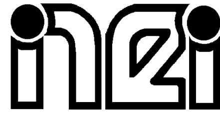


Instituto Nacional de Estadística
e Informática

TECNICAS DE PROGRAMACION

COLECCION INFORMATICA FACIL

INSTITUTO NACIONAL DE ESTADISTICA E INFORMATICA
Elaborado por la Sub-Jefatura de Informática
Dirección Técnica de Desarrollo Informático
Teléfono 433-4223 – anexos 314 - 315
Telefax 433-6106 – 433-5568
INTERNET *infoinei@inei.gob.pe*

Impreso en los talleres de la Oficina de Impresiones de la Oficina Técnica de Difusión Estadística y Tecnología Informática del Instituto Nacional de Estadística e Informática (INEI).

Edición	**:**
Dirección, Redacción y Talleres	**:** Av. General Garzón N° 658 Jesús María.
Orden	**:** 605 - 99 – OI - OTDTI

PRESENTACION

El Instituto Nacional de Estadística e Informática (INEI), como ente rector del Sistema Nacional de Informática, continuando con la publicación de la colección "Informática Fácil", presenta en esta oportunidad su Trigésimo Noveno Número titulado "Técnicas de Programación"

La presente publicación consta de cinco capítulos, los que se da una visión de las Técnicas de Programación Clásicas utilizadas para la práctica de esta disciplina. Pero antes se hace una revisión obligada de los fundamentos básicos de programación como son: los sistemas de procesamiento de información, el diseño de algoritmos sus características y conceptos, variables y tipos de datos primitivos, etc. También, el conocimiento de las herramientas de programación, de los programas y como están estructurados, los tipos de instrucciones. Además sobre las Técnicas de Programación, referidas a la Programación Modular, Programación Estructurada y los diferentes tipos de estructuras que existen. En el capitulo V, y último, se desarrolla el tema de la programación Orientada a Objetos (POO), y los conceptos que se encuentran dentro de este tipo de programación.

El Instituto Nacional de Estadística e Informática, pone a disposición de sus lectores la presente publicación, esperando que su aporte sea de gran utilidad en el conocimiento de la ciencia informática.

Econ. Félix Murillo Alfaro
JEFE
INSTITUTO NACIONAL DE ESTADISTICA E INFORMATICA

CONTENIDO

CAPITULO I - Algoritmos y Programas 9

1. Los sistemas de procesamiento de la información 9
2. Concepto de algoritmo 11
   2.1 Características de los Algoritmos 11
3. Los lenguajes de programación 15
   3.1 Instrucciones a la computadora 15
   3.2 Lenguajes máquina 16
   3.3 Lenguajes de bajo nivel 16
   3.4 Lenguajes de alto nivel 16
   3.5 Traductores de lenguaje 16
   3.6 Los compiladores y sus frases 16
4. Variables y tipos de datos primitivos. 17
   4.1 Datos numéricos. 17
   4.2 Datos lógicos. 25
   4.3 Datos tipo caracter. 25
   4.4 Datos tipo cadena. 25
5. Constantes variables 26
6. Expresiones 26
   6.1 Expresiones aritméticas. 26
   6.2 Reglas de prioridad 27
   6.3 Expresiones lógicas (booleanas) 27
7. Funciones internas. 27
8. La Operación de Asignación 28
9. Entrada y salida de la información 28

CAPITULO II - Computadoras y Herramientas de Programación Claves en la solución de problemas. 29

1. Solucionando el problema. 29
2. Análisis del problema. 29
3. Diseño del algoritmo. 29
   3.1 Escritura inicial del algoritmo 29
4. Solución de problemas por computadoras 30
5. Representación gráfica de los algoritmos. 30
   5.1 Diagrama de flujo. 31
   5.2 Diagrama de Nassi-Schneiderman (N-S) 32
   5.3 Pseudocódigo 32

CAPITULO III – Programa y su Estructura General 34
1. Programa. 34
2. Constitución de un programa y sus partes. 34
3. Instrucciones. 34
4. Tipos de instrucciones. 34
5. Programa y sus elementos básicos. 35
   5.1 Bucles 36
   5.2 Contadores 37
   5.3 Acumulador. 37
   5.4 Decisión o selección. 37
   5.5 Interruptores. 37
6. Escritura de algoritmos/programas 38
   6.1 Cabecera del programa o algoritmo. 38
   6.2 Declaración de variables. 38
   6.3 Declaración de constantes numéricas. 39
   6.4 Declaración de constantes y variables caracter. 39
   6.5 Comentarios. 39
   6.6 Estilo de escritura de algoritmos/programas. 39

CAPITULO IV – Las Técnicas de Programación 41
Introducción 41
1. Introducción a la programación estructurada 42
2. Programación modular. 42
   2.1 Medida de los módulos. 43
   2.2 Implementación de los módulos 43
3. Programación estructurada. 43
   3.1 Recursos abstractos. 44
   3.2 Diseño descendente (top-down) 44
   3.3 Teorema de la programación estructurada:
   Estructuras básicas. 44
4. Estructura secuencial. 45
5. Estructuras selectivas. 45
   5.1 Alternativa simple (si-entonces/if-then) 45
   5.2 Alternativa doble (si-entonces-sino/if-then-else) 46
   5.3 Alternativa múltiple (según-sea, casode/case) 46
6. Estructuras repetitivas. 47
   6.1 Estructura mientras (“while”) 47
   6.2 Estructura repetir (“repeat”) 48
   6.3 Estructura desde/para (“for”) 49
   6.4 Salidas internas de los Bucles 49
7. Estructuras de decisión anidadas 50
8. Estructuras repetitivas anidadas 50
9. Instrucción ir-a (“goto”) 50
10. Métodos de Programación Estructurada 51
    1. Método Jackson 51
    2. Método Bertiní 52
    3. Método Warnier 53

CAPITULO V – Programación Orientada a Objetos. 55
Introducción 55
1. Abstracción. 55
2. Encapsulamiento 55
3. Modularidad. 56
4. Clases y Objetos. 56
5. Herencia. 61
6. Polimorfismo. 63

Conclusiones 67
Bibliografía 68

TECNICAS DE PROGRAMACION

CAPITULO I.
ALGORITMOS Y PROGRAMAS

Hoy en día las personas sienten la necesidad de hacer uso de computadoras para la solución de problemas y, debido a ésto, aprenden lenguajes y técnicas de programación. Por eso para llegar a la solución de un problema se deben dar los siguientes pasos:

1. Definición o análisis del problema.
2. Diseño del algoritmo.
3. Transformación del algoritmo en un programa.
4. Ejecución y validación del programa.

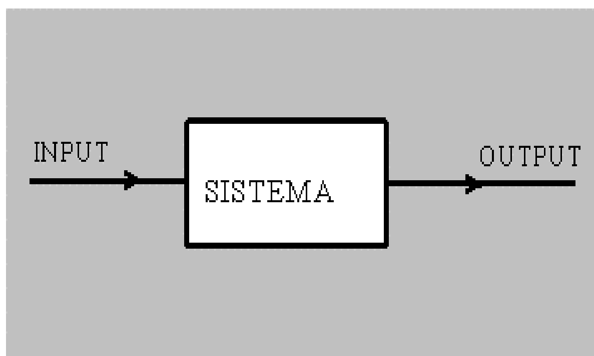
Lo que se pretende es lograr un diálogo permanente entre el usuario y la máquina, por lo tanto veremos conceptos como algoritmo y programa, dentro de lo que son los métodos y técnicas de programación.

Algoritmo, conocido como método de solución de un problema. Algoritmo viene de Mohammed al-kowarizmi, un matemático persa que introdujo este término al mundo, su apellido fue traducido al latín como algorismus, el que posteriormente se conoció como algoritmo.

1. LOS SISTEMAS DE PROCESAMIENTO DE LA INFORMACION

Antiguamente se definía a la Computadora como: **"Una máquina provista de dispositivos electrónicos, los que eran capaces de ejecutar operaciones repetitivas de gran complejidad y a grandes velocidades".** Hoy, la idea ha cambiado y se define como: **"procesador de datos y sistemas de procesamiento de la información"**

La definición general de sistema se da como **"conjunto de componentes conectados e interactivos, con un propósito o fin único",** mientras que sistemas de procesamiento de información son sistemas que transforman datos brutos en información ordenada, con un significado lógico y útil.



El procesamiento de la información presenta tres componentes, que son: entrada, procesador y salida.
La información es ingresada al procesador, el cual quien se representa como una caja negra, luego ésta es procesada y sale como resultado.
Si el procesador es una computadora, el algoritmo se expresa de tal forma que recibe el nombre de programa. Un

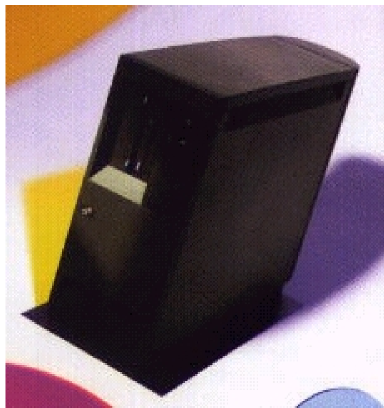
programa se escribe en lenguaje de programación, y a la actividad de expresar un algoritmo en forma de programa se llama programación. El programa se rige de instrucciones y normas siguiendo una secuencia lógica, que especifica operaciones que debe realizar la computadora.

Dos conceptos Importantes:

HARDWARE : Grupo de componentes físicos de la computadora (equipo físico).

SOFTWARE : Grupo de programas que controlan el funcionamiento de la computadora (equipo lógico).

El Hardware está compuesto de:



Unidad Central de Proceso, UCP (Central Processing Unit, CPU): Es el conjunto de circuitos electrónicos capaces de ejecutar operaciones sencillas como: cálculos matemáticos como (suma o multiplicación de números). La potencia de la computadora depende de la velocidad y fiabilidad de la UCP.

Memoria Central: Centro de almacenamiento de información procesada por la UCP hasta que terminan los cálculos. También son almacenados en la memoria central los programas de computadoras.

Dispositivo de Almacenamiento Secundario (memoria auxiliar): Se conoce de esta forma a dispositivos como discos (disquettes y CD´s) y cintas magnéticas, que pueden ser manejados de forma portátil. Los datos son almacenados en dispositivos de memoria auxiliar para luego llevarse a la memoria central.

Periféricos o Dispositivos de Entrada/Salida E/S: Medios por el que el usuario se comunica con la computadora. Estos pueden ser teclado, monitor, impresora, etc.

2. CONCEPTO DE ALGORITMO

Los programadores son, personas que resuelven problemas mediante el uso de computadoras, y la práctica constante del conocimiento de las técnicas de programación.

Un ***algoritmo*** es un método para resolver un problema. Debe presentarse como una secuencia ordenada de instrucciones que siempre se ejecutan en tiempo finito y con una cantidad de esfuerzo también finito. Los *algoritmos* tienen un inicio y un final, son únicos y deben ser fácilmente identificables.

Pasos a seguir para la solución de problemas:

- Diseño del algoritmo que describe la secuencia ordenada de pasos que conducen a la solución de un problema dado (Análisis del problema y desarrollo del programa).
- Expresar el algoritmo como un programa en un lenguaje de programación adecuado (fase de codificación).
- Ejecución y validación del programa por la computadora.

Sin algoritmo no podrá existir un programa, por eso es necesario diseñar el algoritmo previamente, para su realización.

En cada problema el algoritmo se puede expresar como un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo el algoritmo será siempre el mismo.
Los algoritmos son muy importantes, ya que de éstos va a depender el buen funcionamiento de un programa. Los lenguajes de programación sólo expresan lo que un algoritmo quiere decir y las computadoras llevan a cabo la ejecución.

2.1 CARACTERISTICAS DE LOS ALGORITMOS

Todo algoritmo debe cumplir:

- Debe ser preciso indicando la realización de cada paso ordenadamente.
- Debe estar bien definido. Si se sigue un algoritmo mas de una vez, los resultados deben ser los mismos.
- Debe tener un fin; es decir deberá ser finito. Si se sigue un algoritmo, éste debe terminar en algún momento, o sea debe tener un número determinado de pasos.

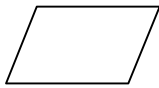
Existen muchas Técnicas. Las más conocidas para el desarrollo de un algoritmo, son el **Pseudocódigo** y el **Diagrama de flujo o flujograma.**

Pseudocódigo: Es la representación de un algorítmo en una secuencia lógica de actividades, a que llevarán en conjunto a la solución de un problema.

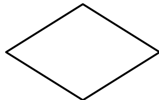
Diagrama de Flujo: Son símbolos gráficos que representan a un algoritmo. Estos símbolos están estandarizados, teniendo cada uno un significado universal.

Aquí algunos de ellos:

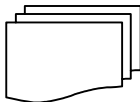
Es un terminal, puede representar el **inicio** o el **final.**

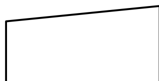

Entrada/Salida, representa la entrada de datos en la memoria o la salida por un periférico.

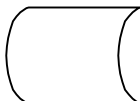
Proceso, cualquier tipo de operación.

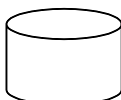

Decisión, función de decisión por cualquier comparación lógica. La decisión puede ser simple o puede ser múltiple

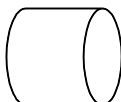
Documento, también se usa como **impresora.**

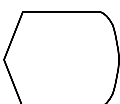

Multidocumento.


Ingreso manual de datos


Datos almacenados.


Disco magnético.


Almacenamiento de acceso directo.


Pantalla.

Un algoritmo debe describir tres partes:

- Entrada : ingredientes y utensilios empleados.
- Proceso : elaboración de la receta en la cocina.
- Salida : terminación del plato.

Ejemplo de un algoritmo:

Determinar la suma de los N primeros números enteros de acuerdo a la siguiente fórmula:

$$\textbf{Suma} = \frac{N*(N+1)}{2}$$

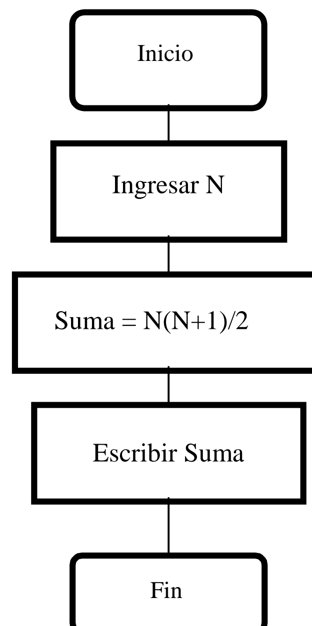
Se definen: E/S

Entrada: Número entero (N)
Salidas: Suma

PSEUDOCODIGO:

inicio
Ingresar un número entero: *N*
Calcular suma por fórmula:
Suma *= N*(N+1)/2*
Escribir suma
fin

DIAGRAMA DE FLUJO:



Ejemplo:
Se desea diseñar un algoritmo para saber si un número es primo o no.

Solución:
Un número es primo si sólo puede dividirse por sí mismo y por la unidad (es decir, no tiene más divisores que el mismo y la unidad). Por ejemplo 3,5,7,11,13,17,19, etc., Así, 3 es divisible por 3 y por 1, 17 lo es por 17 y por 1, etc.
El algoritmo de resolución del problema pasa por dividir sucesivamente el número 2,3,4…, etc.

1. Inicio.
2. Poner X igual a 2 (X=2, X, variable que representa a los divisores del número que se busca N).
3. Dividir N por X (N/X).
4. Si el resultado de N/X es entero, entonces N no es un número primo y bifurcar al punto 7, en caso contrario continuar el proceso.
5. Suma 1 a X (X X + 1).
6. Si X es igual a N, entonces N es un número primo; en caso contrario, bifurcar al punto 3.
7. Fin.

Si N es 131, los pasos anteriores serían :

1. Inicio.
2. X=2

3 y 4. 131/2. Como el resultado no es entero, se continúa el proceso.

