Temario

1- Lenguajes de programación. Intérpretes y compiladores.

2- Tipos de Datos y Expresiones: Concepto, Tipos y manejo de datos. Variables. Constantes. Expresiones: aritméticas y lógicas. Asignación.

3- Variables de trabajo: acumuladoras, contadoras e interruptores.Arreglos y registros. Clasificación de estructuras de datos. Definición de registro. Operaciones sobre registros. Arreglos unidimensionales (vectores) y bidimensionales (matrices). Operaciones con arreglos.

4- Conceptos básicos de programación: Diseño de algoritmos: flujo de control de algoritmos (secuenciación, iteración, selección).

5- Elementos de un programa: Concepto de programa. Partes constitutivas de un programa. Tipos de instrucciones. Estructuras de control. Estructuras anidadas.

6- Subprogramas: procedimientos y funciones. Modularización. Ámbito de las variables. Declaración e invocación de los módulos. Comunicación de módulos. Procedimientos y funciones predefinidas.

7- Software para resolución de problemas matemáticos: Resolución de problemas matemáticos (orientados a graficación de funciones, cálculo de integrales, geometría, poliedros, conversores de magnitudes, etc.) mediante el uso de distintos programas de software utilitario de oficina.

1- Lenguajes de programación. Intérpretes y compiladores.

Explicar los diferentes tipos de lenguajes de programación (Web, Cliente Servidor)

DEFINICIÓN
Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones, y es utilizado para controlar el comportamiento físico y lógico de una máquina. (Porque físico y lógico, porque un software puede controlar un dispositivo físico, como ser una estufa, una cafetera, una cámara web, una persiana, una impresora, otra PC, y cualquier otro dispositivo que pueda tener conectividad y software instalado y lógico porque puede controlar otro programa)

Aunque muchas veces se usan los términos 'lenguaje de programación' y 'lenguaje informático' como si fuesen sinónimos, no es del todo correcto, ya que los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como por ejemplo HTML que es un lenguaje para el marcado de páginas web.

Un lenguaje de programación permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo estos datos deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, tal como sucede con el lenguaje Léxico.

CLASIFICACIÓN

Los lenguajes de programación se pueden clasificar atendiendo a varios criterios, los principales son:
- Según el nivel de abstracción
- Según la forma de ejecución
- Según el paradigma de programación que poseen cada uno de ellos

SEGÚN EL NIVEL DE ABSTRACCIÓN

Lenguajes de máquina y de bajo nivel
Los lenguajes de máquina están escritos en códigos (código máquina) directamente inteligibles por la máquina (computadora), siendo sus instrucciones cadenas binarias (0 y 1).

"Lenguaje de máquina" hace referencia al lenguaje específico de una computadora, mientras que "código máquina" hace referencia al modo en que se escriben los diferentes lenguajes de máquina.

Los lenguajes de bajo nivel son lenguajes de programación que se acercan al funcionamiento de una computadora. Los lenguajes de más bajo nivel son los lenguajes de máquinas. A éste nivel le sigue el lenguaje ensamblador, ya que al programar en ensamblador se trabajan con los registros de memoria de la computadora de forma directa.

La programación en un lenguaje de bajo nivel tiene como ventajas una mayor adaptación al equipo, además de la posibilidad de obtener la máxima velocidad con el mínimo uso de memoria.
Sin embargo tiene importantes inconvenientes, como la imposibilidad de escribir código independiente de la máquina y la mayor dificultad en la programación y en la comprensión de los programas.

Lenguajes de medio nivel
Minoritariamente en algunos textos se diferencian algunos lenguajes como de medio nivel, como el lenguaje C, ya que tienen ciertas características que los acercan a los lenguajes de bajo nivel, como gestión de punteros de memoria y registros, pero con sintaxis, vocabulario y gramática de alto nivel.

