

Tema 1.- ALGORITMOS Y PROGRAMAS. ELEMENTOS DE UN PROGRAMA.

- 1.1. Sistemas de procesamiento de la información
 - 1.2. Algoritmos: Características y diseño
 - 1.3. Ciclo de vida de una aplicación
 - 1.4. Errores
 - 1.5. Documentación de los programas
 - 1.6. Elementos básicos de un programa
 - 1.7. Estructura general de un programa
 - 1.8. Escritura de algoritmos
-

1.1.- SISTEMAS DE PROCESAMIENTO DE INFORMACIÓN

La principal razón para que las personas aprendan lenguajes y técnicas de programación es *utilizar el ordenador como una herramienta para resolver problemas*.

Sistema de procesamiento de información es el que transforma datos brutos en información organizada, significativa y útil.

En el lenguaje diario, datos e información son esencialmente sinónimos, sin embargo, los informáticos cuando hablan de datos se refieren a la representación de algún hecho, concepto o entidad real, mientras que la información implica datos procesados y organizados.

1.2.-ALGORITMOS: CARACTERÍSTICAS Y DISEÑO

Definimos algoritmo como el conjunto de instrucciones que especifican la secuencia de operaciones a realizar en orden, para resolver un determinado problema.

La resolución de un problema exige:

- ◆ Definición y análisis del problema
- ◆ Diseño del algoritmo
- ◆ Transformación del algoritmo en un programa
- ◆ Ejecución y validación del problema

Entrada = datos

Procesador

Salida = información

Cuando el procesador es un ordenador, el algoritmo ha de expresarse en forma de programa. Un programa se escribe en un lenguaje de programación y a la actividad de expresar un algoritmo en forma de programa se le denomina programación.

Al conjunto de métodos y técnicas que ayudan a que el desarrollo de los programas cumpla estos requisitos, se denomina Metodología de la Programación. Las técnicas que vamos a estudiar son:

- Programación modular. Se basa en la realización de una serie de descomposiciones sucesivas del algoritmo principal. El programa quedará formado por una serie de módulos, cada uno de los cuales realiza una parte de la tarea total.
- Programación estructurada. Se basa en el uso exclusivo de las estructuras secuencial, alternativa y repetitiva, para el control del flujo de ejecución de las instrucciones. Los programas así diseñados serán fáciles de verificar, depurar y mantener.
- Programación orientada a objetos. Es una evolución lógica de la programación estructurada, en la que el concepto de variable local se hace extensible a los propios subprogramas que acceden a estas variables.

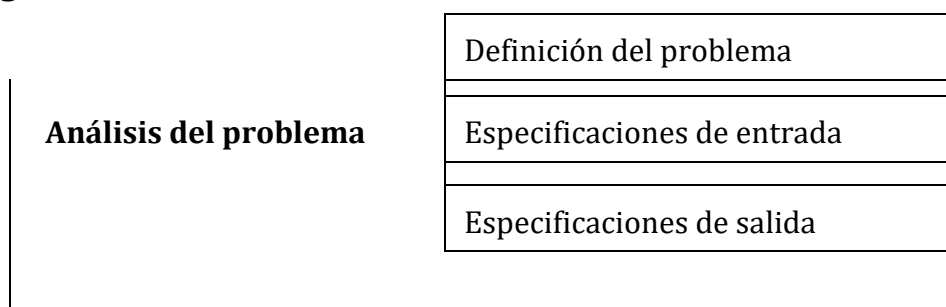
Cada paso en el algoritmo está expresado por medio de una instrucción en el programa, por lo tanto, un programa consta de una secuencia de instrucciones, cada una de las cuales especifica las operaciones que debe realizar el ordenador.