5. X 2 + 1, luego X=3.
6. Como X no es 131, se bifurcará al punto 3.
   131/3 resultado no es entero.
   X 3 + 1, X=4.
   131/4….,etc.

3. LOS LENGUAJES DE PROGRAMACION

Para la realización de un proceso, el procesador deberá recibir el algoritmo adecuado, siendo éste capaz de interpretarlo.

- Entender las instrucciones que cada paso expresa.
- Ejecutar las operaciones correspondientes.

El algoritmo se expresará por medio de un formato que se denomina programa. El programa se desarrolla en un lenguaje de programación y las operaciones que llevarán a expresar un algoritmo en forma de programa se llaman programación.

Existen importantes razones para el estudio de las Técnicas de Programación, y éstas son:

- El desarrollo de algoritmos eficientes es una habilidad que se trata de mejorar siempre.
- Se busca la mejora en el uso del lenguaje de programación disponible.
- Enriquecer el vocabulario con construcciones útiles sobre programación.
- Tener la capacidad para elegir el mejor lenguaje de programación.
- Facilitar el aprendizaje de cualquier otro lenguaje.
- Facilitar el diseño de un lenguaje nuevo.

Actualmente existen tres tipos de lenguaje utilizados:

- Lenguaje de máquina.
- Lenguaje de bajo nivel.
- Lenguaje de alto nivel.

3.1 Instrucciones a la Computadora

Un algoritmo se compone de muchos pasos, todos diferentes, los que son interpretados como instrucciones, sentencias o proposiciones. Cuando se habla del término instrucción se está refiriendo a los lenguajes máquina y bajo nivel, tomando en consideración que las sentencias o proposiciones son para los lenguajes de alto nivel. Entonces, en un programa, la secuencia de instrucciones especifica las operaciones que la computadora debe realizar.

Aquí se hará referencia a los tipos fundamentales de instrucciones que una computadora es capaz de manipular y ejecutar. En casi todos los lenguajes de programación las instrucciones básicas y comunes pueden dividirse en cuatro grupos:

- **Instrucciones de Entrada /Salida:** Transferencia de datos e información entre dispositivos periféricos (teclado, impresora, unidad de disco, etc.) y memoria central.
- **Instrucciones Aritmético-Lógicas:** Tienen la función de ejecutar operaciones aritméticas (suma, resta, multiplicación, división, potenciación), lógicas (operaciones ***and, or, not,*** etc.).
- **Instrucciones Selectivas:** Estas permiten la selección de tareas alternativas en función de los resultados de diferentes expresiones condicionales.
- **Instrucciones Repetitivas:** Permiten la repetición de secuencias de instrucciones, un número determinado o indeterminado de veces.

3.2 Lenguajes Máquina
Son lenguajes que están expresados en lenguajes directamente intelegibles por la máquina (computadora), siendo sus instrucciones cadenas binarias (códigos binarios, caracteres 0 y 1) que especifican una operación.
Las instrucciones en lenguaje máquina dependen del ***Hardware*** de la computadora, pues diferirán de una computadora a otra.

3.3 Lenguajes de Bajo Nivel
Son lenguajes más fáciles de usar que los lenguajes de máquina, pero también dependen de la máquina en particular. El lenguaje de bajo nivel es por excelencia el **ensamblador** (assembler language). Estas instrucciones son conocidas como **nemotécnicos** (mnemonics).

3.4 Lenguajes de Alto Nivel
Estos son los más usados por los programadores. Han sido diseñados para que las personas puedan escribir y entender de manera más fácil los programas que los lenguajes máquina y ensambladores.

Un lenguaje de alto nivel es totalmente independiente de la máquina, es decir que sus instrucciones no dependen del diseño de la máquina. Lo que significa que los lenguajes de alto nivel son portables o transportables, quiere decir que pueden ser ejecutados en diferentes tipos de computadoras.

3.5 Traductores de Lenguaje
Son programas que traducen a su vez los programas fuente. Estos están escritos en lenguajes de alto nivel a código de máquina.
Los traductores se dividen en:

- Compiladores.
- Intérpretes.
- Lee y traduce instrucción por instrucción el programa fuente.

3.5.1 Interpretes:
Un lenguaje que tenga la capacidad de soportar un traductor de tipo intérprete es denominado lenguaje interpretado. BASIC es el modelo por excelencia de lenguaje interpretado.

3.5.2 Compiladores
Es un programa traductor que lee el programa y traduce en forma total, y genera un archivo ejecutable (EXE), en los programas fuente escritos en lenguajes de alto nivel como PASCAL, FORTRAM,…, a lenguajes máquina. Los programas escritos en lenguajes de alto nivel son llamados programas fuente y los programas que son traducidos, programas o código objeto.

3.6 Los Compiladores y sus frases
La compilación es el proceso de traducción de programas fuente a programas objeto. El programa objeto no ha sido traducido normalmente a código máquina sino a ensamblador.

Debe hacerse uso de un programa *montador* o *enlazador* (*linker*) para obtener el programa máquina real. Este proceso hace que un programa en lenguaje máquina sea directamente ejecutable.

4. VARIABLES Y TIPOS DE DATOS PRIMITIVOS

El motivo principal de la computadora es, primordialmente, el manejo de los datos o de la información ingresada para luego de procesarla arrojar los resultados. La computadora opera con objetos, los cuales son descritos por expresiones denominados datos. Son varios los tipos de datos con los que la computadora puede trabajar.

Existen dos grupos de tipos de datos: Los que no tienen estructura, llamados **Simples,** y los que son estructurados, llamados **Compuestos**. La variedad con respecto a los tipos de datos son representados de distintas formas en la computadora. A nivel de máquina un dato es una secuencia o conjunto de bits (dígitos 0 ó 1).

Dato: Cuando se habla de dato, o se da la expresión general de dato, el cual describe los objetos con los que opera una computadora. Los algoritmos y programas correspondientes operan sobre datos. Los datos de entrada se transforman gracias al programa y se traducen a lenguaje máquina, los mismos que son ejecutados por la computadora.

Los tipos de datos que existen son dos: Simples y Compuestos

4.1 Datos Numéricos: Se llama así al conjunto de valores numéricos y son representados de distintas formas:

- **Tipo de dato numérico entero (*integer*):**

  El tipo entero es un subconjunto finito de los números enteros. Los enteros se forman con la representación de 2 bytes, con el 1er bit del extremo izquierdo como bit de signo, o sea (0) positivo y (1) negativo.

  Ej: 5, 6, -15, 4, 2.480

  En ocasiones se llaman a éstos, números de punto y coma fijo. Siendo los números enteros máximos y mínimos de una computadora, se encuentran entre –32768 a +32767. Que es resultado de $-2^{n-1}-1$ y $2^{n-1}-1$, para cuando n= 16 o sea 2 bytes.

  Un número entero es cualquier número que pertenece al conjunto definido como:

  $$-2^{n-1}-1 , 2^{n-2}, ..., -2^1, -2^0, 0, 2^0, 2^1, ..., (2^{n-1})$$

  Aquí se observa que los números enteros no contienen partes fraccionarias, y de acuerdo a esta definición pueden ser positivos o negativos. Entonces se podrá representar cualquier número entero sin signo por medio de una secuencia de dígitos que tengan la forma:

  $$d_n\ d_{n-1}.......d_2\ d_1\ d_0$$

  cuyo valor o magnitud es determinado por la suma de:

  $$d_nb^n + d_{n-1}b^{n-1} + ... + d_2b^2 + d_1b^1 + d_0b^0$$

donde ***b*** *es la base del sistema de numeración* en cuestión. Como lo que interesa es la forma de almacenar (representar) los datos en la computadora, se utiliza el sistema de numeración binaria. La magnitud del número 101112 es la suma de los términos:

$$1x2^4 + 0x2^3 + 1x2^2 + 1x2^1 + 1x2^0$$

Ejemplo: Empleando el método de representación de enteros positivos visto, mostrar cómo se almacenaría el número 6 en la computadora. Asignando los valores a cada bit de acuerdo a su posición se observa que:

$$1x2 + 1x2 + 0x2 = 4 + 2 + 0 = 6 \quad 110_2$$

Con respecto a los números enteros negativos, para su representación se hace necesario establecer convenciones, como por ejemplo el método de signo y magnitud o el del complemento.

Signo y Magnitud: Los símbolos (+, -) deben escribirse precediendo a la magnitud del número. Para su representación en la computadora, por lo general se usa dígitos binarios de extrema izquierda para denotar el signo y los restantes se utilizan para la magnitud.

Número Positivo	Signo y Magnitud	Número Negativo	Signo y Magnitud
0	0 000	0	1 000
1	0 001	-1	1 001
2	0 010	-2	1 010
3	0 011	-3	1 011

Para evitar que el cero se duplique, generado por el bit de signo se suele ocupar la notación por complemento a dos.

Complemento a 2 = 2ˆn - |número|, donde número está en base 2 y n es la cantidad de dígitos binarios de la representación.

Ejemplo: representar el número 4 en base 10 como negativo en complemento a 2 utilizando 4 bits.

$$4 = 0100$$
$$-4 = 10000 – 0100 = 1100$$

o también, si manejar operaciones con dígitos binarios se pueden tomar los valores decimales y al resultado final se pasa a como lo representaría la computadora, es decir:

$$-4 = 2^4 - |4|$$
$$-4 = 16 – 4 = 12 \leftrightarrow \mathbf{1100_2}$$

- **Tipo de dato numérico real (*real*):**
  El tipo real consiste en un subconjunto de los números reales. Los números reales siempre tienen un punto decimal y pueden ser positivos o negativos. Los números reales constan de una parte entera y una decimal.

  Ej: **0.006, 9.3632, 6471.71, 3.0, -8.17, -63.3387**

se ha tomado en cuenta que cuando se quieren hacer aplicaciones científicas se debe tener una representación especial, las que nos permitirá manejar números o cantidades grandes, como por ejemplo cuando se quiere calcular la masa de la tierra o como para calcular la masa de un electrón. La representación de los números es limitada ya que sólo se pueden representar números que tengan como máximo ocho dígitos, provocando problemas a la hora de representar o almacenar números grandes o números muy pequeños.

Ej: **7877377558797** ó **0.70548058505725**

La representación llamada **notación exponencial o científica** es usada para números muy grandes o pequeños, entonces:

896591384456

También existe una representación que lleva el nombre de representación en coma flotante, la misma que es una generalización de notación científica.

Representación de números de coma flotante:
Para expresar los números fraccionarios se utiliza la notación científica o exponencial:

signo mantisa

$$N = s + M + b^{e}$$

En donde:
S = + 6 –
M = Mantisa de M, es un número tal que $0<M<1$
b = base, es un entero positivo >= 2
e = exponente, es un entero con signo.
El número M también se expresa en base b:
M = 0.d1 d2......
Con los dígitos d1 tomando valores en 0,1,....,b-1

NOTA:
1. en la representación anterior se excluye el número 0.0
2. puesto que para un mismo número hay varias representaciones científicas, por lo tanto varias mantisas y exponentes, se suele normalizar la mantisa **"M"** de manera que el primer dígito d1 sea distinto de 0, para lo cual basta desplazar el punto de fracción hasta que se cumpla dicha condición, y desde luego modificar el exponente **"e"** para que se mantenga el valor del número.

Ejemplos:

1. Expresar el número –123.05 (base 10) en forma científica con base 10.

-123.05 = - 0.12305 * 10^3 (mantisa normalizada)
= - 0.0012305 * 10^5

2. Expresar el número 0.003ª (base 16) en notación científica con base 2:

0.003ª = + 0.3A * 16^{-2}
= + 0.0011 1010 * 2^{-8} (pasando a base 2)
= + 0. 111010 * 2^{-10} (mantisa normalizada)

3. Expresar el número –123.025 (base 10) en notación científica con base 2 y mantisa normalizada con 24 dígitos.
   Para reducir el número de operaciones primero obtenemos la representación en base 16 del número sin signo.

Paso1. Parte entera (por divisiones)

123	16
B	7

$123_{(10)} = 7B_{(hex)}$

Paso 2: Parte fraccionaria (por multiplicaciones)

		Parte entera (dígito frac.)
0.025	* 16 = 0.4	, 0
0.4	* 16 = 6.4	, 6
0.4	* 16 = 6	, 6
.....		

en donde en cada paso se retira la parte entera y la parte fraccionaria se continúa multiplicando por 16.

Luego,

$0.025_{(10)}$ = $0.0666...6..._{(16)}$ (fracción periódica)

Paso 3: $-123.025_{(10)} = -7B.066..._{(16)}$ (i)

Pasando ahora a base 2:

- $111\ 1011.0000\ 0110\ 0110\ 0110\ 0110\ ...._{(2)}$

luego (con mantisa normalizada):

- $0.1111\ 0110\ 0000\ 1100\ 1100\ 1100\ 1100_{(2)}$ ...* 2^7 (ii)

Representación en Memoria:
El número N = s M b^e, dado en notación científica y con mantisa normalizada, se suele representar en la memoria del computador usando celdas de longitud L:

---------L bits--------
s e + EX mantisa

usando un bit para el signo, bm bits para la mantisa, y el resto, (L -bm-1) bits, para el exponente desplazado (o sumado) una cantidad constante EX denominada exceso. El exceso es utilizado para considerar exponentes positivos y negativos.

(bm), da la precisión de la representación elegida; en efecto, significa el número de dígitos significativos en base 2: dos números con iguales partes hasta el bit bm e iguales exponentes son considerados iguales.

NOTA: Por convenio, el número 0.0 es representado en memoria exclusivamente con bits 0:

0 0....0 0....0

Ejemplos

1. El procesador PDP-11 representa un número de coma flotante de precisión simple utilizando el siguiente formato de 32 bits:

- un bit de signo.
- Exponente (8bits) con exceso a 128 = 80 (hex)
- Mantisa normalizada de 24 bits
- Base 2

En este caso, en la mantisa normalizada se omite el primer bit (pues siempre es 1) y en su lugar se almacena el bit de signo:

8 bits	1 bit	23 bits
e + 80	s	mantisa normalizada sin primer bit

Ejemplo: Representar el número -$123.025_{(10)}$ usando este formato.

Según el resultado obtenido en (ii), en base 2 el número es:

-0.1111 0110 0000 1100 1100 1100 1100 ....*2^7
23 bits parte truncada

8 bits	1 bit	23 bits de mantisa
7 + 80h	1	111 0110 0000 1100 1100 1101
		bit de signo reemplaza al primer bit de la mantisa

Por tanto, se obtiene: 87 F6 0C CD (hex)

En este formato se conviene en redondear la mantisa resultante: se le suma 1 si la parte truncada empieza con 1, como en efecto se ha hecho en el presente caso

2. El computador IBM 370 representa los números de coma flotante con 32 bits, en la forma:
- Un bit de signo.
- Exponente (7bits) con exceso a 64 = 44 (hex)
- Mantisa normalizada de 24 bits.
- Base 16

Así, los 8 primeros bits contienen el signo y el exponente desplazado:

2 dig. Hex.		6 dig. Hex.
1 bit	7 bits	24 bits
s	e + 44	mantisa normalizada

Ejemplo: La representación en memoria del número anterior según la presente especificación.

Por (i), el número en base 16 y con mantisa normalizada es:

$$- 7B.006... = - 0.7B06\underline{6666\ ...} * 16^2$$

parte truncada.

1 bit	7 bits	6 dig. hex.
1	2 + 44h	7B0666
..	46h	
..	100 1100	
..	bit de signo, el byte completo es: 1 100 1010 = C6	

Luego, C6 7B 06 66

También se aplica el método de redondeo: Simplemente se suprime la parte truncada: 666 ....
Pues 6 = 0110 empieza con 0.

3. Turbo Pascal (para IBM PC y compatibles) utiliza 6 bytes (48 bits) para representar los números de tipo REAL:

8 bits	1 bit	39 bits
e +80h	s	mantisa normalizada, sin primer bit

con base 2.

