REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD EXPERIMENTAL POLITÉCNICA
DE LA FUERZA ARMADA NACIONAL
(UNEFA)

Algoritmos

Adaptación del Manual "**El algoritmo, una iniciación a la programación**" realizado por Wilder Urbáez, hecha por Ing° Luis Castellanos MSc

Índice

luiscastellanos@yahoo.com

1 Algoritmos

1.1 Introducción

- El desarrollo de algoritmos es un tema fundamental en el diseño de programas o soluciones. Por lo cual, el alumno debe tener buenas bases que le sirvan para poder crear de manera fácil y rápida sus programas.
- La siguiente guía pueden servir de apoyo a profesores, en su labor cotidiana de enseñanza y al estudiante, para facilitarles el desarrollo de su capacidad analítica y creadora, para de esta manera mejorar su destreza en la elaboración de algoritmos que sirven como base para la codificación de los diferentes programas que tendrá que desarrollar a lo largo de su carrera.

1.2 ¿Qué es Algoritmo?

- La palabra algoritmo se deriva de la traducción al latín de la palabra árabe "*Alkhowarizmi*", nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.
- Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

1.3 ¿Cuáles son los Tipos de Algoritmos que existen?

- Existen dos tipos y son llamados así por su naturaleza:
    - Cualitativos: Son aquellos en los que se describen los pasos utilizando palabras.
    - Cuantitativos: Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.
- Lenguajes Algorítmicos:
    - Un Lenguaje algorítmico es una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso.
    - Tipos:
        - Gráficos: Es la representación gráfica de las operaciones que realiza un algoritmo (diagrama de flujo).
        - No Gráficos: Representa en forma descriptiva las operaciones que debe realizar un algoritmo (seudo código).

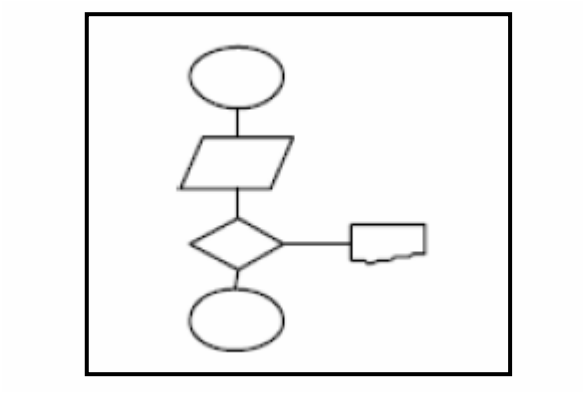


Diagrama de Flujo

```
INICIO
Edad: Entero
ESCRIBA “cual es tu edad?”
Lea Edad
SI Edad >=18 entonces
ESCRIBA “Eres mayor de Edad”
FINSI
ESCRIBA “fin del algoritmo”
FIN
```

Seudo Código

1.4 Para comenzar a Programar

- El computador es una máquina que por sí sola no puede hacer nada, necesita ser programada, es decir, introducirle instrucciones u ordenes que le digan lo que tiene que hacer. Un programa es la solución a un problema inicial, así que todo comienza allí: en el Problema. El proceso de programación es el siguiente: Dado un determinado problema el programador debe idear una solución y expresarla usando un algoritmo (aquí es donde entra a jugar); luego de esto, debe codificarlo en un determinado lenguaje de programación y por último ejecutar el programa en el computador, el cual refleja una solución al problema inicial. Esto es a grandes rasgos lo que hace el programador de computadores.

1.5 Metodología para la solución de problemas por medio de computadora

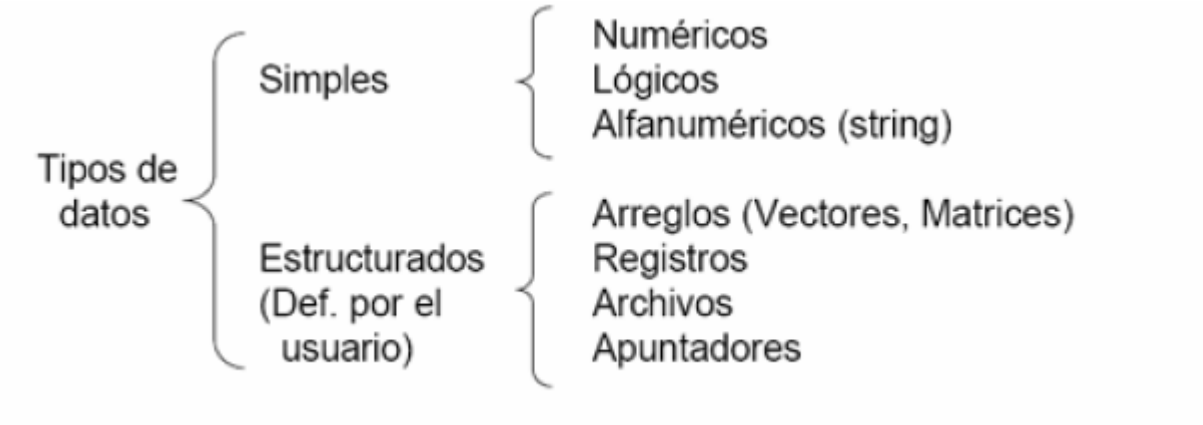
- DEFINICIÓN DEL PROBLEMA
  - Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se

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

- ANÁLISIS DEL PROBLEMA
  - Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:
    - Los datos de entrada.
    - Cual es la información que se desea producir (salida)
    - Los métodos y fórmulas que se necesitan para procesar los datos.
    - (Una recomendación muy práctica es el de colocarse en el lugar de la computadora y analizar qué es lo que se necesita que se ordene y en qué secuencia para producir los resultados esperados.)
- DISEÑO DEL ALGORITMO
  - Las características de un buen algoritmo son:
    - Debe tener un punto particular de inicio.
    - Debe ser definido, no debe permitir dobles interpretaciones.
    - Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.
    - Debe ser finito en tamaño y tiempo de ejecución.
- Prueba de escritorio o Depuración
  - Se denomina prueba de escritorio a la comprobación que se hace de un algoritmo para saber si está bien hecho. Esta prueba consiste en tomar datos específicos como entrada y seguir la secuencia indicada en el algoritmo hasta obtener un resultado, el análisis de estos resultados indicará si el algoritmo está correcto o si por el contrario hay necesidad de corregirlo o hacerle ajustes.

1.6 Entidades primitivas para el desarrollo de algoritmos

- Todos estos elementos con los cuales se construyen dichos algoritmos se basan en una disciplina llamada: Programación Estructurada.
- Empecemos por conocer las reglas para cambiar fórmulas matemáticas a expresiones válidas para la computadora, además de diferenciar constantes e identificadores y tipos de datos simples.
- Tipos de Datos:
    - Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como ‘b’, un valor entero tal como 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable.



- Tipos de Datos Simples:
    - Datos Numéricos: Permiten representar valores escalares de forma numérica, esto incluye a los números enteros y los reales. Este tipo de datos permiten realizar operaciones aritméticas comunes.
    - Datos lógicos: Son aquellos que solo pueden tener dos valores (cierto o falso) ya que representan el resultado de una comparación entre otros datos (numéricos o alfanuméricos).
    - Datos alfanuméricos (string): Es una secuencia de caracteres alfanuméricos que permiten representar valores identificables de forma descriptiva, esto incluye nombres de personas, direcciones, etc. Es posible representar números como alfanuméricos, pero estos pierden su propiedad matemática, es decir no es posible hacer

operaciones con ellos. Este tipo de datos se representan encerrados entre comillas.

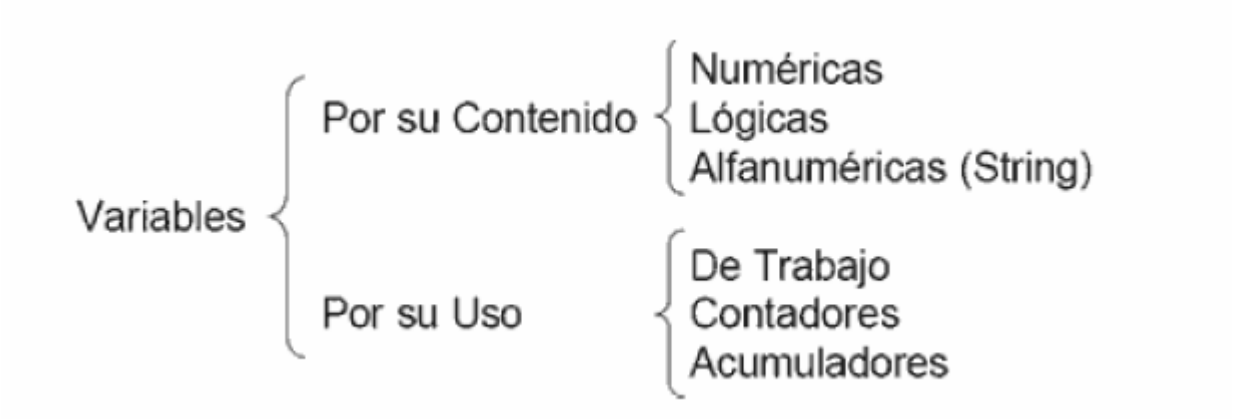
- Identificadores:
  - Los identificadores representan los datos de un programa (constantes, variables, tipos de datos). Un identificador es una secuencia de caracteres que sirve para identificar una posición en la memoria de la computadora, que permite acceder a su contenido.
  - Ejemplo:
    - Nombre
    - Num_hrs
    - Calif2
  - Reglas para formar un identificador
    - Debe comenzar con una letra (A a Z, mayúsculas o minúsculas) y no deben contener espacios en blanco
    - Letras, dígitos y caracteres como la subraya (_) están permitidos después del primer carácter
    - La longitud de identificadores puede ser de varios caracteres. Pero es recomendable una longitud promedio de 8 caracteres
    - El nombre del identificador debe dar una idea del valor que contiene.

