

Lógica de Programación. Nivel I

noviembre, 2017



Objetivos del nivel

- Entender los conceptos básicos de la algoritmia
- Entender técnicas para resolución de problemas
- Aprender a crear algoritmos

Prerrequisitos del nivel

- Desarrollo de Software Nivel I

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADIF1). Para obtener más información sobre este u otros cursos visite nuestra sitio Web www.cadif1.com, escribanos a la dirección de correo cadi@cadif1.com o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños. Copyright 2017. Todos los derechos reservados.

ACADEMIA DE SOFTWARE



Contenido del nivel

Capítulo 1. Algoritmos e Instrucciones

- 1.1.- Definición.
- 1.2.- Algoritmo en la Vida Real.
- 1.3.- Instrucción y Programación.

Capítulo 2. Algoritmos Usando Bloques

- 2.1.- Concepto.
- 2.2.- Ejercicios Iniciales Usando Bloques.

Capítulo 3. Flujogramas. Parte 1

- 3.1.- Concepto y Objetivo.
- 3.2.- Simbología.
- 3.3.- Ejemplos de Flujogramas.

Capítulo 4. Flujogramas. Parte 2

- 4.1.- Antes de Diagramar y la Diagramación.
- 4.2.- Consideraciones de Diagramación.
- 4.3.- Ejemplo de Buena Diagramación.

Capítulo 5. Lenguaje y Pseudolenguaje

- 5.1.- Lenguaje y Pseudolenguaje.
- 5.2.- El Código Fuente y el Ejecutable.
- 5.3.- Entornos de Desarrollo: IDE.

Capítulo 6. Pseudocódigo y PSeInt

- 6.1.- Pseudocódigo.
- 6.2.- Pseudo Intérprete.
- 6.3.- Palabras Reservadas.
- 6.4.- Los Comentarios.

Capítulo 7. Variables

- 7.1.- Dato y Variable.
- 7.2.- Tipos de Datos.
- 7.3.- Declaración de Variables.

Capítulo 8. Instrucciones Básicas en Pseudocódigo

- 8.1.- Concepto.
- 8.2.- Instrucción de Salida.
- 8.3.- Instrucción de Entrada.
- 8.4.- Instrucciones Utilitarias.

Capítulo 9. Operadores Aritméticos

- 9.1.- Operadores Aritméticos.
- 9.2.- Precedencia de Los Operadores.
- 9.3.- Expresiones no Lineales.

Capítulo 10. Expresiones en Cálculos Comunes

- 10.1.- Introducción.
- 10.2.- Promedios.
- 10.3.- Porcentajes.
- 10.4.- Aumentos y Descuentos.

Capítulo 11. Asignaciones

- 11.1.- Operador de Asignación.
- 11.2.- Asignación de Valores Literales.
- 11.3.- Asignación de Variables.
- 11.4.- Asignación de Expresiones.

Capítulo 12. Resolución de Problemas y Algoritmos

- 12.1.- Problema y su Resolución.
- 12.2.- Entradas, Proceso y Salidas.

12.3.- Fases de la Resolución de Problemas.

12.4.- Tabla de Variables.

Capítulo 13. Algoritmos Secuenciales

13.1.- Introducción.

13.2.- Construcción Progresiva: Algoritmo y Programa.

13.3.- Ejemplo de Algoritmo Secuencial.

Capítulo 14. Funciones Internas

14.1.- Concepto.

14.2.- Funciones Con Números.

14.3.- Funciones de Cadena.

Capítulo 15. Depuración de Programas

15.1.- Concepto.

15.2.- Ejecución Paso a Paso.

15.3.- Evaluación de Variables.