Ejemplo: El número – $123.025_{(10)}$ se representa como REAL.

En Turbo Pascal por: 87 F6 0C CC CC CC
(usando dígitos hexadecimales)

Este método no utiliza redondeo.

NOTA: Realmente los bytes se almacenan con los 5 últimos bytes en orden invertido: 87 CC CC OC F6

4.Turbo C emplea 4 bytes (32 bits) para representar los números de tipo float:

1 bit	8 bits	23 bits
s	e + 7Eh	mantisa normalizada, sin primer bit

siendo base 2

Ejemplo:

Aplicar este formato:

Como en el ejemplo 1 se tiene:

-0.1111 0110 0000 1100 1100 1100 1100 ....* 2^7

23 bits parte truncada

1 bit	8 bits	23 bits (mant. nor. Sin primer bit)
1	7 + 7E	111 0110 0000 1100 1100 1101
...	85h	
...	1000 0101	
bit de signo.		

Luego,

1 1000 0101 111 0110 0000 1100 1100 1101

1100 0010 1111 0110 0000 1100 1100 1101
C 2 F 6 0 C C D (hex)

Esto es, C2 F6 0O CD (hex)
Este formato también aplica el redondeo.

Operaciones:
A partir de la representación en memoria de un número de coma flotante se obtienen: el signo, exponente y la mantisa del mismo.

Para sumar dos números de coma flotante:

a. si tienen distintos exponentes se modifica la representación científica del dato de menor exponente de manera que los datos resulten con exponentes iguales al mayor exponente.
b. Si ambos datos tienen iguales signos, se suman las mantisas; si ambos datos tienen signos opuestos se resta la mantisa menor de la mantisa mayor.

c. Se normaliza la mantisa obtenida en b.

Para la sustracción, se modifica el paso b):

b'. Si ambos datos tienen signos opuestos, se suman las mantisas; si ambos datos tienen signos iguales se resta la mantisa menor de la mantisa mayor.
En el primer caso, el signo de la resta es el signo del primer dato; en el segundo caso, el signo de la resta es:

> El signo minuendo si éste tiene la mantisa mayor o el signo opuesto del sustraendo, en caso contrario.

La operación de multiplicación (resp. división) es más sencilla:
- El signo es "+" si ambos tienen iguales signos, o "-" de otra manera
- La mantisa es el producto (resp. cuociente) de las mantisas
- El exponente es la suma (resp. diferencia) de los exponentes.
- Al final hay que normalizar la mantisa resultante.

NOTA:
1. En todos los casos las operaciones con mantisas son realizadas como enteros sin signos.
2. En la exposición dada se excluye el caso en que uno de los datos es el número 0.0, cuyo tratamiento puede ser realizado como un caso simple.

Ejercicios:
1. Expresar los números decimales 1200.05, -456.60 egún los formatos de IBM 370 y Turbo C.
2. Hallar 85 C8 00 00 00 00 + 87 70 00 00 00 00 00 si los factores representan datos REAL de Turbo Pascal.

Rpta. 8C BB 80 00 00 00

3. Hallar los valores de los dos factores del ejercicio b. en el sistema decimal.

Rpta. –25.0, 120.0
Indicación (primer factor)
Exponente = 85 – 80 = 5
Signo dado por C = 1100

Suprimiendo el signo y reponiendo el primer bit 1 suprimido:

8	0	0	0	0	0	0	0	0	(hex)
1 100 1000	0000	0000	0000	0000	0000	0000	0000	0000	(bin)

Luego el número en base 2 es:

$-0.11001 * 2^5 = - 11001$
que es –25.0 en el sistema decimal.

4.2 Datos Lógicos: Conocidos también como datos **booleanos**, este tipo de dato es aquel que sólo admite dos tipos de valores, los que son **verdadero (true)** o **falso (false).** Los dos son usados para la representación de alternativas (si/no) que se dan según la condición que se plantee, es decir, si se quiere que el programa evalúe una condición determinada, ésta puede ser verdadera o falsa, dependiendo de las reglas que se hayan planteado anteriormente.

4.3 Datos Tipo Caracter: Los datos de este tipo forman un conjunto finito y ordenado que la computadora puede reconocer. Un dato tipo caracter contendrá solamente un caracter. No existe un estándar en lo que respecta al reconocimiento de caracteres, pero las computadoras pueden reconocer los siguientes caracteres:

- Caracteres alfabéticos (A,B,C,D,.....)
- Caracteres numéricos (1,2,3,4, ...)
- Caracteres especiales (+, -, *, /,., ;,<, >, ..., etc.)

Representación de Caracteres

Para facilitar la escritura y la lectura de los programas y datos de una computadora, se desarrollaron códigos simbólicos para representar la información con caracteres, en lugar de números.

Cada caracter es representado por un patrón de bits diferente que lo identifica, el número de bits para representar un caracter está determinado por el código utilizado. Los distintos tipos de máquinas han generado cada una su propio juego de caracteres. El más común hasta hace poco era el **ASCII**, en los últimos tiempos Windows ha cambiado al código **OEM**, y con **Windows 95 nos encontramos con UNICODE.** Como quiera que sea, los caracteres han conservado una herencia dada por **ASCII** y **EBCDIC.**

Si 8 bits son usados para representar cada carácter, y la longitud de la palabra de memoria es mayor, es ineficiente almacenar un caracter por palabra. En este caso varios caracteres son almacenados en una palabra.

4.4 Datos tipo Cadena: Un modo de representar los datos tipo carácter, son el tipo **Cadena (string).** Estos se forman por una sucesión de caracteres, encontrándose delimitados por una comilla (apóstrofo) o dobles comillas, ésto va de acuerdo al tipo de lenguaje de programación que se esté tratando, con esto se entiende entonces que el tamaño que tendrá una cadena o su longitud, será la que se encuentre comprendida entre las comillas (delimitadores).

Por ejemplo la cadena **"curso"** tiene una longitud de cinco caracteres, mientras que la cadena " " tiene una longitud de un carácter, que es un espacio solamente. La cadena " **",** es una cadena vacía. La cadena **"win-98"** tiene una longitud de seis caracteres o elementos alfanuméricos.

Otros ejemplos de cadenas:

´hola Rebeca´
´28 de Julio de 1821´
´Sr. Alvarado´

5. CONSTANTES VARIABLES

Las constantes son datos cuyos valores no cambian, pero existen datos cuyos valores sí varían durante la ejecución del programa, a éstos los llamamos variables. En la mayoría de los lenguajes de programación se permiten diferentes tipos de constantes: enteras, reales, caracteres y boolean o lógicas, quienes representan datos de estos tipos.

Entonces una variable se conoce como un objeto, o partida de datos cuyo valor puede cambiar durante la ejecución del algoritmo o programa. A las variables se les conoce o identifica por los atributos siguientes: nombre que lo asigna y tipo que describe el uso de la variable.

6. EXPRESIONES

Son la combinación de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales, idea que puede ser utilizada en notaciones de matemática tradicional. Los valores de las variables nos permitirán determinar el valor de las expresiones, debido a que éstos están implicados en la ejecución de las operaciones indicadas. Estas constan de operandos y operadores. Según el tipo de objetos que manipulan, pueden clasificarse en:

- aritméticas resultado tipo numérico.
- relacionales resultado tipo lógico.
- lógicas resultado tipo lógico.
- caracter resultado tipo caracter.

6.1. Expresiones Aritméticas

Estas expresiones son análogas a las fórmulas matemáticas. Las variables y constantes son numéricas (real o entera) y las operaciones son las aritméticas.

+	suma
-	resta
*	multiplicación
/	división
**,^	exponenciación
div	división entera
mod	módulo o resto

Las palabras clave div y mod se conocen como operadores aritméticos.

Operadores DIV y MOD

El símbolo / se utiliza para la división real, y el operador **div** - en algunos lenguajes, por ejemplo BASIC, se suele utilizar el símbolo \ - representa la división entera.

A div **B**

Sólo se puede utilizar si A y B son expresiones enteras y obtiene la parte entera de A/B. Por consiguiente,

19 div 6 toma el valor de 3, por que los decimales no se consideran.

6.2. Reglas de Prioridad

Cuando existen expresiones que tienen más de dos operandos, se requiere de reglas matemáticas que permitan determinar el orden de las operaciones y éstas, son reglas de prioridad o precedencia y son:

- Operaciones que están encerradas entre paréntesis se evalúan primero. Si existen diferentes paréntesis anidados (interiores unos a otros), las expresiones más internas se evalúan primero.
- Las operaciones aritméticas dentro de una expresión generalmente suelen seguir el siguiente orden de prioridad:

    1. Operador exponencial (^, ↑, o bien **)
    2. Operadores *, /
    3. Operadores +, -.
    4. Operadores div y mod

En caso de coincidir varios operadores de igual prioridad en una expresión o subexpresión, encerrada entre paréntesis, el orden de prioridad es en este caso de izquierda a derecha.

6.3. Expresiones Lógicas (booleanas)

Otro tipo de expresiones son las **expresiones lógicas o booleanas,** cuyo valor es siempre **verdadero** o **falso**. Pues existen dos constantes lógicas, verdadera (true) y falsa (false) y que las variables lógicas pueden tomar sólo estos dos valores, verdadero o falso. Se llama expresión booleana en honor al matemático británico GEORGE BOOLE, quien desarrolló el álgebra de boole.

Estas expresiones se forman combinando constantes lógicas , variables lógicas y otras expresiones lógicas, utilizando los ***operadores lógicos*** **not, and** y **or,** y los operadores relacionales (de relación o comparación) =, <, >, <=, >=, <>.

6.3.1. Operadores de relación

Los operadores relacionales o de relación permiten realizar comparaciones de valores de tipo numérico o caracter. Estos sirven para expresar las condiciones en los algoritmos.
Los operadores de relación se pueden aplicar a cualquiera de los cuatro tipos de datos estándar: entero, real, lógico, caracter.

6.3.2. Operadores lógicos

Los operadores lógicos básicos son **not** (no) **and** (y) y **or** (o)

6.3.3. Prioridad de los operadores lógicos

Los operadores lógicos y relacionales cuentan con un orden de prioridad. Dentro de éstos, los paréntesis y caracteres de esta naturaleza se pueden utilizar y tendrán prioridad sobre cualquier operación.

7. FUNCIONES INTERNAS

Generalmente las operaciones requieren de un número determinado de operadores especiales que se denominan ***funciones internas, incorporadas o estándar.***
Las funciones soportan argumentos reales o enteros y sus resultados están supeditados a la tarea que realice dicha función.

8. LA OPERACION DE ASIGNACION
Se le otorgan valores a una variable. Esta operación de asignación se conoce como **instrucción** o **sentencia** de asignación, si es que está en un lenguaje de programación. La operación de asignación es representada por un símbolo u operador: ←

La acción de asignar puede ser destructiva ya que puede perderse el valor que tuviera la variable antes, siendo reemplazado por el nuevo valor. Las acciones de asignación se clasifican según sea el tipo de expresiones en: **Asignación aritméticas, Asignación lógica y Asignación de caracteres.**

Frente a todo esto, existirá error si es que se quiere asignar valores de tipo caracter a una variable numérica o viceversa.

9. ENTRADA Y SALIDA DE INFORMACION
El ingreso de datos es importante para que la computadora realice los cálculos; esta operación es la entrada, luego, estos datos se convertirán en resultados y serán la salida.

A la entrada se le conoce como operación de **Lectura** (read). Esta operación de lectura se realiza a través de los dispositivos de entrada que son (teclado, unidades de disco, CD-Rom, etc.). La operación de salida se realiza por medio de dispositivos como (monitor, impresora, etc), a esta operación se le conoce como **escritura** (write).

CAPITULO II.
COMPUTADORAS Y HERRAMIENTAS DE PROGRAMACION CLAVES EN LA SOLUCION DE PROBLEMAS

Para la solución de problemas por medio de computadoras, deben cumplirse tres fases:

- Análisis del problema,
- Diseño del algoritmo,
- Resolución del algoritmo en la computadora.

La exposición del problema es importante para el análisis y diseño de la solución del mismo. Para ello se necesita el apoyo de las herramientas de programación, como los diagramas de flujo, diagramas de Nassi - Schneiderman (N-S) o pseudocódigo.

1. SOLUCIONANDO EL PROBLEMA

La computadora es considerada como una herramienta para la solución de problemas, es por eso que las personas están aprendiendo las técnicas de programación, las cuales ayudan a entender mejor los lenguajes de programación, pero para llegar a la solución se deben llevar a cabo los tres pasos anteriormente mencionados.

2. ANALISIS DEL PROBLEMA

Para el análisis del problema, el programador se ayuda de un período de recopilación de información, que es de gran ayuda, pues por medio de este análisis se llega a una mejor comprensión de su naturaleza. Esto ayudará a una mejor definición del mismo y como consecuencia a una solución satisfactoria.

Para la definición del problema con precisión, las entradas y salidas deben estar bien especificadas, siendo estos requisitos fundamentales para una solución eficaz.

3. DISEÑO DEL ALGORITMO

Para que una computadora solucione los problemas es necesario darle pautas o pasos a realizar. A estos pasos se les conoce como hemos visto antes recibiendo el nombre de **algoritmo.**
La información que es ingresada al algoritmo se conoce como entrada y la información producida por éste se conoce como salida.
Para la solución de problemas complejos lo mejor es dividir el problema en subproblemas, así, la solución de éste será más sencilla.

Las ventajas más importantes del diseño son:

- Cuando el problema es dividido en partes más simples, es más fácil su comprensión. A estas partes se les llama módulos.
- Los módulos son más flexibles, en caso de querer hacer modificaciones.
- Las soluciones del problema son fáciles de demostrar.

3.1. Escritura inicial del algoritmo

El sistema para describir (“escribir”) un algoritmo consiste en hacer una descripción paso a paso con un lenguaje natural del citado algoritmo. Siendo estos un conjunto de reglas para solucionar un problema:

Estos tienen las siguientes propiedades:

1. Deberán seguir una secuencia definida por pasos hasta obtener un resultado distinto.
2. Podrán ejecutarse cada vez que se requiera para distintos datos.

Ejemplo:
Queremos determinar la cantidad total de llamadas a pagar en la empresa telefónica, teniendo en cuenta los siguientes aspectos:

1. Cada llamada es de cinco minutos y éstas tienen un costo de 0.60 Nuevos Soles.
2. Pasados los tres minutos por llamada, cada minuto que se adicione tendrá un costo de 0.30 Nuevos Soles.

Solución :
Realizar un algoritmo que solucione el problema planteado:

1. Inicio.
2. Realizar la lectura del número de minutos hablados por teléfono.
3. Realizar la comprobación de que el número de minutos es mayor que cero, pues la llamada ha sido realizada. Si el número de minutos es distinto de cero entonces es positivo. Si el número de minutos es menor que cero, entonces se producirá error.
4. Realizar el cálculo de la conversación de acuerdo a la tarifa:

   - Si el número de pasos es menor que cinco, entonces el precio es de 0.60 Nuevos Soles.
   - Si el número de pasos es mayor que cinco, entonces calcular los pasos que son adicionales a cinco, pues estos tienen un costo de 0.30 Nuevos Soles cada uno. Este resultado deberá ser sumado al costo de la llamada que son 0.60 Nuevos Soles, para así obtener el costo total de la llamada.

4. SOLUCION DE PROBLEMAS POR COMPUTADORA

Después de diseñar el algoritmo y representarlo gráficamente mediante una herramienta de programación (Diagrama de flujo, pseudocódigo o diagrama N-S), se procede a resolver el problema en el computador.
Fase que se descompone en:

- Codificación del algoritmo en un programa.
- Ejecución del problema.
- Comprobación del programa.

