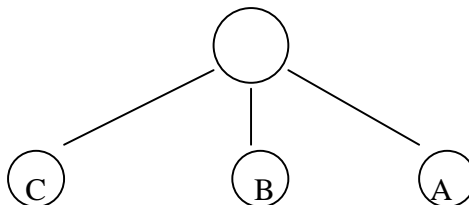
- **Salida** Devuelve un dato al dispositivo de salida

Imprimir (variable)

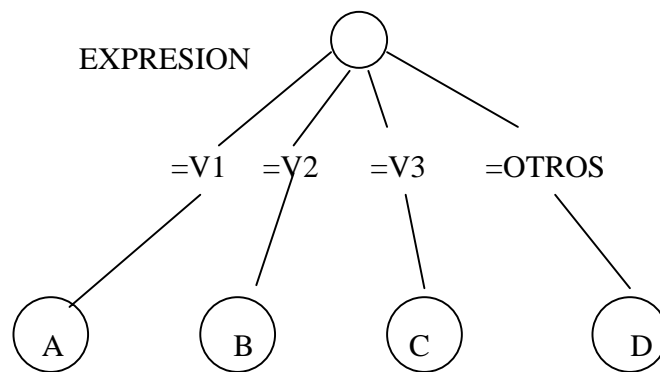
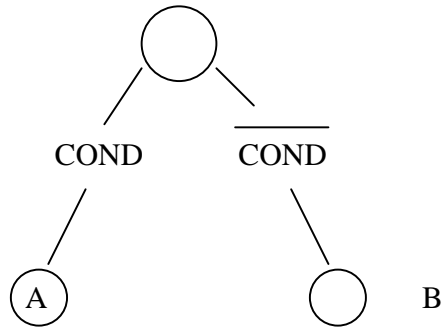
2.11.3.2 Sentencias de control

También se llaman sentencias estructuradas y controlan el flujo de ejecución de otras instrucciones.

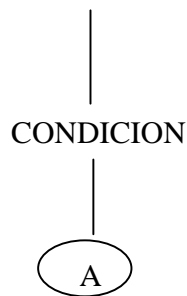
- **Secuencia.** Se ejecutan instrucciones de I1,I2,...,In en el mismo orden en el que aparece



- **Alternativa.** En esta instrucción la condición es booleana



- **Repetición o bucles.** En un bucle hay una o varias acciones que se han de repetir y una condición que determina el número de veces que se repiten las instrucciones.



Mientras

En el bloque condición se escribe **mientras condición**

Repetir

En el bloque condición se escribe **hasta condición**

Para

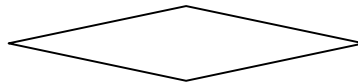
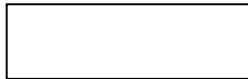
En el bloque condición se escribe **N veces**

2.11.4 Método Tabourier

Se trata de un método de representación de programa en forma de árbol denominado diagrama arborescente de Tabourier, un diagrama de Tabourier consta de :

- Definición detallada de los datos de entrada y salida incluyendo los archivos lógicos utilizados.
- Representación del proceso o algoritmo.

La simbología utilizada es la siguiente



La lectura del árbol se realiza en preorden

- Situar en la raíz (R)
- Recorrer el subárbol izquierdo (I)
- Recorrer el subárbol derecho (D)

2.11.4.1 Acciones simples

Las acciones simples, también denominadas instrucciones primitivas, son aquellas que el procesador ejecuta de forma inmediata.

