Concepto de lenguaje de programación.

Es un conjunto de símbolos junto a un conjunto de reglas para combinar dichos símbolos que se usan para expresar programas. Constan de un léxico, una sintaxis y una semántica.

¿Qué conoces tu por léxico, sintaxis y semántica?

Léxico: Conjunto de símbolos permitidos o vocabulario
Sintaxis: Reglas que indican cómo realizar las construcciones del lenguaje
Semántica: Reglas que permiten determinar el significado de cualquier construcción del lenguaje.

Tipos de lenguajes.

Atendiendo al número de instrucciones necesarias para realizar una tarea específica podemos clasificar los lenguajes informáticos en dos grandes bloques:

· bajo nivel
· alto nivel

Lenguajes de bajo nivel

Es el tipo de lenguaje que cualquier computadora es capaz de entender. Se dice que los programas escritos en forma de ceros y unos están en lenguaje de máquina, porque esa es la versión del programa que la computadora realmente lee y sigue.

Lenguajes de alto nivel

Son lenguajes de programación que se asemejan a las lenguas humanas usando palabras y frases fáciles de entender.
· En un lenguaje de bajo nivel cada instrucción corresponde a una acción ejecutable por el ordenador, mientras que en los lenguajes de alto nivel una instrucción suele corresponder a varias acciones.

· Características de los lenguajes de alto nivel:

Son independientes de la arquitectura física de la computadora. Permiten usar los mismos programas en computadoras de diferentes arquitecturas (portabilidad), y no es necesario conocer el hardware específico de la máquina. La ejecución de un programa en lenguaje de alto nivel, requiere de una traducción del mismo al lenguaje de la computadora donde va a ser ejecutado. Una sentencia en un lenguaje de alto nivel da lugar, al ser traducida, a varias instrucciones en lenguaje entendible por el computador. Utilizan notaciones cercanas a las usadas por las personas en un determinado ámbito. Se suelen incluir instrucciones potentes de uso frecuente que son ofrecidas por el lenguaje de programación.

Generaciones de Lenguajes:

1. lenguajes de máquina
2. Lenguajes ensambladores
3. Lenguajes de procedimientos

4. Lenguajes orientados a problemas
5. Lenguajes naturales

1. Lenguaje de máquina (Primera Generación)

Es el lenguaje que la computadora entiende, su estructura está totalmente adaptada a los circuitos de la máquina y la programación es tediosa porque los datos se representan por ceros y unos. Es de bajo nivel. Es un conjunto de instrucciones codificadas en binario que son capaces de relacionarse directamente con los registros y circuitería del microprocesador de la computadora y que resulta directamente ejecutable por éste, sin necesidad de otros programas intermediarios. Los datos se referencian por medio de las direcciones de memoria donde se encuentran y las instrucciones realizan operaciones simples. Estos lenguajes están íntimamente ligados a la CPU y por eso no son transferibles. (baja portabilidad). Para los programadores es posible escribir programas directamente en lenguaje de máquina, pero las instrucciones son difíciles de recordar y los programas resultan largos y laboriosos de escribir y también de corregir y depurar.

2. Lenguaje ensamblador (Segunda Generación)

Es otro lenguaje de programación de bajo nivel, pero simbólico porque las instrucciones se construyen usando códigos de tipo mnemotécnico, lo cual facilita la escritura y depuración de los programas pero no los acorta puesto que para cada acción se necesita una instrucción. El programa ensamblador va traduciendo línea a línea a la vez que comprueba la existencia de errores. Si localiza alguno da un mensaje de error. Algunas características que lo diferencian del lenguaje de máquina son que permite el uso de comentarios entre las líneas de instrucciones; en lugar de direcciones binarias usa identificadores como total, x, y, etc. Y los códigos de operación se representan por mnemotécnica siempre tienen la desventaja de repertorio reducido de instrucciones, rígido formato para las instrucciones, baja portabilidad y fuerte dependencia del hardware. Tiene la ventaja del uso óptimo de los recursos hardware, permitiendo la obtención de un código muy eficiente. Ejemplo de algunos códigos mnemónicos son: STO para guardar un dato, LOA para cargar algo en el acumulador, ADD para adicionar un dato, INP para leer un dato, STO para guardar información, MOV para mover un dato y ponerlo en un registro, END para terminar el programa, etc. Con la tercera generación avanzamos a los lenguajes de alto nivel, muchos de los cuales se consideran exportables. Esto es, pueden correr en más de un tipo de computadoras, se les puede exportar de una máquina a otra.

3. Lenguaje de procedimientos (Tercera Generación)

Son lenguajes de alto nivel similares al habla humana pero requieren cierta capacitación para su uso.
Ventajas:
a. Independencia de la arquitectura física de la computadora (portabilidad), esto significa que un mismo lenguaje puede funcionar (al menos en teoría) en distintos computadores, por lo que tanto el lenguaje como los programas escritos con él serán transportables de un computador a otro. En la práctica, esta característica resulta limitada por la gran diversidad de versiones y dialectos que se constituyen para cada lenguaje.

b. una sentencia en un lenguaje de alto nivel da lugar, al ser traducida, a varias instrucciones en lenguaje máquina. Se llaman de procedimientos porque están diseñados para expresar la lógica capaz de resolver problemas generales. Entre estos tenemos:
Basic
Pascal
Cobol
C
Fortran
Para que el lenguaje de procedimientos pueda funcionar debe traducirse a lenguaje de máquina a fin de que la computadora lo entienda. Para ello se han de usar programas traductores que realicen dicho proceso. Tienen la capacidad de soportar programación estructurada.

4. Lenguajes orientados a problemas (4GL)

Resultan más eficaces para la resolución de un tipo de problemas a costa de una menor eficiencia para otros. Requieren poca capacitación especial de parte del usuario Son considerados de muy alto nivel Diseñados para resolver problemas específicos
Incluye: lenguajes de consulta y generador de aplicaciones
Lenguajes de consulta:
Permiten a no programadores usar ciertos comandos de fácil comprensión para la búsqueda y generación de reportes a partir de una base de datos.
Generador de aplicaciones:
Quiere decir que cuando se diseña uno de estos lenguajes, se tiene en cuenta que su finalidad es la resolución de problemas, prescindiendo de la arquitectura del computador. Contiene varios módulos que han sido preprogramados para cumplir varias tareas.

5. Lenguajes naturales

Lenguajes orientados a aplicaciones en inteligencia artificial, como lisp y prolog. Dentro de este campo destacan las aplicaciones en sistemas expertos, juegos, visión artificial (Jurasic Park) y robótica. Lisp es un lenguaje para procesamiento de listas y manipulación de símbolos. Prolog es un lenguaje basado en la lógica, para aplicaciones de bases de datos e Inteligencia Artificial. Podemos decir entonces, que los lenguajes de alto nivel, tienen las ventajas de mayor legibilidad de los programas, portabilidad, facilidad de aprendizaje y facilidad de modificación.

PARA ANALIZAR:

1. ¿Cuál es la diferencia fundamental entre los lenguajes de alto nivel y bajo nivel?
2. Investigar analogías y diferencias entre el código máquina y el lenguaje ensamblador.
3. Buscar información que permita decidir cuáles serían los lenguajes de programación más apropiados para realizar: aplicaciones para gestión de oficinas, complejos cálculos científicos, un sistema experto en medicina, un simulador de vuelo, manipulación de bases de datos, control de un robot industrial.
Sugerencias de ampliación:
Clasificación de los lenguajes de alto nivel
Lenguajes de propósito general:

- PL/1
- PASCAL
- LOGO
- BASIC
- MODULA II
- FORTH
- C
- COBOL
- ADA

Lenguajes de Cálculo científico:

- FORTRAN
- ADA
- ALGOL
- APL
- PASCAL

Lenguajes orientados a la gestión:

- COBOL
- RPG

Lenguajes de simulación en general:

- GPSS
- SIMULA
- MIMIC

En el sistema educativo:

- ASSET
- C
- MODULA II
- LOGO
- BASIC
- PILOT
- PASCAL

Lenguajes orientados a objetos:

- SMALLTALK

Lenguajes interrogativos:

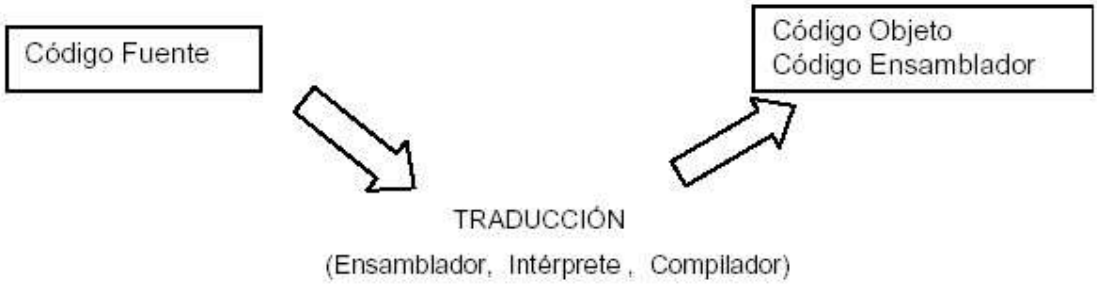
- PROLOG
- DBASE

Lenguajes para Inteligencia Artificial

- LISP
- PROLOG

Algoritmos y diagramas de flujo- Intérpretes y compiladores.

Los compiladores, los intérpretes y los ensambladores se encargan de traducir lo que haya escrito en lenguaje de alto nivel (código fuente) y lo convierten a código objeto (casi ejecutable).



Compilador

Es un programa que traduce un programa escrito en un lenguaje de alto nivel, por ejemplo C, en un programa en lenguaje de máquina que la computadora es capaz de entender y ejecutar

directamente. Un compilador es un tipo especial de programa, en cuanto a que sus entradas o datos son algún programa y su salida es otro programa. Para evitar confusiones, solemos llamar programa fuente o código fuente al programa de entrada, y programa objeto o código objeto a la versión traducida que el compilador produce. Código se usa frecuentemente para referirse a un programa o a una parte de él, sobre todo cuando se habla de programas objeto.