1.7 Constantes, variables y expresiones

- **Constantes:** Una constante es un dato numérico o alfanumérico que no cambia durante la ejecución del programa.
  - Ejemplo: pi = 3.1416
- **Variable:** Es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, su contenido puede cambiar durante la ejecución del programa. Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo.

- Ejemplo: area = pi * radio ^ 2. Las variables son: el radio, el área y la constate es pi.

- Clasificación de las variables:



- Por su contenido:
  - Variables Numéricas: Son aquellas en las cuales se almacenan valores numéricos, positivos o negativos, es decir almacenan números del 0 al 9, signos (+ y -) y el punto decimal. Ejemplo: iva = 0.14 pi = 3.1416 costo = 2500
  - Variables Lógicas: Son aquellas que solo pueden tener dos valores (cierto o falso) estos representan el resultado de una comparación entre otros datos.
  - Variables Alfanuméricas: Esta formada por caracteres alfanuméricos (letras, números y caracteres especiales). Ejemplo: letra = ’a’, apellido = ’lopez’, direccion = ’Av. Delicias #190’
- Por su uso:
  - Variables de Trabajo: Variables que reciben el resultado de una operación matemática completa y que se usan normalmente dentro de un programa. Ejemplo: Suma = a + b /c
  - Contadores: Se utilizan para llevar el control del número de ocasiones en que se realiza una operación o se cumple una condición. Con los incrementos generalmente de uno en uno.

- Acumuladores: Forma que toma una variable y que sirve para llevar la suma acumulativa de una serie de valores que se van leyendo o calculando progresivamente.

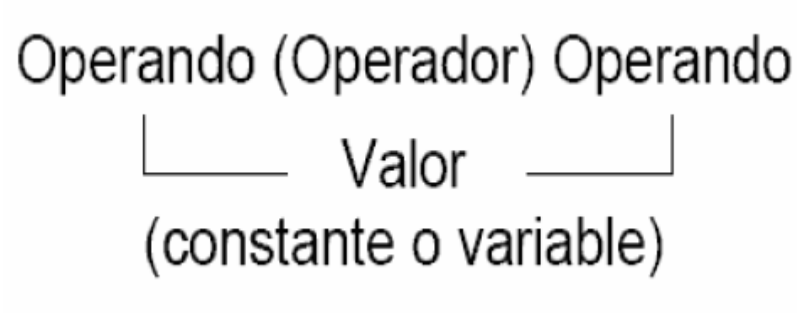
- Expresiones:
  - Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Por ejemplo: a + (b + 3) / c
  - Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.
  - Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican las expresiones en:
    - Aritméticas
    - Relacionales
    - Lógicas

- **Operadores y Operandos**
  - **Operadores:** Son elementos que relacionan de forma diferente, los valores de una o mas variables y/o constantes. Es decir, los operadores nos permiten manipular valores.

    - **Operadores Aritméticos:**
      - Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes).
      - Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales. Si ambos son enteros, el

resultado es entero; si alguno de ellos es real, el resultado es real.

- + Suma
- – Resta
- * Multiplicación
- / División
- mod Modulo (residuo de la división entera)



- Ejemplos:

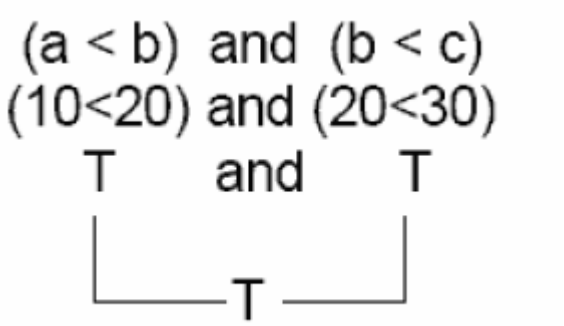
Expresión	Resultado
7 / 2	3.5
12 mod 7	5
4 + 2 * 5	14

- **Prioridad de los Operadores Aritméticos**
  - Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro a fuera, el paréntesis más interno se evalúa primero. Dentro de una misma expresión los operadores se evalúan en el siguiente orden:
    - 1. ^ Exponenciación
    - 2. *, /, mod Multiplicación, división, módulo.
    - 3. +, - Suma y resta.
  - Los operadores en una misma expresión con igual nivel de prioridad se evalúan de izquierda a derecha.

Expresión	Resultado
7 / 2	3.5
12 mod 7	5
4 + 2 * 5	14
4 + 2 * 5	14
23 * 2 / 5	9.2
3 + 5 * (10 - (2 + 4))	23

- **Operadores Relacionales:**
  - Se utilizan para establecer una relación entre dos valores. Luego compara estos valores entre si y esta comparación produce un resultado de certeza o falsedad (verdadero o falso).
  - Los operadores relacionales comparan valores del mismo tipo (numéricos o cadenas). Estos tienen el mismo nivel de prioridad en su evaluación.
  - Los operadores relaciónales tiene menor prioridad que los aritméticos.
  - Tipos de operadores Relacionales
    - > Mayor que
    - < Menor que
    - > = Mayor o igual que
    - < = Menor o igual que
    - < > Diferente
    - = Igual
  - Ejemplos:
    - Si a = 10, b = 20, c = 30
    - a < b < c (10 < 20 < 30)
- **Operadores Lógicos**
  - Estos operadores se utilizan para establecer relaciones entre valores lógicos.

- Estos valores pueden ser resultado de una expresión relacional.
- Tipos de operadores Lógicos
  - And Y
  - Or O
  - Not Negación
- Ejemplo:
  - (Para los siguientes ejemplos T significa verdadero y F falso.)



a + b > c	**F**	a - b < c	**V**
a - b = c	**F**	a * b < > c	**V**

Operador And y Operador Or

Operando1	Operador	Operando2	Resultado
T	AND	T	T
T		F	F
F		T	F
F		F	F

Operando1	Operador	Operando2	Resultado
T	OR	T	T
T		F	T
F		T	T
F		F	F

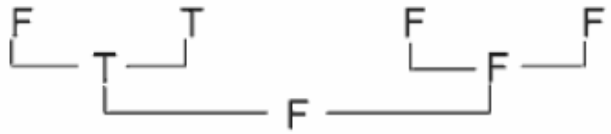
- Prioridad de los Operadores Lógicos:
  - 1. Not
  - 2. And
  - 3. Or

- Prioridad de los Operadores en General
  - 1. ( )

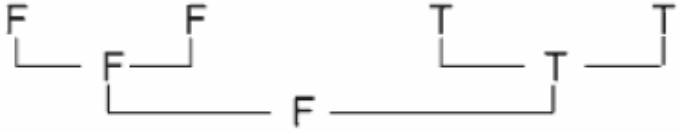
- 2. ^
- 3. *, /, Mod, Not
- 4. +, -, And
- 5. >, <, > =, < =, < >, =, Or

- Ejemplo:
  - Sea: a = 10 b = 12 c = 13 d =10

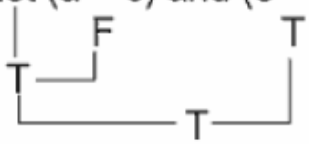
1) ((a > b) or (a < c)) and ((a = c) or (a > = b))



2) ((a > = b) or (a < d)) and ((a > = d) and (c > d))



3) not (a = c) and (c > b)



2 Técnicas de diseño.

2.1 Top Down

- También conocida como de arriba-abajo y consiste en establecer una serie de niveles de mayor a menor complejidad (arriba-abajo) que den solución al problema.
- Consiste en efectuar una relación entre las etapas de la estructuración de forma que una etapa jerárquica y su inmediato inferior se relacionen mediante entradas y salidas de información.
- Este diseño consiste en una serie de descomposiciones sucesivas del problema inicial, que recibe el refinamiento progresivo del repertorio de instrucciones que van a formar parte del programa.
- La utilización de la técnica de diseño Top-Down tiene los siguientes objetivos básicos:
    - Simplificación del problema y de los subprogramas de cada descomposición.
    - Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.
    - El programa final queda estructurado en forma de bloque o módulos lo que hace más sencilla su lectura y mantenimiento.

