Diseño a alto nivel

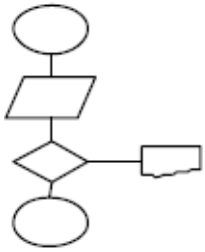
Algoritmos

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

Un Lenguaje algorítmico es una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso.

Tipos de Lenguajes Algorítmicos

- Gráficos: Es la representación gráfica de las operaciones que realiza un algoritmo (diagrama de flujo).



- No Gráficos: Representa en forma descriptiva las operaciones que debe realizar un algoritmo (pseudocódigo).

```
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
```

Metodología de la programación

La primera fase en la resolución de un problema por computadora es la definición o análisis del problema. En donde lo más importante es que conozcamos exactamente lo que debe hacer el programa y "que se desea obtener al final del proceso”

Para poder definir con precisión el problema se requiere que las especificaciones de entrada y salida sean descritas con detalle ya que esto es un requisito para lograr una solución eficaz; por lo que es conveniente hacerse las siguientes preguntas:

1.- ¿Qué entradas se requieren? (Tipo Y Cantidad)

2.- ¿Cuál es la salida deseada? (Tipo Y Cantidad)

3.- ¿Qué método produce la salida deseada?}

Análisis De Los Datos

Una vez que el problema ha sido definido y comprendido, deben analizarse los siguientes aspectos:

1.- Los resultados esperados.

2.- Los datos disponibles.

3.- Herramientas a nuestro alcance para manipular los datos y alcanzar un resultado.

Esta sería un diagrama de la resolución de un problema en su más mínima expresión.

Y mientras esto no se comprenda no puede pasarse a la siguiente etapa.

Diseño De La Solución

Para realizar el diseño de la solución:

Como todos sabemos, una computadora no tiene capacidad para solucionar problemas más que cuando se le proporcionan los sucesivos pasos a realizar, esto se refiere a la obtención de un algoritmo que resuelva adecuadamente el problema.

En caso de que obtengamos varios algoritmos, seleccionaremos uno de ellos utilizando criterios ya conocidos.

Esta etapa incluye:

La descripción del algoritmo resultante en un lenguaje natural, en un diagrama de flujo o natural de programación.

De esta manera, solo se establece la metodología para alcanzar la solución en forma conceptual, es decir; sin alcanzar la implementación en el sistema de cómputo.

Así tenemos que la información proporcionada constituye su entrada y la información producida por el algoritmo constituye su salida.

Los problemas complejos se pueden resolver más eficazmente por la computadora cuando se dividen en subproblemas que sean más fáciles de solucionar.

|

Codificación

Codificación es la escritura en un lenguaje de programación de la representación del algoritmo ***desarrollada en etapas precedentes*** (es decir, primero hay que hacer el diseño, no se puede empezar a escribir código saltándose los pasos previos). Esto se refiere a la obtención de un

programa definitivo que pueda ser comprensible para la máquina.

Y posteriormente, una vez que el algoritmo se ha convertido en un programa fuente. Este programa fuente debe ser traducido a lenguaje máquina, este proceso se realiza con el compilador, y se obtiene el programa objeto, (siempre y cuando el programa fuente sea correcto) que posteriormente se vuelve un programa ejecutable.

Prueba y Depuración

La Prueba se realiza tras la compilación.

Si tras la compilación se presentan errores (errores de compilación) en el programa fuente, es preciso volver a editar el programa, corregir los errores y compilar de nuevo, este proceso se repite hasta que no se producen errores.

De esta manera se obtiene el programa objeto, que todavía no es ejecutable directamente, pero si no contiene errores se debe instruir al sistema para que realice la fase de montaje o enlace del programa objeto con las librerías del programa del compilador; este proceso de montaje produce un programa ejecutable.

La Depuración es el proceso de encontrar los errores del programa y corregir o eliminar dichos errores.

Cuando se ejecuta un programa, se pueden producir tres tipos de errores:

1.- Errores de compilación. Se producen normalmente por un uso incorrecto de las reglas del lenguaje de programación y suelen ser errores de sintaxis, por lo tanto la computadora no puede comprender la instrucción, y obviamente no se obtendrá el programa objeto, y el compilador imprimirá una lista de todos los errores encontrados durante la compilación.

2.- Errores de ejecución. Estos errores se producen por instrucciones que las computadoras pueden comprender, pero no ejecutar. Ejemplos de éstos son: una división por cero, y raíces cuadradas de números negativos; por lo que en este caso se detiene la ejecución del programa y se imprime un mensaje de error.