Ejemplo:
Pascal
Cobol
Fortran
Ada
Código Fuente
Código Objeto
Código Ensamblador

Modula 2

C, C++

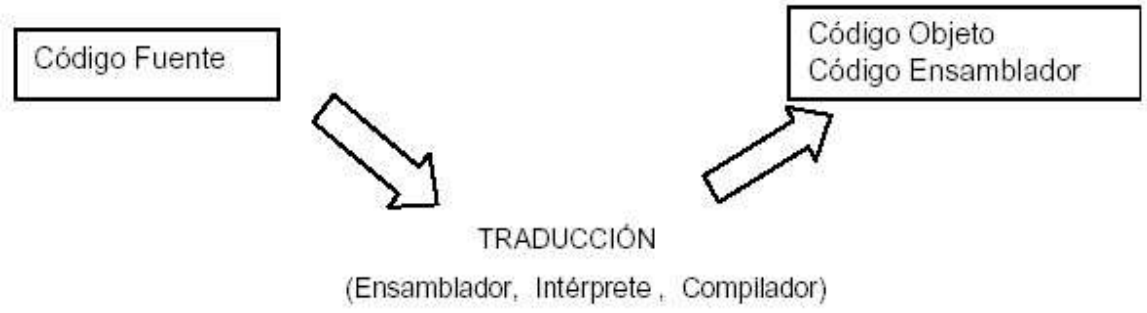
El compilador, informa al usuario de la presencia de errores en el programa fuente, pasándose a crear el programa objeto cuando está libre de errores. El código objeto puede ser ejecutado posteriormente. Una vez traducido un programa, su ejecución es independiente de su compilación. involucra dos pasos en su operación:

1. Convertir código fuente a objeto
2. Ejecutar el código objeto

Ventaja:

Al tener el código objeto, el programa se ejecuta más rápido

Fases de compilación



Análisis: Dependiente del lenguaje. Independiente de la máquina
Sintaxis: Independiente del lenguaje. Dependiente de la máquina.
Intérprete**:** Es el que permite que un programa fuente escrito en un lenguaje vaya traduciéndose y ejecutándose directamente sentencia a sentencia por la computadora. Convierte uno por uno los enunciados del código fuente a código objeto antes de ser ejecutados. Convierte y ejecuta el programa en línea al mismo tiempo. Ejemplo: Basic estándar.

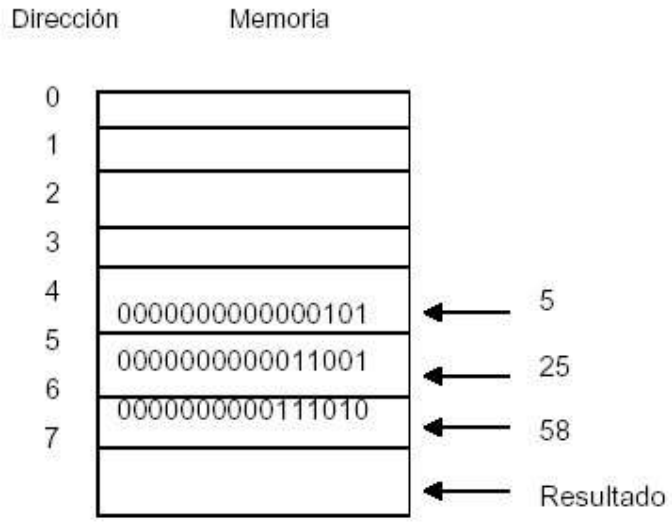
Ventaja:
Las ventajas de los intérpretes son:
· Resulta más fácil localizar y corregir errores (depuración de programas)
- son más pedagógicos para aprender a programar.
· El programa es más fácil de desarrollar.
Traducen programas de alto nivel. No se genera en la mayoría de los ficheros.

Programa Fuente
Código Intermedio
Programa Objeto

Para cada una de las líneas se ejecuta el siguiente proceso:
1. Análisis de la instrucción de esa línea
2. Traducción de esa línea (si ya está correcta) a código objeto
3. Ejecución de esa línea

Con el intérprete, cada vez que necesitamos ejecutar el programa tenemos que volver a analizarlo porque no hay código objeto.
Con el compilador, aunque más lenta la traducción, sólo se realiza una vez.

Ejemplo 4
Supongamos que se han almacenado tres valores 5 = 01012, 25 = 110012 y 58 = 1110102 en las posiciones de memoria con direcciones 4,5 y 6.
Queremos multiplicar los dos primeros valores, sumar el tercero y almacenar el resultado en la palabra de memoria 7.



Para llevar a cabo este cálculo, se deben ejecutar las siguientes instrucciones:
Recuperar el contenido de la palabra de memoria 4 y cargarlo en el registro acumulador de la unidad aritmético lógica.
Recuperar el contenido de la palabra de memoria 5 y calcular el producto de este valor y el valor situado en el acumulador.
Recuperar el contenido de la palabra de memoria 6 y sumar su valor con el valor situado en el registro acumulador.
Almacenar el contenido del registro acumulador en la palabra de memoria 7. Para almacenar estas instrucciones en la memoria de la computadora, deben estar representadas en forma

binaria. Las direcciones de los datos no presentan problemas, puesto que pueden ser convertidos fácilmente a direcciones binarias:

4 = 100
5 = 101
6 = 110
7 = 111

Las operaciones de cargar, multiplicar, sumar, almacenar y otras instrucciones máquina básicas se representan mediante códigos numéricos, llamados códigos de operación, por ejemplo:

LOAD = 16 = 10000
STORE = 17 = 10001
ADD = 35 = 100011
MULTIPLY = 36 = 100100
SUB = 37 = 100101
DIV = 38 = 100110
Usando parte de una palabra para almacenar el código de operación y otra para la dirección del operando, podemos representar nuestra secuencia de instrucciones en lenguaje máquina como sigue:

1. 0001000000000100
2. 0010010000000101
3. 0010001100000110
4. 0001000100000111

Estas pueden ser almacenadas en cuatro palabras consecutivas de memoria. Cuando se ejecuta el programa, la unidad de control recuperará cada una de las instrucciones, la decodificará para determinar la operación y la dirección del operando, recuperará el operando, y entonces ejecutará la operación requerida, usando la unidad aritmético lógica cuando sea necesario. Los programas para las primeras computadoras tuvieron que ser escritos en lenguaje de máquina. Posteriormente fue posible escribirlos en lenguaje ensamblador usando códigos nemotécnicos en lugar de códigos de operación numéricos y nombres de variables en lugar de direcciones numéricas.
Por ejemplo la secuencia de instrucciones anteriores se escribiría así:

1. LOAD A
2. MULT B
3. ADD C
4. STORE X

Luego que se crearon los lenguajes de alto nivel, las instrucciones se escribían en forma más entendible para el programador. El ejemplo anterior podría ser como lo siguiente usando C++:

X = A * B + C

El producto en la programación se representa por asterisco. En cada uno de estos casos (ensamblador y lenguajes de alto nivel), el compilador traduce cada instrucción del programa en una secuencia de cuatro instrucciones máquina y genera un programa objeto.

Ejercicios Propuestos:
Usando mnemónicos de instrucción y códigos de operación, escriba una secuencia de

instrucciones en a) Lenguaje ensamblador b) Lenguaje de máquina, equivalente a las instrucciones de C++:

1. X = (A - B) * C
X = (A+B) / (C+D)
X = (A + B) - C

Para las instrucciones máquina, suponga que los valores de A, B, C y D son almacenados en las palabras de memoria 15,16, 17 y 18 respectivamente, y que los valores de X e Y se almacenarán en la palabra de memoria 23 y 24 respectivamente.

Conceptualización de los tipos de datos.

DATO. Es la expresión general que describe los objetos con los cuales opera el programa. Por ejemplo, la edad y el domicilio de una persona, forman parte de sus datos. Los datos se sitúan en objetos llamados variables.

Las **variables** son zonas de memoria cuyo contenido cambia durante la fase de procesamiento de información. Son objetos cuyo valor puede ser modificado a lo largo de la ejecución de un programa. Las variables llevan un nombre llamado identificador. Este puede ser una cadena de letras y dígitos, empezando siempre con una letra. Por ejemplo: Pi, curso99, nom_alum, etc.

Los Identificadores son palabras creadas por los programadores para dar nombre a los objetos y demás elementos que necesitamos declarar en un programa: variables, constantes, tipos, estructuras de datos, archivos, subprogramas, etc. En C++ las letras mayúsculas se tratan como diferentes y distintas unas de otras. Por ejemplo, contador, Contador y CONTADOR son tres nombres de identificadores distintos. Un identificador no puede ser igual a una palabra reservada, y no debe tener el mismo nombre que una función, ya sea definida por el usuario o de la biblioteca de C.

Constantes: Son objetos cuyo valor permanece invariable a lo largo de la ejecución de un programa. Una constante es la denominación de un valor concreto, de tal forma que se utiliza su nombre cada vez que se necesita referenciarlo. Por ejemplo, si se desea obtener un reporte para cada uno de los empleados de una empresa, con sus datos generales, la fecha y cantidad de dinero que recibieron la última semana, el dato fecha puede ser una constante ya que es el mismo para todos.

Expresiones. Son representaciones de un cálculo necesario para la obtención de un resultado. Estas pueden ser valores constantes, funciones o combinaciones de valores, variables, funciones y operadores que cumplen determinadas reglas de construcción de una expresión. Son un conjunto de operadores y operandos que producen un valor. Por ejemplo:

Cos (pi * X) + 12.56 * SQR(100)

Un **operador** es un símbolo o palabra que significa que se ha de realizar cierta acción entre uno o dos valores que son llamados operandos.

Operación de **Asignación**:

Es el modo de darle valores a una variable. Se representa con el símbolo u operador !, el cual se conoce como instrucción o sentencia de asignación.

Sintaxis: Nombre de la variable ¬ ! expresión.
Ejemplo: cociente ! ¬ cal1/cal2

Tipo : es el conjunto de valores que puede tomar una variable. Es el conjunto de transformaciones y funciones internas y externas definidas sobre el conjunto de datos. Se tienen dos tipos de datos: Simples como numéricos y alfanuméricos y Estructuras de datos que pueden ser internas o externas.

Tipos de datos Numéricos son aquellos cuyo contenido es una serie de dígitos (0-9) que en conjunto nos proporcionan un valor numérico ya sea entero o real y pueden ser precedidos de un signo + ó -. Tipos de datos Alfanuméricos son aquellos cuyo contenido son letras del abecedario, números o caracteres especiales o bien una combinación de ellos.