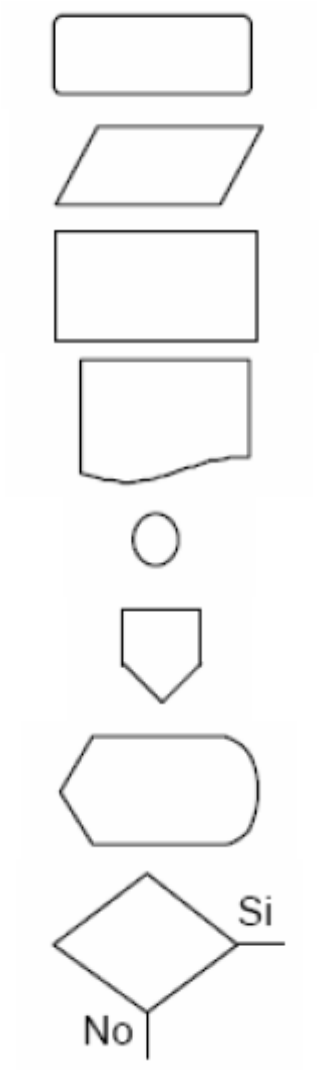
2.2 Bottom Up

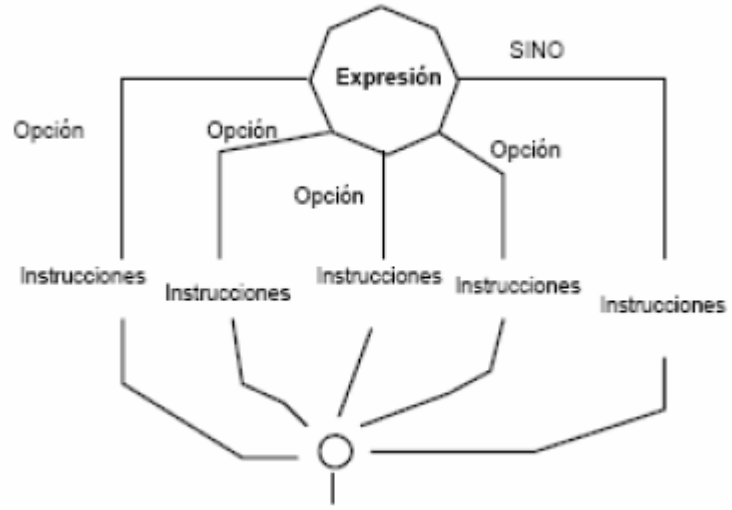
- El diseño ascendente se refiere a la identificación de aquellos procesos que necesitan computarizarse con forme vayan apareciendo, su análisis como sistema y su codificación, o bien, la adquisición de paquetes de software para satisfacer el problema inmediato.
- Cuando la programación se realiza internamente y haciendo un enfoque ascendente, es difícil llegar a integrar los subsistemas al grado tal de que el desempeño global, sea fluido.
- Los problemas de integración entre los subsistemas son sumamente costosos y muchos de ellos no se solucionan hasta que la programación alcanza la fecha límite para la integración total del sistema.

- En esta fecha, ya se cuenta con muy poco tiempo, presupuesto o paciencia de los usuarios, como para corregir aquellas delicadas interfaces, que en un principio, se ignoran.
- Aunque cada subsistema parece ofrecer lo que se requiere, cuando se contempla al sistema como una entidad global, adolece de ciertas limitaciones por haber tomado un enfoque ascendente.
- Uno de ellos es la duplicación de esfuerzos para acceder el software y más aún al introducir los datos.
- Otro es, que se introducen al sistema muchos datos carentes de valor. Un tercero y tal vez el mas serio inconveniente del enfoque ascendente, es que los objetivos globales de la organización no fueron considerados y en consecuencia no se satisfacen.
- La diferencia entre estas dos técnicas de programación se fundamenta en el resultado que presentan frente a un problema dado.
- Imagine una empresa, la cual se compone de varios departamentos (contabilidad, mercadeo, …), en cada uno de ellos se fueron presentando problemas a los cuales se le dieron una solución basados en un enfoque ascendente (Bottom Up): creando programas que satisfacían sólo el problema que se presentaba.
- çCuando la empresa decidió integrar un sistema global para suplir todas las necesidades de todos los departamentos se dio cuenta que cada una de las soluciones presentadas no era compatible la una con la otra, no representaba una globalidad, característica principal de los sistemas.
- Como no hubo un previo análisis, diseño de una solución a nivel global en todos sus departamentos, centralización de información, que son características propias de un diseño Descendente (Top Down) y características fundamentales de los sistemas; la empresa no pudo satisfacer su necesidad a nivel global.
- La creación de algoritmos es basado sobre la técnica descendente, la cual brinda el diseño ideal para la solución de un problema.

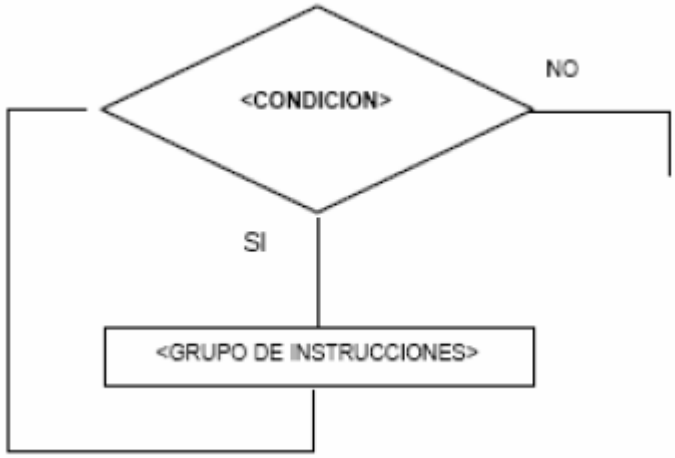
2.3 Técnicas para la formulación de algoritmos. Diagrama de flujo

- Diagrama de Flujo
  - Un diagrama de flujo es la representación gráfica de un algoritmo. También se puede decir que es la representación detallada en forma gráfica de como deben realizarse los pasos en la computadora para producir resultados.
  - Esta representación gráfica se da cuando varios símbolos (que indican diferentes procesos en la computadora), se relacionan entre si mediante líneas que indican el orden en que se deben ejecutar los procesos. Los símbolos utilizados han sido normalizados por el Instituto Norteamericano de Normalización (ANSI):

Símbolo	Descripción
	Indica el inicio y el final de nuestro diagrama de flujo. Indica la entrada y salida de datos. Símbolo de proceso y nos indica la asignación de un valor en la memoria y/o la ejecución de una operación aritmética. Indica la salida de información por impresora. Conector dentro de página. Representa la continuidad del diagrama dentro de la misma página. Conector fuera de página. Representa la continuidad del diagrama en otra página. Indica la salida de información en la pantalla o monitor. Símbolo de decisión. Indica la realización de una comparación de valores.

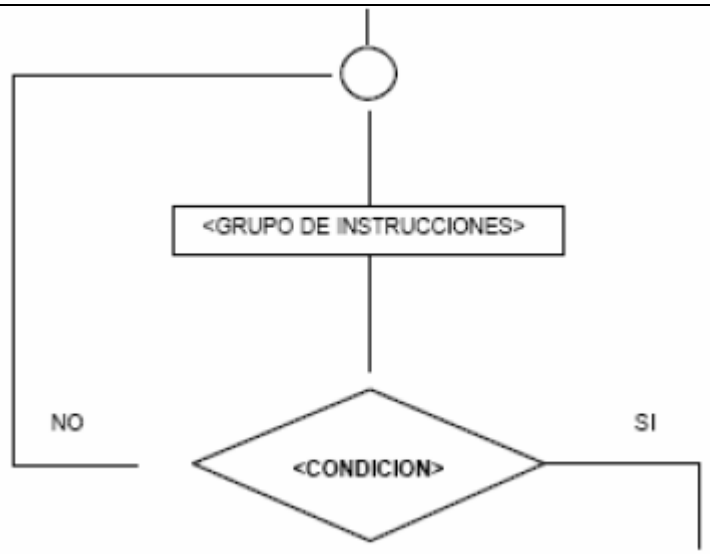


Símbolo de Selección Múltiple. Dada una expresión permite escoger una opción de muchas.



Símbolo del Mientras. Dada una expresión al principio de la iteración esta es evaluada; si la condición es verdadera realizará el ciclo, si es falsa la repetición cesará.

Símbolo del Para. Esta estructura de control repetitiva se usa generalmente cuando se conoce de antemano el número de iteraciones.