- **Asignación** Almacena en una variable el resultado de evaluar una expresión

Variable ← expresión

- **Entrada** Toma un dato del dispositivo de entrada

Leer (Variable)

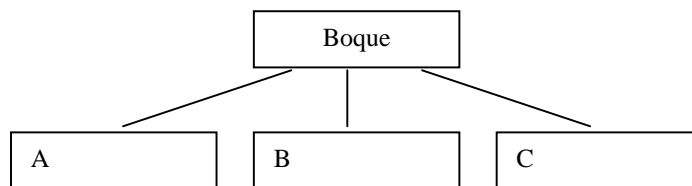
- **Salida** Devuelve un dato al dispositivo de salida

Imprimir (variable)

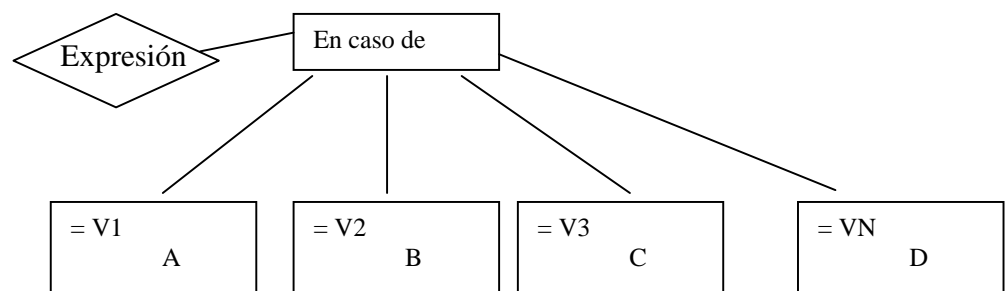
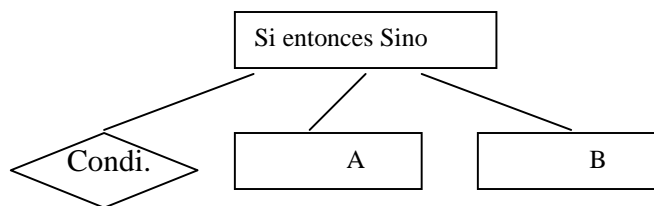
2.11.4.2 Sentencias de control

También se llaman sentencias estructuradas y controlan el flujo de ejecución de otras instrucciones.

- **Secuencia.** Se ejecutan instrucciones de I_1, I_2, \dots, I_n en el mismo orden en el que aparece

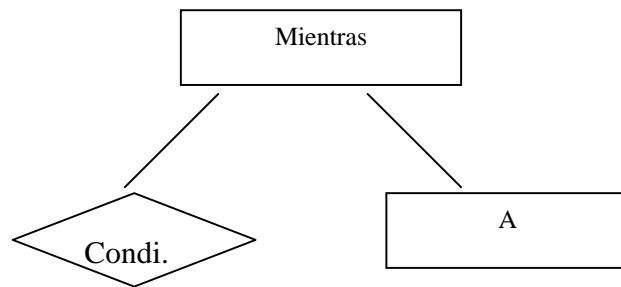


- **Alternativa.** En esta instrucción la condición es booleana

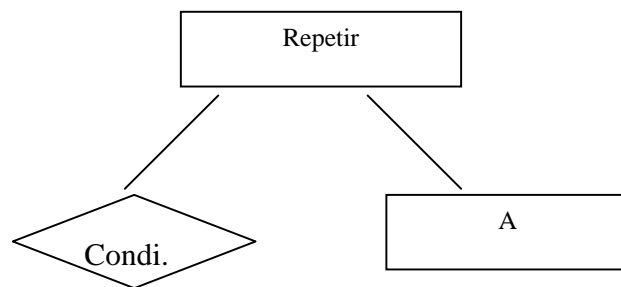


- **Repetición o bucles.** En un bucle hay una o varias acciones que se han de repetir y una condición que determina el número de veces que se repiten las instrucciones.