Lenguajes de alto nivel y de muy alto nivel
Los lenguajes de programación de alto nivel se caracterizan por expresar los algoritmos de una manera adecuada a la capacidad cognitiva humana, en lugar de estar orientados a su ejecución en las máquinas.
Los lenguajes de alto y bajo nivel requieren de conocimientos específicos de programación y del lenguaje concreto (vocabulario, gramática y sintaxis) para realizar las secuencias de instrucciones lógicas.
Los lenguajes de muy alto nivel se crearon para que el usuario común pudiese solucionar ciertos problemas sencillos de procesamiento de datos de una manera más fácil y rápida. (Lenguajes transaccionales para base de datos: select, insert, update, delete, create, drop, etc) (Lenguajes visual basic, if, select case, for, while, etc)

SEGÚN LA FORMA DE EJECUCIÓN

Los procesadores usados en las computadoras son capaces de entender y actuar según lo indican programas escritos en un lenguaje fijo para cada arquitectura, llamado lenguaje de máquina. Todo programa escrito en un lenguaje de alto nivel puede ser ejecutado de dos maneras:

• **Lenguajes compilados**: Antes de poder utilizarse el programa debe utilizarse un traductor llamado "compilador" que se encarga de traducir ("compilar") el programa original ("código fuente") al programa equivalente escrito en lenguaje de máquina o ensamblador ("binario"). Los binarios son los programas ejecutables y los únicos necesarios para el funcionamiento del programa. (Los archivos con extensión .exe, .class → son compilados
• **Lenguajes interpretados**: Cada vez que se usa el programa debe utilizarse un traductor llamado "intérprete" que se encarga de traducir ("interpretar") las instrucciones del programa original ("código fuente") a código máquina según van siendo utilizadas. Para el funcionamiento del programa siempre es necesario disponer del código original y del intérprete. y por ejemplos los archivos .asp, .html, .jsp, .php, .java, etc) → son interpretados

Diferencias entres lenguajes compilados e interpretados

● Los lenguajes compilados se compilan una vez y se utilizan cuantas veces se desee sin necesidad de volver a utilizar el compilador. Los lenguajes interpretados son interpretados, valga la redundancia, cada vez que se ejecutan y necesitan siempre del intérprete. (En la web lo que sucede cuando un servidor o servicio esta bajo, por ejemplo → Page not found, Motor de base de datos está bajo)

● Los compiladores analizan todo el programa y no generan resultados si no es correcto todo el código. Los intérpretes analizan las instrucciones según las necesitan y pueden iniciar la ejecución de un programa con errores e incluso terminar correctamente una ejecución de un programa con errores siempre que no haya sido necesario el uso de las instrucciones que contienen dichos errores.

● Un compilador traduce cada instrucción una sola vez. Un intérprete debe traducir una instrucción cada vez que la encuentra.

● Los binarios son compilados para una arquitectura específica y no pueden ser utilizados en otras arquitecturas no compatibles (aunque pueden existir distintos compiladores para generar binarios para diferentes arquitecturas). Un lenguaje interpretado puede ser utilizado en cualquier arquitectura que disponga de un intérprete sin necesidad de cambios.

● Los lenguajes compilados son más eficientes que los interpretados y además permiten distribuir el programa en forma confidencial mediante binarios.

● Es más sencillo empaquetar lenguajes interpretados dentro de otros lenguajes, como JavaScript dentro de HTML.
Para obtener las ventajas de ambos tipos de lenguajes algunos utilizan una aproximación en dos fases. Primero el programa original (código fuente) es precompilado a un binario confidencial, portable e interpretable. En una segunda fase el binario precompilado es interpretado en cada arquitectura. Ésta aproximación es la que realiza por ejemplo Java.

Hay que hacer notar que algunas aplicaciones permiten ser programadas con lenguajes. Estos lenguajes no tienen por objeto solicitar acciones a la computadora sino solicitar acciones a la aplicación sobre la que se ejecutan. Por tanto aunque algunos de estos lenguajes son lenguajes de programación, no son lenguajes de programación de computadoras y por tanto no necesitan ser traducidos a código máquina. Es el caso por ejemplo de SQL, un lenguaje declarativo de cuarta generación diseñado para trabajar con bases de datos.