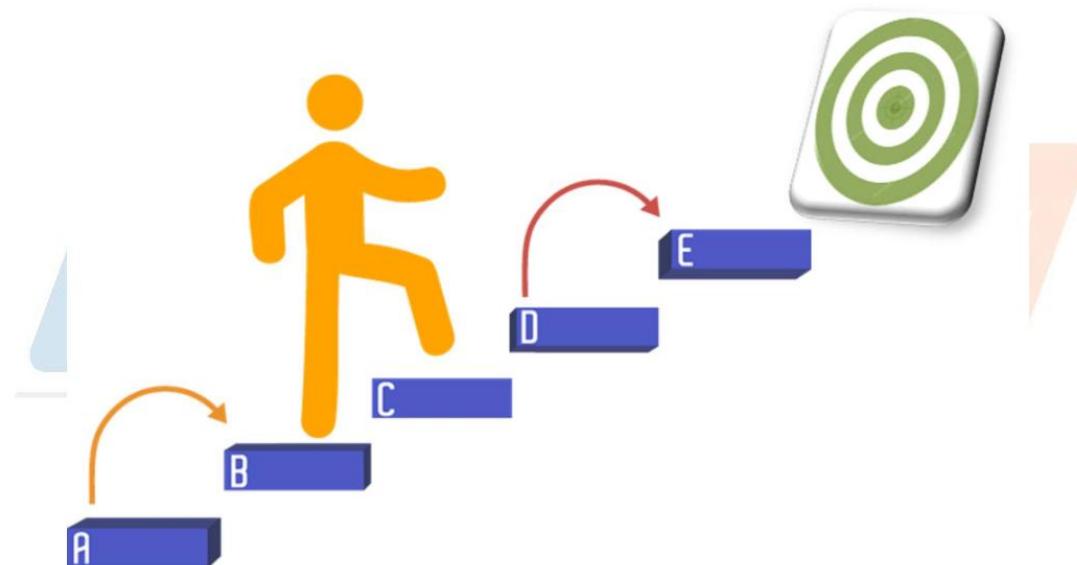


Capítulo 1. ALGORITMOS E INSTRUCCIONES

1.1.- Definición

En términos simples, un ALGORITMO es un conjunto de pasos que deben ser ejecutados en orden para realizar una tarea y/o resolver un problema.

Este término no está relacionado exclusivamente con las ciencias de la computación, como la informática, o la matemática, a partir de la cual se creó.



De un modo más formal, un ALGORITMO es una secuencia finita de operaciones realizables, no ambiguas, cuyo seguimiento paso a paso debe conducir a la solución a un problema en un tiempo finito.

Por lo anterior, se puede decir que un algoritmo es una "fórmula" para resolver un problema, el cual delinea un proceso metódico que establece una serie de pasos no confusos y sencillos que al ser seguido permite lograr uno o varios objetivos.

Un ejemplo de un algoritmo cotidiano es el de hablar por teléfono usando un teléfono de mesa:

```

Levantar el auricular
Esperar tono
Marcar el número de teléfono destino
SI contestan
    saludar
    preguntar por la persona
    hablar con la persona
    despedirse
FIN SI
Colgar el auricular

```

1.2.- Algoritmo en la Vida Real

Es importante concientizar que en la vida real empleamos algoritmos en múltiples ocasiones, ejemplo de ellos son los algoritmos para:

- Resolver un problema matemático.
- Preparar una comida.

En general... ¿Cuales serán los pasos para preparar un platillo siguiendo una receta?

TORTA DE NUEZ
Cacerola con asas (Cód. 4922)

Función: Horno - Masas Porciones: 10

Ingredientes:

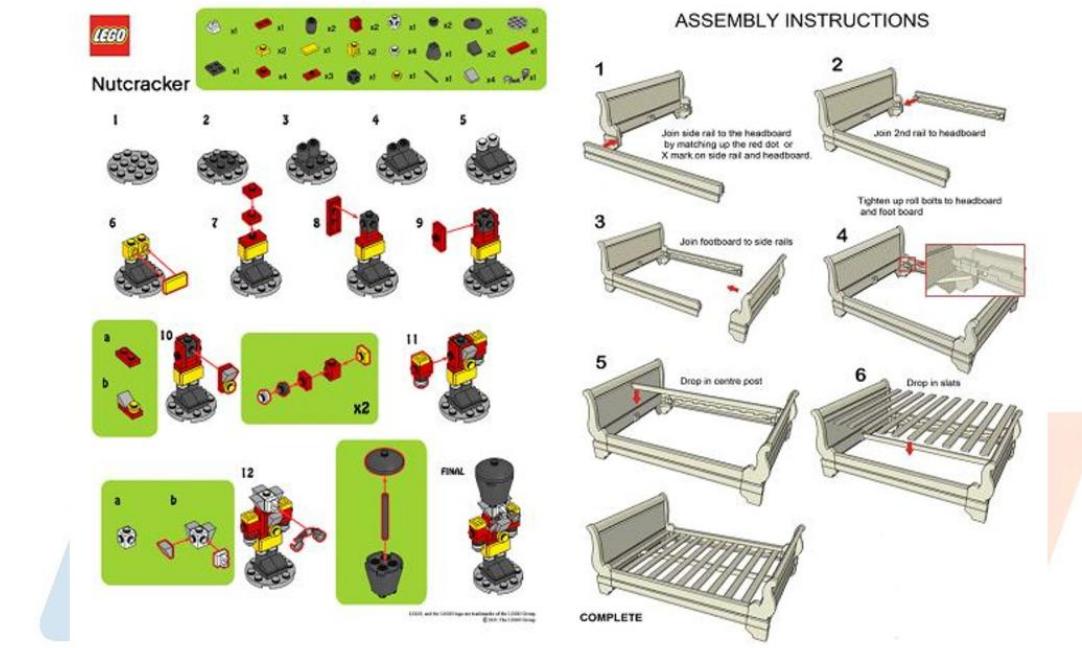
- 150 gr de manteca a temperatura ambiente
- 200 gr de azúcar
- 3 yemas
- 1 cdta. de esencia de vainilla
- Ralladura de 1 limón
- 100 gr de nueces picadas
- 450 gr de harina leudante
- 200 c.c. de leche
- 3 claras batidas a punto de nieve

Preparación:

- Mezclar la manteca con el azúcar hasta lograr una crema.
- Agregar las yemas, la esencia de vainilla, la ralladura de limón y las nueces picadas.
- Añadir en forma alternada la harina tamizada y la leche.
- Por último incorporar las claras batidas a punto de nieve, con movimientos suaves y envolventes.
- Verter la mezcla en la cacerola enmantecada.
- Tapar y cocinar a fuego corona hasta finalizar la cocción.