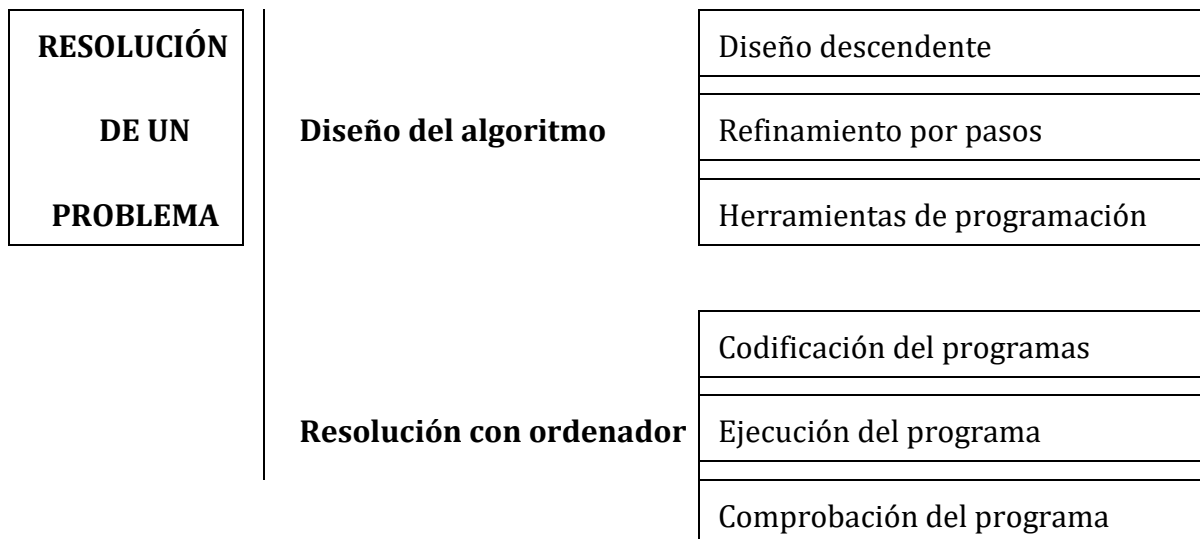
Características de un algoritmo:

- Debe ser preciso e indicar el orden de realización de cada paso.
- Estar definido. Si se sigue un algoritmo dos veces se debe obtener el mismo resultado cada vez.
- Debe ser finito. Tener un número finito de pasos.

La definición de un algoritmo debe describir tres partes: Entrada, Proceso y Salida.

Diseño de algoritmos:










Herramientas de programación. Para representar un algoritmo debemos utilizar un método que permita independizar dicho algoritmo del lenguaje de programación elegido, permitiendo que pueda ser codificado en cualquier lenguaje.

Los métodos para representar un algoritmo son : Diagrama de flujo y Pseudocódigo

Diagrama de flujo.- Un diagrama de flujo (Flowchart) fue una de las técnicas de representación de algoritmos más utilizadas, aunque su empleo disminuyó considerablemente con los lenguajes de programación estructurados. Un diagrama de flujo utiliza símbolos estándar que contienen los pasos del algoritmo escritos en esos símbolos, unidos por flechas denominadas líneas de flujo que indican la secuencia en que deben ejecutarse.

Los símbolos más utilizados son :

	Proceso :cualquier tipo de operación que pueda originar cambio de valor, formato, operaciones aritméticas etc.		Entrada / Salida de información. Este símbolo se puede subdividir en otros de teclado, pantalla, impresora disco etc.
	Decisión : indica operaciones lógicas o de comparación entre datos y en función del resultado determina el camino a seguir.		Dirección del flujo (Indica el sentido de ejecución de las operaciones).
	Terminal : representa el comienza y el final de un programa.		

Pseudocódigo

El pseudocódigo es un lenguaje de descripción de algoritmos, que hace más fácil el paso final de codificación. Nace como medio de representar las estructuras de control de programación estructurada. El pseudocódigo no se puede ejecutar en el ordenador, sino que tiene que traducirse a un lenguaje de programación.

La ventaja del pseudocódigo es que se puede modificar fácilmente si detecta un error en la lógica del programa, y puede ser traducido fácilmente a los lenguajes estructurados.

El Pseudocódigo utiliza para representar las acciones sucesivas, palabras reservadas, - antes en inglés -, y exige la indentación -sangría en el margen izquierdo- de diferentes líneas.