3.- Errores lógicos. Se producen en la lógica del programa y la fuente del error suele ser el diseño del algoritmo. Estos errores son los más difíciles de detectar, ya que el programa puede funcionar y no producir errores de compilación ni ejecución, y solo puede detectarse cuando se advierte un error por la obtención de resultados incorrectos.

En este caso se debe volver a la fase del diseño del algoritmo, modificarlo, cambiar el programa fuente, compilar y ejecutar una vez más.

Documentación

La documentación de un problema consta de las descripciones de los pasos a dar en el proceso de resolución de un problema. La importancia de la documentación es por su decisiva influencia en el producto final.

Programas pobremente documentados son difíciles de leer, más difíciles de depurar y casi imposibles de mantener y modificar. Por ello la importancia de la documentación, sin la documentación es imposible corregir errores futuros o bien cambiar el programa

Mantenimiento

El mantenimiento se define como la modificación del programa por medio de actualizaciones, que mejoran al programa, corrigiendo errores o bien actualizándolos para un mejor funcionamiento.

Por ello la documentación es sin duda muy importante para poder llevar a cabo el mantenimiento.

Programación estructurada

La programación estructurada (en adelante simplemente PE), es un estilo de programación con el cual el programador elabora programas, cuya estructura es la más clara posible, mediante el uso de tres estructuras básicas de control lógico, a saber :

- SECUENCIA.
- SELECCIÓN.
- ITERACIÓN.

Un programa estructurado se compone de funciones,segmentos, módulos y/o subrutinas, cada una con una sola entrada y una sola salida. Cada uno de estos módulos (aún en el mismo programa completo), se denomina programa apropiado cuando, además de estar compuesto solamente por las tres estructuras básicas, tiene sólo una entrada y una salida y en ejecución no tiene partes por las cuales nunca pasa ni tiene ciclos infinitos.

La PE tiene un teorema estructural o teorema fundamental, el cual afirma que cualquier programa, no importa el tipo de trabajo que ejecute, puede ser elaborado ***utilizando únicamente las tres estructuras básicas (secuencia, selección, iteración)***.

Ventajas de la programación estructurada

Con la PE, elaborar programas de computador sigue siendo una labor que demanda esfuerzo, creatividad, habilidad y cuidado. Sin embargo, con este nuevo estilo podemos obtener las siguientes ventajas :

1. Los programas son más fáciles de entender. Un programa estructurado puede ser leído en secuencia, de arriba hacia abajo, sin necesidad de estar saltando de un sitio a otro en la lógica, lo cual es típico de otros estilos de programación. La estructura del programa es más clara puesto que las instrucciones están más ligadas o relacionadas entre si, por lo que es más fácil comprender lo que hace cada función.
2. Reducción del esfuerzo en las pruebas. El programa se puede tener listo para producción normal en un tiempo menor del tradicional; por otro lado, el seguimiento de las fallas o depuración (debugging) se facilita debido a la lógica más visible, de tal forma que los errores se pueden detectar y corregir más fácilmente.
3. Reducción de los costos de mantenimiento.
4. Programas más sencillos y más rápidos.
5. Aumento en la productividad del programador.
6. Se facilita la utilización de las otras técnicas para el mejoramiento de la productividad en programación.
7. Los programas quedan mejor documentados internamente.

Diagramas 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.

La simbología utilizada para la elaboración de diagramas de flujo es variable y debe ajustarse a un patrón definido previamente.

Recomendaciones de 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.

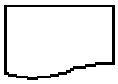
- Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.

Simbología usada en los diagramas de flujo

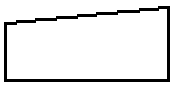
Conector (Conexión entre dos puntos del organigrama situado en páginas diferentes).

Llamada a subrutina o a un proceso predeterminado (una subrutina es un módulo independiente del programa)

Pantalla (se utiliza en ocasiones en lugar del símbolo de E/S).

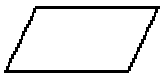


Impresora (se utiliza en ocasiones en lugar del símbolo de E/S).



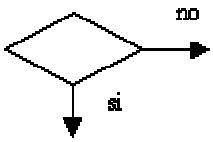
Teclado (Se utiliza en ocasiones en lugar del símbolo de E/S).

Terminal (representa el Inicio y el Final, de un programa, puede representar también una parada o interrupción programada que sea necesario realizar en un programa.

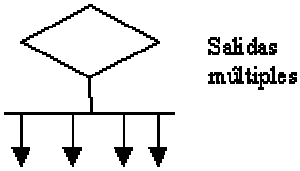


Entrada/Salida (cualquier tipo de introducción de datos)

Proceso (cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas).

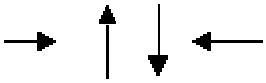


Desición (indica operaciones lógicas o de comparación entre datos -normalmante dos- y en función del resultado de la misma determina cual de los distintos caminos alternativos del programa se debe seguir).