Muy posiblemente muchos de nosotros nos hemos visto siguiendo algoritmos para:

- 1) Armar un juguete tipo LEGO
- 2) Armar un mueble modular

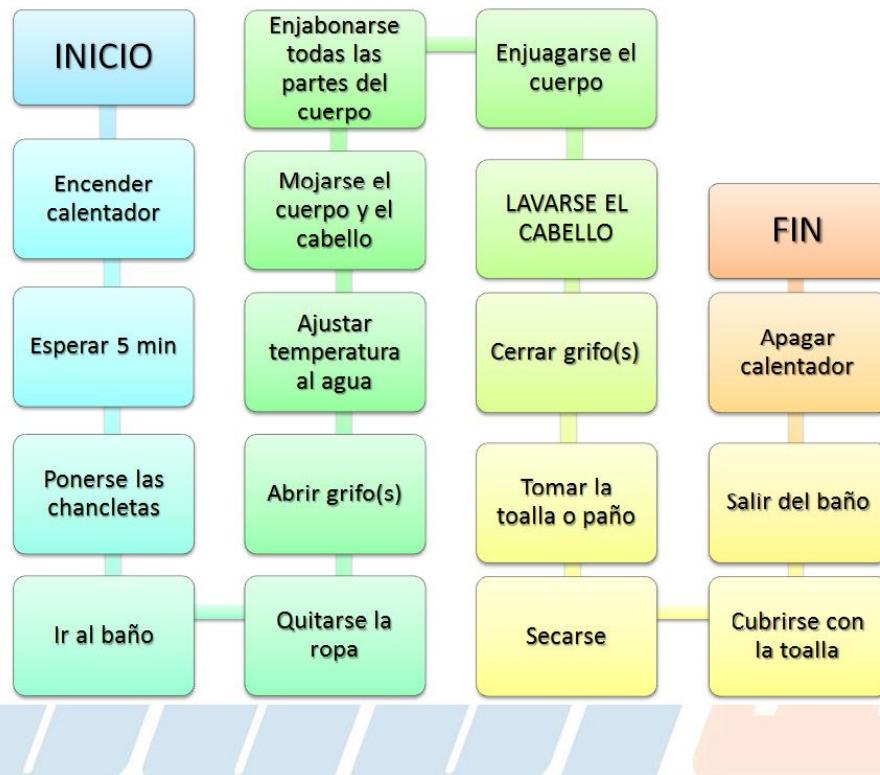


Diariamente todos ejecutamos algoritmos para:

- Comer.
- Vestirse.
- Bañarse.

Veamos este último y apreciemos que como dentro de él se ejecuta un sub-algoritmo.

Igualmente, de manera cotidiana, seguimos algoritmos más complejo, como el de "Prepararse para salir de la casa", que puede implicar la ejecución de otros algoritmos para "Bañarse", "Vestirse" y/o "Comer".



1.3.- Instrucción y Programación

La programación consiste en adaptar algoritmo para que sea entendido por un dispositivo capaz de entenderlo, y cada uno de sus pasos son referidos con el término instrucción.

Una instrucción es una acción específica entendible por el dispositivo programable y en consecuencia realizable o ejecutable por el mismo. IMPORTANTE: El algoritmo es independiente de la implementación en cualquier lenguaje de programación.

Las instrucciones se clasifican según sus tipos:

- 1) Declarativas.
- 2) De Entrada.
- 3) De Salida.
- 4) De Asignación.
- 5) De Control (condicionales y ciclos).

Capítulo 2. ALGORITMOS USANDO BLOQUES

2.1.- Concepto

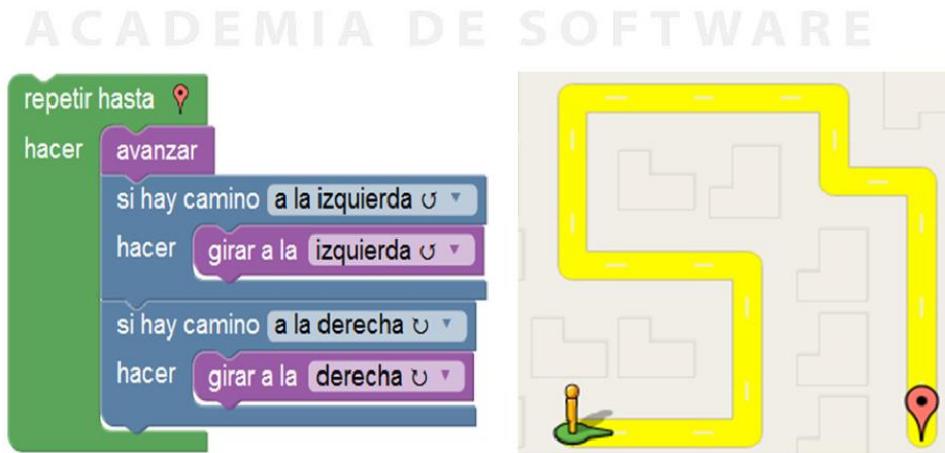
La PROGRAMACIÓN CON BLOQUES es una estrategia lúdica para desarrollar la lógica algorítmica usando la computadora, la cual consiste en conectar bloques, que encajan como piezas de un rompecabezas.