Las líneas precedidas por // se denominan comentarios, es decir una información al lector del algoritmo que no realiza ninguna instrucción ejecutable.

Palabras reservadas del pseudocódigo:

leer, escribir,
si <condición> entonces <acción-1>
si_no <acción2>
fin_si

mientras <condición >	repetir	para valor-inicial hasta valor-final
<acción>	<acción>	<acción>
fin_mientras	hasta_que <condición>	fin_para, etc.

1.3 .- CICLO DE VIDA DE UNA APLICACIÓN INFORMÁTICA .

Una aplicación es un conjunto de uno o más programas interrelacionados que tienen por objeto la realización de una determinada tarea mediante el uso de un sistema informático.

ciclo de vida de una aplicación informática Es el proceso que se sigue desde el planteamiento de un problema o tarea, hasta que se tiene una solución instalada en el ordenador y en funcionamiento por el usuario, mientras esta sea de utilidad.

Se compone de varias fases que agrupamos en dos bloques.

Fase de Diseño	Análisis	Especificaciones
	Programación	Algoritmo
	Codificación	Programa
Fase de Instalación	Edición	Programa Fuente
	Compilación	Programa objeto
	Montaje	Programa ejecutable
	Prueba de ejecución	Aplicación
	Explotación y Mantenimiento	

1.4.- ERRORES. Según el momento en que se detectan pueden ser:

- Errores de compilación - > sintaxis
- Errores de ejecución - > operaciones no permitidas
- Errores de lógica -> resultados no correctos, los más difíciles de corregir.
- Errores de especificación -> especificaciones incorrectas, son los más costosos

CALIDAD DE LOS PROGRAMAS.

Para un determinado problema se pueden construir diferentes algoritmos de resolución. La elección del más adecuado se basa en una serie de requisitos de calidad que adquieren importancia a la hora de evaluar el coste de su diseño y mantenimiento.

Las características generales que debe cumplir un programa son:

Legibilidad.- Ha de ser claro y sencillo de tal forma que facilite su lectura y comprensión

Fiabilidad.- Ha de ser robusto, es decir capaz de recuperarse frente a errores o usos inadecuados.

Portabilidad.- Su diseño debe permitir la codificación en diferentes lenguajes de programación.

Modificabilidad.- Ha de facilitar su mantenimiento, es decir, poder hacer las modificaciones y actualizaciones necesarias para adaptarlo a una nueva situación.

El conjunto de métodos y técnicas que ayudan al desarrollo de programas que cumplan estos requisitos se denomina *Metodología de la Programación*.

Los métodos que vamos a estudiar son: La programación modular y programación estructurada.

Programación modular. Se basa en la realización de una serie de descomposiciones sucesivas del algoritmo principal. El programa quedará formado por una serie de **módulos**, cada uno de los cuales realiza una parte de la tarea total.

Programación estructurada. Se basa en el uso exclusivo de las estructuras **secuencial, alternativa y repetitiva**, para el control del flujo de ejecución de las instrucciones. Los programas así diseñados serán fáciles de verificar, depurar y mantener..

1.5.- DOCUMENTACIÓN DE LOS PROGRAMAS.

Con el fin de facilitar la explotación y el mantenimiento de un programa es fundamental que este se acompañe de una documentación clara, amplia y precisa. Existen dos clases de documentación según su ubicación:

Documentación interna:

Comentarios. Son frases explicativas que se insertan en cualquier lugar del programa fuente y son ignoradas por el compilador

Código autodocumentado.- Las palabras reservadas de los lenguajes constituyen en si mismas parte de la documentación.

Se puede mejorar la documentación de un programa :

- Utilizando identificadores adecuados para nombrar variables, constantes, subprogramas etc.
- Declarando constantes para valores fijos.
- Sangrando, paginando e intercalando líneas en blanco para facilitar la lectura del programa.

Documentación externa. Es el conjunto de documentos que acompañan al programa sin formar parte de el.