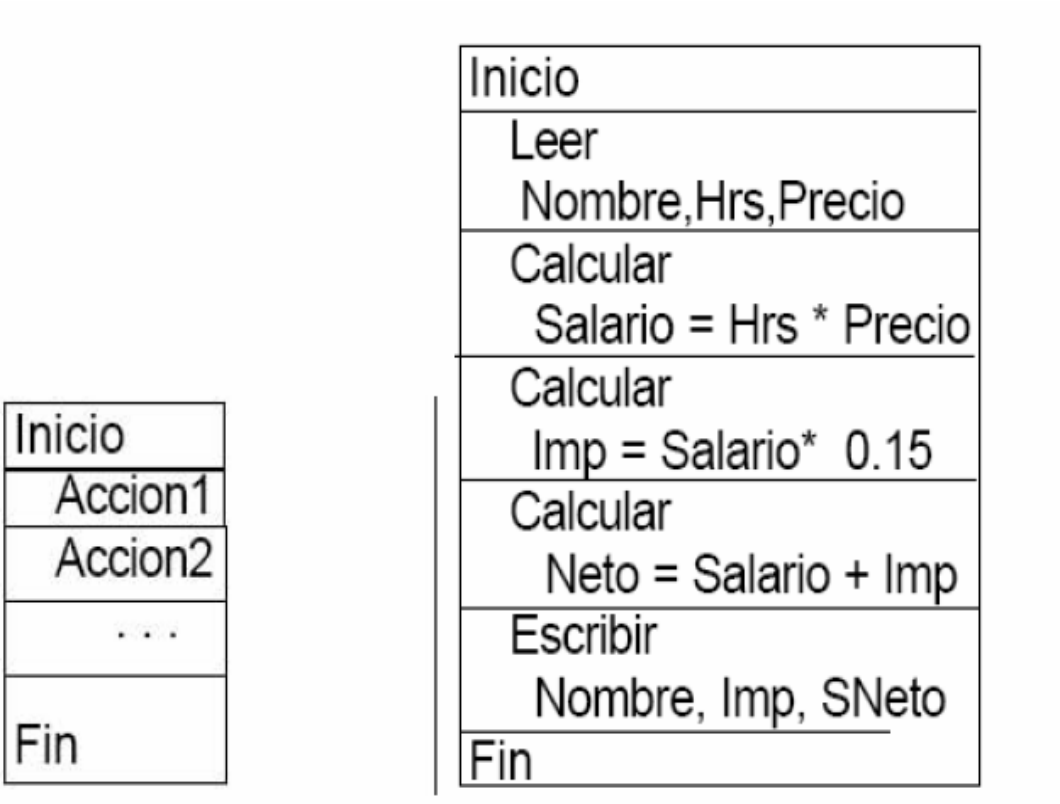
	Símbolo Repita Hasta. Funciona igual que la estructura Mientras, con la diferencia que al menos una vez hará el grupo de instrucciones y luego evaluará una condición. Si la condición evaluada es falsa continua dentro del ciclo y si es verdadera termina la iteración.

- **Recomendaciones para el diseño de Diagramas de Flujo**
  - Se deben usar solamente líneas de flujo horizontales y/o verticales.
  - Se debe evitar el cruce de líneas utilizando los conectores.
  - Se deben usar conectores sólo cuando sea necesario.
  - No deben quedar líneas de flujo sin conectar.
  - Se deben trazar los símbolos de manera que se puedan leer de arriba hacia abajo y de izquierda a derecha.
  - Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.

2.4 Seudo código, diagramas estructurados y estructuras algorítmicas

- **Seudo código**
  - Mezcla de lenguaje de programación y español (o ingles o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa.
  - En esencia, el Seudo código se puede definir como un lenguaje de especificaciones de algoritmos.
  - Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado.
  - El Seudo código utiliza palabras que indican el proceso a realizar.

- **Ventajas de utilizar un Seudo código a un Diagrama de Flujo**
  - Ocupa menos espacio en una hoja de papel
  - Permite representar en forma fácil operaciones repetitivas complejas
  - Es muy fácil pasar de Seudo código a un programa en algún lenguaje de programación.
  - Si se siguen las reglas se puede observar claramente los niveles que tiene cada operación.
- **Diagramas estructurados (Nassi-Schneiderman)**
  - El diagrama estructurado N-S (también conocido como diagrama de Chapin) es como un diagrama de flujo en el que se omiten las flechas de unión y las cajas son contiguas.
  - Las acciones sucesivas se pueden escribir en cajas sucesivas y como en los diagramas de flujo, se pueden escribir diferentes acciones en una caja.
  - Un algoritmo se represente en la siguiente forma:



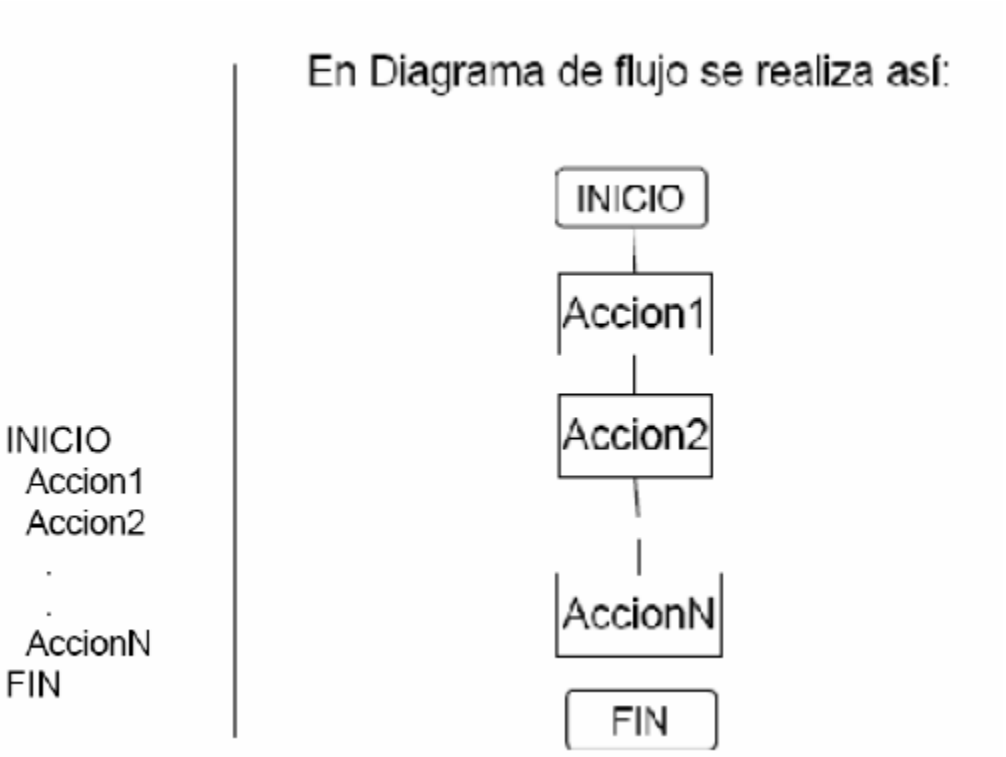
3 Estructuras Básicas

3.1 Estructuras Algorítmicas.

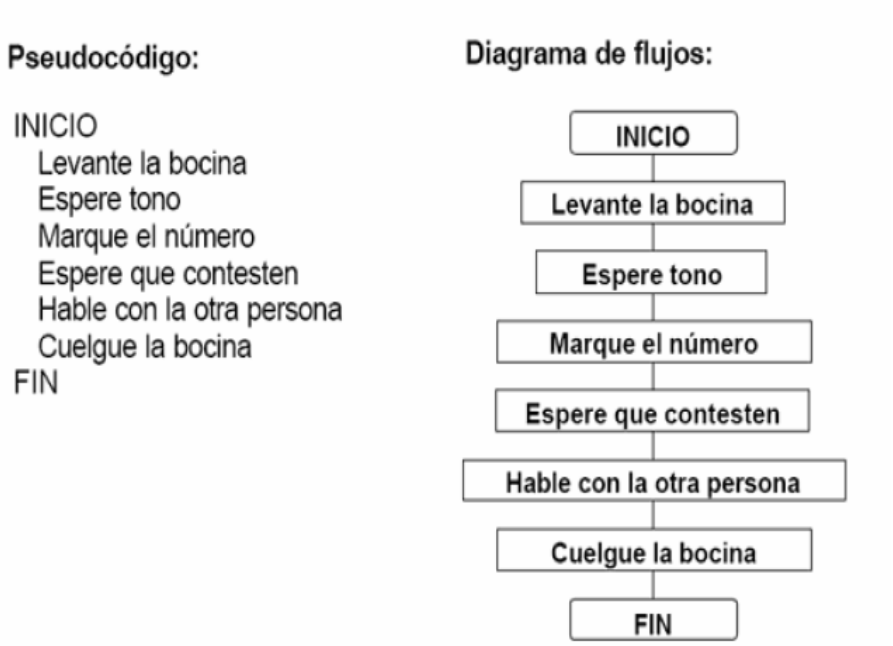
- Las estructuras de operación de programas son un grupo de formas de trabajo, que permiten, mediante la manipulación de variables, realizar ciertos procesos específicos que nos lleven a la solución de problemas.
- Estas estructuras se clasifican de acuerdo con su complejidad en:

3.2 Estructuras secuenciales

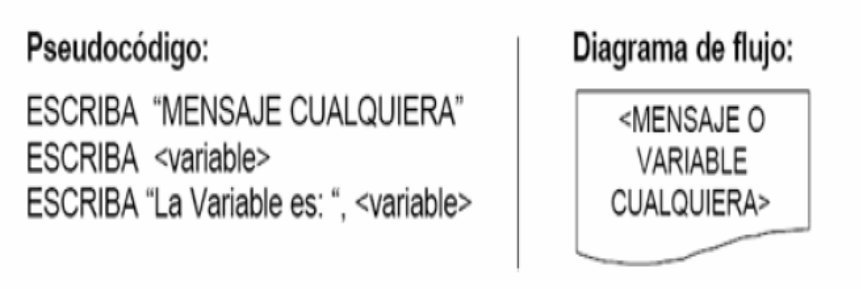
- La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. En Seudo código una Estructura Secuencial se representa de la siguiente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



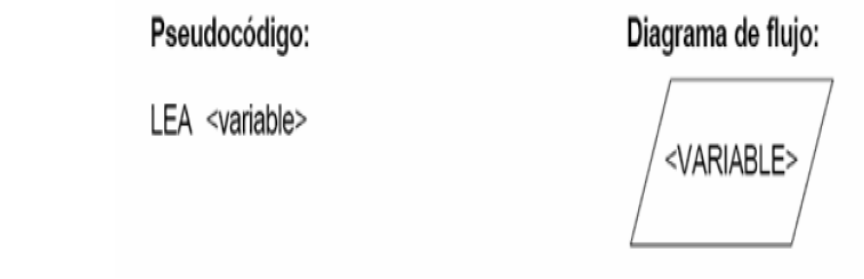
- Observe el siguiente problema de tipo cotidiano y sus respectivos algoritmos representados en Seudo código y en diagramas de flujos:
    - Tengo un teléfono y necesito llamar a alguien pero no sé como hacerlo.



- El anterior ejemplo es un sencillo algoritmo de un problema cotidiano dado como muestra de una estructura secuencial. Ahora veremos los componentes que pertenecen a ella:
  - Asignación
    - La asignación consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor. La asignación se puede clasificar de la siguiente forma:
      - Simples: Consiste en pasar un valor constante a una variable (a ←15)
      - Contador: Consiste en usarla como un verificador del numero de veces que se realiza un proceso (a ← a + 1)
      - Acumulador: Consiste en usarla como un sumador en un proceso (a ← a + b)
      - De trabajo: Donde puede recibir el resultado de una operación matemática que involucre muchas variables (a ← c + b*2/4).
      - En general el formato a utilizar es el siguiente: < Variable > ←<valor o expresión > (< Variable > = <valor o expresión >)
      - El símbolo ← debe leerse “asigne”.
  - Escritura o salida de datos
    - Consiste en enviar por un dispositivo de salida (p.ej. monitor o impresora) un resultado o mensaje. Esta instrucción presenta en pantalla el mensaje escrito entre comillas o el contenido de la variable. Este proceso se representa así como sigue:



- Lectura o entrada de datos
  - La lectura o entrada de datos consiste en recibir desde un dispositivo de entrada (p.ej. el teclado) un valor o dato. Este dato va a ser almacenado en la variable que aparece a continuación de la instrucción. Esta operación se representa así:



- Declaración de variables y constantes
  - La declaración de variables es un proceso que consiste en listar al principio del algoritmo todas las variables que se usarán, además de colocar el nombre de la variable se debe decir qué tipo de variable es.
    - Contador: ENTERO
    - Edad, I: ENTERO
    - Direccion : CADENA_DE_CARACTERES
    - Salario_Basico : REAL
    - Opcion : CARACTER
  - En la anterior declaración de variables Contador, Edad e I son declaradas de tipo entero; Salario_Basico es una variable de tipo real, Opcion

es de tipo carácter y la variable Direccion está declarada como una variable alfanumérica de cadena de caracteres.

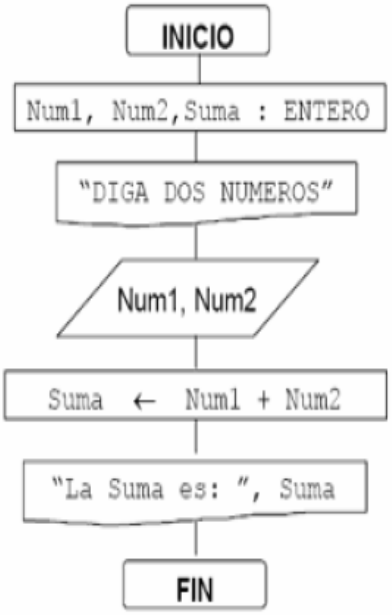
- En el momento de declarar constantes debe indicarse que lo es y colocarse su respectivo valor.
  - CONSTANTE Pi 3.14159
  - CONSTANTE Msg “Presione una tecla y continue”
  - CONSTANTE ALTURA 40
- Cuando se trabaja con algoritmos por lo general no se acostumbra a declarar las variables ni tampoco constantes debido a razones de simplicidad, es decir, no es camisa de fuerza declarar las variables. Sin embargo en esta guía se presenta para todos los algoritmos que se realicen, con esto logramos hacerlos más entendibles y organizados y de paso permite acostumbrarnos a declararlas ya que la mayoría de los lenguajes de programación (entre ellos el Pascal y el C++) requieren que necesariamente se declaren las variables que se van a usar en los programas.

- Veamos algunos ejemplos donde se aplique todo lo que hemos visto hasta el momento sobre algoritmos:
  - Ejemplo 1: Escriba un algoritmo que pregunte por dos números y muestre como resultado la suma de estos. Use Seudo código y diagrama de flujos.

Pseudocódigo:

```
INICIO
   Num1, Num2,Suma  : ENTERO
   ESCRIBA  "Diga dos números: "
   LEA Num1, Num2
   Suma ← Num1 + Num2
   ESCRIBA "La Suma es:", Suma
FIN
```

Diagrama de flujo:

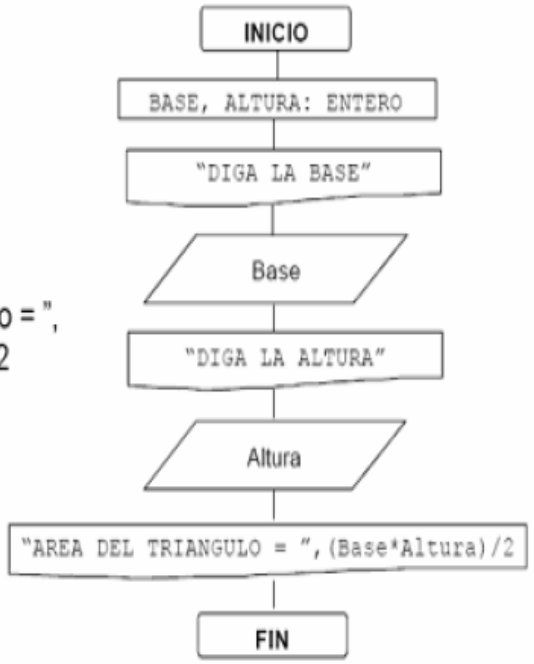


- Ejemplo 2: Escriba un algoritmo que permita conocer el área de un triángulo a partir de la base y la altura. Exprese el algoritmo usando Seudo código y diagrama de flujos.

Pseudocódigo:

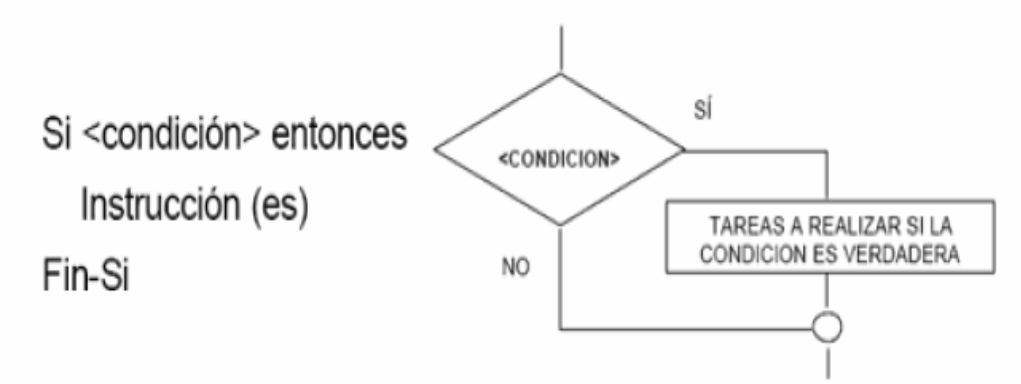
```
INICIO
    Base, Altura: ENTERO
    ESCRIBA "Diga la Base: "
    LEA Base
    ESCRIBA "Diga la Altura"
    LEA Altura
    ESCRIBA "Area del Triangulo = ",
            (BASE*ALTURA)/2
FIN
```