Tipos de Instrucciones:

a) Instrucciones de inicio/fin (ejemplo inicio - fin)
b) Instrucciones de asignación (ejemplo B ! 7)
c) Instrucciones de lectura (entrada) (ejemplo leer)
d) Instrucciones de escritura (salir) (ejemplo escribir)
e) Instrucciones de bifurcación (ejemplo Goto fin)

Tipos de Operadores:

a) Aritméticos (su resultado es un número): potencia, * , / , mod, div, + , -
b) Relacionales (su resultado es un valor de verdad): =, <, >, <=, >=, <>
c) Lógicos o Booleanos (su resultado es un valor de verdad): not, and, or
d) Alfanuméricos : + (concatenación)
e) Asociativos. El único operador asociativo es el paréntesis ( ) , el cual permite indicar en qué orden deben realizarse las operaciones. Cuando una expresión se encuentra entre paréntesis, indica que las operaciones que están dentro de ellos debe realizarse primero. Si en una expresión se utilizan más de un paréntesis se deberá proceder primero con los que se encuentren más hacia el centro de la expresión.

Jerarquía de Operaciones:
( )
signo
Potencia
Producto y división
Div
Mod
Suma y resta
Concatenación
Relacionales
Negación
And
Or
Datos de tipo entero tienen los operadores +, -, *, /, div, mod, abs, sqr, sqrt, ln, exp, sin, cos, tan, pow, etc. Los datos de tipo real tienen los mismos operadores enteros y además trunc, round, int, y otros. La suma y multiplicación de datos de tipo real cumplen la propiedad conmutativa, pero no siempre la asociativa y la distributiva.

Para resolver una expresión aritmética se deben seguir las siguientes reglas:
· Primero se resuelven las expresiones que se encuentran entre paréntesis.
· Se procede aplicando la jerarquía de operadores.
· Al evaluar una expresión, si hay dos operadores con la misma jerarquía, se procede a evaluar de izquierda a derecha.
· Si hay expresiones relacionales, se resuelven primero paréntesis, luego se encuentran los valores de verdad de las expresiones relacionales y por último se aplica jerarquía de operadores lógicos. En caso de haber iguales, proceder de izquierda a derecha.

EJERCICIOS.
Aplicando la jerarquía de los operadores, encontrar el valor de cada una de las siguientes expresiones:

1) 4 + 1 * 5 ^ 2 - 1
2) 9 / 3 + 4 ^ 2 - 5 * 1 + 9 / -2 + 3
3) 5 / 2 + 3 - 4 * 5 / 2
4) (4 + 1) * 5 ^ 2 - 1
5) 17 / 2 + 3 ^ 2 ^ 2 - 2 * 2 / 2

Aplicando la jerarquía de operadores, encontrar el valor de verdad de cada una de las siguientes expresiones:

1. Not ((M > N and R > S) or (NOT(T < V and S > M))) para M=8, N=9, R=5, S=5 , T=4 y V= 2
2. (3 * 2 ^ 2 - 4 / 2 * 1) > (3 * 2 ^ -4 / 2 * 1) and (5 > 9 / 3)

Herramientas de programación.

Algoritmo

Es una serie de operaciones detalladas a ejecutar paso a paso, que conducen a la resolución de problemas. Es un conjunto de reglas para resolver determinado problema describiendo de forma lógica su solución. Cada una de las acciones de que consta un algoritmo es denominada sentencia y éstas deben ser escritas en términos de cierto lenguaje comprensible para el computador, que es el lenguaje de programación. Para diseñar un algoritmo se debe comenzar por identificar las tareas más importantes para resolver el problema y disponerlas en el orden en que han de ser ejecutadas.

Criterios que debe satisfacer un algoritmo (características):

1. Entrada. Son cero o más cantidades las cuales son externamente sustituidas.
2. Salida. Al menos una cantidad es producida.
3. Exactitud/precisión. Cada instrucción debe ser clara y sin ambigüedad.
4. Finito. Terminará después de un número finito de pasos.
5. Eficiente. Cada instrucción puede ser verificada por una persona con una prueba manual que satisfaga los requerimientos planteados por el problema.

Partes de un Algoritmo

Representación gráfica de algoritmos.

a) Descripción Narrada
b) Pseudocódigo
c) Diagramas de Flujo

Descripción Narrada
Este algoritmo es caracterizado porque sigue un proceso de ejecución común y lógico, describiendo textualmente paso a paso cada una de las actividades a realizar dentro de una actividad determinada.
Ejemplo 1 Algoritmo para asistir a clases:

1. Levantarse
2. Bañarse
3. Vestirse
4. Desayunar
5. Cepillarse los dientes
6. Salir de casa
7. Tomar el autobús
8. Llegar al ITCA
9. Buscar el aula
10. Ubicarse en un asiento

Descripción en Pseudocódigo
Pseudo = falso. El pseudo código no es realmente un código sino una imitación y una versión abreviada de instrucciones reales para las computadoras. Es una técnica para diseño de programas que permite definir las estructuras de datos, las operaciones que se aplicarán a los datos y la lógica que tendrá el programa de computadora para solucionar un determinado problema. Utiliza un pseudolenguaje muy parecido a nuestro idioma, pero que respeta las directrices y los elementos de los lenguajes de programación. Se concibió para superar las dos principales desventajas de los flujogramas: lento de crear y difícil de modificar sin un nuevo redibujo.

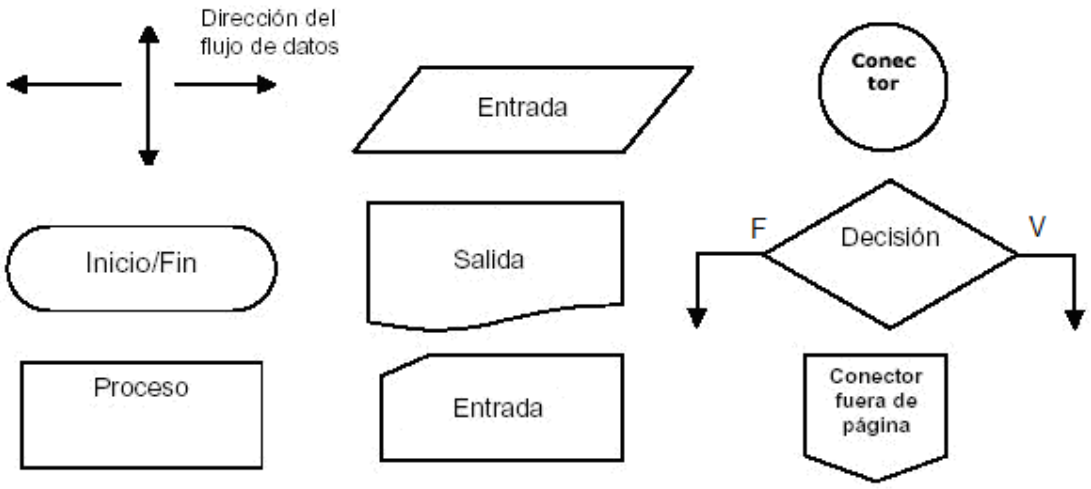
Diagramas de Flujo.
Son la representación gráfica de la solución algorítmica de un problema. Para diseñarlos se utilizan determinados símbolos o figuras que representan una acción dentro del procedimiento. Utilizan unos símbolos normalizados, con los pasos del algoritmo escritos en el símbolo adecuado y los símbolos unidos con flechas, denominadas líneas de flujo, que indican el orden en que los pasos deben ser ejecutados.

Para su elaboración se siguen ciertas reglas:
Se escribe de arriba hacia abajo y de izquierda a derecha
Siempre se usan flechas verticales u horizontales, jamás curvas
Evitar cruce de flujos
En cada paso expresar una acción concreta

Secuencia de flujo normal en una solución de problema:

Tiene un inicio
Una lectura o entrada de datos
El proceso de datos
Una salida de información
Un final

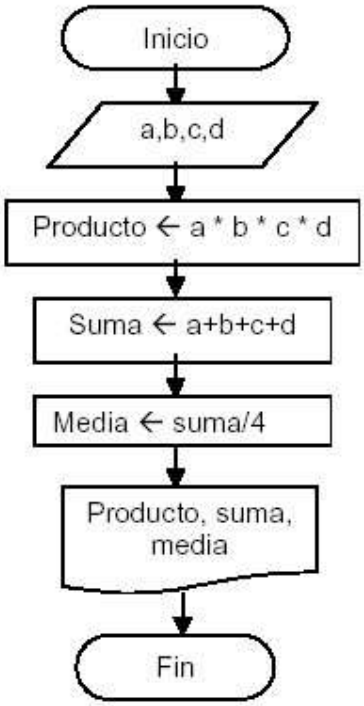
Simbología para diseñar flujogramas.



VENTAJAS DE USAR FLUJOGRAMAS
Rápida comprensión de las relaciones
Análisis efectivo de las diferentes secciones del programa
Pueden usarse como modelos de trabajo en el diseño de nuevos programas o sistemas
Comunicación con el usuario
Documentación adecuada de los programas
Codificación eficaz de los programas
Depuración y pruebas ordenadas de programas

DESVENTAJAS DE LOS FLUJOGRAMAS
Diagramas complejos y detallados suelen ser laboriosos en su planteamiento y diseño
Acciones a seguir tras la salida de un símbolo de decisión, pueden ser difíciles de seguir si existen diferentes caminos
No existen normas fijas para la elaboración de los diagramas de flujo que permitan incluir todos los detalles que el usuario desee introducir.

Representando el ejemplo como flujograma tenemos:



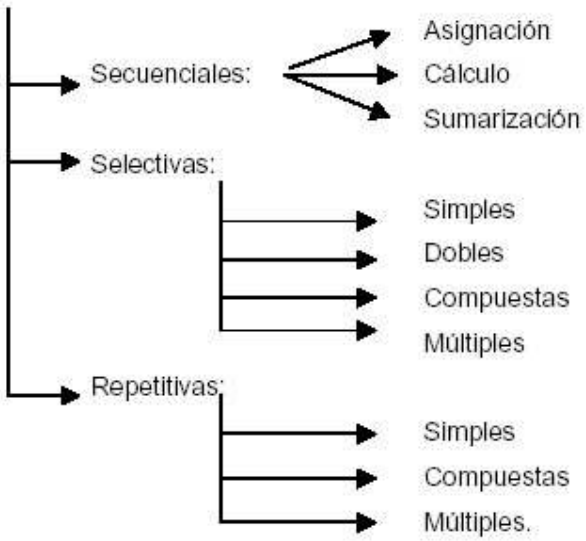
Tipos de estructuras de programación. Estructuras básicas y secuencial.