Este lenguaje SQL es interpretado por el motor de la Base de Datos, no por la CPU.

SEGÚN EL PARADIGMA DE PROGRAMACIÓN

Un paradigma de programación representa un enfoque particular o filosofía para la construcción del software. Si bien puede seleccionarse la forma pura de estos paradigmas a la hora de programar, en la práctica es habitual que se mezclen, dando lugar a la programación multiparadigma.

Los diferentes paradigmas de programación son:

• **Algorítmico, Imperativo o Por procedimientos**. El más común y está representado, por ejemplo, por C o por BASIC.
Describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea.
La implementación de hardware de la mayoría de computadores es imperativa ya que el *hardware* está diseñado para ejecutar código de máquina que es imperativo.

• **Declarativo o Predicativo**. Basado en la utilización de predicados lógicos (lógico) o funciones matemáticas (funcional), su objetivo es conseguir lenguajes expresivos en los que no sea necesario especificar cómo resolver el problema (programación convencional imperativa), sino qué problema se desea resolver. Los Interpretes de los lenguajes declarativos tienen incorporado un motor de inferencia genérico que resuelve los problemas a partir de su especificación.
○ **Lógico**. Un ejemplo es PROLOG. El mecanismo de inferencia genérico se basa en los procedimientos de deducción de formulas válidas en un sistema axiomático
○ **Funcional**. Representado por la familia de lenguajes LISP (en particular Scheme), ML o Haskell. El mecanismo de inferencia genérico se basa en la reducción de una expresión funcional a otra equivalente simplificada.

• **Orientado a Objetos**. Cada vez más utilizado, sobre todo en combinación con el imperativo. De hecho los lenguajes orientados a objetos permiten la programación imperativa. Algunos ejemplos de lenguajes orientados a objetos son C++, Java, Python. Usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento.

Por todo esto, es más lento el desarrollo de programas comparables en Lenguaje Ensamblador que en un lenguaje de alto nivel, pues el programador goza de una menor abstracción.

Entonces existen dos tipos principales de traductores de los lenguajes de programación de alto nivel:

- **Compilador**, que analiza el programa fuente y lo traduce a otro equivalente escrito en otro lenguaje (por ejemplo, en el lenguaje de la máquina). Su acción equivale a la de un traductor humano, que toma un libro y produce otro equivalente escrito en otra lengua.
- **Intérprete**, que analiza el programa fuente y lo ejecuta directamente, sin generar ningún código equivalente. Su acción equivale a la de un intérprete humano, que traduce las frases que oye sobre la marcha, sin producir ningún escrito permanente. Intérpretes y compiladores tienen diversas ventajas e inconvenientes que los hacen complementarios:
    - Un intérprete facilita la búsqueda de errores, pues la ejecución de un programa puede interrumpirse en cualquier momento para estudiar el entorno (valores de las variables, etc.). Además, el programa puede modificarse sobre la marcha, sin necesidad de volver a comenzar la ejecución.
    - Un compilador suele generar programas más rápidos y eficientes, ya que el análisis del lenguaje fuente se hace una sola vez, durante la generación del programa equivalente. En cambio, un intérprete se ve obligado generalmente a analizar cada instrucción tantas veces como se ejecute (incluso miles o millones de veces).
    - Un intérprete permite utilizar funciones y operadores más potentes, como por ejemplo ejecutar código contenido en una variable en forma de cadenas de caracteres. Usualmente, este tipo de instrucciones es imposible de tratar por medio de compiladores. Los lenguajes que incluyen este tipo de operadores y que, por tanto, exigen un intérprete, se llaman interpretativos. Los lenguajes compilativos, que permiten el uso de un compilador, prescinden de este tipo de operadores.