Debe incluir al menos:

- Especificaciones del análisis
- Descripción del diseño del programa
- Descripción del programa principal y subprogramas
- Manual de usuario
- Manual de mantenimiento

1.6.- ELEMENTOS BÁSICOS DE UN PROGRAMA .

- **OBJETOS DE UN PROGRAMA.** Son objetos de un programa todos aquellos manipulados por las instrucciones.

Todo objeto tiene tres atributos:

- . **Nombre.-** Es el identificador del mismo
- . **Tipo.-** Conjunto de valores que puede tomar
- . **Valor.-** Elemento del tipo que se le asigna.

En programación se debe diferenciar entre el diseño del algoritmo y su implementación en un lenguaje específico. Por ello, se debe distinguir claramente entre los conceptos de programación y el medio en que ellos se implementan en un lenguaje específico. Sin embargo, una vez que se comprendan los conceptos de programación y como utilizarlos, aprender un nuevo lenguaje es relativamente fácil.

Los lenguajes de programación—como los restantes lenguajes—tienen elementos básicos que se utilizan como bloques constructivos, así como reglas para que esos elementos se combinen. Estas reglas se denominan *sintaxis* del lenguaje.

Solamente las instrucciones sintácticamente correctas, pueden ser interpretadas por la computadora, y los programas que contengan errores de sintaxis son rechazados por la máquina.

- **Los elementos básicos de un programa o algoritmo son:**

- ❑ *Palabras reservadas* (inicio, fin, si -entonces . . ., etc.),
- ❑ identificadores (nombres de variables esencialmente),
- ❑ *Caracteres especiales* (coma, apóstrofo, etc.),
- ❑ Constantes,
- ❑ *Variables*,
- ❑ Expresiones,
- ❑ Instrucciones.

Además de estos elementos básicos, existen otros elementos que forman parte de los programas, cuya comprensión y funcionamiento será vital para el correcto diseño de un algoritmo y naturalmente la codificación del programa.

- Estos elementos son:**
- *bucles*,
 - *contadores*,
 - *acumuladores*,
 - *interruptores*,
 - *estructuras: —secuenciales, —selectivas, —repetitivas.*

El amplio conocimiento de todos los elementos de programación y el modo de su integración en los programas constituyen las técnicas de programación .

Bucles. Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan bucles, y se llama iteración al hecho de repetir la ejecución de una secuencia de acciones. Las dos principales preguntas a realizarse en el diseño de un bucle son: ¿ que contiene el bucle? y ¿ cuantas veces se debe repetir?.

Para detener la ejecución de los bucles se utiliza una condición de parada. En pseudocódigo lo normal es que la condición se indique al final o al principio del bucle

, y así se consideran tres tipos de instrucciones repetitivas: **Mientras, Repetir, Para.**

Un bucle consta de tres partes:

- decisión.
- cuerpo del bucle,
- salida del bucle.

En un algoritmo pueden existir varios bucles, y estos pueden ser anidados o independientes. Son anidados cuando están dispuestos de tal manera que unos son interiores a otros. Son independientes cuando son externos unos a otros.

Contadores. Los procesos repetitivos son la base del uso de las computadoras. En estos procesos se necesitan normalmente contar los sucesos o acciones internas del bucle, como pueden ser, el numero de iteraciones a realizar por el bucle, la cantidad de entradas realizadas etc. Una forma de controlar un bucle es mediante un contador, que es una variable cuyo valor se incrementa o decrementa en una cantidad constante en cada iteración. Ejem. $C \leftarrow C + 1$

Acumulador. Un acumulador o totalizador es una variable cuya misión es almacenar cantidades variables resultantes de sumas sucesivas. Realiza la misma función que un contador con la diferencia de que el incremento o decremento de cada suma es variable en lugar de constante, como en el caso del contador. Se representa por la instrucción $S \leftarrow S + N$, donde N es una variable y no una constante.