Un problema se puede dividir en acciones elementales o instrucciones, usando un número limitado de estructuras de control (básicas) y sus combinaciones que pueden servir para resolver dicho problema.

Las Estructuras Básicas pueden ser:

Secuenciales: cuando una instrucción del programa sigue a otra.
Selección o decisión: acciones en las que la ejecución de alguna dependerá de que se cumplan una o varias condiciones. Repetición, Iteración: cuando un proceso se repite en tanto cierta condición sea establecida para finalizar ese proceso.

ESTRUCTURAS BÁSICAS.

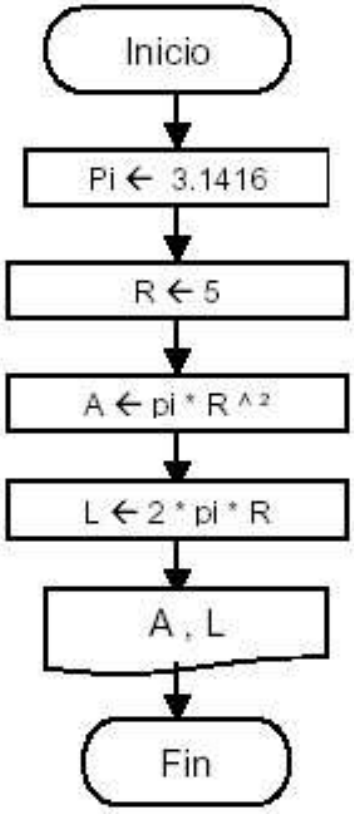


Estructura Secuencial.
Se caracteriza porque una acción se ejecuta detrás de otra. El flujo del programa coincide con el orden físico en el que se han ido poniendo las instrucciones. Dentro de este tipo podemos encontrar operaciones de inicio/fin, inicialización de variables, operaciones de asignación, cálculo, sumarización, etc. Este tipo de estructura se basa en las 5 fases de que consta todo algoritmo o programa:
Definición de variables (Declaración)
Inicialización de variables.
Lectura de datos
Cálculo
Salida

Ejemplo 1.
Se desea encontrar la longitud y el área de un círculo de radio 5.

Solución.
El objetivo del ejercicio es encontrar la longitud y el área de un círculo con un radio conocido y de valor 5. Las salidas serán entonces la longitud y el área. (Fase 5 del algoritmo) Sabemos que la longitud de un círculo viene dada por la fórmula 2 * pi * radio y que el área viene dada por pi * radio al cuadrado. (Fase 4 del algoritmo) Si definimos las variables como: (fase 1 del algoritmo)
L = Longitud A = área R = radio pi = 3.1416 hagamos el algoritmo:
Inicio
Pi ¬ 3.1416 (definición de un valor constante)
R ¬ 5 (radio constante ya que es conocido su valor)
A ¬ pi * R ^ ² (asignación del valor del área)
L ¬ 2 * pi * R (asignación del valor de la longitud)
Escribir (A, L) (salida del algoritmo)
Fin

Representación en Diagrama de Flujo para el ejemplo:



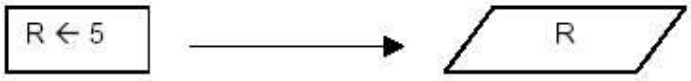
En este ejercicio no existen datos de entrada ya que para calcular el área y la longitud necesitamos únicamente el radio y el valor de Pi los cuales ya son dados en el problema Modificar el problema anterior para que sea capaz de calcular el área y la longitud de un círculo de cualquier radio requerido.

Solución.

El problema es el mismo con la variante de que ahora ya existe un dato de entrada, puesto que el radio puede ser cualquiera y será necesario que el usuario sea quien lo introduzca de teclado. Usando las misma definición de variables tenemos:

Algoritmo:
Inicio
Pi ¬ 3.1416 (fase de inicialización)
Leer (R) (fase de lectura)
Area ¬ pi * R ^ ² (fase de cálculos)
L ¬ 2 * pi * R
Escribir (A, L) (fase de salida)
Fin

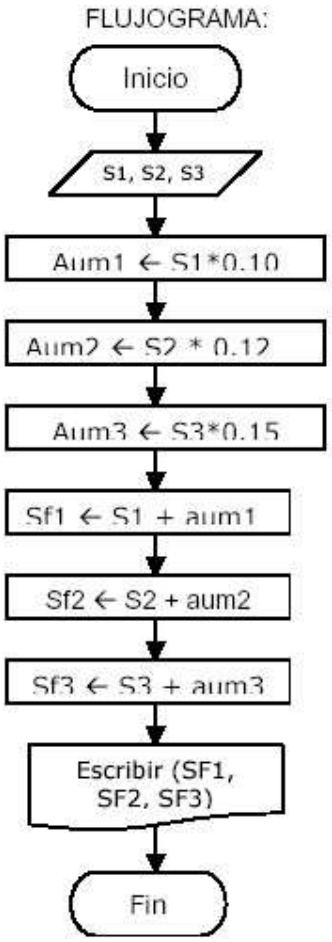
Note que la instrucción de asignación fue cambiada por la instrucción leer. En el flujograma deberán cambiarse también los símbolos que los representan:



Ejemplo 3.
Leer el sueldo de tres empleados y aplicarles un aumento del 10, 12 y 15% respectivamente.
Desplegar el resultado.
Salidas: Sueldos finales
Entradas: Salarios de los empleados
Datos adicionales: aumentos del 10, 12 y 15%
Cálculos:
Sueldo final = sueldo inicial + aumento
Aumento = sueldo inicial * porcentaje/100
Definición de variables:
Sf1, Sf2, Sf3 = los sueldos finales
S1, S2, S3 = salarios de los empleados
Aum1, aum2, aum3 = aumentos

ALGORITMO
Inicio
Leer (S1,S2,S3)
Aum1 ¬ S1 * 0.10
Aum2 ¬ S2 * 0.12
Aum3 ¬ S3 * 0.15
Sf1 ¬ S1 + Aum1
Sf2 ¬ S2 + Aum2
Sf3 ¬ S3 + Aum3
Escribir (SF1,SF2,SF3)
Fin

FLUJOGRAMA



Tipos de estructuras selectivas. Estructura simple.

La especificación formal de algoritmos tiene realmente utilidad cuando el algoritmo requiere una descripción más complicada que una lista sencilla de instrucciones. Este es el caso cuando existen un número de posibles alternativas resultantes de la evaluación de una determinada condición.

Estas estructuras se identifican porque en la fase de solución del problema existe algún punto en el cual es necesario establecer una pregunta, para decidir si ciertas acciones deben realizarse o no.

Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con palabras en pseudocódigo (if - then - else o en español si - entonces - sino) y en flujograma con una figura geométrica en forma de rombo.

Las estructuras selectivas o alternativas se clasifican en:

a) Simples
b) Dobles

c) Compuestas
d) Múltiples

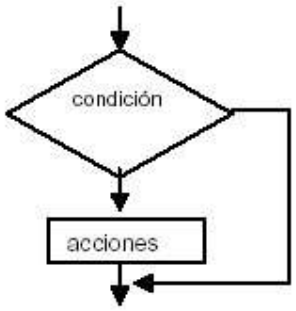
ESTRUCTURAS SELECTIVAS SIMPLES.

Se identifican porque están compuestos únicamente de una condición. La estructura si - entonces evalúa la condición y en tal caso:

Si la condición es verdadera, entonces ejecuta la acción Si (o acciones si son varias).
Si la condición es falsa, entonces no se hace nada.

Español	Inglés
Si <condición>	If <condición>
Entonces	then
<acción Si>	<acción Si>
fin_si	endif

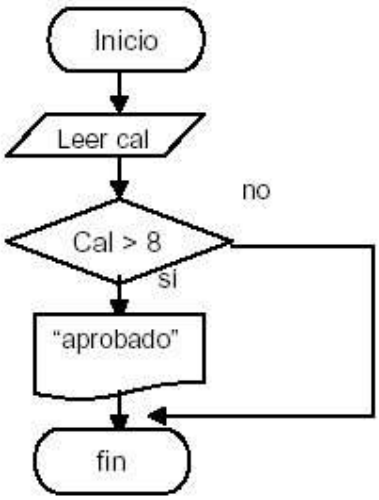
Representación en Flujograma:



Ejemplo 1.
Construir un algoritmo tal, que dado como dato la calificación de un alumno en un examen, escriba "Aprobado" en caso que esa calificación fuese mayor que 8.
Salidas: mensaje de aprobado si se cumple la condición.
Entradas: calificación

Datos adicionales: un alumno aprueba si la calificación es mayor que 8
Variables:
Cal = calificación
Algoritmo:
Inicio
Leer (cal)
Si cal > 8 entonces
Escribir ("aprobado")
Fin_si
Fin

Flujograma:



Estructura de selección doble.

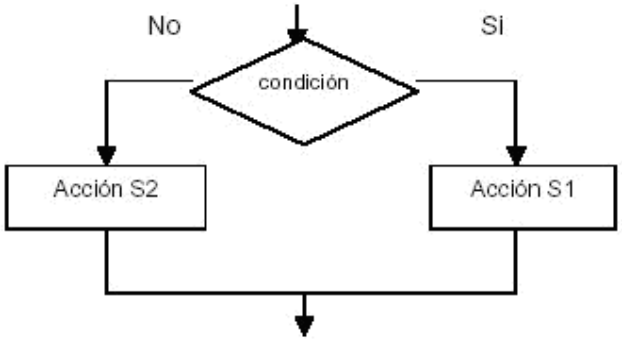
Son estructuras lógicas que permiten controlar la ejecución de varias acciones y se utilizan cuando se tienen dos opciones de acción, por la naturaleza de estas se debe ejecutar una o la otra, pero no ambas a la vez, es decir, son mutuamente excluyentes.

Representación pseudocodificada.

Español	Inglés
Si <condición> entonces	If <condición> then
<acción S1>	<acción S1>
sino	else
<acción S2>	<acción S2>
Fin_Si	End_if

Entonces, si una condición C es verdadera, se ejecuta la acción S1 y si es falsa, se ejecuta la acción S2.