Programa
• Un programa informático es un conjunto de instrucciones escritas en un lenguaje de programación, para que sean ejecutadas en un ordenador.

• Un programa es el resultado final de un proceso que se inicia con el planteamientode un problema (que deberá ser posteriormente resuelto por un ordenador)

Para que el ordenador entienda y pueda ejecutar un programa, éste deberá estar escrito en un lenguaje especial, comprensible para el ordenador

• Los ordenadores no entienden matices, no resuelven contradicciones, ni realizan deducciones fruto de la experiencia. Por ello, los programas deberán estar escritos de una forma rigurosa
- Sin errores sintácticos
- Sin contradicciones
- Sin omisiones

• Los errores sintácticos producen programas incomprensibles para el ordenador En programas que no se pueden ejecutar
• Las contradicciones u omisiones (errores lógicos), producen programas que posiblemente no se ejecuten como el programador ha previsto. Suelen ser por ello, más difíciles de detectar.
• Los ordenadores son quisquillosos. Una simple coma (,) de más, de menos, o fuera de lugar, puede producir errores sintácticos o errores lógicos (dependiendo del contexto, del lenguaje, etc.)

Tipos de instrucciones:

- E/S: Pasar información del exterior al interior del ordenador y al revés.
- Aritmético-lógicas: Aritméticas: +,-,*,... ; Lógicas: or, and, <, >, ...
- Selectivas: Permiten la selección de una alternativa en función de una condición.
- Repetitivas: Repetición de un número de instrucciones un número finito de veces.

Tipos de lenguajes:

- Lenguaje máquina: Todo se programa con 1 y 0, que es lo único que entiende el ordenador.

Ventaja: No necesita ser traducido.

Explicación sistema binario: http://www.youtube.com/watch?v=KySjjvBEDaA

Inconveniente: La dificultad, la confusión, para corregir errores, es propia de cada máquina.

- De bajo nivel o ensamblador: Se utilizan mnemotécnicos (abreviaturas).

Ventaja: No es tan difícil como el lenguaje máquina.

Inconvenientes: Cada máquina tiene su propio lenguaje, necesitamos un proceso de traducción.

- El programa escrito en ensamblador se llama programa fuente y el programa que se obtiene al ensamblarlo se llama programa objeto.

- Lenguajes de alto nivel: Los más cercanos al lenguaje humano.

Ventaja: Son independientes de cada maquina (los compiladores aceptan las instrucciones estándar, pero también tienen instrucciones propias).

Inconveniente: El proceso de traducción es muy largo y ocupa más recursos. Aprovecha menos los recursos internos.

Proceso de traducción y ejecución de un programa escrito en un lenguaje a alto nivel:

Usamos un editor y obtenemos el programa fuente, y el compilador es el que traduce el programa al lenguaje máquina. El compilador internamente ha sido diseñado para traducir.
El compilador obtiene el programa o el fichero objeto. El compilador tiene que buscar los errores.
Normalmente no sale un ejecutable, sino que necesita elementos, librerías, ...
Mediante un linkador juntamos el programa objeto y las librerías, y se forma un programa ejecutable.

Cuando se ejecuta el programa, el cargador lleva al programa a memoria para que éste pueda ser ejecutable.

Debbuger: Depura el programa ejecutándolo paso a paso, viendo la memoria paso a paso para encontrar el error.

2- Tipos de Datos y Expresiones: Concepto, Tipos y manejo de datos. Variables. Constantes. Expresiones: aritméticas y lógicas. Asignación.

DATOS, TIPOS DE DATOS Y OPERACIONES PRIMITIVAS:

- Dato:

Es un objeto o elemento que tratamos a lo largo de diversas operaciones.

Un dato es una representación de un objeto del mundo real, mediante la cual se puede modelar aspectos de un problema que se desea resolver.