Interruptores. Un interruptor (switch) es una variable que puede tomar diversos valores a lo largo de la ejecución del programa y que permite comunicar información de una parte a otra del mismo. Los interruptores pueden tomar dos valores diferentes 1 y 0. (De ahí su nombre de interruptor, encendido /apagado, abierto / cerrado)

◆ TIPOS DE DATOS SENCILLOS

✓ **Identificadores.** Son palabras creadas por el programador para dar nombre a los objetos y demás elementos que necesitamos declarar en un programa, como variables, estructuras de dato, archivos, subprogramas, etc. En general se utiliza una cadena de letras y dígitos que empiece por letra..

Un dato es la expresión general que describe los objetos con los cuales opera un ordenador. En el proceso de solución de problemas, el diseño de la estructura de datos es tan importante como el diseño del algoritmo y del programa.

Existen dos clases de tipos de datos: simples (sin estructura) y compuestos (estructurados).

Tipos de datos	Simple	Numéricos (entero, real).	
		Lógicos (boolean).	
		Carácter (char, string).	
	Estructuras de datos	Internas	Estáticas (Vectores, Matrices, tablas)
			Dinámicas (Listas, Pilas, Colas, Árboles)
		Externas	Ficheros
			Bases de Datos

- **Datos numéricos.** El tipo numérico es el conjunto de los valores numéricos. Se pueden representar de dos formas distintas:

- Tipo numérico entero. Es un subconjunto finito de los números enteros. No tienen componentes fraccionarios o decimales y pueden ser positivos o negativos. Se denominan en ocasiones números de coma fija.

- Tipo numérico real. Consiste en un subconjunto de los números reales. El número real consta de una parte entera y una parte decimal y pueden ser positivos o negativos

- **Datos lógicos (booleanos) .**El tipo lógico es aquel dato que sólo puede tomar uno de dos valores: cierto o verdadero (true) y falso (false). Este tipo se utiliza para representar las alternativas (si/no) a determinadas condiciones.

- **Datos tipo carácter y tipo cadena.** El tipo carácter es el conjunto finito y ordenado de caracteres que reconoce el ordenador, Un dato tipo carácter contiene un solo carácter.

Los caracteres que reconocen los diferentes ordenadores no son estándar, sin embargo la mayoría reconocen los siguientes caracteres:

- * Caracteres alfabéticos (A....Z) (a..z)

- * Caracteres numéricos (1,2.....0)
- * Caracteres especiales (+, -, *, /, ^, ., ;, >, <, \$,)

Una cadena (string) de caracteres es una sucesión de caracteres que se encuentran delimitados por una comilla o dobles comillas, según el lenguaje de programación.

- **Constantes y variables** . Los programas contienen ciertos valores que no deben cambiar durante la ejecución del programa, estos valores se llaman **constantas**.

Una constante tipo carácter o constante de caracteres consiste en un carácter valido encerrado entre apóstrofes, si se desea incluir el apóstrofo en la cadena, entonces debe aparecer como un par de apóstrofes encerrados entre comillas simples.

Constantas lógicas (boolean) sólo existen dos verdad y falso.

La mayoría de lenguajes de programación permiten diferentes tipos de constantes: enteras, reales, caracteres y boolean o lógicas.

Una variable es un objeto o partida de datos cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa. Hay de diferentes tipos enteras, reales, caracteres y boolean o lógicas y cadena. Una variable que es de un cierto tipo puede tomar únicamente valores de ese tipo

Una variable se identifica por los siguientes atributos: nombre y tipo que describe el uso de la variable. Los nombres a veces conocidos como identificadores suelen constar de varios caracteres alfanuméricos de los cuales el primero normalmente es una letra. Los nombres de las variables elegidas para el algoritmo deben ser *significativas y tener relación con el objeto que representan*.

- **Expresiones**. Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.