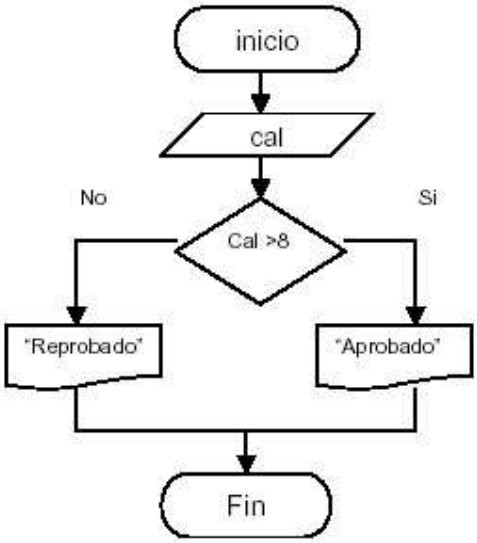
Flujograma:



Ejemplo 1

Dado como dato la calificación de un alumno en un examen, escriba "aprobado" si su calificación es mayor que 8 y "Reprobado" en caso contrario.

Algoritmo:
Inicio
Leer (cal)
Si cal > 8 entonces
Escribir ("aprobado")
Sino
Escribir ("reprobado")
Fin_si
Fin



Ejemplo 2.

Dado como dato el sueldo de un trabajador, aplicar un aumento del 15% si su sueldo es inferior a $1000 y 12% en caso contrario, luego imprimir el nuevo sueldo del trabajador.

Algoritmo:

```
Inicio
        Leer (sue)
        Si sue < 1000 entonces
                Nsue ← sue *1.15
        Sino
                Nsue ← sue *1.12
        Fin_si
        Escribir (Nsue)
Fin
```

EXPRESIONES LÓGICAS

Sirven para plantear condiciones o comparaciones y dan como resultado un valor booleano verdadero o falso, es decir, se cumple o no se cumple la condición. Se pueden clasificar en

simples y complejas. Las simples son las que usan operadores relacionales y las complejas las que usan operadores lógicos.

Ejemplos:
Un ejemplo en el cual usamos el operador lógico AND sería:
Una escuela aplica dos exámenes a sus aspirantes, por lo que cada uno de ellos obtiene dos calificaciones denotadas como C1 y C2. El aspirante que obtenga calificaciones mayores que 80 en ambos exámenes es aceptado; en caso contrario es rechazado.
En este ejemplo se dan las condiciones siguientes:

```
Si (C1 >= 80) y (C2 >= 80) entonces
Escribir ("aceptado")
Sino
Escribir ("rechazado")
Fin_si
```

Note que también usa operadores relacionales. Por lo general cuando hay operadores lógicos, éstos van acompañados de operadores relacionales. Un ejemplo usando el operador lógico OR sería:
Una escuela aplica dos exámenes a sus aspirantes, por lo que cada uno de ellos obtiene dos calificaciones denotadas como C1 y C2. El aspirante que obtenga una calificación mayor que 90 en cualquiera de los exámenes es aceptado; en caso contrario es rechazado.
En este caso se dan las condiciones siguientes:

```
Si (C1 >=90) or (C2 >=90) entonces
Escribir ("aceptado")
Sino
Escribir ("rechazado")
Fin_si
```

La instrucción equivale a OR ya que nos dice que puede ser en cualquiera de los exámenes no necesariamente en los dos. En el ejemplo 1 la palabra ambos equivalía a seleccionar la instrucción AND. Si la instrucción nos dijera que obtenga una nota en cualquiera de los exámenes pero no en ambos, nos estaría indicando una instrucción XOR que es un tipo de OR pero exclusivo. Es decir, no puede considerarse el caso en que tenga la misma nota en los dos exámenes, solo en uno de los dos.

Estructuras selectivas compuestas.

En la solución de problemas encontramos numerosos casos en los que luego de tomar una decisión y marcar el camino correspondiente a seguir, es necesario tomar otra decisión. Dicho proceso puede repetirse numerosas veces. En aquellos problemas en donde un bloque condicional incluye otro bloque condicional se dice que un bloque está anidado dentro del otro.

Ejemplo 1.
Determinar la cantidad de dinero que recibirá un trabajador por concepto de las horas extras trabajadas en una empresa, sabiendo que cuando las horas de trabajo exceden de 40, el resto se consideran horas extras y que éstas se pagan al doble de una hora normal cuando no exceden de 8; si las horas extras exceden de 8 se pagan las primeras 8 al doble de lo que se paga por una hora normal y el resto al triple.

Solución.

Lo primero que hay que determinar es si el trabajador trabajó horas extras o no. Encontrar las horas extras de la siguiente forma:
Horas extras = horas trabajadas - 40
En caso que sí trabajó horas extras:
Si horas extras > 8 entonces a horas extras excedentes de 8 = horas extras -8 y pago por horas extras = pago por hora normal * 2 * 8 + pago por hora normal * 3 * horas extras excedentes de 8
De otra forma (solo horas al doble) pago por horas extras = pago por hora normal * 2 * horas extras.
Finalmente, pago total que recibirá el trabajador será:
Pago = pago * hora normal * 40 + pago por horas extras.
Si no trabajó horas extras tendremos:
Pago = pago por hora normal * horas trabajadas.
Datos de salida: Pago.
Datos de entrada: número de horas trabajadas y pago por hora normal.
Definición de variables:
ht = horas trabajadas het = horas extras que exceden de 8
ph = pago por hora normal phe = pago por horas extras
he = horas extras pt = pago que recibe el trabajador

Algoritmo:

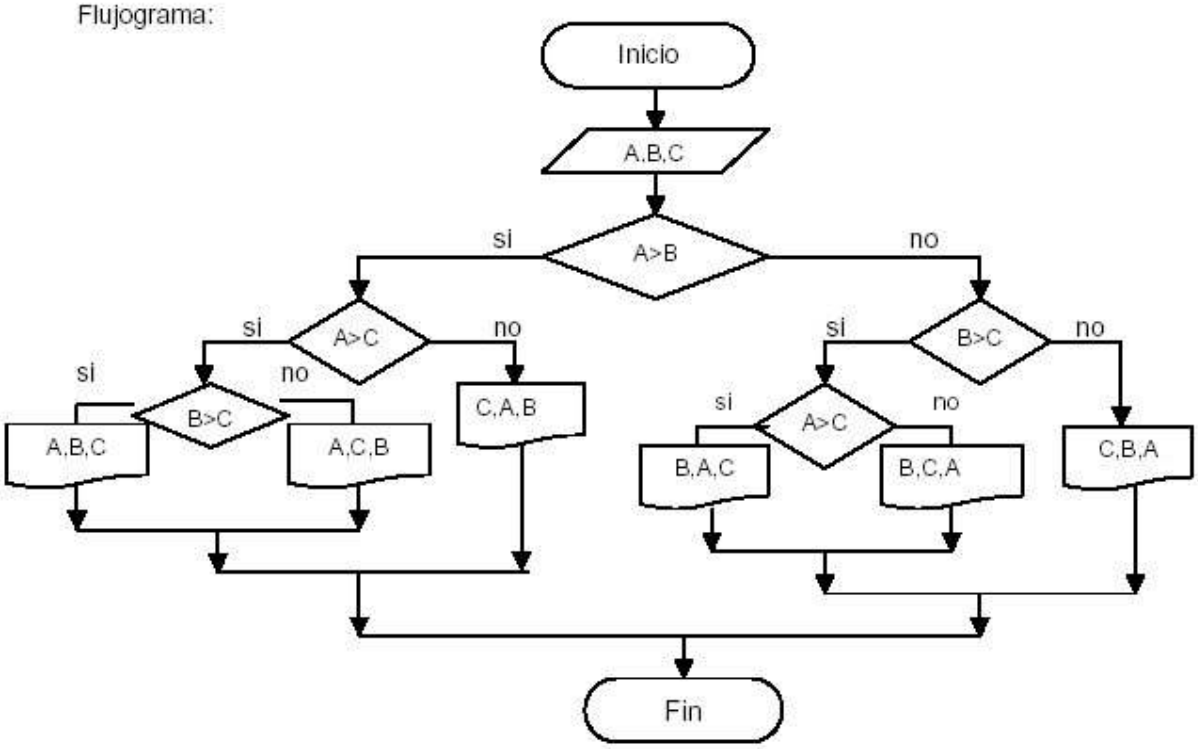
Inicio
Leer (ht, ph)
Si ht >40 entonces
He ¬ ht - 40
Si he > 8 entonces
Het ¬he - 8
Phe ¬ph * 2 * 8 + ph * 3 * het
Sino
Phe ¬ ph * 2 * he
Fin_si
Pt ¬ph * 40 + phe
Sino
Pt ¬ ph * ht
Fin_si
Escribir (pt)
Fin

Ejemplo 2.

Dados los datos A, B y C que representan números enteros diferentes, construir un algoritmo para escribir estos números en forma descendente. Este es un ejemplo de los algoritmos conocidos como de Lógica Pura, ya que poseen muchas decisiones y muchas bifurcaciones.

Salida: A, B y C ordenados descendentemente.
Entradas: A, B y C.
La dinámica del problema es comparar dos números a la vez para conocer cuál es el mayor.

Flujograma:

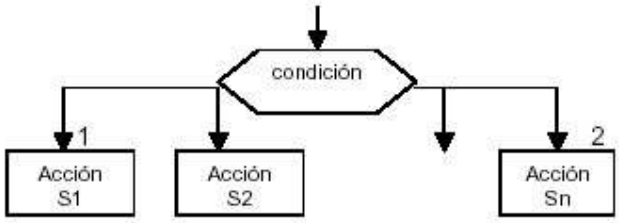


Estructura selectiva múltiple.

Con frecuencia es necesario que existan más de dos elecciones posibles. Este problema se podría resolver por estructuras selectivas simples o dobles, anidadas o en cascada, pero si el número de alternativas es grande puede plantear serios problemas de escritura y de legibilidad.

Usando la estructura de decisión múltiple se evaluará una expresión que podrá tomar n valores distintos, 1, 2 , 3, ...,n y según que elija uno de estos valores en la condición, se realizará una de las n acciones o lo que es igual, el flujo del algoritmo seguirá sólo un determinado camino entre los n posibles.