5. REPRESENTACION GRAFICA DE LOS ALGORITMOS

La representación de un algoritmo se logra mediante la independización de éste, del lenguaje de programación elegido. Con esto se logra que pueda ser representado en cualquier lenguaje. Para ello debe ser representado gráfica y numéricamente, así cualquier lenguaje de programación será capaz de interpretar su codificación.
Un algoritmo se representa tomando las siguientes consideraciones:

1. Diagrama de flujo.
2. Diagrama N-S (Nassi – Schneiderman)
3. Lenguaje de especificación de algoritmos: pseudocódigo.

4. Lenguaje español.
5. Fórmulas.

5.1 Diagrama de flujo

Esta es una de las técnicas más antiguas y de mayor uso, pero hoy su utilización ha disminuido con la aparición de los lenguajes estructurados. Un diagrama de flujo hace uso de símbolos estándar que están unidos por flechas, que tienen una secuencia en que se deben ejecutar. En los diagramas de flujo del sistema se representan operaciones manuales y automáticas con diferentes dispositivos del sistema informático.

Representación gráfica de Un flujograma o diagrama de flujo:

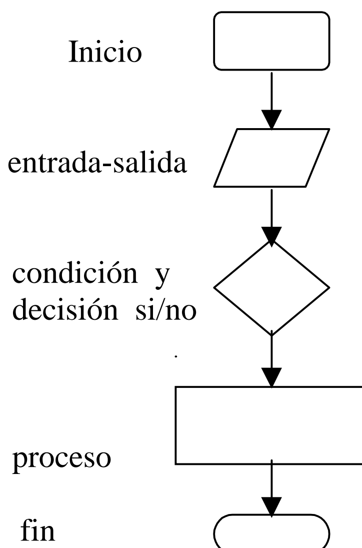


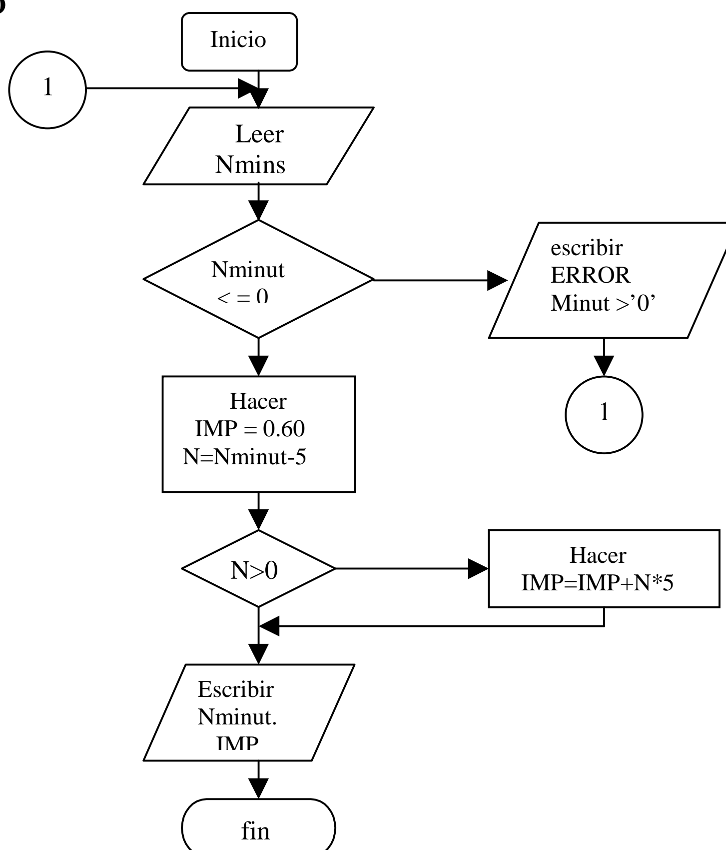
DIAGRAMA DE FLUJO

Ejemplo: 1

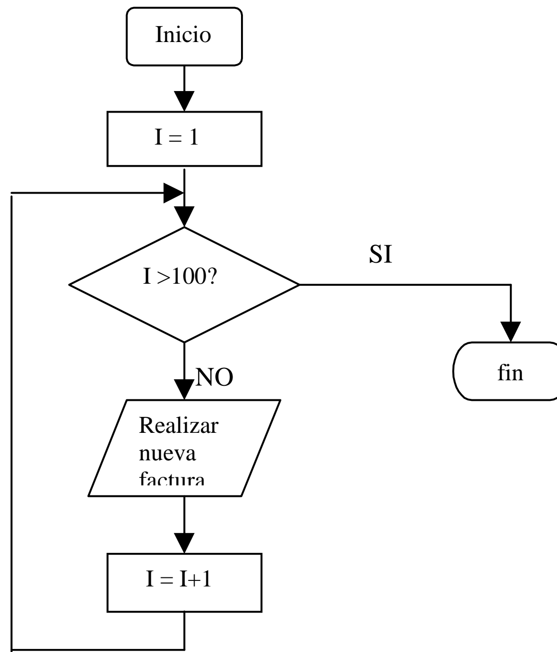
En la representación gráfica del ejemplo de la sección 3.1 será la siguiente:

Variables:

NPASOS : Número de pasos de la llamada.
N : Número de pasos que exceden de cinco.
FACT : Importe total de la llamada.



Ejemplo: 2
Se desea realizar 100 facturas para cien clientes. El diagrama de flujo sería el siguiente:
Solución:



5.2 Diagramas de Nassi - Schneiderman (N-S)
También conocido como diagrama de Chapín, es un diagrama de flujo pero donde las flechas de unión son omitidas, las casillas donde se indican las operaciones se unen una detrás de la otra. Según el orden en que se unen las cajas, las instrucciones son sucesivas, al igual que en los diagramas de flujo en una caja pueden escribirse acciones diferentes.

6. PSEUDOCODIGO
Conocido como lenguaje de especificación de algoritmos, el pseudocódigo se traduce posteriormente a un lenguaje de programación. La computadora no puede ejecutar el pseudocódigo. Su uso tiene ventajas por que permite al programador una mejor concentración de la lógica y estructuras de control, y no preocuparse de las reglas de un lenguaje de programación específico. El pseudocódigo también ofrece la ventaja de poder hacer modificaciones cuando se detectan errores en la lógica del programa, lo que no es posible o es muy difícil cuando se ve esto en un lenguaje de programación. Dentro de las ventajas del pseudocódigo es que da la facilidad de traducción por lenguajes como **PASCAL, COBOL, C, FORTRAN 77 o BASIC estructurado** (ANSI, True, Quick, Turbo, etc.).

El pseudocódigo hace uso de palabras que son reservadas para las acciones sucesivas, las cuales son similares a sus homónimas en los lenguajes de programación tales como: **star, end, stop, if-then-else, while-when, repeat-until,** etc. El pseudocódigo exige la indentación (sangría en el margen izquierdo) de diferentes líneas.

Ejemplo de un Pseudocódigo:

1. inicio
2. establecer CONTADOR a 1
3. establecer SUMA a 0
4. mientras CONTADOR < 100 hacer lo siguiente
   4.1 sumar CONTADOR a SUMA
   4.2 incrementar CONTADOR en 1
5. Visualizar suma

Algunos alcances para escribir pseudocódigos:

- Bifurcaciones:
  - If (condición) then (actividad)
  - If (condición) then (actividad_1) else (actividad_2)
- Bucles:
  - while (condición) do (actividad)
  - repeat (actividad) until (condición)
- Funciones:
  - procedure nombre(argumentos)
- Diseño modular:
  - El programa se divide en módulos con una finalidad bien definida.
  - Cada uno de estos módulos se puede descomponer en otros módulos más simples.
  - Se puede llegar al grado de detalle que se desee.
  - Se puede utilizar niveles de detalle distintos, en las diferentes fases del desarrollo.

Unas primitivas pueden ir dentro de otras, utilizando paréntesis y corchetes si es necesario. Conviene indentar.

CAPITULO III.
PROGRAMA Y SU ESTRUCTURA GENERAL

1. PROGRAMA
Está definido como un conjunto de instrucciones, que ejecutarán una tarea determinada, es decir, mediante procedimientos lógicos, el programa realizará lo que el programador desea. Es un medio para conseguir un fin, el cual será la información necesaria para solucionar un problema.
Para la realización y desarrollo de un programa se requiere de las siguientes fases:

- Definición y análisis del problema
- Diseño de algoritmos
    - Diagrama de flujo
    - Diagrama N-S
    - Pseudocódigo
- Codificación del programa
- Depuración y verificación del programa
- Documentación
- Mantenimiento

2. CONSTITUCION DE UN PROGRAMA Y SUS PARTES
Un programa debe contener un conjunto de especificaciones las que deben ser establecidas por el programador, éstas son: entrada, salida y algoritmos de resolución que tendrán las técnicas para obtener las salidas a partir de las entradas.

El algoritmo de resolución o caja blanca como se le conoce, es en realidad el conjunto de códigos que transforman las entradas del programa (datos), en salidas (resultados).
Las entradas al programa deben ser establecidas por el programador. Las entradas se darán por medio de un dispositivo de entrada, puede ser teclado, disco, teléfono, etc., a este proceso se le conoce como ***entrada de datos***, ***operación de lectura o acción de leer***.

Las salidas de datos se presentan en dispositivos periféricos de salida, puede ser pantalla, impresora, discos, etc. La operación de salida de datos se conoce también como ***escritura*** o acción de ***escribir.***

3. INSTRUCCIONES
El diseño del algoritmo y luego la codificación de programas consiste en la definición de acciones o instrucciones que van a dar solución al problema que se está tratando.
Luego de escribirse las acciones o instrucciones, se deberá ir al proceso de almacenamiento en la memoria conservando el orden lógico para su posterior ejecución; es decir en secuencia. Un programa puede ser lineal o no lineal.

Un programa es lineal cuando las instrucciones siguen una secuencia, sin bifurcaciones, decisión ni comparaciones. Cuando el programa es no lineal, significa que la secuencia es interrumpida mediante instrucciones de bifurcación.

4. TIPOS DE INSTRUCCIONES
De acuerdo al tipo de lenguaje de programación en que se trabaja las instrucciones van a ser diferentes. Pero las instrucciones básicas son independientes del lenguaje.
Las instrucciones básicas pueden clasificarse de la siguiente forma:

- Instrucción de inicio/fin.
- Instrucción de asignación.
- Instrucción de lectura.
- Instrucción de escritura.
- Instrucción de bifurcación.
- Instrucciones de repetición o ciclos.

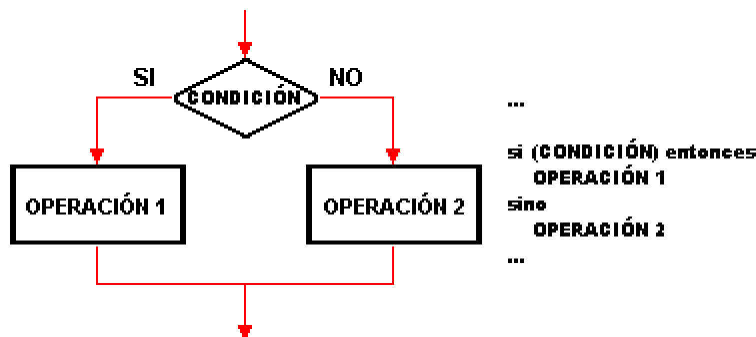
La instrucción de lectura de datos (entrada). Esta instrucción lee datos que son ingresados por medio de un dispositivo de entrada.

La instrucción de escritura de resultados (salida). Estas se escriben en un dispositivo de salida.

Instrucciones de bifurcación. Cuando se ejecuta esta instrucción un programa detiene su desarrollo lineal. Las bifurcaciones pueden ser hacia delante o hacia atrás, según el punto del programa donde se bifurca. Las bifurcaciones en el desarrollo de un programa serán de modo condicional, de acuerdo al resultado de la evaluación de la condición.

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

Bifurcación condicional: Esta depende de que se cumpla una condición determinada. La estructura de bifurcación condicional permite elegir una de dos opciones en una alternativa, dependiente del resultado obtenido al evaluar la condición. Véase el siguiente fragmento de algoritmo:



La palabra clave "sí" indica que estamos en una sentencia de bifurcación condicional. Si la condición es verdadera se ejecuta la operación 1, de otro modo se ejecuta la operación 2.

5. PROGRAMA Y SUS ELEMENTOS BASICOS

Los lenguajes de programación se componen de elementos básicos, los cuales se usan como bloques constructivos; también existen reglas que harán uso de la combinación de estos elementos. A estas reglas les llamamos la ***Sintaxis*** del lenguaje, sólo las instrucciones que contienen una sintaxis correcta pueden ser interpretadas por la computadora y los programas que contengan errores de sintaxis son rechazados por esta misma.

Los elementos básicos que constituyen un programa o algoritmo son:

- palabras reservadas (***inicio, fin, si-entonces, etc.***)
- identificadores (**nombres de variables, esencialmente**)
- caracteres especiales (**coma, apóstrofe, etc**.)
- constantes
- variables

- expresiones
- instrucciones

Hay otros elementos que también son básicos y forman parte de los programas. Son de mucha importancia la comprensión y el funcionamiento, por que gracias a ellos se llegará a un correcto diseño del algoritmo.

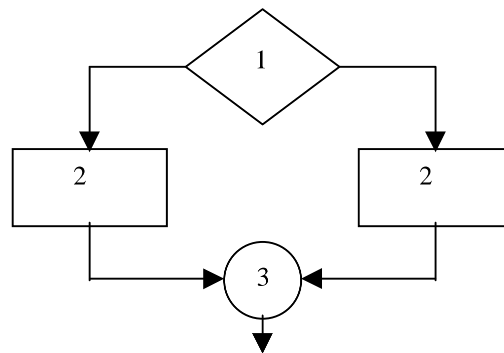
Estos elementos son:
- bucles
- contadores
- acumuladores
- interruptores
- estructuras
  - secuenciales
  - selectivas
  - repetitivas

5.1 Bucles:

Los bucles son elementos que tiene instrucciones que se repiten un cierto número de veces, mientras una condición es cumplida; se conoce como lazo (loop). La condición es un mecanismo que determina las tareas repetitivas el cual debe ser establecido por el programador, éste puede ser verdadero o falso y se comprueba una vez a cada paso o iteración del bucle.

Un bucle está compuesto de tres partes:

1. decisión
2. cuerpo del bucle
3. salida del bucle

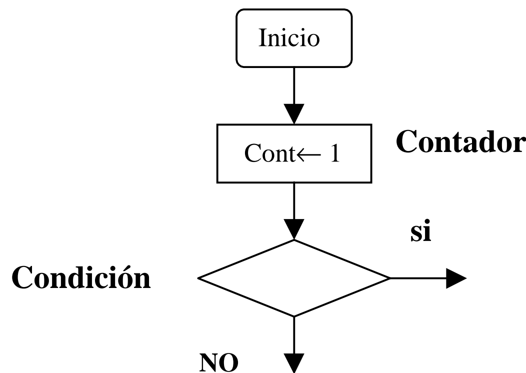


Bucles Anidados: La forma como se disponen los bucles también lleva un nombre, de allí que pueden ser anidados o independientes. Los que son anidados son cuando se encuentran uno contenido en otro, y los independientes son los que se encuentran libres, sin la influencia de otro.

Ejemplo:

5.2 Contadores: Es un dispositivo que sirve para un control permanente del bucle. Este dispositivo llamado Contador es una variable en la que su valor se incrementa o decrementa en una cantidad constante por cada iteración.

Ejemplo:

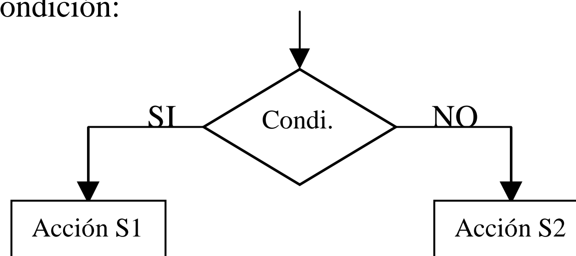


5.3 Acumulador: Conocido también como totalizador, almacena cantidades variables que resulten de sumas sucesivas. Al igual que el contador, cumplen la misma función, con la diferencia de que este último cuenta cantidades no constantes, sino que son variables.