Una expresión consta de operandos y operadores, y según sea el tipo de objetos que manipulan, las expresiones se clasifican en :

- Aritméticas
- Lógicas
- Carácter

♦ **Expresiones aritméticas**. Son análogas a las fórmulas matemáticas. Las variables y constantes son numéricas (enteras o reales) y las operaciones son aritméticas.

operadores:		
+	Suma	Operadores div . El símbolo / se utiliza para la división real y el operador div en algunos
-	resta y cambio de	

*	signo producto	lenguajes representa la división entera. Ejemplo A div B Sólo se puede realizar si A y B son variables enteras. 19 div 6 = 3 19 mod 3 = 1 15 div 6 = 2 15 mod 6 = 3
/	cociente	
** ^	potencia	
div entera	división	
mod	módulo o resto	

- **Reglas de prioridad.** Las expresiones que tienen dos o más operandos requieren unas reglas matemáticas que determinen el orden de las operaciones, estas reglas de prioridad son las siguientes.

- 1.- Las operaciones encerradas entre paréntesis se evalúan primero. Si existen diferentes paréntesis anidados, las expresiones más internas se evalúan primero.
- 2.- Las operaciones aritméticas dentro de una expresión suelen seguir el siguiente orden:
 - operador exponencial (^ ↑ **)
 - producto y cociente * /
 - operadores div y mod
 - operador suma y resta

Cuando varios operadores de igual prioridad coinciden en una expresión, el orden de prioridad es de izquierda a derecha.

♦ **Expresiones lógicas.** Sólo pueden tomar dos valores verdadero y falso. Se forman combinando constantes lógicas, variables lógicas y otras expresiones lógicas, utilizando los operadores lógicos (not and or) y los operadores de relación.

Operadores lógicos. La definición de los operadores lógicos se resume en tablas conocidas como tablas de verdad:

A	no A	A	B	A y B	A	B	A o B
verdad	falso	verdad	verdad	verdad	verdad	verdad	verdad
falso	verdad	verdad	falso	falso	verdad	falso	verdad
		falso	verdad	falso	falso	verdad	verdad
		falso	falso	falso	falso	falso	falso

Operadores de relación. = < > <= >= <> . Los operadores de relación realizan comparaciones de valores de tipo numérico o carácter. Se utilizan para expresar las condiciones en los algoritmos, y el resultado de la operación será verdadero o falso. Se pueden aplicar a cualquiera de los cuatro tipos de datos, enteros, reales, lógicos y carácter.

Cuando se utilizan los operadores de relación con valores lógicos, la constante false (falso) es menor que la constante true (verdad). **Ejemplo**

A=4 B=3	A > B	verdad
	(A - 2) < (B - 4)	falso
	(5 > 10) o ('A' < 'B')	verdad
	(1 < 5) y (5 < 10)	verdad
N=5	(N=1) o (7 >= 4)	verdad

- ♦ **Expresiones de tipo carácter.** Utilizan como operandos variables o constantes tipo carácter o cadena, y el operador que utilizan es la concatenación, & o + .

Funciones internas . Las operaciones necesarias en los programas exigen en muchas ocasiones además de las operaciones aritméticas básicas, un determinado número de operadores especiales llamadas *funciones internas* ., por ejemplo, raíz cuadrada, redondeo, seno de x, logaritmo de x, cuadrado de x etc. Estas funciones utilizan argumentos reales o enteros y los resultados dependen de la tarea que realice la función.

Función	Descripción	Tipo argumento	Resultado
abs(x)	valor absoluto de x	entero o real	igual que el argumento
Redondeo(x)	Redondeo de x	real	entero
Cuadrado(x)	Cuadrado de x	entero o real	igual que el argumento
raiz2(x)	raíz cuadrada de x	entero o real	real
trunc(x)	Truncamiento de x	real	entero