Este tipo de programación se basa en crear un algoritmo usando bloques que conduzcan a lograr el objetivo del escenario presentado.

Por lo anterior, cada bloque que se utilice tiene un efecto en la ejecución del algoritmo que debe ser creado para lograr objetivos establecidos.

Bajo la modalidad simplificada se trabaja con bloques que no están atados a ningún lenguaje de programación, es decir, en el bloque no está escrita la instrucción en la sintaxis de ningún lenguaje, aunque todas ellas puede ser traducidas a un lenguaje de programación particular, por ejemplo, JavaScript.

El siguiente es un ejemplo de un algoritmo hecho con bloques, que guiará al personaje en el mapa hasta llegar al punto de destino:



2.2.- Ejercicios Iniciales Usando Bloques

En la Programación en Bloques, hay bloques cuyo propósito es de naturaleza muy diversa. Cada bloque representa una instrucción.

Por ejemplo, en los ejercicios con los que se practicará en clase el fin último es conducir a un personaje desde un punto origen hasta un punto destino, deberemos usar los bloques cuyo objetivo se explica a continuación:

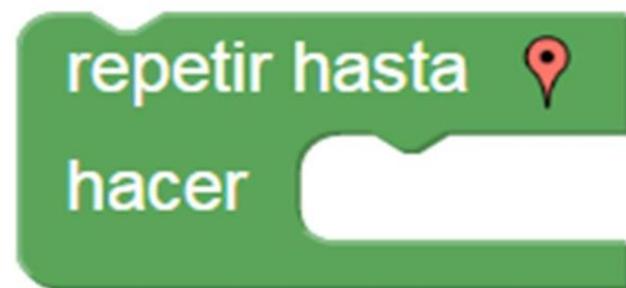
AVANZAR: Hace que el personaje dé un paso en la dirección hace la que esté orientado.

GIRAR A LA IZQUIERDA: Hace que el personaje dé un giro (sin avanzar) a la izquierda.

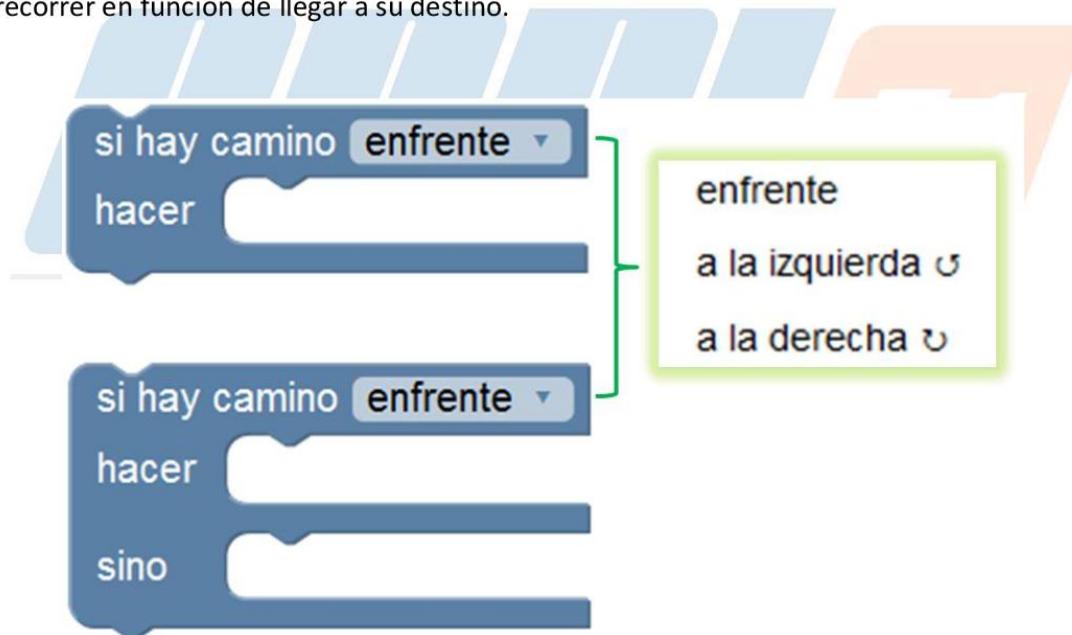
GIRAR A LA DERECHA: Hace que el personaje dé un giro (sin avanzar) a la derecha.