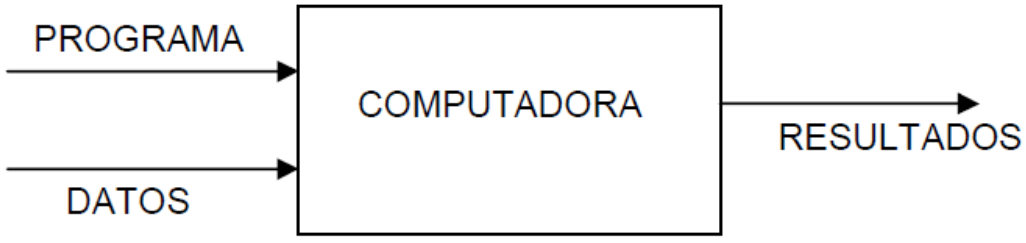
Dato es un símbolo físico que representa información, tal como un número o un carácter.

Tenemos que representar la información como datos antes de poder usarla en la computadora. Estos datos serán procesados estando almacenados internamente en memoria, almacenados en un disco o cinta o introducidos desde una terminal.

El término Dato desde el punto de vista del Problema

En particular, cuando se desea resolver un problema, se debe hacer una correcta interpretación del enunciado del mismo. Se debe determinar cuáles son los elementos de que se dispone (los datos del problema) y cuáles son los objetivos deseados (los resultados que hay que hallar).

Para que la máquina pueda ejecutar las acciones y resolver el problema hay que ingresar el programa y en el momento de ejecución del mismo, se deben ingresar los datos con los que trabajará. Aquí estamos haciendo referencia al término dato hablando de los objetos que se deben ingresar durante la ejecución del programa. Estos datos serán procesados, se encontrarán resultados intermedios y finalmente se encontrarán los resultados finales deseados.



Desde el punto de vista del problema, diremos que los datos, serán los objetos ingresados durante la ejecución del programa. En cambio, desde el punto de vista de la computadora, todos los objetos procesados serán datos, tanto los que se ingresan, como los resultados intermedios y los resultados finales.

Tienen 3 caracteristicas:

- Un nombre que los diferencia del resto.
- Un tipo que nos determina las operaciones que podemos hacer con ese dato.
- Un valor que puede variar o no a lo largo de la operación.

Existen diferentes tipos de datos.

En Pascal (como en la mayoría de los lenguajes) cada elemento de dato debe ser de un tipo específico.

El tipo de dato determina:

☐ un rango de valores posibles

☐ cómo se representarán internamente los datos en la computadora, es decir, qué tamaño y forma tendrá la variable donde se almacena.

☐ qué tipo de procesamiento (operaciones) podrá ejecutar sobre ellosla computadora.

Esto significa que, antes de que podamos escribir cualquier instrucción para manipular datos de un tipo particular, hemos de escribir una declaración que informe de qué tipo es la variable que lo contiene. Toda variable en un programa debe estar asociada a un tipo de dato y sólo a uno.

Un tipo de dato define el conjunto de valores que puede asumir una variable.
Tipo de datos es una familia o conjunto de datos que tienen las mismas propiedades, al que se le asocia un nombre para identificarlo.

O bien

Tipo de datos es la forma general de una clase de elementos de datos.

Hay algunos tipos de datos que se utilizan frecuentemente y el Pascal, como los otros lenguajes, los define automáticamente. Estos son los tipos de datos simples o estándares:

1	Numéricos	Enteros
2		Reales
3	Carácter o Alfanumérico	
4	Lógicos o Booleanos	
5	Cadena o String	

1) Tipo de dato Entero (Integer)

Es el conjunto de los números enteros positivos y negativos (sin parte decimal). Constan de un signo y dígitos.
+22 -16 1 -426 2500

Cuando se omite el signo se asume que el número es positivo.

Teóricamente no hay límite para el tamaño de los enteros, pero dado que la computadora tiene memoria finita, la cantidad de valores que se pueden representar en ella son finitos, por esto se deduce que existe un número entero máximo y otro mínimo.