1.7 ESTRUCTURA GENERAL DE UN PROGRAMA.

- **Concepto de programa.** Un *programa de ordenador* es un conjunto de instrucciones (órdenes dadas a la maquina) que producirán la ejecución de una determinada tarea. El desarrollo de un programa requiere el análisis y diseño previo del algoritmo.
 - **Instrucciones y tipos de instrucciones.** El proceso de diseño del algoritmo y posteriormente de codificación del programa consiste en definir las acciones o instrucciones que resolverán el problema. Un programa es *lineal* si las instrucciones se ejecutan secuencialmente, sin bifurcaciones, decisiones ni comparaciones. Un programa es no lineal cuando se interrumpe la secuencia mediante instrucciones de bifurcación.
 - **Tipos de instrucciones**
 1. Instrucciones de Inicio / Fin
 2. Instrucciones de asignación
 3. Instrucciones de lectura
 4. Instrucciones de escritura
 5. Instrucciones de bifurcación
- **Estructuras dentro de un algoritmo.** En mayo de 1966 Böhm y Jacopini demostraron que un programa puede ser escrito utilizando solamente tres tipos de estructuras de control:
 - ❑ Secuenciales
 - ❑ Selectivas
 - ❑ Repetitivas

Estructura secuencial.- 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 final del proceso.

Estructuras selectivas.- Las estructuras selectivas se utilizan para tomar decisiones lógicas , como consecuencia de la evaluación de una determinada condición. También se denominan estructuras de decisión o alternativas. *Las estructuras selectivas o alternativas pueden ser:*

• Simples Ejecuta una o más acciones si la condición es cierta.	• Dobles Ejecuta una o más acciones si la condición es cierta, y otras si la condición es falsa.	• Múltiples .Ejecuta una o más acciones dependiendo del valor de la <i>expresión</i>
si condición entonces <acción1> < acción2>	si condición entonces <accion1> si-no	segun sea expresion hacer valo1 : <acción1> valor2 : < acción2>

fin-si	<acción2>	:
	fin-si	fin_segun

Estructuras repetitivas.

mientras condición hacer acción S1 acción S2 acción Sn fin_mientras	repetir <acciones> hasta_que <condición>	Para v= vi hasta vf [incremento decremento < valor >] hacer < acciones> fin_para v: variable índice vi, vf: valores inicial y final de la variable
---	---	---

- **La estructura repetitiva mientras** (while o do while) es aquella en que el cuerpo del bucle se repite mientras se cumple una determinada condición. **Ejecución de un bucle cero o mas veces.** En una estructura mientras la primera acción es evaluar la expresión booleana, si es falsa el bucle nunca se ejecuta.

Bucles infinitos. Algunos bucles no exigen fin y otros no encuentran el fin por error en su diseño. Un bucle que nunca se termina se denomina bucle infinito. Estos bucles sin fin no intencionados son perjudiciales para la programación y se deben evitar siempre.

Regla practica: Las pruebas o tests en las expresiones booleanas es conveniente que sean mayor o menor que en lugar de pruebas de igualdad o desigualdad. En el caso de un lenguaje de programación, esta regla debe seguirse rígidamente en el caso de comparación de números reales, ya que como esos valores se almacenan en cantidades aproximadas las comparaciones de igualdad de valores reales normalmente plantean problemas. Siempre que realicemos comparaciones de números reales, usaremos las relaciones <, <=, > o >=.

Terminación de bucles con datos de entrada

Suma ← 0 escribir (`Existen mas números en la lista s/n`) leer (Resp) //variable Resp, tipo carácter mientras (Resp = `S`) o (Resp = `s`) hacer escribir ('numero') leer (N) Suma ← Suma + N escribir (`Existen mas números (s/n)`) leer (Resp) fin_mientras	<ul style="list-style-type: none"> Este método a veces es aceptable y es muy útil en ciertas ocasiones, pero suele ser tedioso para listas grandes; en este caso, es preferible incluir una señal de parada. 	<ul style="list-style-type: none"> Tal vez el método mas correcto para terminar un bucle que lee una lista de valores es con un centinela. <p>Un valor centinela es un valor especial usado para indicar el final de una lista de datos</p>
--	---	--

- **La estructura repetir (*repeat*)** se ejecuta hasta que se cumpla una condición determinada que se comprueba al final del bucle. El bucle repetir-hasta_que se repite mientras el valor de la expresión booleana de la condición sea falsa, justo la opuesta de la sentencia mientras. Con una estructura repetir el cuerpo del bucle se ejecuta siempre al menos una vez, a continuación, se evalúa la expresión booleana resultante de la condición. Si se evalúa como falsa, el cuerpo del bucle se repite y la expresión booleana se evalúa una vez más.