Desición múltiple (en función del resultado de la comparación se seguirá uno de los diferentes caminos de acuerdo con dicho resultado).

Conector (Sirve para enlazae dos partes cualesquiera de un organigrama a través de un conector en la salida y otro conector en la salida)



Indicador de dirección o línea de flujo (indica el sentido de ejecución de las operaciones)

Línea conectora (sirve de unión entre dos símbolos).

Pseudocódigo

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 esencial, el Pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos. Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El Pseudocódigo utiliza palabras que indican el proceso a realizar.

Ventajas de utilizar un Pseudocódigo 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 Pseudocódigo a un programa en algún lenguaje de programación.
- Si se siguen las reglas se puede observar claramente los niveles que tiene cada operación.

Estructuras básicas

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.

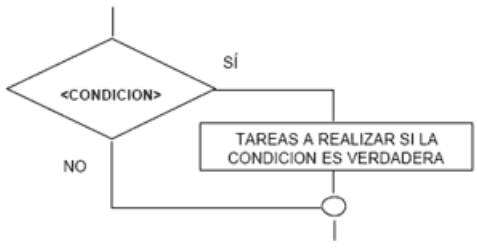
Simples:

Las estructuras condicionales simples se les conoce como "Tomas de decisión". Estas tomas de decisión tienen la siguiente forma:

Pseudocódigo:

```
Si <condición> entonces
   Instrucción (es)
Fin-Si
```

Diagrama de flujo:



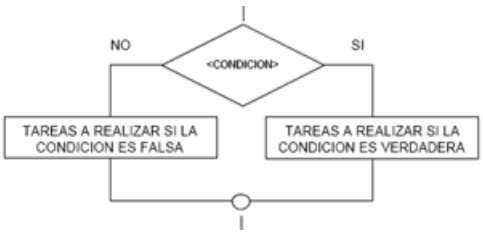
Dobles:

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:

Pseudocódigo:

```
Si <condición> entonces
        Instrucción (es)
   Si no
        Instrucción (es)
Fin-Si
```

Diagrama de flujo:



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

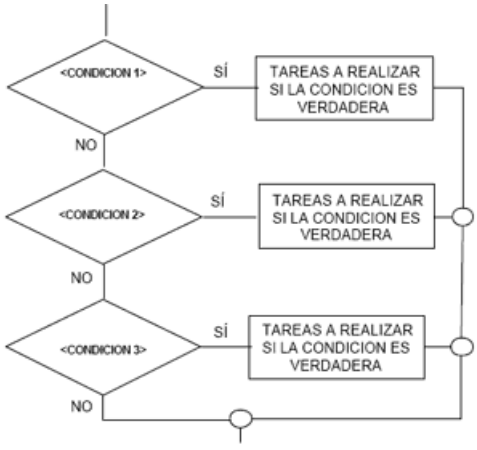
Instrucción(es):Son las acciones a realizar cuando se cumple o no la condición

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.

Múltiples:

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 especificas. La forma común es la siguiente:

Pseudocódigo:	Diagrama de flujo:
Si <condición> entonces Instrucción(es) Si no Si <condición> entonces Instrucción(es) Si no . . Varias condiciones .	

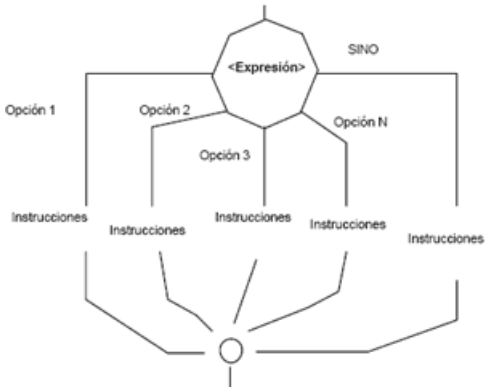
Múltiples (En caso de):

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:

Pseudocódigo:

```
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
```

Diagrama de flujo:



Detalle sobre la especificación de algoritmos.