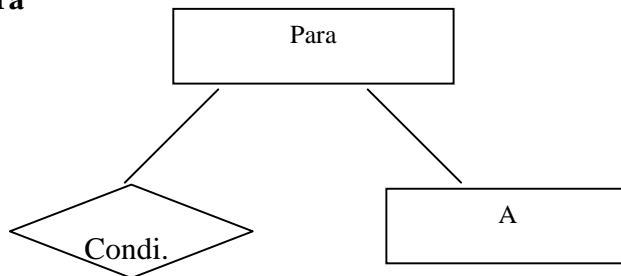
Mientras



Repetir



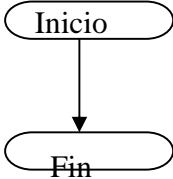
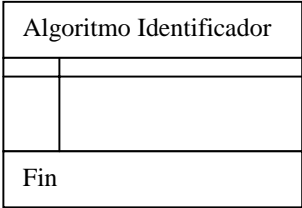
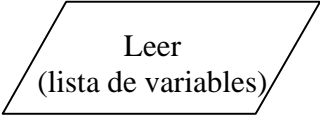
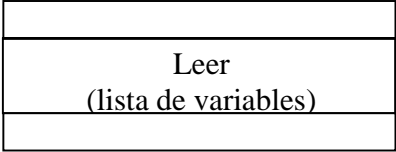
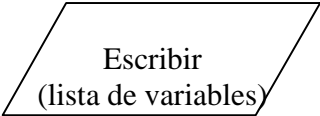
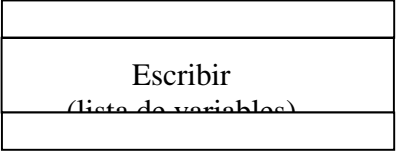
Para

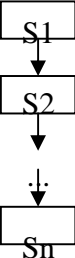
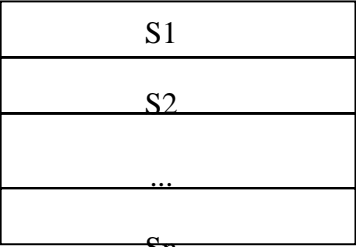
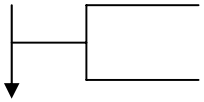
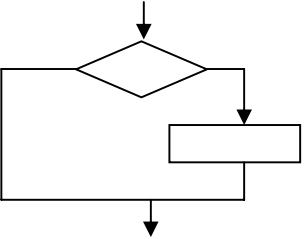
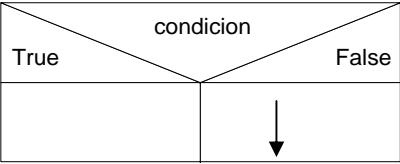
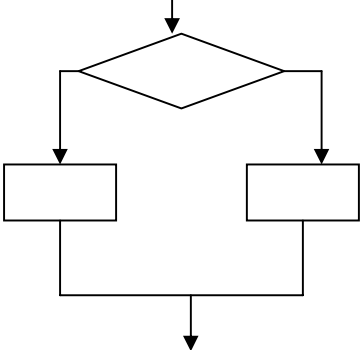
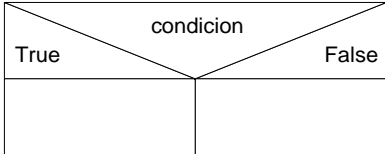


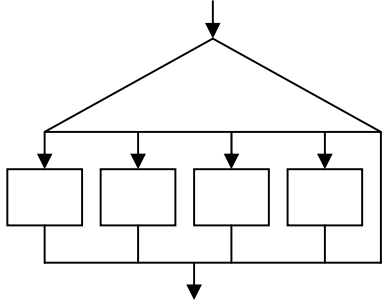
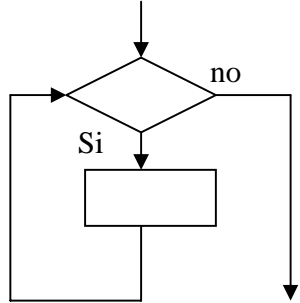
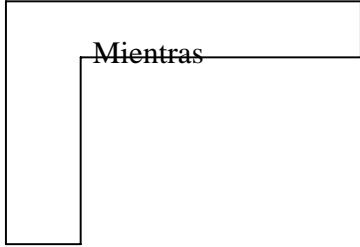
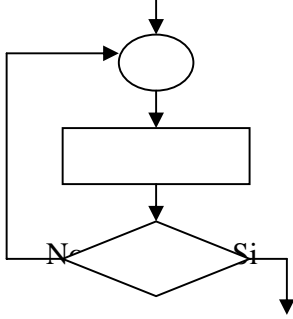
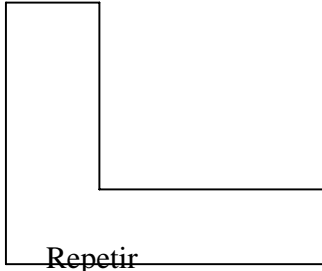
2.12 CONVERSIÓN DE ALGORITMOS