Se establece una cantidad fija de bytes (conjunto de 8 bits) para almacenar todo dato entero, por ello existe un valor máximo que se pueda representar.

Existe un identificador predefinido Maxint, que es el mayor valor entero que pueda representarse.

Pascal dispone generalmente de 2 bytes para almacenar un entero, entonces

Maxint es 32767, por lo cual el rango de los enteros permitidos será:
- Maxint hasta Maxint (- 32768 hasta 32767)

Maxint puede ser diferente de una máquina a otra, por lo que puede imprimirse par ver cuál es su valor.
Las expresiones enteras son las que permiten obtener un dato entero. La tabla siguiente muestra los operadores que se pueden utilizar para generar operaciones con enteros. Cinco de ellos dan resultado de tipo entero y uno devuelve un resultado de tipo real.

Operadores			
Operador	Efecto	Tipo de Operando	Tipo de resultado
+	Adición	Entero	Entero
-	Resta	Entero	Entero
*	Producto	Entero	Entero
/	División	Entero	Real
Div	División Entera	Entero	Entero
Mod	Resto de la división entera	Entero	Entero

Las reglas de jerarquía de estos operadores es la misma que la de la aritmética y para igual jerarquía se ejecutan de por orden de aparición. En Pascal no existe la potenciación. Remitirse a la Unidad 2 para ver las funciones que se pueden aplicar a enteros.

2) Tipo de dato Real (Real)

Los datos de tipo real representan a cantidades numéricas reales, es decir, tienen una parte entera y una parte decimal.

Existen dos formas de representarlos:
1. Con parte entera y parte decimal separadas por coma o punto (-348,91)
2. Notación científica (-0.34891 E + 03)

Internamente en la computadora los reales se representan en notación científica con base 2, denominada coma flotante, por ser conveniente para la aritmética binaria, definiendo a cada número como una mantisa (parte decimal) y un exponente (posición de la coma).

Se debe tener en cuenta que el tipo de dato real tiene una representación finita de los números reales, dicha representación tiene una precisión fija, es decir, un número fijo de dígitos significativos. Esta condición establece una diferencia con la representación matemática de los números reales. En este caso se tienen infinitos números reales, en tanto que la cantidad de representaciones del tipo de dato real está limitada por el espacio en memoria disponible.
En Pascal se destinan 4 u 8 bytes para representarlos. Dependerá de la cantidad de bytes para tener mayor o menor precisión (cantidad de dígitos significativos) y magnitud (tamaño en valor absoluto) en el número.

Las expresiones reales permiten obtener datos reales. Si en una expresión al menos uno de los operandos es de tipo real, los operadores producen un resultado real.

Operadores	
+	Adición
-	Resta
*	Producto
/	División (ambos operando pueden ser enteros pero el resultado es siempre real)

3) Tipo de dato Alfanumérico o Carácter (Char)

Un dato de tipo carácter es un carácter simple encerrado entre apóstrofos, este carácter puede ser una letra, un dígito o un símbolo especial.

Por ejemplo: 'A' 'a' '8' '-' '$' ' ' '?'

Cada máquina tiene un conjunto de caracteres alfanuméricos que pueden ser representados en ella. Este conjunto se llama el conjunto de caracteres de la máquina e incluye:

☐ Letras 'A'…'Z', 'a'…'z'
☐ Dígitos '0'…'9'
☐ Caracteres especiales '&', '+' , '_' , ' ' , '?'

Obsérvese que cada carácter está encerrado entre apóstrofos o comillas simples. El compilador necesita los apóstrofos para diferenciar entre el dato carácter '8' ó '+' y el entero 8 ó el operador suma +.

4) Tipo de dato Lógico o Boolean (Boolean)

Booleano es un tipo con sólo dos valores: True y False (Verdadero y Falso).
Los valores true y false representan un conjunto ordenado, donde false precede a true, es decir: false < trae.

Combinando operandos con operadores lógicos se obtienen expresiones lógicas o booleanas.