5.4 Decisión o selección: Este tipo de estructura es usada cuando el programador quiere dar dos o más caminos como alternativa en un algoritmo a seguir, según los parámetros especificados. Una instrucción de decisión evalúa una condición y, en función del resultado, se bifurcará el algoritmo a un determinado punto.

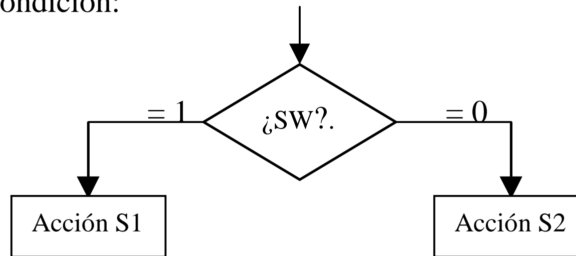
Ejemplo de una decisión o selección:

Gráfico de una condición:



5.5 Interruptores: Variable que puede tomar diversos valores a lo largo de la ejecución de un programa, permitiendo comunicar la información de una parte a otra. Estos pueden tomar dos valores 1 y 0. (“encendido” / “apagado”, “abierto”/ “cerrado”).El interruptor es llamado también conmutador (switch).

Gráfico de una condición:



6. ESCRITURA DE ALGORITMOS/PROGRAMAS

Un algoritmo debe ser sencillo, es decir escrito de una manera clara y entendible, estructurado de modo que su lectura facilite de manera considerable su posterior codificación en un lenguaje de programación cualquiera, los algoritmos deben ser escritos en un lenguaje similar a los programas.

Existen varias maneras de representar algoritmos, la más evidente es el lenguaje natural; sin embargo, no siempre es lo suficientemente preciso. Como alternativas tenemos al Pseudocódigo y el Diagrama de Flujo.
Un algoritmo está constituido por dos componentes: Una cabecera de programa y un bloque algoritmo.

El pseudocódigo es una manera de representar un algoritmo, mediante sentencias similares al lenguaje natural, pero tienen una precisión mayor.

6.1 Cabecera del programa o algoritmo

Existen pasos o procedimientos que se realizan para comenzar un programa. Siempre que se haga debe ponerse un encabezado de programa, en donde debe expresarse el identificador o nombre correspondiente con la palabra reservada que señale el lenguaje, generalmente ésta suele ser **program** que en algoritmia significa **algoritmo.**

6.2 Declaración de Variables

En este punto se describen todas las variables que son usadas en el algoritmo, haciendo una lista de sus nombres y especificando a qué tipo corresponde cada uno. En esta sección se comienza con la palabra reservada **var** (abreviatura de variable) y es de la forma

Var

lista de variables-1: tipo-1
lista de variables-2: tipo-2
.
.
lista de variables-n: tipo-n

siendo cada lista de variables una variable simple o una lista de variables separadas por comas y cada tipo es uno de los tipos de datos básicos (entero, real, char o boolean).

Sería lo más recomendable en la práctica de programación el uso de nombres de variables cuyo significado nos sugieran lo que ellos representan, eso hará al programa

más fácil de entender. También la inclusión de breves comentarios que indiquen cómo se usa la variable, es una buena práctica.

6.3 Declaración de Constantes Numéricas

En este punto se declararán todas las constantes de carácter estándar; es decir, que tengan nombre y un valor ya conocido o valores que ya no pueden variar en el transcurso del algoritmo.

Por ejemplo:

```
const
    Pi =3.141592
    Tamaño = 43
    Horas = 6.50
```

El valor de cada una de estas constantes será completamente invariable.

6.4 Declaración de Constantes y Variables caracter

Las constantes de caracter simple y de cadenas de caracteres se pueden declarar en la sección del programa **const,** igualmente sucede con las constantes numéricas.

```
Const
    castillo = '*'
    frase = 'el caballo es blanco'
    mensaje = 'te veo luego'
```

Las variables de caracter se declaran de dos modos:

Almacenando un solo caracter.

Var *nombre, inicial, nota: carácter*

Se declaran nombre, inicial, nota y letra quien solamente almacenará un caracter.

Almacenando múltiples caracteres (cadenas).

El almacenamiento de caracteres múltiples va a depender del lenguaje de programación en que se esté trabajando. Entonces, en:

BASIC nombre variable cadena = cadena de caracteres
NOMBRES = "Luis Ricardo Alvarez Carrasco"

6.5 Comentarios

El contenido de un programa es el conjunto de la información interna y externa al programa que facilitará su mantenimiento posterior y puesta a punto.

El contenido interno es el que se acompaña en el código o programa fuente y se realiza a base de comentarios significativos. Estos comentarios son representados con diferentes notaciones, según sea el lenguaje de programación en que se trabaje.

6.6 Estilo de escritura de algoritmos/programas

Un método para escribir algoritmos será el siguiente:

Algoritmo identificador {cabecera}
{sección de declaraciones}

var lista de identificadores: tipo de datos
const lista de identificadores = valor

inicio
<sentencia S1>
<Sentencia S2> {cuerpo del algoritmo}
.
.
<Sentencia Sn>
fin

Debemos Recordar:

- La declaración de constantes y variables serán omitidas o descritas en una tabla de variables, que hace las mismas funciones, pero en ciertas ocasiones.
- Para escribir las cadenas de caracteres, éstas serán encerradas entre comillas simples.
- Se recomienda utilizar sangrías en los bucles o en aquellas instrucciones que harán una mejor legibilidad al programa como **inicio** y **fin**.

CAPITULO IV
LAS TECNICAS DE PROGRAMACION

INTRODUCCION

El diseño de un programa que se realiza sin seguir una metodología puede funcionar, pero se debe tener en cuenta que con el tiempo se convertirá en un conjunto de instrucciones. Es decir que las consecuencias de no utilizar un método determinado llevará a cometer errores que pueden costar el buen funcionamiento del mismo.

Las diferentes etapas del programa suelen tener discontinuidad y son difícilmente identificables. En consecuencia existe una difícil fase de desarrollo y mantenimiento.
Aquí se identifican algunos de los problemas que suelen presentarse:

- Se presenta un exceso en la rigidez del programa, lo que implica que sean difícilmente adaptables a cualquier tipo de configuración.
- Se pierde mucho tiempo en la corrección de errores.
- Los programas generalmente son propios de cada programador, lo que implica que no sean muy manejables por otros.
- Para cuando se realice la documentación final, existirán deficiencias por presentarse la ausencia de diagramas, habrán descripciones que no estén completas o simplemente no las habrá, y la documentación no estará actualizada.

Existe una larga lista de problemas que pueden presentarse pero en este caso sólo se han citado algunos.
Es de suma importancia poder prevenir las modificaciones que puedan realizarse en el futuro, así como también la actualización de la documentación.
Para esto, se citan algunas que son importantes como:

- Incrementar el volumen de datos y estructuras.
- Modificación en la forma como se organiza la información.
- Modificación por actualización de los documentos.
- Ampliación, reducción o sustitución en el sistema del proceso de datos.

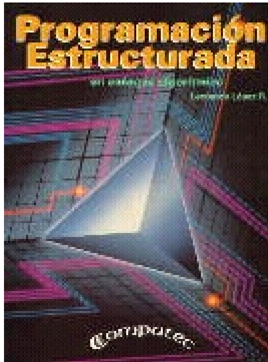
La creación de programas debe tener la flexibilidad suficiente para ser modificables en el momento en que se requiera. Estos deben ser claros, simples, con el fin de poder ser leídos e interpretados de forma fácil.

Con lo dicho anteriormente, se entiende que para la programación deberá asumirse ciertas normas que permitan la estandarización de la programación, implicando una disminución en costos, independencia del programador y seguridad.

Cuando existen problemas con cierto grado de complejidad, el diseño del algoritmo requiere de una reducción y simplificación en la legibilidad del algoritmo. Las técnicas de programación **Modular y Estructurada** son de gran ayuda para la solución de problemas de este tipo, consiguiendo mayor rapidez y eficacia. Para el diseño de un programa, el problema se descompone en módulos (independientes cada uno), se hace la programación de cada módulo mediante métodos estructurados, los que posteriormente son unidos mediante el uso de procedimientos ascendentes o descendentes.

1. INTRODUCCION A LA PROGRAMACION ESTRUCTURADA

Actualmente, dado el tamaño considerable de las memorias centrales y las altas velocidades de los procesadores, la forma de escritura de los programas ha sido considerada una de las características más sobresalientes en las técnicas de programación. El entendimiento de los algoritmos y luego de los programas, exige que su diseño sea fácil de comprender y su flujo lógico un camino fácil de seguir.



La descomposición de programas en módulos más simples de programar se dará a través de la programación modular, y la programación estructurada permitirá la escritura de programas fáciles de leer y modificar. En un programa estructurado, el flujo lógico se gobierna por las estructuras de control básicas.

1. Secuenciales.
2. Repetitivas.
3. Selección.

2. PROGRAMACION MODULAR

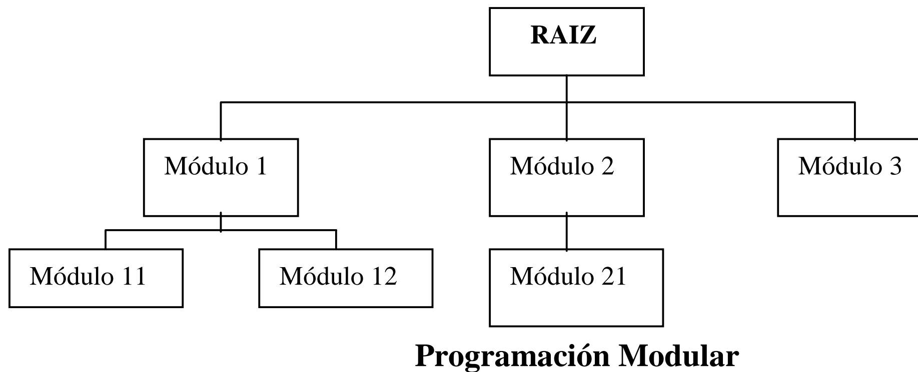
Este es uno de los métodos para el diseño más flexible y de mayor performance para la productividad de un programa. En este tipo de programación el programa es dividido en módulos, cada uno de las cuales realiza una tarea específica, codificándose independientemente de otros módulos. Cada uno de éstos son analizados, codificados y puestos a punto por separado.

Los programas contienen un módulo denominado módulo principal, el cual supervisa todo lo que sucede, transfiriendo el control a submódulos (los que son denominados subprogramas), para que puedan realizar sus funciones. Sin embargo, cada submódulo devolverá el control al módulo principal una vez completada su tarea. Si las tareas asignadas a cada submódulo son demasiado complejas, se procederá a una nueva subdivisión en otros módulos más pequeños aún.

Este procedimiento se realiza hasta que cada uno de los módulos realicen tareas específicas. Estas pueden ser entrada, salida, manipulación de datos, control de otros módulos o alguna combinación de éstos. Puede ser que un módulo derive el control a otro mediante un proceso denominado bifurcación, pero se debe tomar en cuenta que esta derivación deberá ser devuelta a su módulo original.

En cuanto a la seguridad podemos decir que los módulos son independientes, de modo que ningún módulo puede tener acceso directo a cualquier otro módulo, excepto el módulo al que llama y sus submódulos correspondientes. Sin embargo, los resultados producidos por un módulo pueden ser utilizados por otro módulo cuando se transfiera a ellos el control.

Dada la ventaja de ser independientes el programa puede ser trabajado por diferentes programadores a la vez. Con ésto se ahorra tiempo en el diseño del algoritmo y en su posterior codificación. También un módulo puede ser codificado sin afectar a los demás, incluso sin alterar su función principal.



Programación Modular

2.1. Medida de los Módulos

La determinación del tamaño de los módulos significa un obstáculo para la programación modular. No existe un estándar qué determine que tamaño debe tener un módulo, pero existen alternativas que deben tomarse en cuenta, como una aproximación del tamaño físico de una página, es por eso que los programadores tienen la tarea difícil de poder hacer que los módulos tengan aproximadamente esta medida

2.2 Implementación de los Módulos

Este tipo de programación puede implementarse utilizando módulos que toman diferentes nombres, según el lenguaje de programación en el cual estén expresados: ***Subrutinas*** en ***BASIC, procedimientos*** en Pascal, ***subrutinas*** en ***FORTRAN***, **secciones** en ***COBOL*** y **funciones** como módulos en todos los lenguajes.

3. PROGRAMACION ESTRUCTURADA

Cuando hablamos de Programación Estructurada, nos referimos a un conjunto de técnicas que con el transcurrir del tiempo han evolucionado. Gracias a éstas, la productividad de un programa se ve incrementada de forma considerable y se reduce el tiempo de escritura, de depuración y mantenimiento de los programas. Aquí se hace un número limitado de estructuras de control, se reduce la complejidad de los problemas y se minimiza los errores.

Gracias a la programación estructurada, es más fácil la escritura de los programas, también lo es su verificación, su lectura y mantenimiento. Esta programación es un conjunto de técnicas que incorpora:

- diseño descendente (*top-down*)
- recursos abstractos
- estructuras básicas

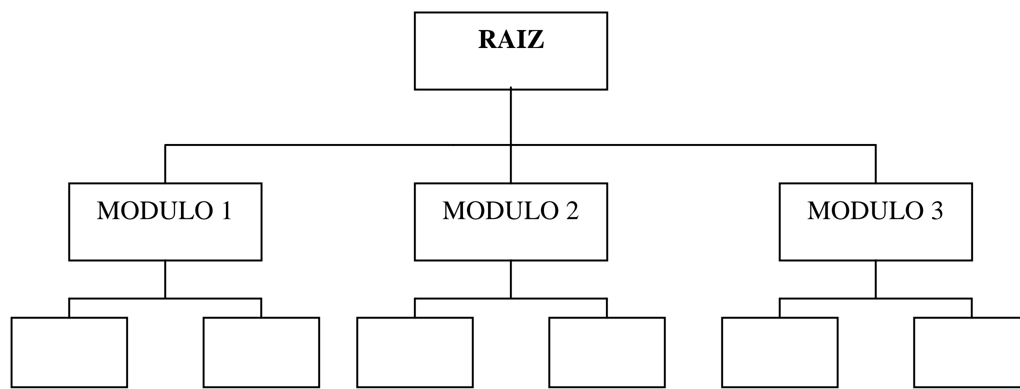
3.1 Recursos Abstractos

Los recursos abstractos son utilizados como un apoyo en la programación estructurada, en vez de los recursos concretos de los que se dispone (lenguaje de programación determinado).

Para disgregar un programa en términos de recursos abstractos debemos descomponer acciones complejas en acciones más simples, las que son capaces de ejecutar o constituyen instrucciones de computadora disponible.

3.2 Diseño descendente (Top-Down)

Este es un proceso en el cual el problema se descompone en una serie de niveles o pasos sucesivos (stepwise). Esta metodología consiste en crear una relación entre las etapas de estructuración, las que son sucesivas, de tal forma que se interrelacionen mediante entradas y salidas de información. Considerando los problemas desde dos puntos de vista: ¿que hace? y ¿cómo lo hace?



Este es un típico diseño descendente

**3.4 Teorema de la Programación Estructurada:
Estructuras Básicas**

Un programa propio puede ser escrito utilizando sólo tres tipos de estructuras de control:

- **secuenciales**
- **selectivas**
- **repetitivas**

Podemos definir un programa como propio si cumple con las siguientes características:

- Tiene un solo punto de entrada y uno de salida o fin de control del programa.
- Existen caminos desde la entrada hasta la salida que se pueden seguir y que pasan por todas partes del programa.
- Todas las instrucciones son ejecutables y no existen lazos o bucles infinitos (sin fin).