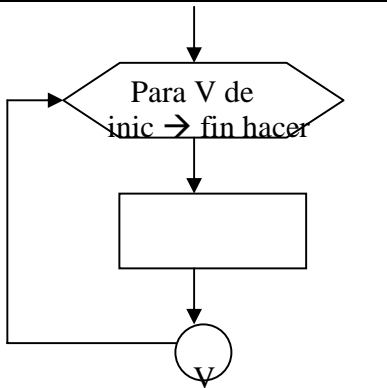
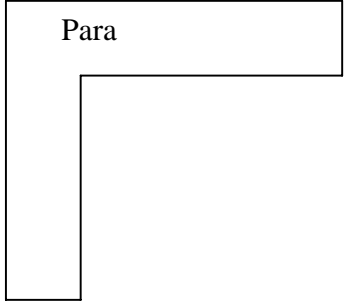
Se tiene las siguientes analogías entre las herramientas de presentación de algoritmos

- Diagramas de flujo
- Pseudocódigo
- Diagramas N-S

| Instrucciones, acciones | Pseudocódigo | Diagrama de flujo | Diagrama N-S |
|--------------------------|---|---|---|
| Estructura del Algoritmo | Algoritmo Identificador {Sección de declaración de variables} inicio fin |  |  |
| Declaración de variables | Descripción de variables y tipos en la tabla de variables | Se escriben las variables y tipos en la tabla de variables junto con el diagrama de flujo | Escritura de una tabla de variables |
| Asignación | Variable ← Expresión | Variable ← Expresión | Variable ← Expresión |
| Entrada de datos | Leer (lista de variables) |  |  |
| Salida de datos | Escribir (lista de variables) |  |  |

| | | | |
|-------------------------------------|---|--|--|
| <p>Instrucción compuesta</p> | <p>S1 S2 S3 ... SN</p> |  |  |
| <p>Comentarios</p> | <p>{ FASE DESCRIPTIVA DE COMENTARIO }</p> |  | |
| <p>Selectiva alternativa simple</p> | <p>Si condición entonces S1 S2 ... S3 Finsi</p> |  |  |
| <p>Selectiva alternativa doble</p> | <p>Si condición entonces S1 S2 ... S3 Sino S1 S2 ... S3 Finsi</p> |  |  |

| <p>Selectiva alternativa múltiple</p> | <p>Según_sea expresión hacer E1 : S1,S2,...Sn E2 : S1,S2,...Sn E3 : S1,S2,...Sn EN : S1,S2,...Sn En_otro_caso EN : S1,S2,...Sn Fin_según_sea</p> |  | <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="8">Condición</th> </tr> <tr> <th>1</th> <th>2</th> <th>3</th> <th>4</th> <th>5</th> <th>6</th> <th>7</th> <th>8</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> <td> </td> <td> </td> <td> </td> <td> </td> <td> </td> <td> </td> </tr> </tbody> </table> | Condición | | | | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | | | | | | |
|---------------------------------------|---|--|--|-----------|---|---|---|--|--|--|--|---|---|---|---|---|---|---|---|--|--|--|--|--|--|--|--|
| Condición | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <p>Repetitiva mientras</p> | <p>Mientras condición S1 S2 ... S3 Fin_Mientras</p> |  |  | | | | | | | | | | | | | | | | | | | | | | | | |
| <p>Repetitiva repetir</p> | <p>Repetir S1 S2 ... S3 Hasta condición</p> |  |  | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | |
|-------------------------------|---|---|---|
| <p>Repetitiva (desde)</p> | <p>para</p> <p>Para Var de ValInc Hasta ValFin [incremento x] hacer S1 S2 ... Sn Fin_Para</p> |  |  |
|-------------------------------|---|---|---|