Los datos booleanos no pueden ingresarse como datos (como los otros tres tipos estándares), pero sí pueden imprimirse.

Remitirse a la tabla de la Unidad 2 para observar las funciones que producen resultados lógicos.

5) Tipo de dato Cadena (String)

El tipo de dato String es una sucesión de caracteres que se almacenan en un área contigua de la memoria y que puede leído o escrito.

Este tipo represente un dato de tamaño dado que resulta de la concatenación (unión) de los caracteres que lo forman. Al declarar la variable de tipo string se debe indicar la longitud que es el número máximo de caracteres que puede contener.

```
Var Nombre_Variable_string : string [ longitud]
Ejemplo:
CADENA: string [10];
…
CADENA := 'lunes' ;
```

Las operaciones válidas sobre string son la asignación y la concatenación, ésta permite adosar un string a continuación de otro.

Los operadores relacionales pueden utilizarse con los string (=, <, <=, >, >=. <>), no es necesario que las dos variables string tengan la misma longitud.

CONSTANTES Y VARIABLES:

- Constantes: Tienen un valor fijo que se le da cuando se define la constante y que ya no puede ser modificado durante la ejecución.
- Variables: El valor puede cambiar durante la ejecución del algoritmo, pero nunca varia su nombre y su tipo.

Antes de usar una variable hay que definirla o declararla, al hacerlo hay que dar su nombre y su tipo. El nombre que le damos tiene que ser un nombre significativo, va a ser un conjunto de caracteres que dependiendo del lenguaje hay restricciones.

Tiene que empezar por una letra, y el tamaño depende del lenguaje.
Identificador: Palabra que no es propia del lenguaje.

El valor de la variable si al declararla no se la inicializa, en algunos lenguajes toma una por defecto. En cualquier caso el valor de la variable podemos darle uno incial o podemos ir variandolo a lo largo de la ejecución.

Las constantes pueden llevar asociados un nombre o no, si no lo llevan, se llaman literales. Su valor hay que darlo al definir la constante y ya no puede cambiar a lo largo de la ejecución, y en cuanto al tipo, dependiendo de los lenguajes en algunos hay que ponerlo, y en otros no hace falta ponerlo porque toma el tipo del dato que se le asigna. Const PI=3,1416.

La ventaja de usar constantes con nombre es que en cualquier lugar donde quiera que vaya la constante, basta con poner su nombre y luego el compilador lo sustituira por su valor. (En caso de modificación de ese dato, sólo se altera en un lugar y no hay que alterar todo el programa)

Las constantes sin nombres son de valor constante: 5, 6, `a', "hola".
Cuando una cadena es de tipo carácter, se encierra entre `' `a'.

Relación entre variables y constantes en memoria:
Al detectar una variable o una constante con nombre, automáticamente se reserva en memoria espacio para guardar esa variable o constante. El espacio reservado depende del tipo de la variable.
En esa zona de memoria es en la que se guarda el valor asociado a la variable o constante y cuando el programa use esa variable, ira a esa zona de memoria a buscar su valor.

Las operaciones que se realicen sobre estas variables y/o constantes, están definidas por una serie de operadores, entre los cuales se encuentran:

Operadores: Aritméticos.

- Potencia. ^ **
- Producto. *
- División. / Div Mod
- Suma. +
- Resta. -

Operadores: Alfanuméricos.

- Concatenación. + &

Ejm.

'UN' + 'AD' → 'UNAD'

Operadores: Relacionales.

- Igual a. =
- Menor que. <
- Menor o igual que. <=
- Mayor que. >
- Mayor o igual que. >=
- Distinto a. <>

Operadores: Lógicos.

- Negación. Not no
- Conjunción/producto. And y
- Disyunción/suma. Or o

Operadores: Paréntesis.