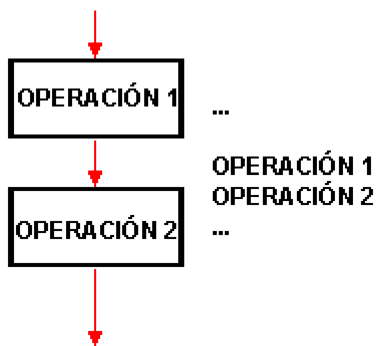
4. ESTRUCTURA SECUENCIAL

Es la estructura en donde una acción (instrucción) sigue a otra de manera secuencial. Las tareas se dan de tal forma que la salida de una es la entrada de la que sigue y así en lo sucesivo hasta cumplir con todo el proceso. Para la realización de esta estructura secuencial nos apoyamos en unas estructuras a las que llamaremos *Estructuras de Control.*

Las *Estructuras de Control* determinan la secuencia en que deben ejecutarse las instrucciones de un algoritmo.

Existen tres *Estructuras de control* básicas o primitivas, y combinándolas se puede escribir cualquier algoritmo. Estas estructuras primitivas son: la *secuencia*, la *bifurcación condicional* y el *ciclo*.

Secuencia:



La estructura de control más simple es la *secuencia*. Esta estructura permite que las instrucciones que la constituyen se ejecuten una tras otra en el orden en que se listan. Por ejemplo, considérese el siguiente fragmento de un algoritmo:

En este fragmento se indica que se ejecute la *operación 1* y a continuación la *operación 2*.

5. ESTRUCTURAS SELECTIVAS

Es de gran utilidad la especificación formal de los algoritmos, para cuando éste requiera una descripción más complicada que una lista sencilla de instrucciones. Este es el caso cuando existe un número de posibles alternativas que resulten de la evaluación de una determinada condición.

Este tipo de estructuras son utilizadas para tomar decisiones lógicas, llamándose por esta razón ***estructuras de decisión*** o ***alternativas***.

En esta estructura es evaluada una condición y de acuerdo al resultado el algoritmo opta por una de las alternativas. Las condiciones son especificadas utilizando expresiones lógicas. Para representar una estructura selectiva se hace uso de palabras en pseudocódigo.

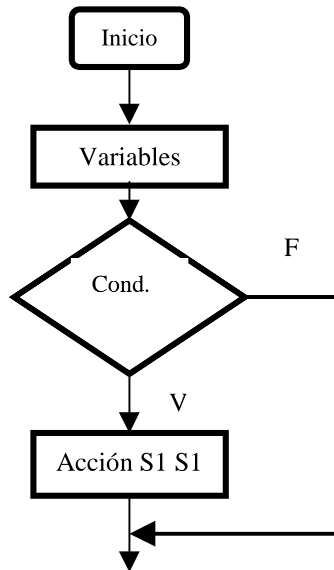
Las estructuras selectivas o alternativas pueden ser:

- Simples
- Dobles
- Múltiples

5.1 Alternativa simple (si-entonces/if-then)

La estructura alternativa simple si-entonces (en inglés if-then) lleva a cabo una acción al cumplirse una determinada condición. La selección si-entonces evalúa la condición y

- si la condición es verdadera, ejecuta la acción SI
- si la condición es falsa, no ejecuta nada.



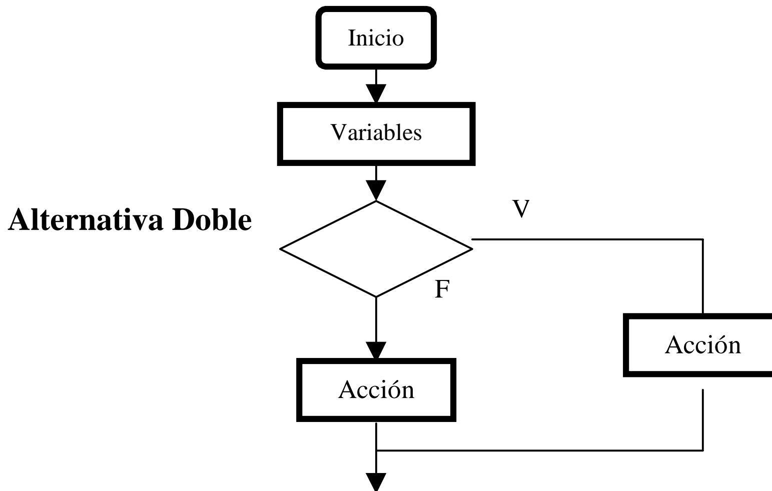

```
En español:
Si<condición>
        Entonces<acción S1>
Fin_si

En Inglés:
If<condición>
        Then <acción S1>
End_if
```

Alternativa Simple

5.2 Alternativa doble (si-entonces-sino/if-then-else)

Existen limitaciones en la estructura anterior, y se necesitará normalmente una estructura que permita elegir dos opciones o alternativas posibles, de acuerdo al cumplimiento o no de una determinada condición.

- si la condición es verdadera, se ejecuta la acción S1
- si la condición es falsa, se ejecuta la acción S2



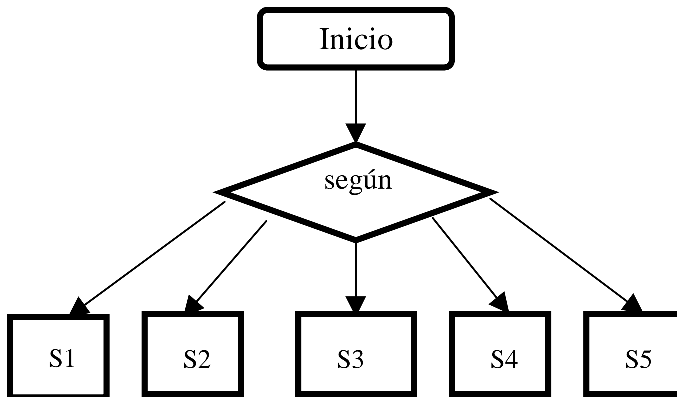

```
En español:
Si <condición>
        entonces <acción S1>
        sino <acción S2>
fin_Si

En Inglés:
If <condición>
        then<acción>
        else<acción S2>
end_if
```

5.3 Alternativa múltiple (según_sea, caso de/case)

Generalmente es necesario que existan más de dos alternativas de las cuales poder elegir. Esta opción podría solucionar problemas que requieren de decisiones no usuales, por estructuras alternativas simples o dobles, anidadas o en cascada. Sin embargo, se pueden plantear serios problemas de escritura del algoritmo y naturalmente de legibilidad, si el número de alternativas es grande.

Esta estructura evaluará una expresión que podrá tomar n valores, distintos todos 1, 2, 3, 4,.., n. Se realizará una de las n acciones de acuerdo a la elección de uno de estos

valores, o lo que es igual el flujo del algoritmo seguirá un determinado camino entre los n posibles.



Alternativa múltiple

En español:
Según_sea expresión **(E) hacer**
e1: acción S1
acción S2
.
.
e2: acción S21
acción S22
.
.
en: acción S31
acción S32

otros: acción Sx
fin_según

En Inglés:
Case expresión **of**
[e1]: acción S1
[e2]: acción S2
.
.
[en]: acción Sn
otherwise
acción Sx
end_case

6. ESTRUCTURAS REPETITIVAS

El diseño de las computadoras está hecho especialmente para aquellas aplicaciones en las que una operación o conjunto de operaciones deben repetirse muchas veces. Para ello es importante la estructura del algoritmo, necesario para repetir una o varias acciones, un número determinado de veces.

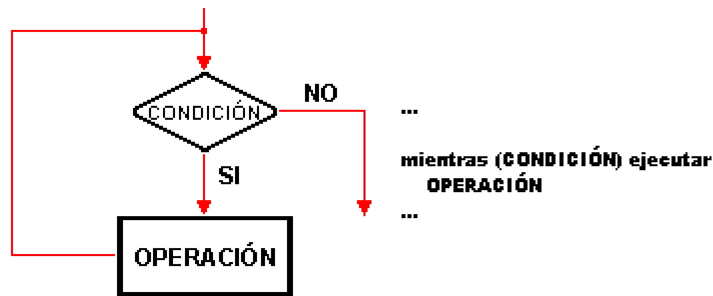
Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan ***bucles*** y, al hecho de repetir la ejecución de una secuencia de acciones se denomina ***iteración.***

6.1 Estructura mientras (“while”)

Esta estructura repetitiva **mientras**, es en la que el cuerpo del bucle se repite en tanto se da una determinada condición.
Los ciclos son estructuras de control que permiten ejecutar varias veces una operación. Existen varios tipos de ciclos:

Ciclo Mientras:

En este ciclo se repite una operación, mientras se cumpla una cierta condición. Por ejemplo:



Las palabras clave "mientras" y "ejecutar" señalan que se trata de un **ciclo mientras**. La condición se verifica antes de ejecutar la operación.

Ejecución de un Bucle cero veces

Si se observa en una estructura mientras, lo primero que se da es la evaluación de una expresión booleana. Si es falsa, entonces el cuerpo del bucle no se ejecuta nunca. Esta acción puede parecer inútil, afectará a ningún valor o salida. Sin embargo puede ser la acción deseada.

Bucles Infinitos

En algunas ocasiones los bucles no exigen final y en otras, no tienen fin por errores en su diseño. Los programas o bucles correrán siempre o mientras la máquina esté encendida, e incluso habrán situaciones en las que nunca se cumpla la condición, eso significa que el bucle no se detendrá nunca. A estos bucles se les llama bucles ***sin fin*** o ***infinito,*** y por su condición deben evitarse siempre, por que afectan en forma perjudicial a la programación.

Terminación de bucles con datos de entrada

Si un algoritmo lee una lista de valores con un bucle *mientras*, se debe incluir algún tipo de mecanismo para terminar el bucle. Para llevar ésto a cabo, existen cuatro métodos típicos para terminar un bucle de entrada:

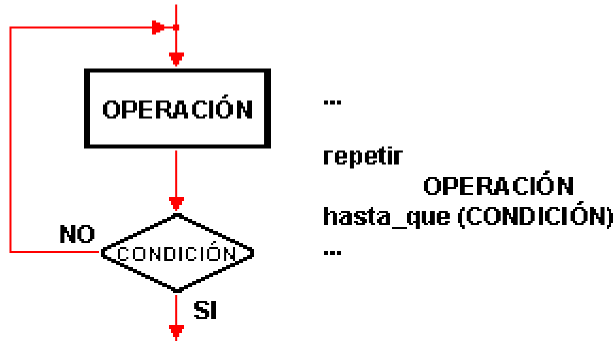
1. Preguntas antes de la iteración.
2. Encabezar la lista de datos con su tamaño.
3. Finalizar la lista con su valor de entrada
4. Agotar los datos de entrada.

6.2 Estructura repetir (“repeat”)

Generalmente se quiere que un bucle se ejecute, mínimo una vez, antes de que la condición de repetición sea cumplida o se compruebe. En la estructura mientras, si la condición es falsa, no se dará ninguna ejecución. Esta estructura se ejecutará mientras se cumpla una condición determinada, la cual es comprobada al final del bucle.

Ciclo Repetir:

En este ciclo se ejecuta una operación hasta que se cumpla una condición.



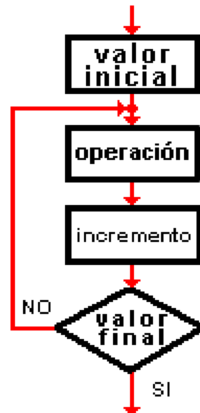
Ahora se ejecuta primero la operación y se verifica la condición después.

6.3 Estructura desde/para ("for")

Muchas veces ya se conoce el número de veces que se deben ejecutar las acciones de un bucle. Pero cuando se quiere que el número de iteraciones sea un número determinado, los programadores utilizan un tipo de estructura ***desde*** o ***para*** (for, en inglés).

Ciclo Desde:

En este ciclo se ejecuta una operación un cierto número de veces, especificando en un contador el incremento unitario, desde un Valor Inicial hasta un Valor Final que marcará la condición de salida del ciclo. Por ejemplo:



Esta estructura accionará el bucle las veces que se requiera, llevando de manera automática un control sobre el número de iteraciones o pasos a través del cuerpo del bucle.

```
...
desde <variable de control>←<valor inicial> hasta <valor final>
  OPERACIÓN
...
```

6.4 Salidas internas de los Bucles

En ciertas ocasiones es necesario contar con una estructura repetitiva que permita la salida de un punto intermedio del bucle al cumplirse una condición. Sólo algunos lenguajes de programación específicos pueden disponer de esta estructura. Deberá tener un nombre que la diferencie de la instrucción ***repetir_hasta,*** la que ya es conocida y se llama ***salir_si*** o ***iterar***. Estas salidas pueden ser válidas para las estructuras ***mientras, repetir*** y ***desde.***

Siendo su formato el siguiente:

```
repetir
        <acciones>
        Salir_si <condición> entonces salir bucle
        <acciones>
fin_repetir
```

La instrucción salir_si no garantiza producir un programa legible y comprensible como lo hacen ***mientras*** y ***repetir***. Lo que sucede es que la salida de un bucle ocurre en el medio del bucle, por eso es que carece de claridad, siendo lo normal que la salida del bucle es el inicio o el final del mismo.

7. ESTRUCTURAS DE DECISION ANIDADAS

Por medio de las estructuras ***si-entonces*** y ***si-entonces-sino*** se llegará a tener que elegir entre dos alternativas, teniendo que elegir una sola. Esta estructura puede ser utilizada también para diseñar estructuras de selección que contengan más de dos opciones. Se podrá tener entonces una estructura si-entonces, dentro de otra estructura si-entonces y otra dentro de esta misma, y así sucesivamente un número indeterminado de veces, existiendo acciones diferentes dentro de cada una de estas estructuras.

Una estructura selectiva múltiple estará conformada de una serie de estructuras si, unas dentro de otras. Se puede utilizar indentación (sangría o sangrado), para evitar complejidad y así el algoritmo sea más claro, creando una correspondencia entre palabras reservadas ***si*** y ***fin_si***, por un lado y ***entonces*** y ***sino,*** por otro.
De acuerdo al lenguaje con que se esté trabajando, la escritura de las estructuras puede variar en unos y en otros.

8. ESTRUCTURAS REPETITIVAS ANIDADAS

Así como las estructuras de selección pueden ser anidadas, es posible que un bucle pueda ser insertado dentro de otro. Para los dos casos, las reglas de construcción de estructuras repetitivas anidadas son iguales: la estructura interna deberá estar totalmente dentro de la externa no pudiendo existir solapamiento.
Las variables índices o de control de los bucles toman valores, de modo tal que por cada valor de la variable índice del ciclo externo se debe ejecutar totalmente en el bucle interno.

9. INSTRUCCIÓN ir_a ("goto")

Existe una sentencia adicional que permite transferir la secuencia de un algoritmo a cualquier parte de él mismo. Es la sentencia ***ir_a(goto)***. Aunque la posibilidad de pasar a cualquier parte de un programa da una sensación de libertad de acción, tiene el terrible inconveniente de perder fácilmente el seguimiento del programa.

Un algoritmo tiene un flujo de control siempre secuencial, con excepción a las estructuras de control que han sido estudiadas con anterioridad, pues realizan transferencias de no control secuenciales.

Esta programación ayuda al usuario a realizar programas fáciles de comprender y legibles utilizando estructuras que ya conocidas, siendo estas tres: **secuenciales, selectivas y repetitivas.** Pero hay situaciones en que se tiene que recurrir a bifurcaciones incondicionales; recurriéndose entonces a la instrucción ***ir_a(goto).***

Siendo **goto** una instrucción ha causado ciertos problemas, por lo que en muchas ocasiones se ha considerado a la instrucción como nefasta y perjudicial para los programadores, recomendando su no uso en sus algoritmos y programas.