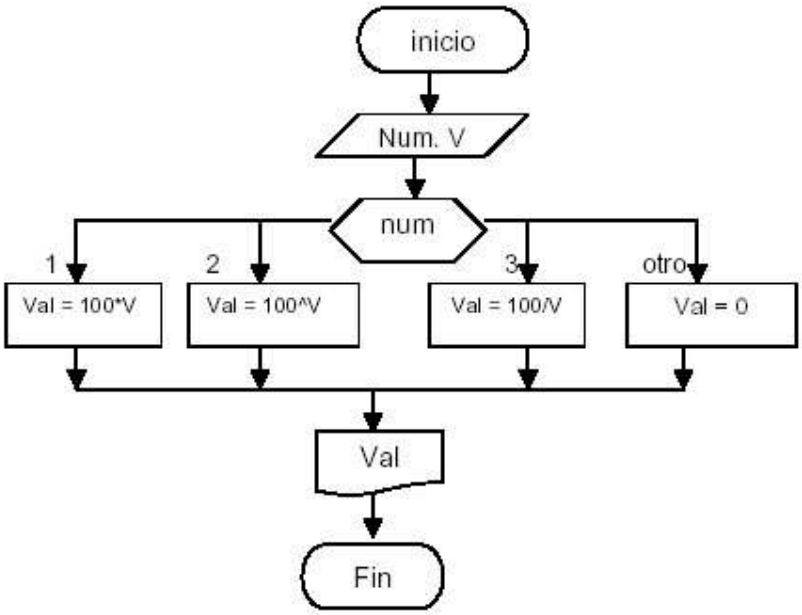
Esta estructura se representa por un selector el cual si toma el valor 1 ejecutará la acción 1, si toma el valor 2 ejecutará la acción 2, si toma el valor N realizará la acción N.



Ejemplo 1:

Diseñar un algoritmo tal que dados como datos dos variables de tipo entero, obtenga el resultado de la siguiente función:

$$Val = \begin{cases} 100 * V & \text{si Num = 1} \\ 100 \wedge V & \text{si num = 2} \\ 100 / V & \text{si num = 3} \\ 0 & \text{cualquier otro valor de num} \end{cases}$$




```
Inicio
Leer (num,V)
En caso que Num sea
        1:  hacer val = 100*V
        2: hacer val = 100 ΛV
        3: hacer val = 100/V
De otra forma:
        Val = 0
Fin_caso_que
Escribir (val)
Fin
```

Ejemplo 2.
Dados como datos la categoría y el sueldo de un trabajador, calcule el aumento correspondiente teniendo en cuenta la siguiente tabla. Imprimir la categoría del trabajador y el nuevo sueldo.

INCREMENTOS	
CATEGORÍA	AUMENTO
1	15%
2	10%
3	8%
4	7%

Definición de variables:

Cate = categoría
Sue = sueldo
Nsue = nuevo sueldo

ALGORITMO

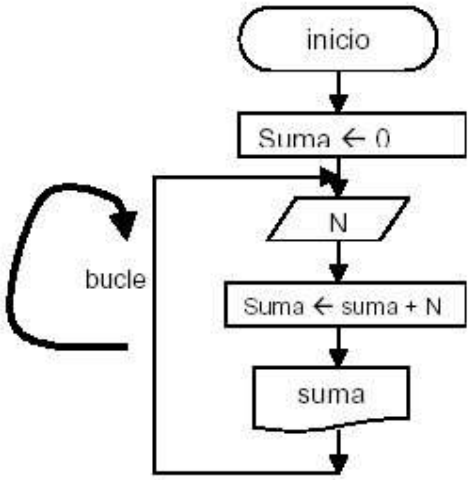
Inicio
Leer (cate, sue)
En caso que cate sea
1: hacer nsue <-- sue * 1.15
2: hacer nsue <-- sue * 1.10
3: hacer nsue <-- sue * 1.08
4: hacer nsue <-- sue * 1.07
Fin_caso_que
Escribir (cate, nsue)
Fin

Estructuras repetitivas e iterativas.

Son operaciones que se deben ejecutar un número repetido de veces. El conjunto de instrucciones que se ejecuta repetidamente cierto número de veces, se llama Ciclo, Bucle o Lazo.
Iteración es cada una de las diferentes pasadas o ejecuciones de todas las instrucciones contenidas en el bucle.

Fases de un Programa Cíclico :
1. Entrada de datos e instrucciones previas
2. Lazo o bucle
3. Instrucciones finales o resto del proceso
4. Salida de resultado

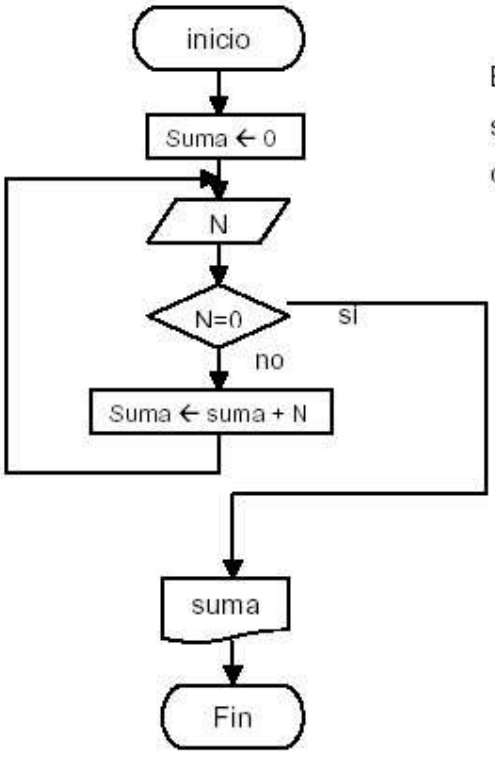
Ejemplo de bucle infinito:



En este flujograma, el bucle se estará repitiendo indefinidamente ya que no existe ninguna condición que nos permita finalizar en algún momento.

En el flujograma anterior, observa que la flecha que se regresa hacia arriba nos está indicando que hay que volver a evaluar la expresión. En ese caso como el bucle es infinito, no se tiene una condición para terminar y se estará haciendo siempre. En el siguiente ejemplo, ya se agregó una condición, la cual nos permitirá finalizar la ejecución del bucle en el caso en que la condición se cumpla.

Ejemplo de bucle finito:



En este ejemplo, el bucle finalizará cuando se cumpla la condición de que N sea igual a cero.

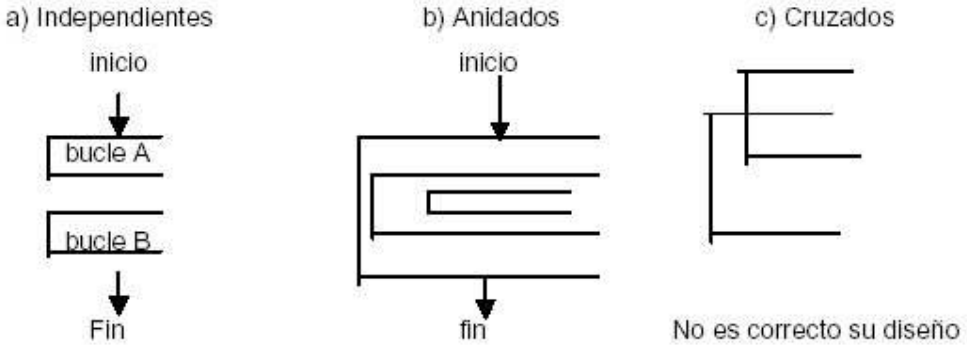
Bucles Repetitivos:

A continuación, te muestro tres diseños de estructuras cíclicas: las independientes son cuando los bucles se realiza uno primero hasta que se cumple la condición y solo en ese caso se entra al bucle B.

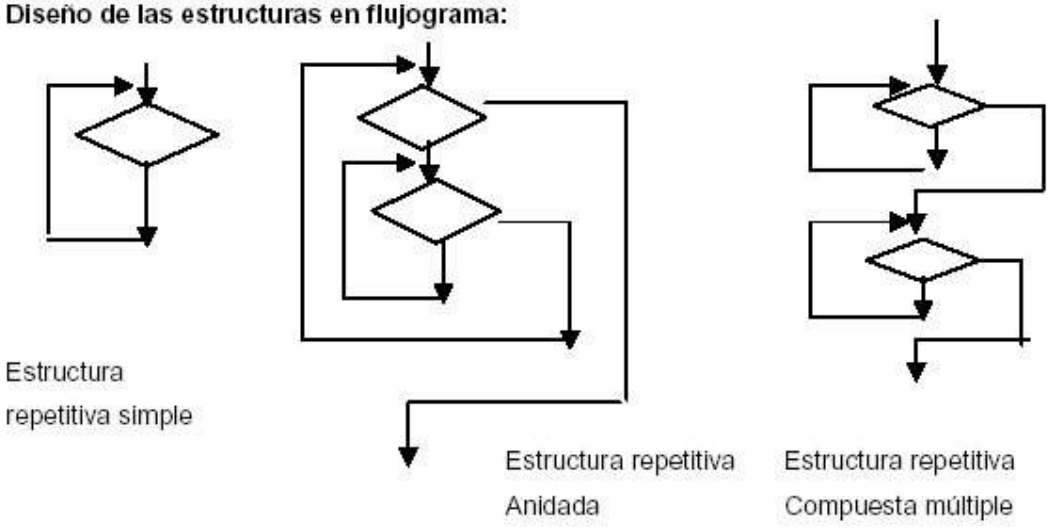
En los ciclos anidados, al entrar a una estructura de repetición, dentro de ella se encuentra otra. La más interna se termina de realizar y se continúa con la externa hasta que la condición se cumple.

En los bucles cruzados, los cuales no son convenientes de utilizar, se tiene que iniciamos un bucle y no se ha terminado cuando empezamos otro, luego utilizamos estructuras goto (saltos) para pasar al bucle externo y se quedan entrelazados.

Esto puede ocasionar que el programa pierda el control de cuál proceso se está ejecutando y podamos obtener resultados erróneos. Veamos gráficamente el diseño de estas tres formas cíclicas:



Diseño de las estructuras en flujograma:



Estructuras básicas.

Durante las siguientes lecciones estaremos estudiando tres estructuras básicas que son:

Estructura Desde/Para
Estructura Mientras
Estructura Repetir

En esta lección estudiaremos la forma general de la estructura Desde/Para, su uso y ejemplos.

Estructura Desde/Para:
Se usa frecuentemente cuando se conoce de antemano el número de veces que se ejecutarán las acciones de un bucle. Esta es una de sus características.