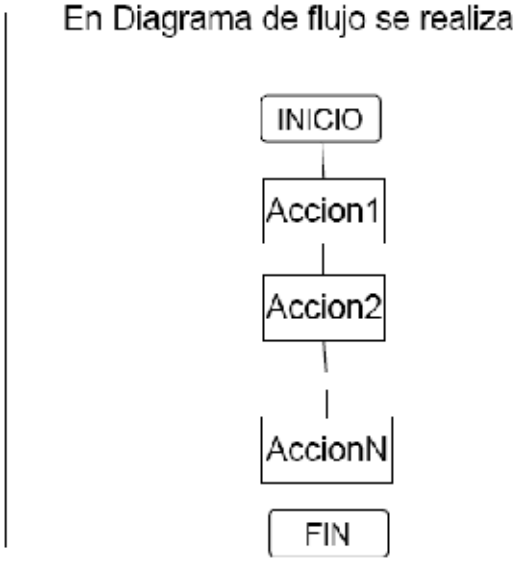
En las siguientes paginas se describen de forma más detallada tanto las estructuras más usadas en programación, como la utilización de entradas y salidas, uso de variables. Se compara la forma de hacerlo con pseudocódigo y diagramas de flujo, y se ilustra con varios ejemplos.

Es importante incidir en que cuando se programa es necesario empezar especificando el algoritmo (con pseudocódigo o diagramas de flujo) antes de empezar a escribir una sola línea de código.

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 Pseudocódigo una Estructura Secuencial se representa de la siguiente forma:

Pseudocódigo	***Diagrama de flujo***
INICIO Accion1 Accion2 . . AccionN FIN	

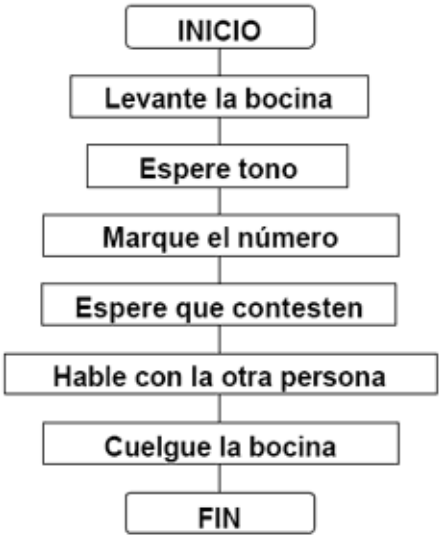
Observe el siguiente problema de tipo cotidiano y sus respectivos algoritmos representados en Pseudocódigo y en diagramas de flujo:

• Tengo un teléfono y necesito llamar a alguien pero no sé como hacerlo.

Pseudocódigo:

```
INICIO
   Levante la bocina
   Espere tono
   Marque el número
   Espere que contesten
   Hable con la otra persona
   Cuelgue la bocina
FIN
```

Diagrama de flujos:



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 >

El símbolo ← debe leerse “asigne”.

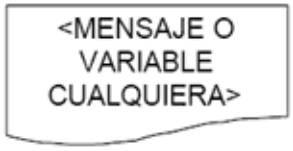
Escritura o salida de datos

Consiste en mandar 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:

Pseudocódigo:

```
ESCRIBA  "MENSAJE CUALQUIERA"
ESCRIBA  <variable>
ESCRIBA "La Variable es: ", <variable>
```

Diagrama de flujo:



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:

Pseudocódigo:

```
LEA  <variable>
```

Diagrama de flujo:

DECLARACION 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

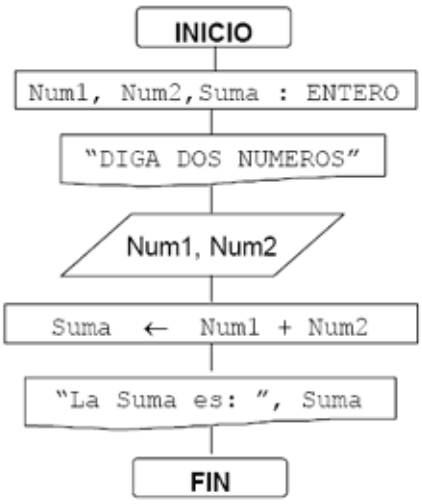
Cuando se trabaja con algoritmos por lo general no se acostumbra a declarar las variables ni tampoco constantes debido a razones de simplicidad, es decir, no es camisa de fuerza declarar las variables. Sin embargo en este curso lo haremos para todos los algoritmos que realicemos, con esto logramos hacerlos más entendibles y organizados y de paso permite acostumbrarnos a declararlas ya que la mayoría de los lenguajes de programación (entre ellos el 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 Pseudocódigo y Diagrama de flujo.