Tomando en cuenta que ***ir_a(goto)*** es una instrucción que todos los lenguajes de programación tienen dentro de sus instrucciones, existiendo muchos que dependen más de ellos que otros, tales como **BASIC Y FORTRAN.** Los programas que hacen uso de ésta instrucción pueden ser escritos nuevamente para hacer lo mismo, sin necesidad de recurrir a la instrucción ***ir_a***. Pues es mucho más difícil leer un programa que utiliza instrucciones ir-a, que un programa bien escrito que no usa estas instrucciones, o si las usa son muy pocas, ya que muy pocas veces suelen ser útiles.

10. METODOS DE PROGRAMACION ESTRUCTURADA

Existen diferentes métodos de programación estructurada. Entre los más conocidos se encuentran: Jackson, Bertini y Warnier.

10.1 Método Jackson:

Esta es una metodología creada por Michael Jackson de nacionalidad Inglesa. Esta metodología está basada en que estructuralmente el programa depende o se encuentra en función de la estructura de los datos que se manejan. Michael Jackson se ayuda con la implementación de módulos, siguiendo un orden jerárquico de acuerdo a los diferentes niveles en que se encuentra. Cada módulo se convierte en un dato o en un conjunto de datos.

Las estructuras básicas en este método vienen representadas en el siguiente orden:

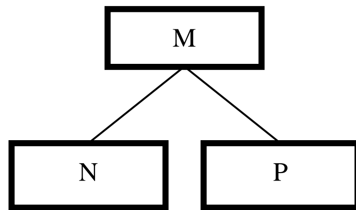
- **Secuencial:** Los módulos en un número determinado se ejecutan una sola vez, respetando un orden jerárquico preestablecido.
- **Repetitiva:** Los módulos son ejecutados desde cero hasta ***n*** veces. Cuando existe un proceso repetitivo, se indica con un asterisco **(*)**.
- **Alternativa:** Uno de los módulos es seleccionado entre todos para ser ejecutado. El proceso se indica por medio de la letra **O.**

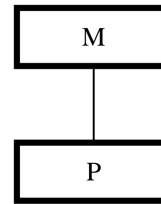
Gracias al desarrollo de estas estructuras se pueden podremos desarrollar otras que intervengan en el diseño del programa.
Este método determina la lectura de arriba hacia abajo y de izquierda a derecha.

Los problemas a resolver por medio de este método, deben seguir un determinado número de pasos:

1° Se definen los datos de entrada y salida.
2° A partir de diferentes estructuras de datos se crea la del programa.
3° El método posee recursos que pueden ser explotados a fin de obtener resultados.
4° Finalmente, se escribe el pseudocódigo y se codifica.

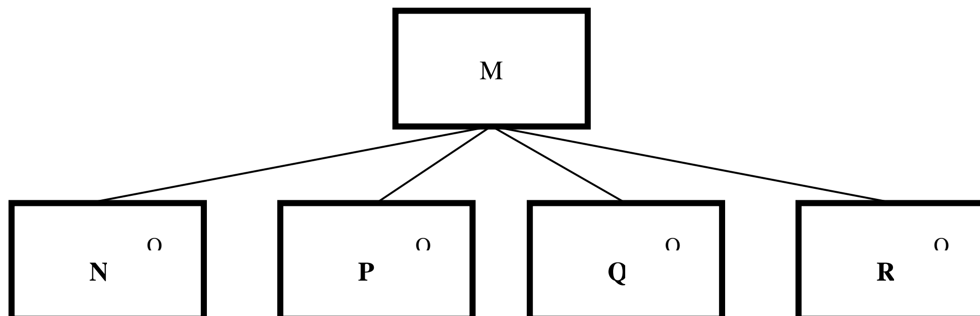
Estructuras Básicas del método de Jackson





Secuencial

Repetitiva



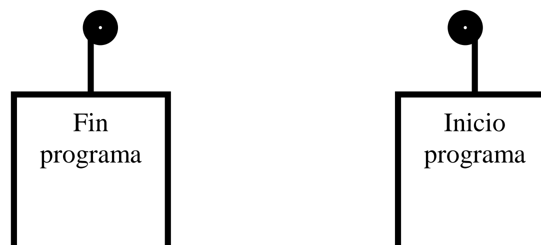
Alternativa

10.2 Método Bertini:

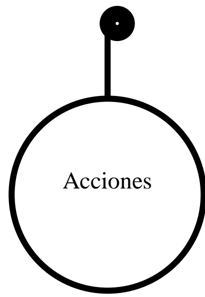
Bertiní aplicó una metodología que consistió en descomponer el problema en niveles. Cada nivel tiene un inicio, una cantidad de procesos y un final. Esta metodología representa la estructura de los programas y no las operaciones del tratamiento.

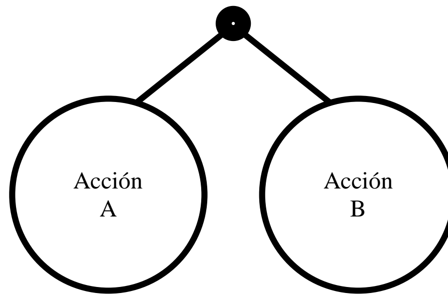
De acuerdo con este método el proceso de ejecución de las instrucciones es de derecha a izquierda, no hay problema si es que el programador quiere leerlo al revés, por que puede hacerlo. El diagramas de flujo del problema anterior, de obtención del máximo y mínimo de una serie de números, aplicando éste método, sería el siguiente:

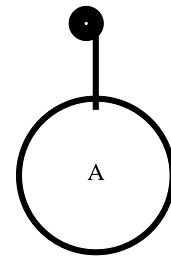
Estructuras Básicas:



secuencial







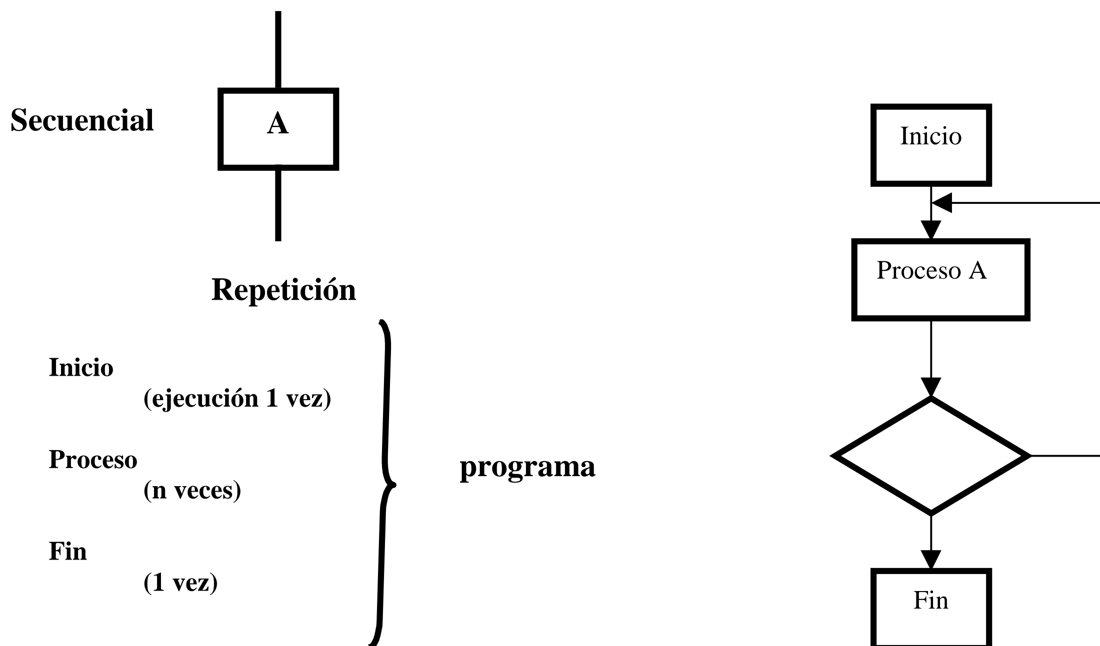
Repetitiva **Alternativa** **Secuencial**

10.3 Método Warnier

Está basado en métodos matemáticos estableciendo un lenguaje único de comunicación entre usuarios, analistas y programadores, lo que permite que cualquier programador pueda entenderlo de manera simple. La representación de cualquier proceso puede hacerse mediante llaves.

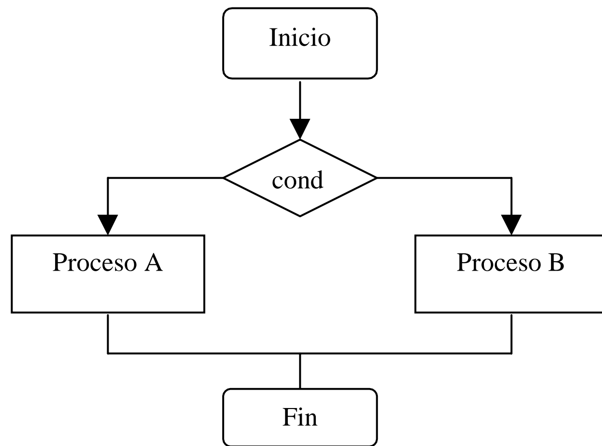
Este método está basado en la descomposición por niveles del problema. En cada nivel los tratamientos son explicados de forma clara para la resolución del problema planteado. Las estructuras que se usan en esta metodología son idénticas a las que se usan en el método de Jackson, aunque su representación puede variar.

Estructuras Básicas:



Alternativa

Inicio
(1 vez)
Proceso A
(0-1 Vez)
Proceso B
(0-1 vez)
Final



CAPÍTULO V.
PROGRAMACION ORIENTADA A OBJETOS

INTRODUCCION

En los últimos años la frase "orientado (a) a objetos", se ha vuelto muy popular, escuchándose a cada momento frases como sistemas "operativos orientados a objetos", "lenguajes orientados a objetos", "programación orientada a objetos" (POO), etc. Sin embargo, el concepto ya tiene un tiempo de aproximadamente 25 años, cuando se dio su nacimiento con la creación del lenguaje Simula. Su redescubrimiento y reciente popularidad se deben a C++, lenguaje que fue creado por Stroustrup y fue basado en un lenguaje que es usado ampliamente como C. En esta parte se dará a conocer los principales conceptos más íntimamente relacionados con la POO, así como la forma de implementarlos en C++.

Se hará mención de los conceptos generales más utilizados en el modelo orientado a objetos, los cuales son: **abstracción, encapsulación** y **modularidad.** Y con respecto a programación, son: **objeto, clase, método, envío y recepción de mensajes, herencia y polimorfismo.**

1. ABSTRACCION:

Es una descripción o especificación simplificada de un sistema que hace énfasis en algunos detalles significativos y suprime los irrelevantes.
La abstracción debe enfocarse más en qué es un objeto y qué hace antes, de pensar en la implementación. Por ejemplo, un automóvil puede abstraerse como un objeto que sirve para desplazarse a mayor velocidad, sin importar cómo lo haga.

Una característica de la abstracción es que un objeto puede abstraerse de diversas formas, dependiendo del observador. Así, el automóvil que se mencionaba puede ser visto como un objeto de colección por un coleccionista, una herramienta de trabajo por un corredor profesional, una mercancía por un vendedor, etc.

2. ENCAPSULAMIENTO:

Típicamente, la información acerca de un objeto está encapsulada por su comportamiento. Esto significa que un objeto mantiene datos acerca de cosas del mundo real a las que representa en su sentido verdadero.
Típicamente a un objeto se le debe "pedir" o "decir" que cambie sus propios datos con un mensaje, en vez de esperar que tales datos de procesos extremos cambien la naturaleza de un objeto.

Al encapsular u ocultar información se separan los aspectos externos de un objeto (los accesibles para todos) de los detalles de implementación (los accesibles para nadie). Con esto se trata de lograr que al tener algún cambio en la implementación de un objeto no se tengan que modificar los programas que utilizan tal objeto.

Siguiendo con el ejemplo del automóvil, se sabe que existen diversos mecanismos para que funcione éste, en particular se tiene el sistema de frenado que todo mundo sabe que sirve para detener el auto al pisar el pedal de freno, pero sólo el mecánico sabe los detalles de la implementación. Por otro lado, si en algún momento se cambia, para el conductor es transparente.

3. MODULARIDAD:

La modularidad consiste en dividir un programa en partes llamadas módulos, los cuales pueden trabajarse por separado. En términos de programación, los módulos pueden compilarse por separado y la división no depende de cierto número de líneas sino es una división en términos de integrar en un módulo un conjunto de procedimientos relacionados entre sí, junto con los datos que son manipulados por tales procedimientos. El objetivo de la modularidad es reducir el costo de elaboración de programas al poder dividir el trabajo entre varios programadores.

Por ejemplo, un automóvil está constituido por un conjunto de módulos tales como un sistema eléctrico, uno mecánico y uno de frenado. Cada módulo se trabaja por separado y el especialista sólo conoce la forma en que se relaciona el módulo con los otros, pero no tiene por qué saber los detalles de funcionamiento de otros módulos o sistema.

Estos conceptos no son exclusivos de la POO, pues se han desarrollado desde la programación estructurada, sólo que en ésta se pueden omitir, desde luego bajo responsabilidad del programador, pues hacerlo, lleva a tener grandes programas en un solo archivo y sin estructura alguna, lo cual causa grandes pérdidas de tiempo al desear modificar tal programa. La POO no puede lograrse sin hacer uso de los mecanismos mencionados.

4.CLASES Y OBJETOS

A pesar de que el punto central en esta nueva metodología de programación es el concepto de objeto, resulta difícil tratar de definirlo. En un diccionario se puede encontrar la siguiente definición:

Un objeto es cualquier cosa que se ofrece a la vista y afecta los sentidos. Es una entidad tangible que exhibe algún comportamiento bien definido.
En términos de programación, un objeto no es necesariamente algo tangible (por ejemplo: un proceso). Lo que sí puede decirse de todo objeto es que tiene estado, comportamiento e identidad.

El estado de un objeto abarca todas las propiedades o características distintivas del mismo y los valores de cada una de estas propiedades. En términos de programación, puede decirse que las propiedades son las variables que sirven para describir tal objeto.

El comportamiento es la forma como actúa o reacciona un objeto en términos de cambio de estado, envío y recepción de mensajes. Está formado por la definición de las operaciones (funciones y procedimientos) que puede realizar este objeto. Los tipos más comunes de operaciones, o en POO métodos, son: modificar, seleccionar, iterar, construir y destruir. El conjunto de operaciones que un objeto puede realizar sobre otro, se conoce como protocolo.

Identidad es la propiedad de un objeto que lo distingue de todos los demás. Es un programa, normalmente se trata de un identificador.

En resumen, un objeto es un conjunto de localidades en memoria con un conjunto de subprogramas (en POO se conocen como métodos) que definen su comportamiento y un identificador asociado. Lo más común es que el programa tenga más de un objeto con

propiedades y comportamientos similares, así que en lugar de repetir la definición de un objeto se agrupan las características comunes de los objetos en una clase.

La clase representa la esencia del objeto y él es una entidad que existe en el tiempo y el espacio. El objeto se define también como instancia de la clase a que pertenece. La clase tiene dos vistas: la exterior, en la cual se hace énfasis en la abstracción y se oculta en la estructura y secretos de comportamiento, y la vista interior o implementación. Aquí se nota que es indispensable hacer uso del concepto de encapsulación.

La vista exterior de un objeto en C++ tiene la siguiente sintaxis:

Ejemplo:

```
Class {
        Private:
        //      Representación de los objetos de esta clase.
        Public:
        //      Declaración de los métodos que 'entienden'
        //      los objetos de esta clase
        protected:
        //      elementos que heredan las clases derivadas de ésta
        } ;
```

En la sección privada de una clase usualmente se definen las estructuras de datos con la que se representará el objeto, y algunos métodos que se emplearán en la definición de la interfaz.