Representación pseudocodificada:

Español	**Inglés**
Desde var = valor inicial hasta valor final hacer Acciones Fin_desde	For var=valor inicial to valor final do acciones end_for

A la estructura Desde/Para se le conoce como Repetitiva. Para utilizar esta estructura en algoritmos, debemos hacer uso de contadores y algunas veces de acumuladores, cuyos conceptos se describen a continuación:

CONTADOR:
Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante cada vez que se produce un determinado suceso o acción. Los contadores se utilizan con la finalidad de contar sucesos o acciones internas de un bucle; deben realizar una operación de inicialización y posteriormente las sucesivas de incremento o decremento del mismo. La inicialización consiste en asignarle al contador un valor. Se situará antes y fuera del bucle.
Representación:
<nombre del contador> ¬ nombre del contador> + <valor constante>
Si en vez de incremento es decremento se coloca un menos en lugar del más.
Ejemplo: i = i + 1

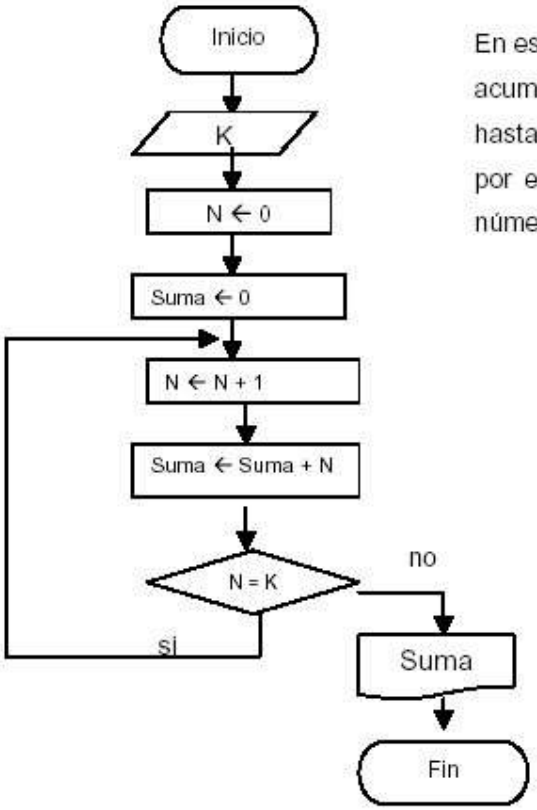
ACUMULADOR O TOTALIZADOR **:**
Es una variable que suma sobre sí misma un conjunto de valores para de esta manera tener la suma de todos ellos en una sola variable. La diferencia entre un contador y un acumulador es que mientras el primero va aumentando de uno en uno, el acumulador va aumentando en una cantidad variable.
Representación: <Nombre del acumulador> ¬ <nombre del acumulador> + <valor variable>

Ejemplo:
Calcular la suma de los cuadrados de los primeros 100 enteros y escribir el resultado. Se desea resolver el problema usando estructura Desde, Mientras y luego Repetir.

1. Usando la estructura Desde:

```
Inicio
        Suma ← 0
        Desde I = 1 hasta 100 hacer
                Suma ← suma + I * I
        Fin_desde
        Escribir (suma)
Fin
```

Ejemplo 2.
Elaborar un flujograma para encontrar la suma de los K primeros números enteros.



En este caso, utilizamos un contador (N) y un acumulador (suma). N cuenta cada número hasta llegar a K que es el valor máximo dado por el problema. Suma lleva la suma de los números enteros que se van generando

En este ejemplo hemos utilizado un bucle repetir, el cual estudiaremos en otra lección. Lo que queremos hacer notar por el momento, es cómo funcionan el contador y el acumulador. Nota que N es el contador, el cual se inicializa en este caso, con cero, antes de entrar al bucle. Dentro del bucle podrás notar que N se incrementa en 1.

También observa la variable suma, la cual es un acumulador que lleva la suma de los números generados. También debe inicializarse con cero, ya que para sumar valores debemos partir de cero, es decir, que al inicio no tenemos nada. Dentro del bucle, suma se incrementa en un número N, pero la diferencia con el contador N, es que a suma le sumamos N más ella misma.

EJERCICIO:

Trata de elaborar un flujograma para encontrar el cuadrado de los primeros 25 números naturales, usando la estructura Desde/Para.

¿Qué necesitas para resolver el problema contadores o acumuladores? Modifica el flujograma del ejercicio anterior para que también te muestre la suma de dichos cuadrados.
¿Qué necesitas agregar ahora?

RESUMEN

En esta lección aprendimos un poco del uso de contadores y acumuladores. También aprendimos a elaborar flujogramas o algoritmos usando la estructura Desde. Hay un número importante de reglas que deben seguirse cuando se utilizan instrucciones

Desde:

Los valores inicial y final de la variable de control se determinan antes de que empiece la repetición y no pueden cambiarse durante la ejecución de la instrucción Desde. Dentro del cuerpo del bucle Desde, los valores de las variables que especifican los valores inicial y final pueden cambiar, pero esto no va a afectar al número de repeticiones. La instrucción del cuerpo del bucle de una instrucción Desde puede utilizar el valor de la variable de control, pero no debe modificar este valor. Esta estructura se puede usar únicamente en aquellos casos en que conocemos el número de veces que se va a realizar el ciclo.

Esta estructura hace el incremento automáticamente y se inicializa en la instrucción desde.

Estructuras iterativas. Estructura mientras.

Se llama Mientras a la estructura algorítmica que se ejecuta mientras la condición evaluada resulte verdadera. Se evalúa la expresión booleana y, si es cierta, se ejecuta la instrucción especificada, llamada el cuerpo del bucle. Entonces se vuelve a evaluar la expresión booleana, y si todavía es cierta se ejecuta de nuevo el cuerpo. Este proceso de evaluación de la expresión booleana y ejecución del cuerpo se repite mientras la expresión sea cierta.

Cuando se hace falsa, finaliza la repetición. En la lección anterior iniciamos con las estructuras repetitivas. La estructura While y la estructura Repeat, se conocen como Iterativas. Se usan cuando no se conoce con anticipación el número de veces que se ejecutará la acción.

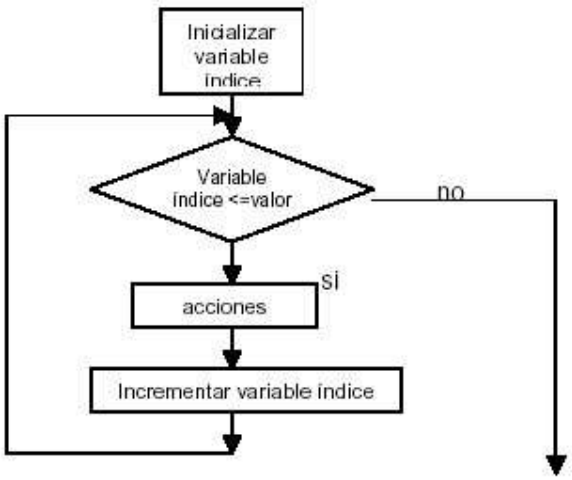
La diferencia entre ambas es que la condición se sitúa al principio (Mientras) o al final (Repetir) de la secuencia de instrucciones. Entonces, en el primero, el bucle continúa mientras la condición es verdadera (la cual se comprueba antes de ejecutar la acción) y en el segundo, el bucle continúa hasta que la condición se hace verdadera (la condición se comprueba después de ejecutar la acción, es decir, se ejecutará al menos una vez).

La estructura Desde/Para suele utilizarse cuando se conoce con anterioridad el número de veces que se ejecutará la acción y se le conoce como Estructura Repetitiva en lugar de iterativa, para diferenciarla de las dos anteriores.

Las estructuras Mientras y Para/Desde suelen en ciertos casos, no realizar ninguna iteración en el bucle, mientras que Repetir ejecutará el bucle al menos una vez.

Existe otro caso de estructura conocida como Salto (Goto), la cual no es muy recomendable de usar ya que su uso dificulta la legibilidad de un programa y tiende a confundir por el hecho de recurrir a numerosas etiquetas o números de línea.

Representación gráfica:



Observa en el flujograma, que se necesita una variable contadora (un índice), para llevar la cuenta de las veces que entramos al cuerpo del ciclo. También es importante notar que esta variable se inicializa antes de entrar al cuerpo del ciclo y dentro del cuerpo se incrementa en una cantidad constante, por lo general en uno.

Esta variable a la vez, nos sirve para compararla con el valor dado en la condición, cuando se cumple la condición, se sale del ciclo.

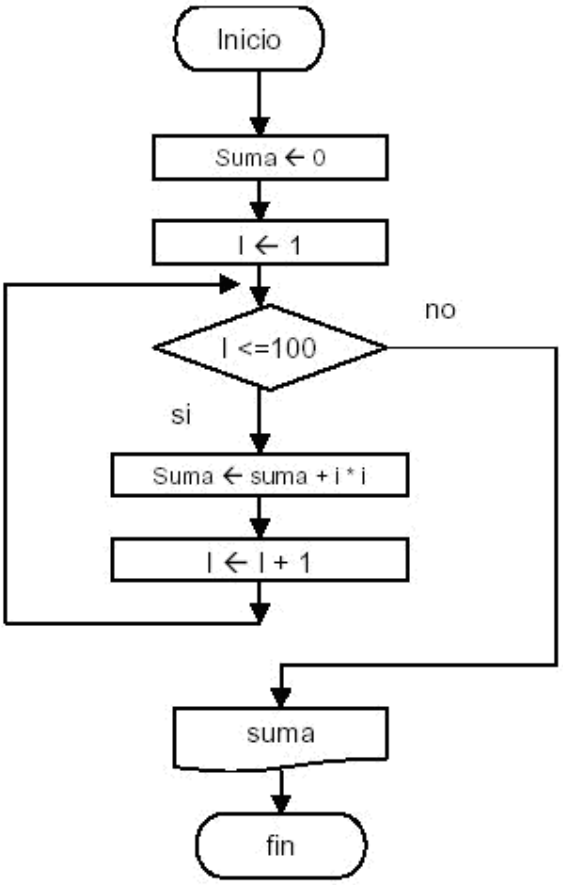
Representación pseudocodificada:

Español	Inglés
Mientras <condición>	While <condición> do
Acciones	Acciones
Fin_mientras	end_while

EJEMPLO:
Calcular la suma de los cuadrados de los primeros 100 números enteros y escribir el resultado.
Solución.
Como recordarás, resolvimos este ejercicio en la lección anterior pero utilizando la estructura Desde. Hoy lo haremos con la estructura Mientras.Que tendremos de diferente?

Hagamos el algoritmo:

```
Inicio
  Suma ← 0
  I ← 1
  Mientras I < = 100 hacer
              Suma ← suma + I * I
              I ← I + 1
  Fin_mientras
  Escribir (suma)
Fin
```



CENTINELAS Y BANDERAS.
Cuando no se conoce a priori el número de iteraciones que se van a realizar, el ciclo puede ser controlado por centinelas.