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:

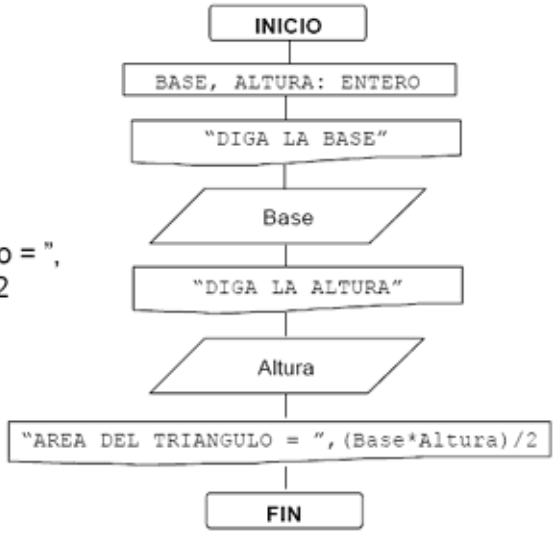


Ejemplo 2: Escriba un algoritmo que permita conocer el área de un triángulo a partir de la base y la altura. Exprese el algoritmo usando Pseudocódigo y Diagrama de flujo.

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:



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.

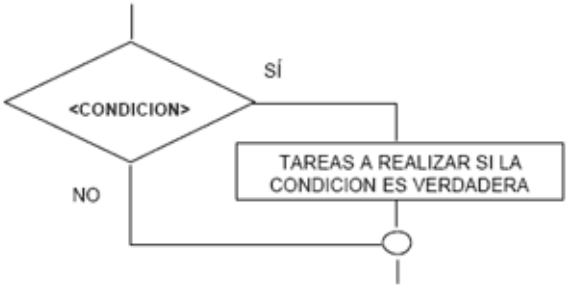
Simples:

Las estructuras condicionales simples se les conoce como "Tomas de decisión". Estas tomas de decisión tienen la siguiente forma:

Pseudocódigo:

```
Si <condición> entonces
   Instrucción (es)
Fin-Si
```

Diagrama de flujo:



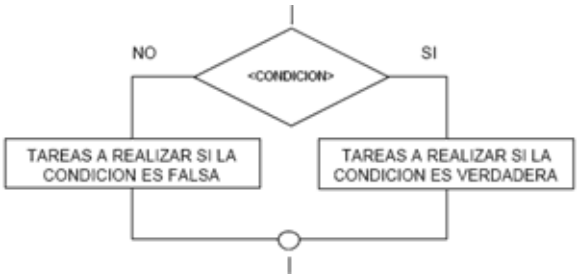
Dobles:

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:

Pseudocódigo:

```
Si <condición> entonces
        Instrucción (es)
    Si no
        Instrucción (es)
Fin-Si
```

Diagrama de flujo:



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.

Múltiples:

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 especificas. La forma común es la siguiente:

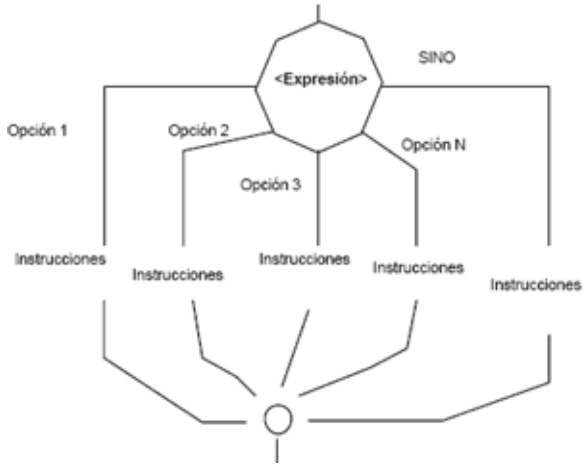
Múltiples (En caso de):

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:

Pseudocódigo:

```
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
```

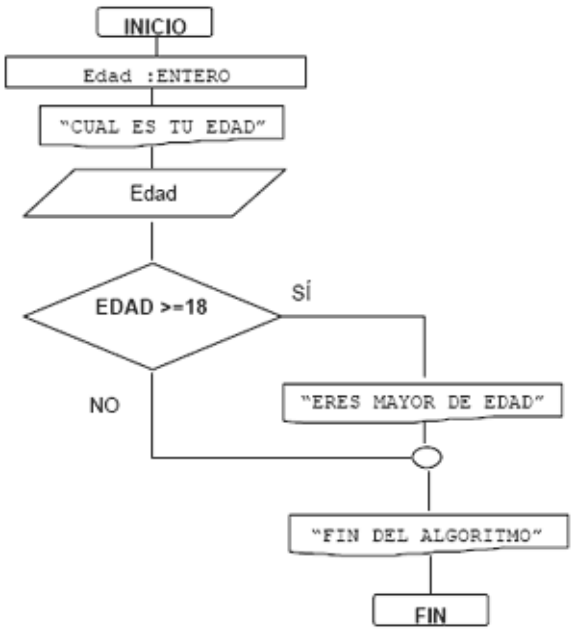
Diagrama de flujo:



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 Pseudocódigo y Diagrama de flujo.