Los datos y métodos aquí definidos no pueden ser accesados directamente fuera de la implementación de la interfaz.

La parte pública o interfaz tiene dos clases de métodos:

Funciones de acceso, las cuales regresan abstracciones con significado acerca de una instancia y funciones de transformación, las que llevan a una instancia de un estado válido a otro.

El ocultamiento de datos implica que todos los datos dentro de una clase deben ser privados a ella, ésto garantiza que la interfaz es una abstracción.

A continuación se presenta un ejemplo: En programación es sabido que un "contador" es un objeto que sirve para ayudar a contar cosas, por tanto se puede incrementar, decrementar o simplemente preguntar por su valor. Esto es una abstracción del objeto contador, y una implementación de ésta, podría ser:

La definición se tiene un archivo, por ejemplo "contador.h" y la implementación de otro llamado con esto se aplica la modularidad.

Ejemplo:

```
//definición de la clase contador. Archivo contador.h
class contador {
private:
        unsigned int valor;
public:
        contador ( ) ;
void incrementa ( ) ;
void decrementa ( ) ;
unsigned int accesa_valor ( ) ;
} ;
```

En este caso se define que cada objeto de la clase contador es un entero sin signo y que tal objeto puede incrementar, decrementar y accesar su valor. Antes de mostrar la implementación, es conveniente recalcar que nadie, excepto el implementador de esta clase, sabe cuál es la representación interna de los objetos de esta clase, por eso se pone en la sección **private.** Otro aspecto que se debe resaltar es, que dentro de los métodos existe uno cuyo nombre es idéntico al de la clase, este método se conoce como constructor y se invoca automáticamente al crear un objeto de esta clase.

La implementación de métodos se tiene en otro archivo y por tanto en el encabezado de cada método, el nombre del mismo va precedido del nombre de la clase a la que pertenece el método y un par de dos puntos.

Ejemplo:

```
//Implementación de la clase contador. Archivo contador. c
#include "contador.h"

contador::contador ( ) {
        valor = 0;
}
void contador::decrementa ( ) {
        if (valor<MAXIMO) valor ++;
}
```

Ejemplo:

```
Void contador::decrementa ( ) {
        If (valor>0) valor -- ;
}

unsigned int contador::accesa_valor ( ) {
        return valor;
}
```

En este ejemplo el constructor únicamente asigna valor inicial al contador.

Ejemplo

```
//Programa de prueba que usa la clase contador
#include <STDIO.H
#include ``contador.h"
main ( ) {
contador c1, c2;
int i;

for (i = 1; i < 15; i++) {
  c1.incrementa( );
  printf("c1 = \%u ", c1.accesavalor( ) );
  c2.incrementa();
}
```

ejemplo

```
printf("Despues del ciclo c2 = \%u ", c2.accesa valor( ) );

for (i = 1; i < 10; i++)
  c2.decrementa( );
printf("El valor final de c2 es = \%u ", c2.accesa valor( ) );
}
```

En el ejemplo se crean dos objetos c1 y c2, de la clase contador. Al hacerlo, automáticamente se llama a su constructor y éste inicializa su valor en cero, como se vio en la implementación.

La forma de enviar mensajes, es decir, de utilizar los métodos es escribiendo el nombre del objeto, un punto y luego el nombre del método que se requiere y entre paréntesis los parámetros que requiera tal método.

Es posible que se piense que es tomarse demasiadas molestias para crear un contador, pero al crearlo como objeto se evitan incongruencias como tt i = 1000 + c1;} es decir, mezclar un contador con cualquier variable entera, como podemos hacer en la programación tradicional.

En la definición de la clase, los constructores pueden tomar diferentes formas como en el siguiente ejemplo, en que se crea la clase de pilas:

Ejemplo:

```
//Interfaz de la pila
class pila {
private:
  int tamano max;
  int tope;
  char * p;
public:
  pila(int tamano);
```

```
    pila(void);
    pila(int tamano, char str[]);
    void push (char c) ;
    int  pop () ;
    \~{}pila() ;
  } ;
```

Ejemplo:

```
  Implementación de la pila
  pila::pila(int tamano) {

p = new char [tamano];
    tamano_max = tamano;
    tope = 0;
  }
  pila::pila(void) {
    p = new char [100];
    tamano_max = 100;
    tope = 0;
  }
  pila::pila(int tamano, char *s) {
    p = new char [tamano];
    tamano_max = tamano;
    strcpy(p, str);
    tope = 0;
  }

  void pila::push(char c)
  { ... }
  int pila::pop()
  { ... }
  pila ::{}pila(void) { delete p; }
```

En este caso se tienen tres posibilidades de constructores: sin parámetros, con un entero que represente el tamaño de la pila y con el tamaño y valor inicial de la pila. Esto permite tener más opciones al crear pilas.

{\ejemplo pila p1, p2(20), p3(5,"Hola");

En este caso p1 es una pila de 100 elementos, p2 de 20 y p3 de 5 con valor cada uno de los caracteres de la cadena ``Hola".

El destructor es sólo uno y tiene como nombre el mismo que la clase, pero precedido por una tilde. Este método se invoca automáticamente al salir del alcance donde se crea el objeto, y su función es liberar el espacio de memoria solicitado para tal objeto en la creación del mismo.

5. HERENCIA

La *herencia* es la contribución más importante de la POO, pues mediante este mecanismo es posible lograr la principal meta de la POO que es la reutilización de código.

La herencia permite proporcionar una jerarquía de clases. En tal jerarquía, algunas clases son subordinadas a otras llamadas subclases, o como en C++, clases derivadas. Una subclase define el comportamiento de un conjunto de objetos que heredan algunas de las características de la clase padre, pero adquieren características especiales no compartidas por el padre, en este sentido se dice que la subclase es una especialización de la clase padre.

La herencia define relaciones entre clases, donde una clase comparte la estructura o comportamiento definidos en una o más clases.

Por ejemplo, si se desea crear un sistema que maneje datos de estudiantes y de profesores, es fácil notar que independientemente de que sean alumnos o profesores todos son personas y por tanto tienen nombre y dirección.

Persona:
- **char*ap_paterno;**
- **char*ap_materno;**
- **char*nombre;**
- **char*calle_num;**
- **char*ciudad;**
- **char*edo;**
- **char*cp;**

Profesor:
- **char*ap_paterno;**
- **char*ap_materno;**
- **char*nombre;**
- **char*calle_num;**
- **char*ciudad;**
- **char*edo;**
- **char*cp;**
- **char*depto;**
- **float sueldo;**

Estudiante:
- **char*ap_paterno;**
- **char*ap_materno;**
- **char*nombre;**
- **char*calle_num;**
- **char*ciudad;**
- **char*edo;**
- **char*cp;**
- **char*carrera;**
- **long num_cta;**
- **int semestre;**

Pasante:
- **char*ap_paterno;**
- **char*ap_materno;**
- **char*nombre;**
- **char*calle_num;**
- **char*ciudad;**
- **char*edo;**
- **char*cp;**
- **char*carrera;**
- **long num_cta;**
- **int semestre;**
- **char*tesis;**
- **int avance;**

En C++ todos los miembros declarados como protegidos permanecen ocultos, de la misma forma que los privados, excepto para sus subclases. La definición de estos objetos en C++ se daría de la siguiente forma:

Ejemplo:

Clases y subclases de personal universitario (profesores y alumnos)

```
Class persona
Private:
Char * ap_paterno;
Char * ap_materno;
Char * nombre;
Char * calle_num;
Char * ciudad;

Char = edo;
        Char = cp;
        Public:
```

Uso del mecanismo de herencia, en la definición de la clase se pone después del nombre de la clase de la cual es heredera. La palabra **public** indica que los objetos de la clase derivada pueden usar todos los métodos de sus padres a menos que éstos se redefinan en la clase derivada.

```
Class estudiante : public persona {
        private:
                char * carrera;
                long num_cta;
                int grado;
        public:
                estudiante ( ) :  ( ) { }
                void inserta_carrera (char * s);
                void inserta_num_cta (long id);
                void inserta_grado (int g);
                void imprime ( );
} ;
                void estudiante: : imprime ( ) {
                        persona: : imprime ( ) :
                        cout<<"carrera → "<<carrera;
>>>
```

Pasante es un caso particular de **estudiante,** así que se define como una subclase de **estudiante** y por lo tanto hereda las propiedades de **estudiante** y también de **persona.**

Ejemplo:

```
class pasante : public estudiante {
        char *tesis;
        int avance;
public:
```

```
        pasante ( ) : ( ) { }
        void inserta_tesis (char *s);
        void inserta_avance (int a);
        void imprime ( );
}

void pasante: : imprime( ) {
estudiante::imprime( );
cout <<"Titulo de la tesis>>: "<<Tesis<<"Porcentaje avance de>>> "<<avance;>}
```

Finalmente se define **profesor** como un caso particular de **persona.**

Ejemplo:

```
class profesor :public persona {
private:
        char *depto;
        float sueldo;
public:
        profesor ( ) : ( ) { }
        void inserta_depto (char*d) ;
        void inserta_sueldo (float s);
        void imprime ( );
};

void profesor : : imprime ( ) {
persona : : imprime ( );
cout <<"Depto- ->>: "<<depto<<"salario   >>>>$"<<sueldo; >>}
```

En el método **imprime** de la clase **profesor** se utiliza el método **imprime** de la clase **persona,** para que imprima el nombre y la dirección del profesor, y en la segunda línea se agrega la impresión de los datos específicos del profesor.
Como se pudo apreciar, el uso de la herencia evita el tener que modificar programas existentes o peor aún, reescribirlos.

En C++ el mecanismo de herencia no sólo contempla herencia sencilla(donde se tiene un solo padre) como la mostrada en el ejemplo anterior, también se tiene la posibilidad de que una clase herede más de una clase, es decir se permite tener herencia múltiple. En ese caso, en la definición de la clase se especifica cuáles serán los padres de la clase hijo como antes.

```
Class hijo: public padre, public madre {....};
```

6. POLIMORFISMO

Otro de los mecanismos aportados por la POO es el de *polimorfismo,* el cual es la capacidad de tener métodos con el mismo nombre pero que su implementación sea diferente. En la vida diaria se presenta el siguiente ejemplo de polimorfismo: al tratar de frenar un vehículo siempre se debe oprimir el pedal del lado izquierdo y el vehículo se detendrá sin importar si los frenos son de tambor o de disco.

Una forma de polimosfismo en POO se da al usar un operador para aplicarlo a elementos de diferente tipo. Por ejemplo, al pretender sumar enteros, reales o complejos, se emplea el mismo símbolo +, esto se conoce como sobrecarga de operadores. En este caso el compilador se encarga de determinar cuál es el método que se está invocando de acuerdo a los objetos involucrados en la operación. Se presenta nuevamente el ejemplo inicial del contador, sólo que ahora usando sobrecarga de operadores en lugar de inventar nombres para las operaciones:

```
class contador {
        private:
                unsigned int valor;
        public:
                contador ( ) ;
                void operator ++ ( ) ;
                void operator -- ( ) ;
                unsigned int operator ( ) ( ) ;
                El operador ( )
};
```

La implementación de las operaciones es igual que antes, sólo cambia el encabezado para indicar que se trata de un operador.

```
#include "cont2.h"
        contador::contador ( ) {
                valor = 0;
}
        void contador::operator ++ ( ) {
                if (valor < 65535) valor++;
}
        void contador::operator -- ( ) {
                if (valor > 0) valor --;
}
        unsigned int contador::operator ( ) ( ) {
                return valor;
}
```

Finalmente se muestra el ejemplo de uso de objetos de esta clase, donde puede observarse el uso más natural de las operaciones con contadores:

Programa de prueba que usa la clase contador

```
#include
#include "cont 2.h"

main ( ) {
        contador c1, c2;
        int i;
        for (i = 1; i < 15; i++) {
                c1++;
                printf("\nc1 = %u ", c1( ) );
                c2++;
        }
```

```
        printf ("\nDespues del ciclo c2 = %u ", c2( ) );
        for (i = 1; i < 10; i++)
        c2--;
        printf("\nEl valor final de c2 es = %u \n", c2( ) );
}
```

Así como hay sobrecarga de operadores, también la hay de métodos, como se ha visto con los constructores de los cuales se pueden tener más de uno por clase, y en el ejemplo de herencia también se usó sobrecarga de métodos con la función imprime, la cual tiene diversas formas dependiendo de la clase que se trate. Pero el polimorfismo puro es el que se da a momento de ejecución como en el siguiente ejemplo, en el cual se tienen diversas figuras geométricas (de momento no interesa su representación, así que se omite) con un método para dibujarlas:

```
class figura {
        public:
                virtual void dibuja (void) = 0;
};
class circulo : public figura {
        public:
                void dibuja(void);
};
class cuadro : public figura {
        public:
                void dibuja(void);
};
class linea : public figura {
        public:
                void dibuja(void);
};
//Implementacion de las tres clases derivadas de FIGURA
#include "polim.h"
void circulo::dibuja(void) {
        cout <<" 00";
        cout <<" 0 0";
        cout <<" 00";
}
void cuadro::dibuja(void) {
        cout <<" +---+";
        cout <<" |      |";
        cout <<" +---+";
}

void linea :: dibuja (void) {
        cout " \\\\ ";
        cout " \\\\ ";
        cout " \\\\ ";
}
```

Ejemplo

Programa que hace uso del polimorfismo

```
#include "polim.h"
main ( ) {
        figura *p[3];
        p[0] = new circulo;
        p[1] = new cuadro;
        p[2] = new linea;
        for (int i =0;
                i <= 2; i++) p[i]->dibuja( );
}
```

En el programa se crean tres figuras diferentes, y no es sino hasta el momento de ejecución en que se sabe a qué método se invoca para imprimirla dependiendo del valor de la variable.

CONCLUSIÓN

Como conclusión podemos apreciar que la programación es muy importante en la actualidad y lo seguirá siendo por un buen tiempo, pero así como recalcamos su importancia decimos que lo más importante no es el lenguaje de programación sino el planteamiento de la solución al problema, ya que podría caerse en un error al pensar que sin las técnicas o métodos que se emplean para la programación vamos a llegar fácilmente a la elaboración correcta de un programa, lo cual no es imposible cuando se trata de programas sencillos y pequeños, pero cuando se tienen que elaborar programas muy grandes y con un alto grado de complejidad hay problemas para su diseño, por que para esto se debe seguir un orden y utilizar técnicas que deben ser tomadas muy en consideración para el diseño y la estructura general de los programas.

En los casos de la Programación Modular, Estructurada y Orientada a Objetos, éstas tienen fundamentos que siguen una secuencia lógica y es por eso que tienen una relación muy estrecha, hay que tomar en cuenta que la lógica es muy importante en la programación ya que el programador tendrá que ayudarse en muchas ocasiones de su ingenio para lograr lo que se pretende, y no debemos olvidar que la práctica constante es la mejor enseñanza, esperamos entonces que ésta publicación les sea de gran ayuda.

BIBLIOGRAFÍA

Fundamentos de Programación.
Luis Joyanes Aguilar.
McGRAW – HILL/INTERAMERICANA DE ESPAÑA S.A.

Lenguajes de Programación Diseño e Implementación.
Terrece W. Pratt and Marvin V. Zelkwitz
A Simon & Schuster Company

Booch, Grady. Object Oriented Design with applications.
The Benjamin/Cummings Publishing Company, Inc. 1991.

Meyer Bertran.
Happy 25 Anniversary Objects SIGSPublications. 1989.

Pohl, Ira. C++ for C Programmers.
TheBenjamin/Cummings Publishing Company, Inc. 1989.

Stroustrup, Bjarne.
The C++ Programming Language. Addison
WesleyPublishing Company. 1987.

Stroustrup, Bjarne.
What is Object Oriented Programming? IEEE Software. Mayo 1988.

Wiener, Richard; Pinson, Lewis.
An introduction to object-oriented programming Addison-Wesley Publishing Company. 1990.