Hay un bloque cuyo comportamiento es radicalmente diferente a los descritos anteriormente. Permite repetir la ejecución de acciones cíclicamente, por su comportamiento es llamado Repetir-Hasta. Gracias a éste bloque, se puede lograr que el personaje repita una o varias acciones hasta llegar al punto destino.



Hay otro tipo de bloque llamado CONDICIONAL, que permitirá condicionar acciones ante el cumplimiento o no de la condición evaluada. En los ejercicios a realizar, permitirán decidir el movimiento del personaje dependiendo de lo que resulte de la evaluación de en qué dirección está el siguiente tramo de camino que el personaje debe recorrer en función de llegar a su destino.

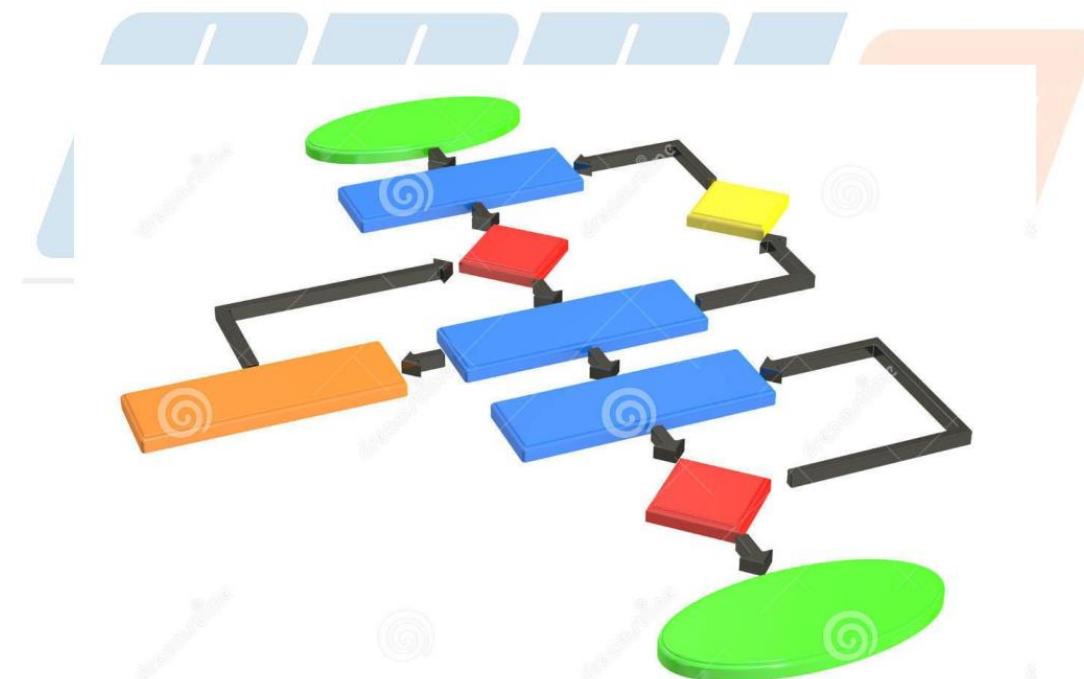


Capítulo 3. FLUJOGRAMAS. PARTE 1

3.1.- Concepto y Objetivo

Al escribir los algoritmos se debe buscar una forma de hacerlos entendibles de forma más intuitiva, de manera tal que se pueda a simple vista determinar si el algoritmo es correcto o no. Existen varias técnicas para hacer esto, una de ellas es emplear flujogramas, donde se muestran los datos que se van a usar y los procesos que se deben realizar durante el algoritmo.

Un FLUJOGRAMA o Diagrama de Flujo, en inglés "flowchart", es una representación gráfica de un algoritmo, la cual debe presentar la información de manera clara, concisa y ordenada. También son llamados Diagramas de Flujos de Datos (DFD).



El objetivo de este tipo de diagrama es presentar la secuencia ordenada de los pasos lógicos, que son necesarios para completar una tarea, para lo cual se debe usar simbología estandarizada en la representación dichos pasos.

Dada la estandarización, cada símbolo tiene un propósito bien definido y casi todos ellos deben tener escrito dentro de él un paso en la secuencia.

Usar flujogramas resulta adecuado cuando se desea visualizar de manera más intuitiva y con mayor detalle un algoritmo, tal como incluir una o más bifurcaciones y/o acciones que deben repetirse (ciclos).