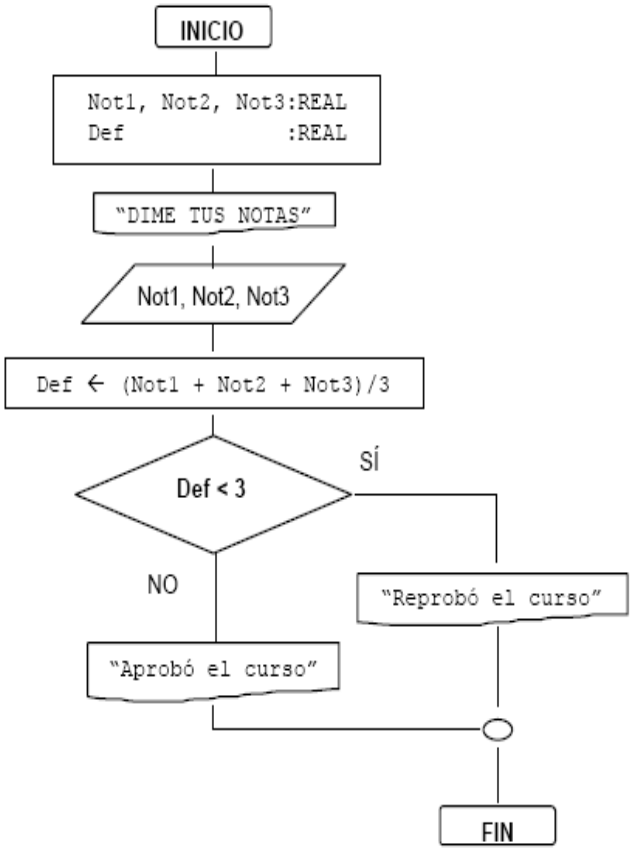
Diagrama de flujo



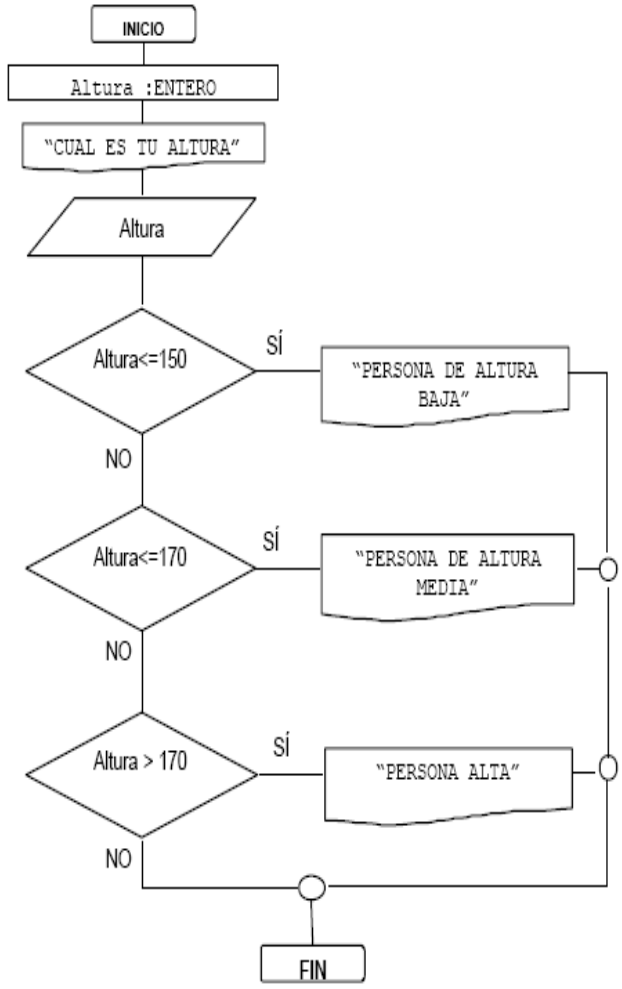
pseudocódigo

```
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 Pseudocódigo y Diagrama de flujo.

Diagrama de flujo	***pseudocódigo***
	INICIO Not1, Not2, Not 3 :REAL Def: REAL LEA Nota1, Nota2, Nota3 Def ß (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 Pseudocódigo y Diagrama de flujo.

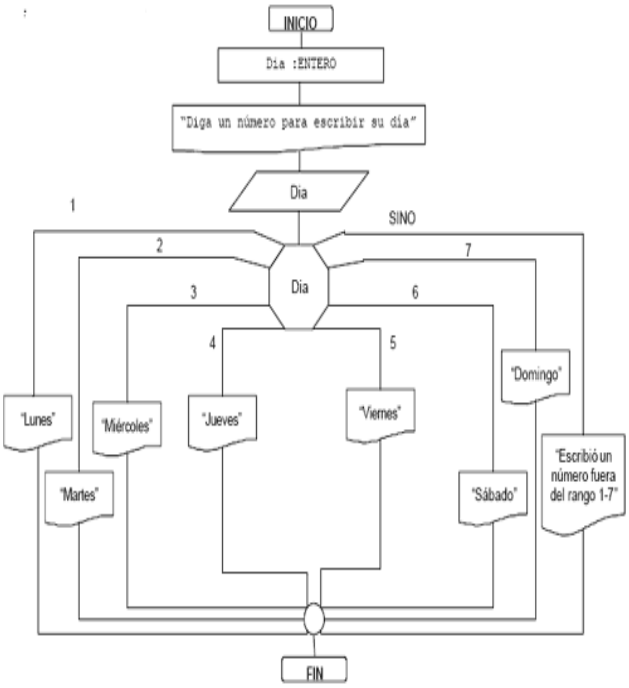
Diagrama de flujo	***pseudocódigo***
	INICIO Altura: ENTERO ESCRIBA “Cuál es tu altura? ” LEA Altura Si Altura <=150 entonces ESCRIBA “persona de altura baja” Sino Si Altura <=170 entonces ESCRIBA “persona de altura media” Sino Si Altura>170 ENTONCES ESCRIBA “persona alta” Fin-Si Fin-Si Fin-Si FIN

¡Es importante ser ordenado en el código que se escribe!

Dado un numero 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 Pseudocódigo y Diagrama de flujo.

Diagrama de flujo	***pseudocódigo***
	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

Estructuras de repetición

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:

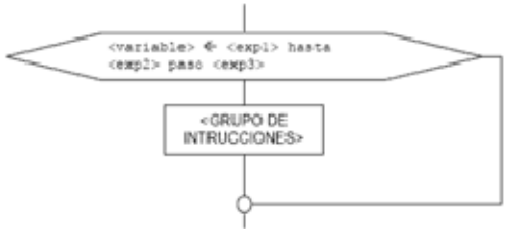
Ciclos con un Numero Determinado de Iteraciones

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

Pseudocódigo