Diagrama de flujo:

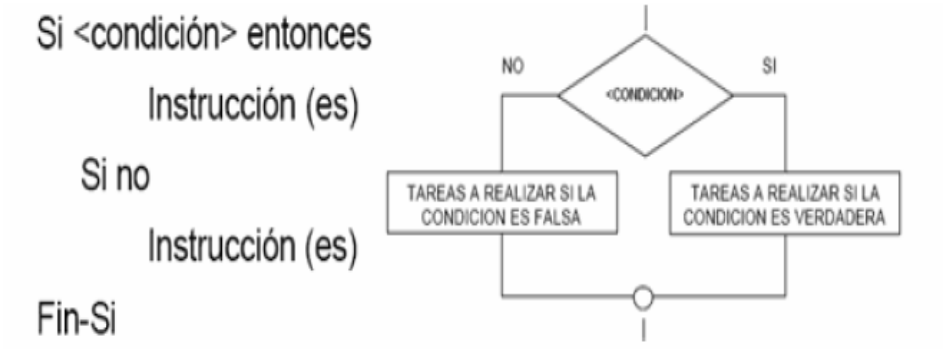


3.3 Estructuras Condicionales

- Las estructuras condicionales comparan una variable contra otro(s)valor (es), para que en base al resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite. Existen tres tipos básicos, las simples, las dobles y las múltiples.
- Tipos de Estructuras Condicionales:
  - Simples: Las estructuras condicionales simples se les conoce como “Tomas de decisión”. Estas tomas de decisión tienen la siguiente forma:



  - Dobles: Las estructuras condicionales dobles permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición. Se representa de la siguiente forma:



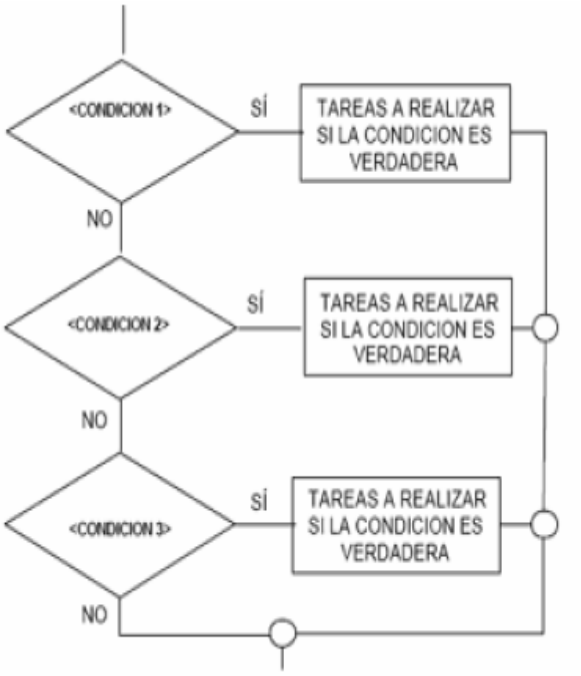
    - Donde:
      - Si: Indica el comando de comparación

- Condición: Indica la condición a evaluar
- Entonces: Precede a las acciones a realizar cuando se cumple la condición
- Instrucción(es): Son las acciones a realizar cuando se cumple o no la condición
- si no: Precede a las acciones a realizar cuando no se cumple la condición
- (Dependiendo de si la comparación es cierta o falsa, se pueden realizar una o más acciones.)

o Múltiples (Si anidado):

- Las estructuras de comparación múltiples, son tomas de decisión especializadas que permiten comparar una variable contra distintos posibles resultados, ejecutando para cada caso una serie de instrucciones específicas. La forma común es la siguiente:

```
Si <condición> entonces
    Instrucción(es)
  Si no
    Si <condición> entonces
        Instrucción(es)
      Si no
          .
          .   Varias condiciones
          .
```

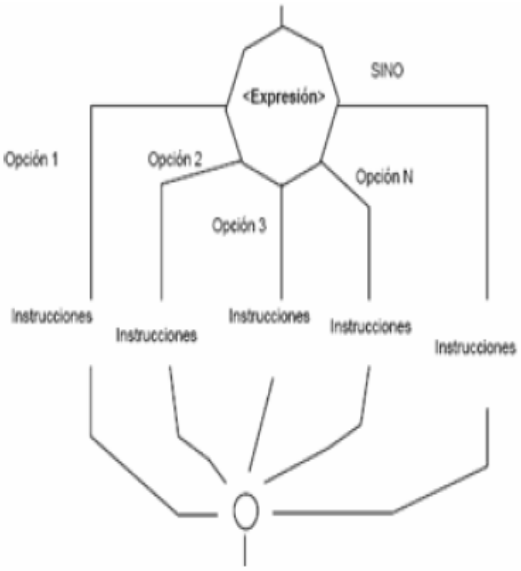


o Múltiples (En caso de):

- Las estructuras de comparación múltiples, es una toma de decisión especializada que permiten evaluar una variable

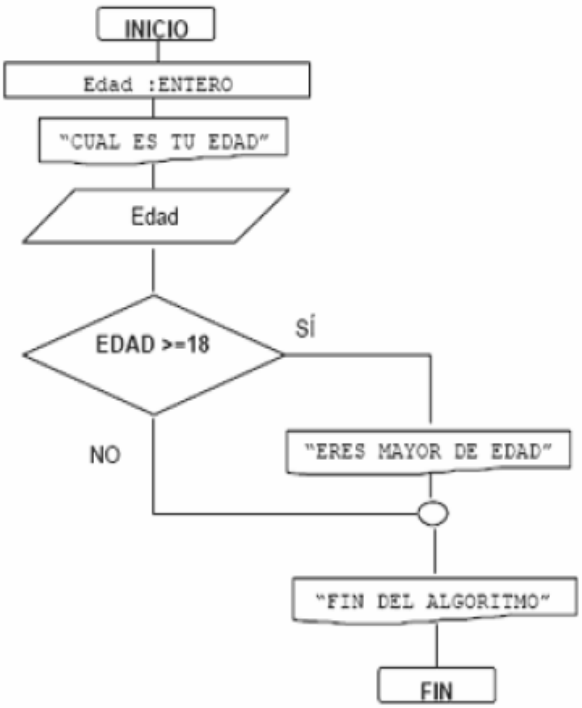
con distintos posibles resultados, ejecutando para cada caso una serie de instrucciones especificas. La forma es la siguiente:

```
En-caso-de <expresión> haga
  Caso <opción 1>:
        <instrucciones>
  caso <opción 2>:
        <instrucciones>
  caso <opción 3>:
        <instrucciones>
...
  caso <opción N>:
       <instrucciones>
SINO <instrucciones a realizar si no
        se ha cumplido Ninguna de
        las condiciones anteriores>
Fin-Caso
```



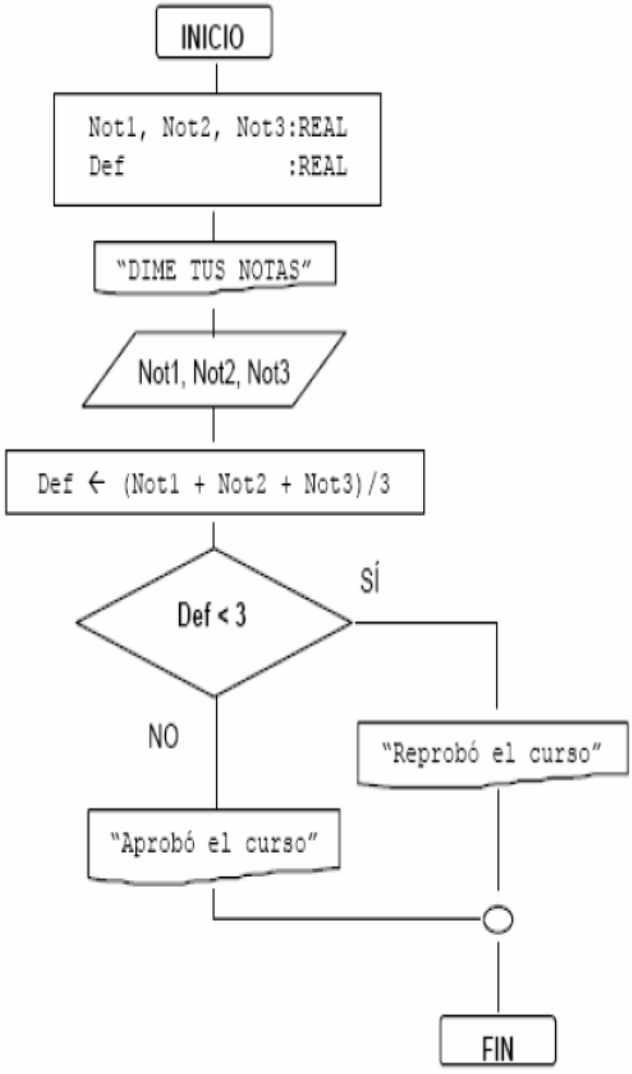
- Veamos algunos ejemplos donde se aplique todo lo anterior:
  - Realizar un algoritmo en donde se pide la edad del usuario; si es mayor de edad debe aparecer un mensaje indicándolo. Expresarlo en Seudo código y Diagrama de flujos.

```
INICIO
   Edad: Entero
   ESCRIBA "Cual es tu edad?"
   LEA Edad
   Si Edad >=18 entonces
      Escriba "Eres mayor de edad"
   Fin-Si
   ESCRIBA "Fin del algoritmo"
FIN
```



- Se pide leer tres notas del alumno, calcular su definitiva en un rango de 0-5 y enviar un mensaje donde diga si el alumno aprobó o reprobó el curso. Exprese el algoritmo usando Seudo código y diagrama de flujos.