CENTINELAS.
En un ciclo While controlado por tarea, la condición de While especifica que el cuerpo del ciclo debe continuar ejecutándose mientras la tarea no haya sido completada.

En un ciclo controlado por centinela el usuario puede suspender la introducción de datos cuando lo desee, introduciendo una señal adecuada llamada centinela. Un ciclo Repetir controlado por centinela es cuando el usuario digita una letra para salir como por ejemplo S o N para indicar si desea continuar o no. El bucle debe repetirse hasta que la respuesta del usuario sea "n" o "N".

Cuando una decisión toma los valores de -1 o algún posible valor que no esté dentro del rango válido en un momento determinado, se le denomina centinela y su función primordial es detener el proceso de entrada de datos en una corrida de programa.

Por ejemplo, si se tienen las calificaciones de un test (comprendida entre 0 y 100); un valor centinela en esta lista puede ser -999, ya que nunca será una calificación válida y cuando aparezca este valor se terminará de ejecutar el bucle.

Si la lista de datos son números positivos, un valor centinela puede ser un número negativo. Los centinelas solamente pueden usarse con las estructuras Mientras y Repetir, no con estructuras Desde/Para. ¿PODRÍAS DECIR POR QUÉ?

Ejemplo:

Suponga que debemos obtener la suma de los gastos que hicimos en nuestro último viaje, pero no sabemos exactamente cuántos fueron.
Si definimos gasto1, gasto2, gasto3, ..., -1 donde gastoi: real es el gasto número i y sumgas: real es el acumulador de gastos efectuados. -1 es el centinela de fin de datos.

Algoritmo:

```
Inicio
Sumgas . 0
Leer (gasto)
Mientras gasto <> -1 hacer
Sumgas . sumgas + gasto
Leer (gasto)
Fin_mientras
Escribir (sumgas)
Fin
```

BANDERAS.
Conocidas también como interruptores, switch, flags o conmutadores, son variables que pueden tomar solamente dos valores durante la ejecución del programa, los cuales pueden ser 0 ó 1, o bien los valores booleanos True o False. Se les suele llamar interruptores porque cuando toman los valores 0 ó 1 están simulando un interruptor abierto/cerrado o encendido/apagado.

Ejemplo 1:
Leer un número entero N y calcular el resultado de la siguiente serie: 1 - 1/2+ 1/3 - 1/4 +.... +/- 1/N.
Algoritmo:

```
Inicio
Serie . 0
I . 1
Leer (N)
Band . "T"
Mientras I <= N hacer
Si band = "T" entonces
Serie . serie + (1/I)
Band . "F"
Sino
Serie . serie - (1/I)
Band . "T"
Fin_si
I ¬ I + 1
Fin_mientras
Escribir (serie)
Fin
```

Ejemplo 2.
Obtener suma de los términos de la serie: 2, 5, 7, 10, 12, 15, 17, ... 1800.
Sumser de tipo entero, es el acumulador de términos de la serie

Band de tipo carácter, es variable auxiliar que indica si al siguiente término de la serie hay que sumarle 3 ó 2.

Algoritmo:

```
Inicio
I ¬2
Sumser ¬0
 Band ¬T"
Mientras (I <= 1800) hacer
Sumser ¬sumser + I
Escribir (I)
Si band = "T" entonces
I ¬I + 3
Band ¬F"
Sino
I ¬I + 2
Band ¬T"
Fin_si
Fin_mientras
Escribir (sumser)
Fin
```

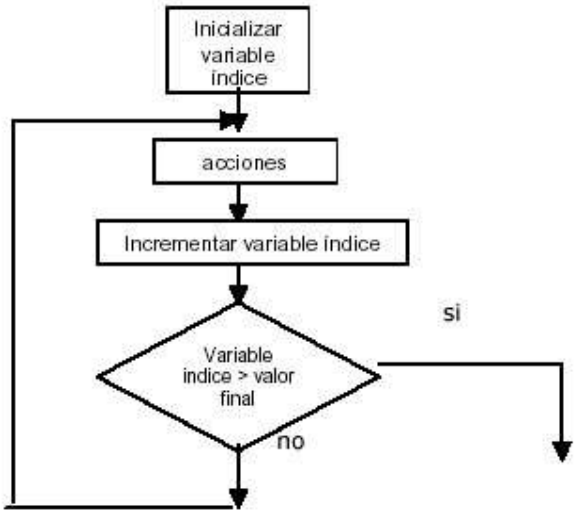
RESUMEN

Hemos estudiado en esta lección que la estructura Mientras tiene una pequeña variante a la estructura Desde en cuanto a la representación algorítmica. Recuerda que la estructura Desde, se inicializa automáticamente en su sintaxis y el incremento también es automático. En cambio, la estructura Mientras usa un contador que es inicializado antes de entrar al ciclo y dentro del ciclo es incrementado. También estudiamos que los centinelas son valores que le damos a la condición para forzar a que un ciclo pueda terminar. También decíamos que los centinelas solamente los podemos usar en estructuras Mientras y Repetir, ya que sirven para finalizar el ciclo cuando no sabemos las veces que lo vamos a realizar, y la estructura Desde es usada cuando ya conocemos a priori el número de veces que se va a realizar el ciclo.

Estructuras iterativas. Estructura repetir.

Se llama Repetir a la estructura algorítmica que se ejecuta un número definido de veces hasta que la condición se torna verdadera:

Representación gráfica:



Representación pseudocodificada :

Español	**Inglés**
Repetir	Repeat
Acciones	Acciones
Hasta que <condición>	until <condición>

EJEMPLO:
Calcular la suma de los cuadrados de los primeros 100 números enteros y escribir el resultado.

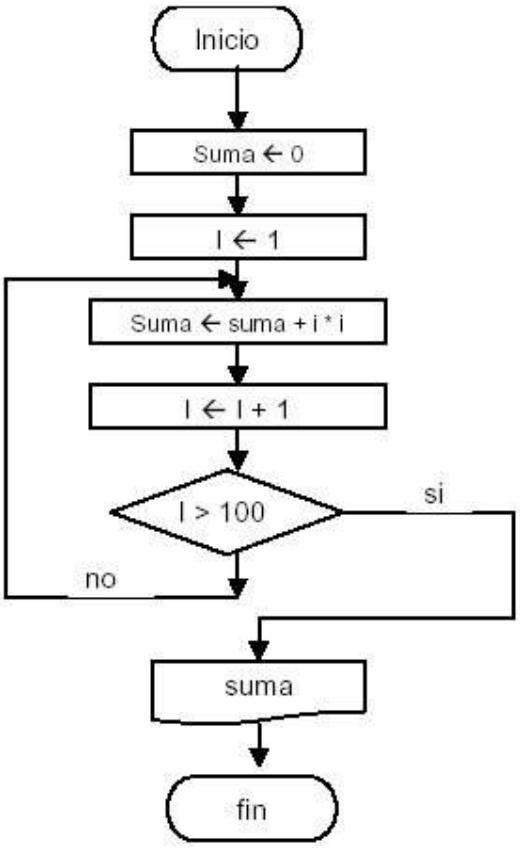
Solución.

Nuevamente resolveremos el ejercicio de las dos lecciones anteriores, ahora utilizando la estructura Repetir.¿Podrás decir cuál será ahora la diferencia? Las reglas para construcción de esta estructura usando Repetir, nos dicen que debemos declarar una variable contador que debe inicializarse antes del ciclo e incrementarse dentro del ciclo. A diferencia de la estructura Mientras, la condición ahora estará colocada al final del bucle para que primero ejecutamos la instrucción y luego preguntamos si la condición se cumple. Esto quiere decir, que en esta estructura el bucle se realizará por lo menos una vez. También podrás observar que la condición está al revés, porque el bucle se repite hasta que la condición se cumpla. En el bucle Mientras, la condición se evaluaba mientras era cierta.

Hagamos el algoritmo:

```
Inicio
  Suma <-- 0
  I <-- 1
  Repetir
     Suma <-- suma + I * I
     I ← I + 1
  Hasta que I > 100
  Escribir (suma)
Fin
```

Flujograma:



BIBLIOGRAFÍA

" Fundamentos de Informática
Luis A. Ureña y otros. 1999 Editorial Alfaomega ra-ma

" Fundamentos de Programación. Libro de Problemas
Luis Joyanes Aguilar, Rodríguez Baena y Fernández Azuela. 1996

Editorial Mc Graw Hill 2ª edición.
" Fundamentos de Programación. Algoritmos y estructuras de datos. 2ª Edición

Luis Joyanes Aguilar 1996. Editorial Mc Graw Hill
" Pascal y Turbo Pascal enfoque práctico

Luis Joyanes Aguilar y Angel Hermoso. Editorial Mc Graw Hill
" Lógica Simbólica.

Irving M. Copi. Editorial C.E.C.S.A.
" Programación en Pascal. 4ª Edición

Sanford Leestma y Larry Nyhoff. Editorial Prentice Hall 1999
" Cómo programar en C/C++

Deitel/Deitel. Editorial Prentice Hall
" Introducción a la Ciencia de las Computadoras.

Tremblay, Jean Paul. Edit. Mc Graw Hill 1985
" Introducción a la Computación para Ingenieros

Steven C. Chapra y Raymond P. Canale
Edit. Mc Graw Hill 1993

" Curso General de Informática.
Javier Gayan y Dolors Segarra. Edit. Gustavo Gili S.A. 1988

" Aprenda Visual C++ Ya.
Mark Andrews. Editorial Mc Graw Hill

" Visual C++ 6.0 Manual de Referencia.
Chris Pappas y William H. Murray. Editorial Mc Graw Hill

" Computación. Metodología, Lógica Computacional y Programación.
Ma. del Rosario Bores Rangel y Román Rosales Becerril. 1ª Edición Editorial Mc Graw Hill 1993

" Programación en C. 1ª Edición
Byron S. Gottfried. Editorial Mc Graw Hill 1991

" Metodología de la Programación. Algoritmos, Diagramas de Flujo y programas
Tomos I y II. Osvaldo Cairó battistutti. 1a Edición
Editorial Alfaomega 1995

" Enciclopedia del Lenguaje C. 1ª Edición
Francisco Javier Ceballos. Editorial Alfaomega ra-ma 1999

DIRECCIONES DE INTERNET:

http://mundovb.net/mundoc/
http://lawebdelprogramador.com/cursos/