3.2.- Simbología

El ANSI (Instituto Nacional de Estadounidense de Estándares) normalizó el uso de varios símbolos para crear estos diagramas, veamos los más utilizados:

Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y el final de un proceso
	Línea de Flujo	Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida
	Proceso	Representa cualquier tipo de operación
	Decisión	Nos permite analizar una situación, con base en los valores verdadero y falso

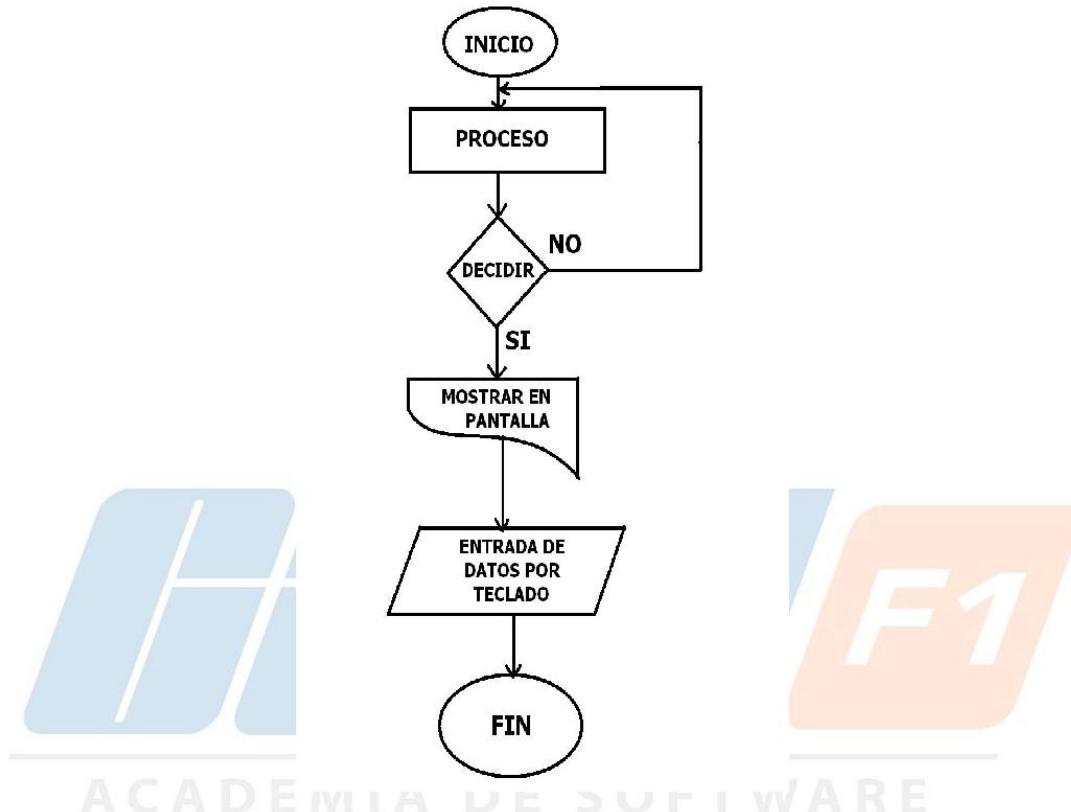
Como fue referido anteriormente cada símbolo tiene un propósito particular y en consecuencia algunos deben contener algo:

Terminador (Inicio/Fin)	<ul style="list-style-type: none"> Dentro del símbolo va la palabra Inicio o Fin según marque el principio o final del algoritmo.
Flechas o Líneas de Flujo	<ul style="list-style-type: none"> Las flechas no contienen texto, pero si pueden ir acompañado de uno, por ej: Si o NO / V o F
Símbolo de Proceso	<ul style="list-style-type: none"> Dentro del rectángulo se escribe la acción que debe realizarse, por ejemplo, una operación aritmética.
Entrada o Salida	<ul style="list-style-type: none"> Dependiendo de lo que se trate dentro del trapecioide debe escribirse: <ul style="list-style-type: none"> Entrada: aquellos datos que deben captarse, por ej: edad. Salida: aquella información que desee mostrarse.
Decisión	<ul style="list-style-type: none"> Dentro del rombo se coloca la condición, la cual será evaluada, para decidir el siguiente paso a ejecutarse.

A modo referencial, en este flujograma genérico se ven representados los símbolos descritos, su propósito.

También se aprecia la incorporación de un símbolo distintivo que representa el despliegue de información por pantalla:

SIMBOLOGÍA BASICA DE LOS DFD



Aquí se pueden apreciar otros símbolos que son usados en los flujogramas.

El símbolo de subprograma, se usa cuando dentro de un algoritmo de necesita representar el uso de otro que es necesario en la secuencia.

Los conectores, que deben identificarse con un número y sirven para darle continuidad al flujograma en otra columna dentro de la misma página (círculo) y en otra página (pentágono).

También distinguen 2 variantes del rombo, que representa posibles decisiones en función de una evaluación:

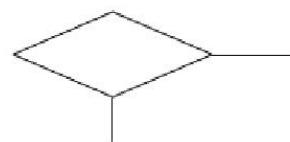
- Con 2 flujos salientes, que representa una bifurcación, donde sólo es posible tomar un camino u otro.
- Con varios flujos salientes, que representa la posibilidad de tomar uno de múltiples caminos.