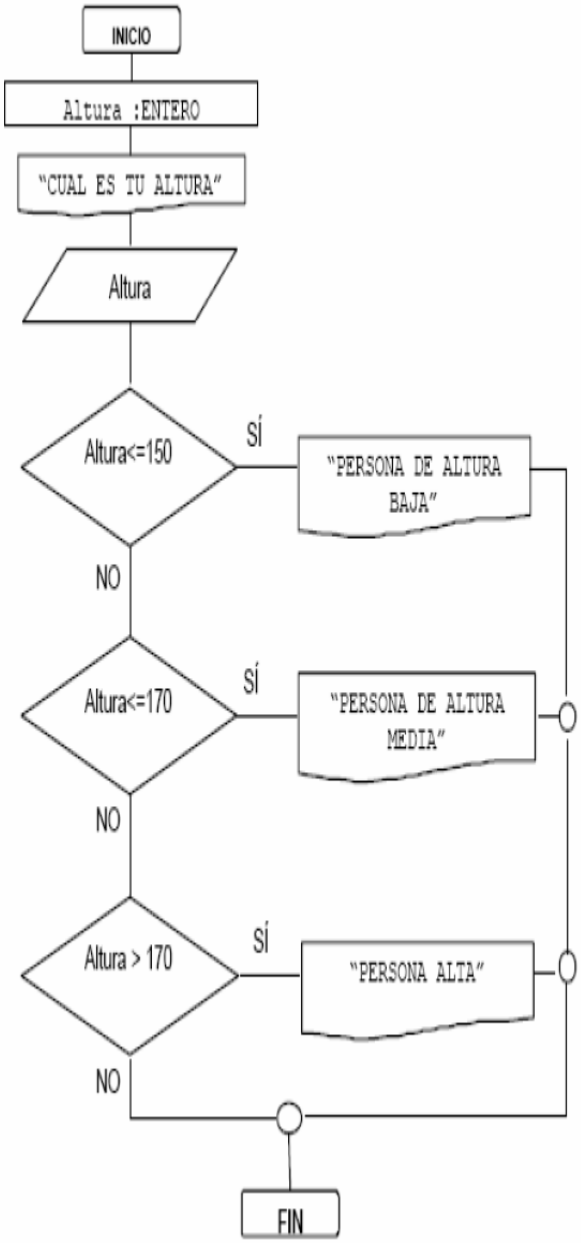
```
INICIO
        Not1, Not2, Not 3 :REAL
        Def: REAL

LEA Not1, Not2, Not3
Def = (Not1 + Not2 + Not3) /3
Si Def < 3 entonces
        Escriba “Reprobó el curso”
   Sino
        Escriba “Aprobó el curso”
Fin-Si
FIN
```



- Se desea escribir un algoritmo que pida la altura de una persona, si la altura es menor o igual a 150 cm envíe el mensaje: “Persona de altura baja”; si la altura está entre 151 y 170 escriba el mensaje: “Persona de altura media” y si la altura es mayor al 171

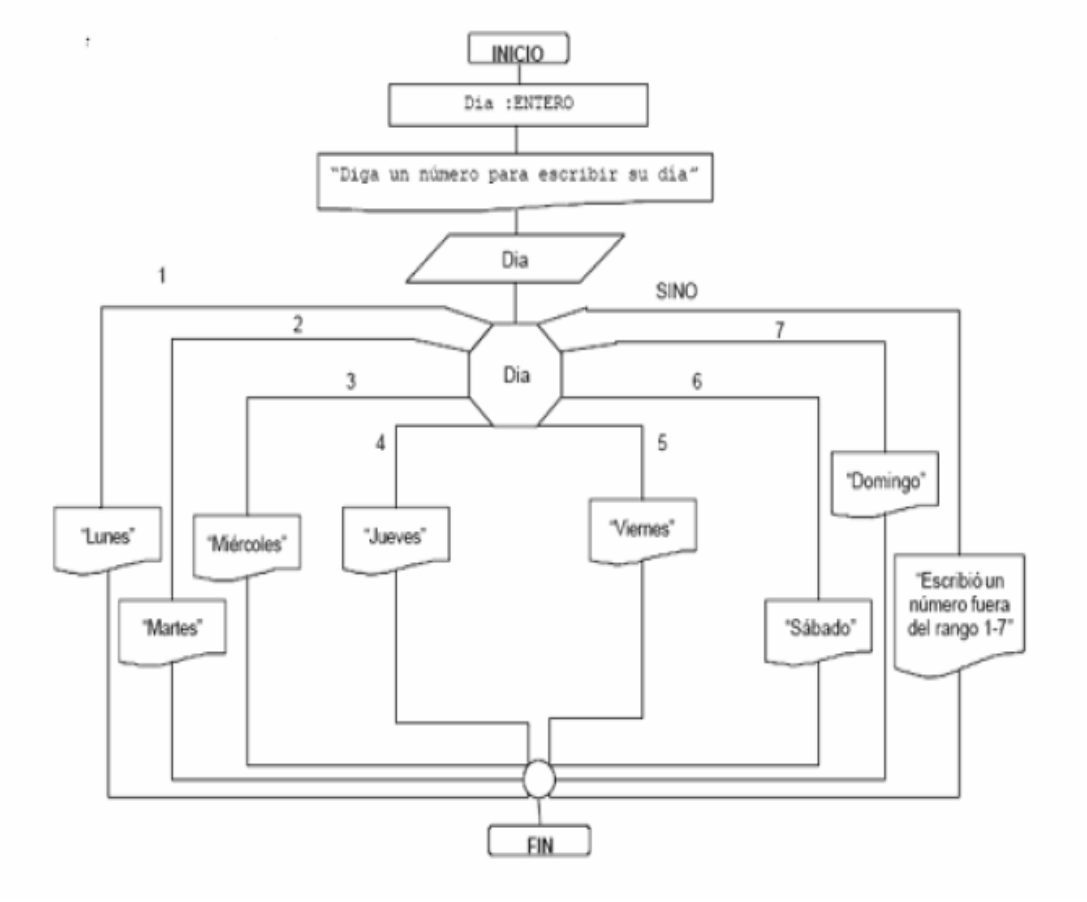
escriba el mensaje: “Persona alta”. Exprese el algoritmo usando Seudo código y diagrama de flujos.

```
INICIO
Altura: ENTERO
ESCRIBA “Cuál es tu altura?”
LEA Altura
Si Altura <=150 entonces
   ESCRIBA “persona de altura baja”
Sino
  Si Altura <=170 entonces
     ESCRIBA “persona de altura media”
  Sino
     Si Altura>170 ENTONCES
        ESCRIBA “persona alta”
     Fin-Si
  Fin-Si
Fin-Si
FIN
```



- Dado un número entre 1 y 7 escriba su correspondiente día de la semana así: 1- Lunes 2- Martes 3- Miércoles 4- Jueves 5- Viernes 6- Sábado 7- Domingo. Exprese el algoritmo usando seudocódigo y diagrama de flujos.