```
Para <var> ← <exp1> hasta <exp2> paso <exp3> haga
        <Tareas a repetir>
Fin-Para
```

Diagrama de Flujo



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

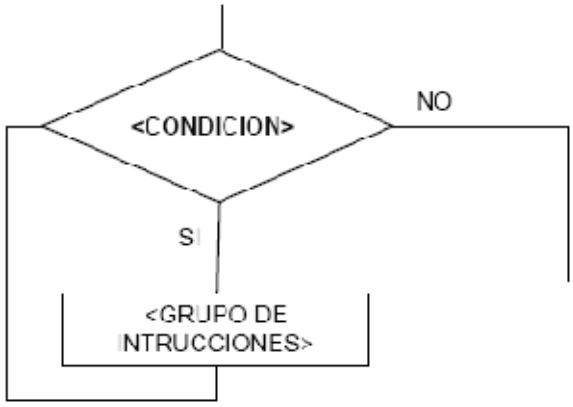
Son aquellos en que el numero de iteraciones no se conoce con exactitud, ya que esta dado en función de un dato dentro del programa.

- 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:

Pseudocódigo

```
Mientras Que <condición>
        Accion1
        Accion2
           .
           .
        AccionN
Fin-Mientras
```

Diagrama de flujo

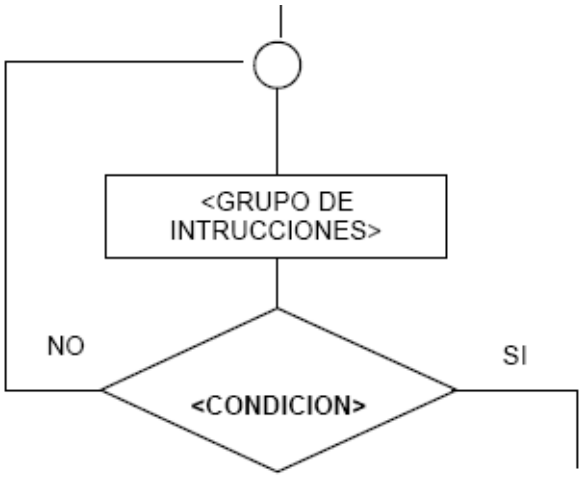


- Repita-Hasta: Esta es una estructura similar en algunas características, a la anterior. Repite un proceso una cantidad de veces, pero a diferencia del Mientras Que, el Repita-Hasta lo hace hasta que la condición se cumple y no mientras, como en el Mientras Que. Por otra parte, esta estructura permite 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:

Pseudocódigo