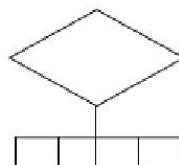
Subprograma



Conectores

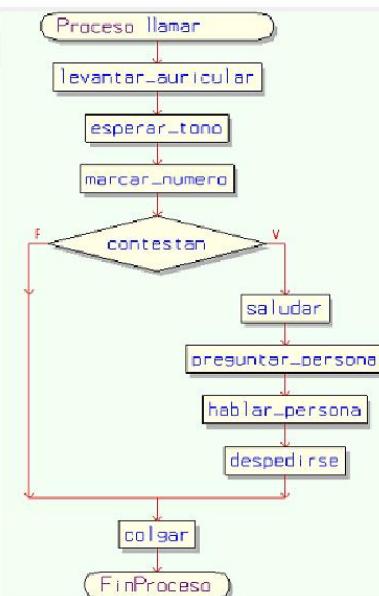


Condición

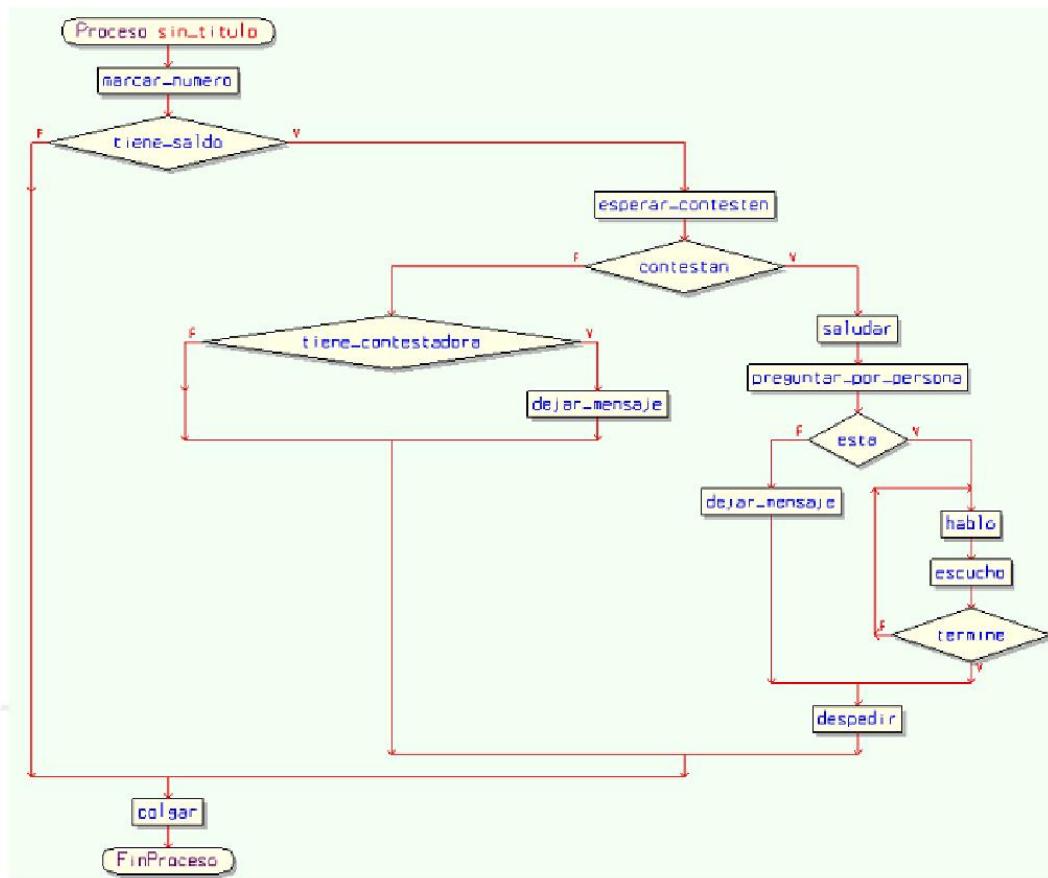


3.3.- Ejemplos de Flujogramas

El siguiente es el flujo de control para llamar por teléfono visto en el capítulo anterior:



El siguiente es el flujograma del algoritmo para llamar por teléfono con unas cuantas instrucciones adicionales:



Capítulo 4. FLUJOGRAMAS. PARTE 2

4.1.- Antes de Diagramar y la Diagramación

Las siguientes son acciones deben ser realizadas antes de la realización del diagrama de flujo:

- 1) Identificar las acciones principales a ser representadas en el diagrama de flujo.
- 2) Identificar los elementos de datos e información que deberán ser usados a través del diagrama.
- 3) Definir qué se espera obtener del diagrama de flujo, es decir, precisar su objetivo.
- 4) Establecer el nivel de detalle requerido.
- 5) Determinar los límites del proceso a describir gráficamente.

Para construir el flujograma se deben seguir los siguientes pasos:

- a) Identificar y listar las principales actividades/subprocesos que están incluidos en el proceso a describir y su orden cronológico.
- b) Si el nivel de detalle definido incluye actividades menores, listarlas también.
- c) Identificar y listar los puntos de decisión.
- d) Construir el diagrama respetando la secuencia lógica y usando los símbolos apropiados.
- e) Una vez finalizado el flujograma, éste debe ser revisado para verificar que está completo y que su ejecución es conducente al logro de el(los) objetivo(s) perseguido(s).