INICIO
Dia: ENTERO
ESCRIBA “Diga un número para escribir su día”
LEA Dia
En-caso-de Dia haga
    Caso 1: ESCRIBA “Lunes”
    Caso 2: ESCRIBA “Martes”
    Caso 3: ESCRIBA “Miércoles”
    Caso 4: ESCRIBA “Jueves”
    Caso 5: ESCRIBA “Viernes”
    Caso 6: ESCRIBA “Sábado”
    Caso 7: ESCRIBA “Domingo”
    SINO: ESCRIBA “Escribió un numero fuera del rango 1-7”
Fin-Caso
FIN



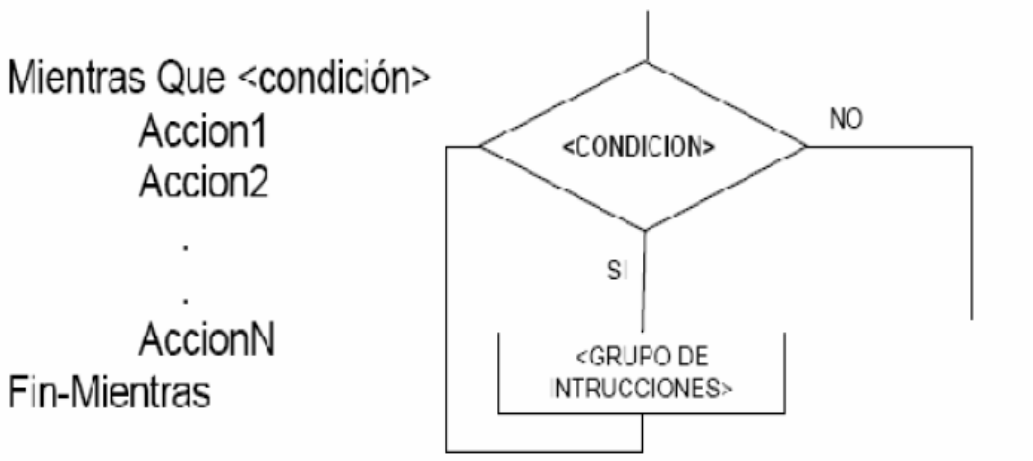
3.4 Estructuras Cíclicas

- Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad específica de veces. Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable

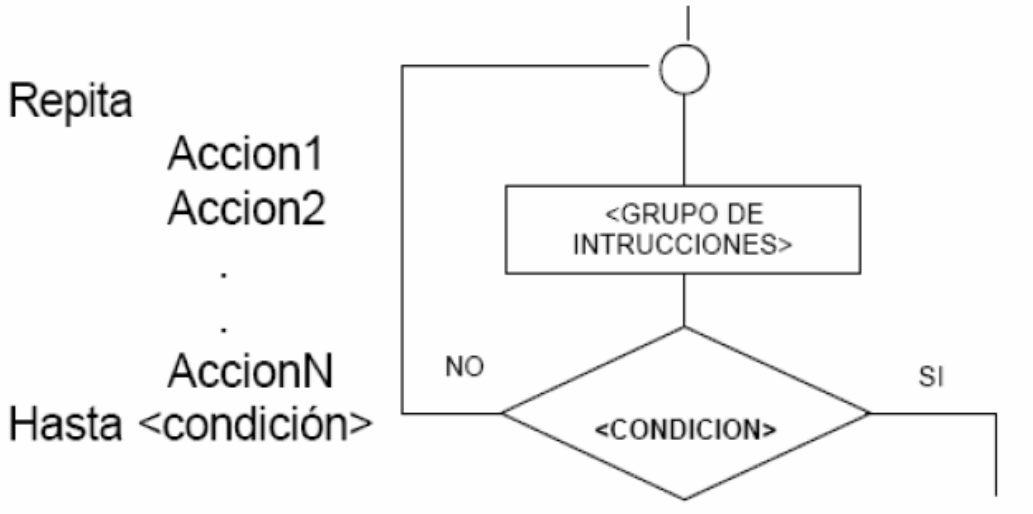
(estar en función de algún dato dentro del programa). Los ciclos se clasifican en:

- Ciclos con un Numero Determinado de Iteraciones
  - (Para): Son aquellos en que el número de iteraciones se conoce antes de ejecutarse el ciclo. La forma de esta estructura es la siguiente:

  - Dado un valor inicial exp1 asignado a la variable esta se irá aumentando o disminuyendo de acuerdo a la exp3 hasta llegar a la exp2; si se omite el paso, significa que la variable aumentará de uno en uno.
- Ciclos con un Número Indeterminado de Iteraciones
  - Son aquellos en que el número de iteraciones no se conoce con exactitud, ya que esta dado en función de un dato dentro del programa.
    - Mientras Que: Esta es una estructura que repetirá un proceso durante “N” veces, donde “N” puede ser fijo o variable. Para esto, la instrucción se vale de una condición que es la que debe cumplirse para que se siga ejecutando. Cuando la condición ya no se cumple, entonces ya no se ejecuta el proceso. La forma de esta estructura es la siguiente:



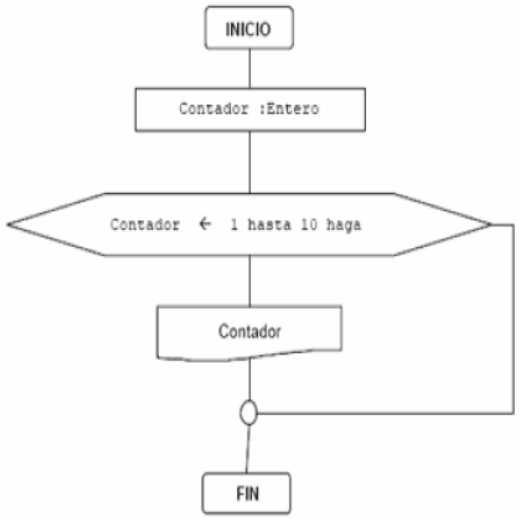
- Repita-Hasta: Esta es una estructura similar en algunas características, a la anterior. Repite un proceso una cantidad de veces, pero a diferencia del Mientras Que, el Repita- Hasta lo hace hasta que la condición se cumple y no mientras, como en el Mientras Que. Por otra parte, esta estructura permite realizar el proceso cuando menos una vez, ya que la condición se evalúa al final del proceso, mientras que en el Mientras Que puede ser que nunca llegue a entrar si la condición no se cumple desde un principio. La forma de esta estructura es la siguiente:



o Ejemplos:

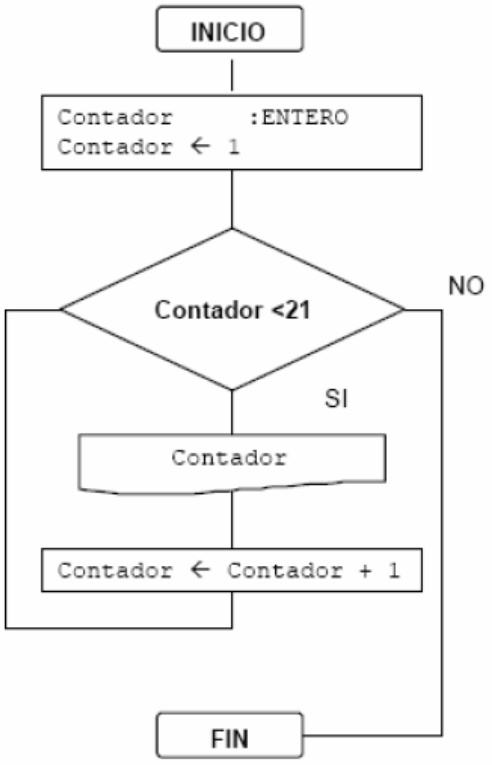
Ejemplo 1:
Realizar un algoritmo que muestre los números de uno en uno hasta diez usando una estructura Para. Exprese el algoritmo usando Seudo código y diagrama de flujos.

```
INICIO
   Contador: Entera
   Para Contador ← 1 hasta 10 haga
        ESCRIBA Contador
   Fin del para
FIN
```



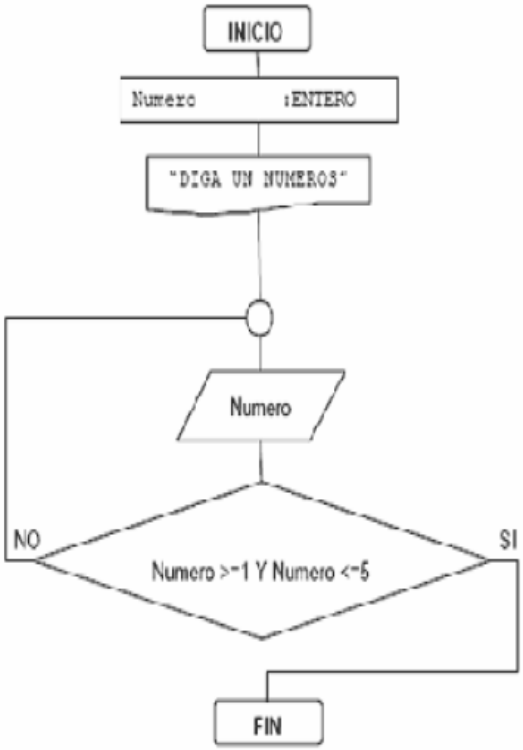
Ejemplo 2:
Usando una estructura Mientras, realizar un algoritmo que escriba los números de uno en uno hasta 20.

```
INICIO
   Contador: ENTERO
   Contador ← 1
   Mientras Que Contador < 27 haga
        ESCRIBA Contador
        Contador ← Contador + 1
   Fin-Mientras
FIN
```



Ejemplo 3:
Realizar un algoritmo que pregunte al usuario un número comprendido en el rango de 1 a 5. El algoritmo deberá validar el número, de manera que no continúe la ejecución del programa mientras no se escriba un número correcto.

```
INICIO
   Numero  : ENTERO
   Escriba "Diga un numero de 1 a 5"
   Repita
        Lea Numero
   Hasta que (Numero > = 1) Y (Numero < 5)
FIN
```



4 Bibliografía

- http://www.desarrolloweb.com/manuales/67

UNEFA

“Excelencia Educativa”