Diferencias de las estructuras mientras y repetir.

- La estructura mientras termina cuando la condición es falsa, mientras que repetir termina cuando la condición es verdadera.
- En la estructura repetir el cuerpo del bucle se ejecuta siempre al menos una vez; por el contrario, mientras es más general y permite la posibilidad de que el bucle pueda no ser ejecutado. Para usar la estructura repetir debes estar seguro de que el cuerpo del bucle - bajo cualquier circunstancia- se repetirá al menos una vez.

- **La estructura para (*for*)** . En muchas ocasiones se conoce de antemano el número de veces que se desean ejecutar las acciones de un bucle. En estos casos en los que el número de iteraciones es fijo, se debe usar la estructura desde o para (*for*). La estructura desde ejecuta las acciones del cuerpo del bucle un número especificado de veces y de modo automático controla el número de iteraciones o pasos a través del cuerpo del bucle.

La estructura **desde** comienza con un valor inicial de la variable índice y las acciones especificadas se ejecutan a menos que el valor inicial sea mayor que el valor final. La variable índice se incrementa en uno y si este nuevo valor no excede al final, se ejecutan de nuevo las acciones. Por consiguiente, las acciones específicas en el bucle se ejecutan para cada valor de la variable índice desde el valor inicial hasta el valor final con el incremento de uno en uno.

1.8.- ESCRITURA DE ALGORITMOS / PROGRAMAS.

La escritura de un algoritmo mediante una herramienta de programación debe ser lo más clara posible y estructurada, de modo que su lectura facilite considerablemente el entendimiento y su posterior codificación en un lenguaje de programación.

Los algoritmos deben ser escritos en lenguajes similares a los programas. Nosotros utilizaremos esencialmente el lenguaje algorítmico, basado en pseudocódigo, y la estructura del algoritmo requerirá la lógica de los programas escritos en el lenguaje de programación estructurado. Un algoritmo constará de dos componentes: una **cabecera de programa** y un **bloque algoritmo**.

La cabecera de programa es una acción simple que comienza con la palabra **algoritmo /programa** . Esta palabra estará seguida por el nombre asignado al programa completo.

El bloque algoritmo es el resto del programa y consta de dos componentes o secciones: **Las acciones de declaración y las acciones ejecutables.**

Las declaraciones definen o declaran las variables y constantes .

Algoritmo cabecera del programa
sección de declaración
sección de acciones ejecutables

Las ejecutables son las acciones que posteriormente deberá realizar el ordenador cuando el algoritmo se convierta en programa .

Cabecera del programa/algoritmo. Todos los algoritmos y programas deben comenzar con una cabecera en la que se exprese el identificador o nombre correspondiente con la palabra reservada que señale el lenguaje .

Declaración de variables. En esta sección se declaran o describen todas las variables utilizadas en el algoritmo, indicando sus nombres y sus tipos.

Esta sección comienza con la palabra reservada **entorno** y cada lista de variables es una variable simple o una lista de variables separadas por comas y cada tipo es uno de los tipos básicos de datos , (entero, real , carácter o lógico) Por ejemplo, la sección de declaración de variables:

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

entorno entera: Numero-Empleado real: Horas real: Impuesto real: Salario	entorno entera: Numero-Empleado real: Horas, Impuesto, Salario	<ul style="list-style-type: none"> Es una buena práctica de programación utilizar nombres de variables significativos que sugieran lo que ellas representan, ya que eso hará más fácil y legible el programa. También es buena práctica incluir breves comentarios que indiquen como se utiliza la variable.
---	---	---

Declaración de constantes numéricas y tipo carácter. En esta sección se declararán todas las constantes que tengan nombre. Su formato es:

const pi = 3.141592 tamaño = 43 horas = 6.50 estrella = ' * ' frase = '12 de octubre'	<ul style="list-style-type: none"> Los valores de estas constantes ya no pueden variar en el transcurso del algoritmo.
--	---