• El paréntesis Permite alterar el orden en que realizan las diferentes operaciones

Ejm. A / (2 * B)

En la ejecución de un programa o algoritmo se hace cumplir una serie de reglas de prioridad que permiten determinar el orden de las operaciones

Orden de evaluación de los operadores

• Paréntesis.

• Cambio de signo.

• Potencias.

• Productos y divisiones.

• Sumas y restas.

• Concatenación.

• Relacionales.

• Negación.

• Conjunción.

• Disyunción.

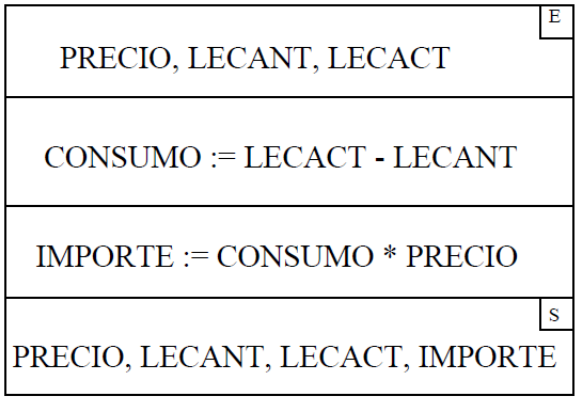
Observación

El operador MOD, permite obtener el residuo de una división

El operador DIV, Permite obtener la parte entera de una división

Realizar un algoritmo en diagrama de Chapin para resolver el siguiente problema:

Dados como datos: el precio del kilowatt-hora, la lectura actual y la lectura anterior de un medidor; calcular el importe que una persona deberá abonar a la E.P.E. Mostrar los datos y el resultado obtenido.-



Explicar básicamente de forma gráfica (Pantalla de ingreso de datos) como sería la interpretación de este programa

4- Conceptos básicos de programación: Diseño de algoritmos: flujo de control de algoritmos (secuenciación, iteración, selección).

¿Qué es un algoritmo?:

El algoritmo trata de resolver problemas mediante programas.

Fases:

- Análisis preliminar o evaluación del problema: Estudiar el problema en general y ver que parte nos interesa.
- Definición o análisis del problema: Ver que es lo que entra y que es lo que sale, las posibles condiciones o restricciones, ...
- Diseño del algoritmo: Diseñar la solución.
- El programa: Codificación del algoritmo en un lenguaje de programación.
- Ejecución del programa y las pruebas: Ver si el programa hace lo que queríamos.

Es una formula para resolver un problema. Es un conjunto de acciones o secuencia de operaciones que ejecutadas en un determinado orden resuelven el problema. Existen n algoritmos, hay que coger el más efectivo.

Características:

- Tiene que ser preciso.
- Tiene que estar bien definido.
- Tiene que ser finito.

La programación es adaptar el algoritmo al ordenador.

El algoritmo es independiente según donde lo implemente.

Prueba de escritorio: seguimiento algorítmico

Probar un algoritmo es ejecutar cada una de las acciones incluidas en él.
La prueba de escritorio o seguimiento de un algoritmo puede efectuarse de la siguiente manera:
a) Proponga un conjunto de datos. Estos datos deben coincidir en cantidad y tipo con las variables que aparezcan en las acciones de lectura.
b) Construya una tabla y coloque – encabezando cada columna de la tabla – cada una de las variables que aparezcan en el algoritmo.
c) Escriba Salida en el encabezado de la última columna de la tabla. Aquí se notarán los resultados que produzca el algoritmo como consecuencia de la acción de Escribir.
d) Comience a ejecutar las acciones del algoritmo. Cuando encuentre una asignación de una variable coloque el valor o dato a asignar en la columna correspondiente a esa variable. e) Si lee una variable, tomo el dato de prueba propuesto para esa variable y colóquelo en la columna correspondiente a esa variable.
f) Si vuelve a asignar o a leer una variable ya creada, continúe anotando en la columna correspondiente.
g) Al terminar de ejecutar las acciones, los resultados o salida del algoritmo deben aparecer en la columna de Salida.