```
Repita
        Accion1
        Accion2
           .
           .
        AccionN
Hasta <condición>
```

Diagrama de flujo



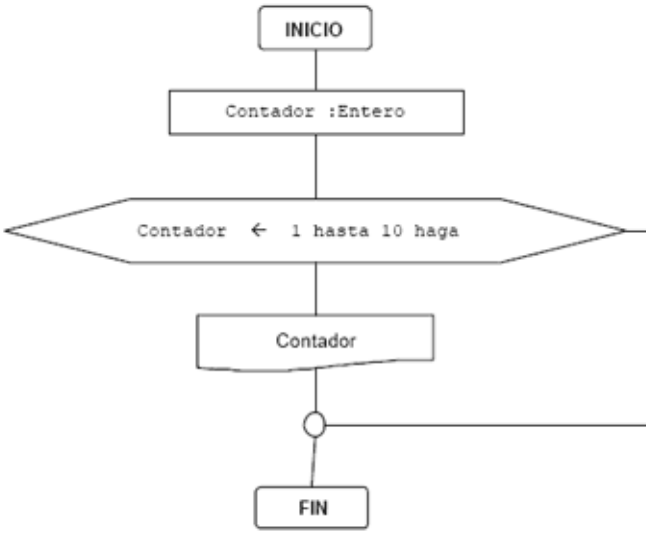
Ejemplo 1:

Realizar un algoritmo que muestre los números de uno en uno hasta diez usando una estructura Para. Exprese el algoritmo usando Pseudocódigo y Diagrama de flujo.

Pseudocódigo

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

Diagrama de flujo



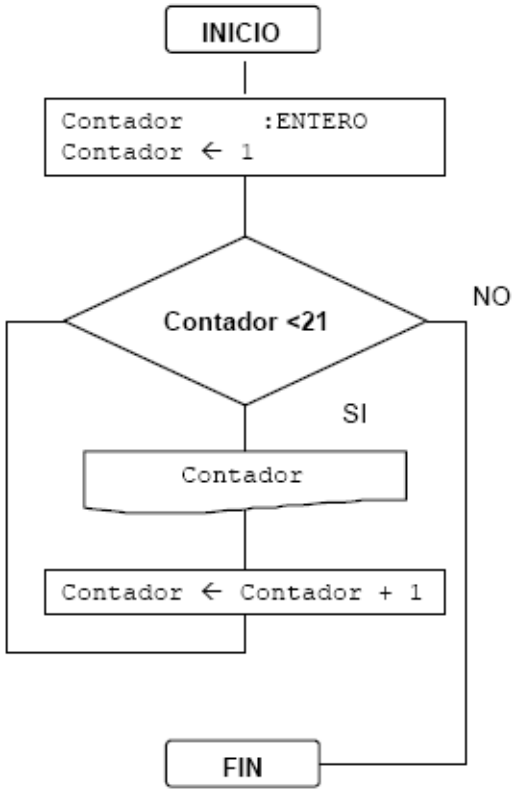
Ejemplo 2:

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

Pseudocódigo

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

Diagrama de flujo



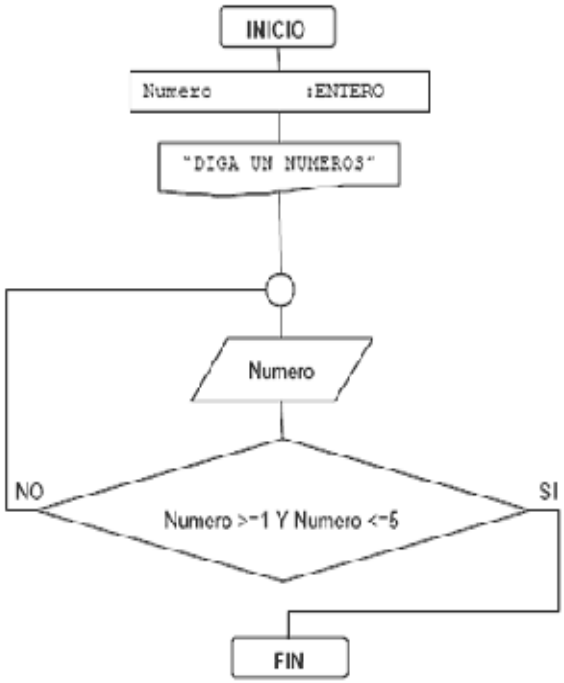
Ejemplo 3:

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

Pseudocódigo

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

Diagrama de flujo



Información recopilada de:

www.itlp.edu.mx/publica/tutoriales/algoritmos

www.desarrolloweb.com

http://html.rincondelvago.com/diseno-estructural-de-algoritmos.html