4.2.- Consideraciones de Diagramación

Al hacer flujogramas se deben tomar en cuenta los siguientes lineamientos:

- 1) El diagrama debe mostrar claramente donde inicia y donde termina.
- 2) Cualquier camino que siga debe conducir al fin.
- 3) Los símbolos deben estar organizados de tal forma, que visualmente el flujo vaya de arriba abajo y de izquierda a derecha. Además, ningún símbolo puede quedar sin conexión al diagrama.

4) Sobre las flechas:

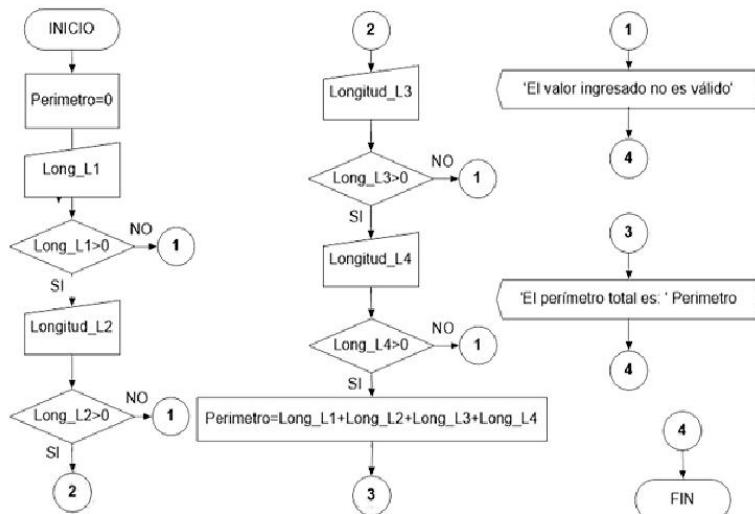
- 4.1) Deben ser verticales u horizontales, nunca diagonales.
- 4.2) No deben cruzarse, para evitar los cruces se utilizan los conectores.
- 4.3) No pueden quedar flechas sin punto de llegada.
- 4.4) A un símbolo solo debe llegar una flecha, si varias flechas se dirigen al mismo símbolo, se deben unir en una sola.

5) Sobre el símbolo de decisión (rombo):

- 5.1) Siempre les debe llegar una sola flecha.
- 5.2) Es el único símbolo desde el que puede salir más de una flecha, según la cantidad de alternativas de decisión.
- 6) Los símbolos de Terminador, y los conectores sólo pueden estar conectados al diagrama por una sola flecha, ya que por su naturaleza es imposible que tenga una entrada y una de salida.

4.3.- Ejemplo de Buena Diagramación

En este flujograma para calcular el perímetro de un cuadrilátero se aprecia el cumplimiento de los lineamientos citados, en particular apréciese el uso de los conectores:



Capítulo 5. LENGUAJE Y PSEUDOLENGUAJE

5.1.- Lenguaje y Pseudolenguaje

Lenguaje de Programación	Pseudolenguaje
Lenguaje artificial que puede ser usado para controlar el comportamiento de una dispositivo, especialmente una computadora	Es un lenguaje de documentación de programas similar al español o al inglés. <i>Su objetivo es tener la forma de escribir un algoritmo neutro a un lenguaje de programación particular</i>
Se componen de un conjunto de reglas sintácticas y semánticas que permiten expresar instrucciones que luego serán procesadas	No hay ningún estándar de la industria definido para pseudolenguaje por lo que cada persona puede desarrollar uno propio.

En un lenguaje de programación el cumplimiento de las reglas, es requerido al momento de escribir un programa que se pretende ejecutar en un dispositivo programable como la computadora. El programador es el encargado de utilizar un lenguaje de programación para escribir un conjunto de instrucciones que, al final, generará un programa o subprograma informático.

Éste le permite al programador especificar de manera precisa:

- sobre qué datos una computadora debe operar,
- cómo deben ser éstos procesados, almacenados y transmitidos, y

- qué acciones debe tomar bajo una variada gama de circunstancias.

Algunos lenguajes de programación

▪ ABAP	▪ Caml	▪ FORTRAN	▪ Lua	▪ Parlog	▪ Seed7
▪ ABC	▪ Clipper	▪ Gambas	▪ MAGIC	▪ Perl	▪ Self
▪ Ada	▪ CLIPS	▪ GML	▪ Mainsail	▪ Python	▪ Sh
▪ ActionScript	▪ CLU	▪ GRAFCET	▪ Mesa	▪ PHP	▪ Simula
▪ Afnix	▪ COBOL	▪ FP	▪ Miranda	▪ PL/I	▪ Smalltalk
▪ ALGOL	▪ CORAL	▪ Haskell	▪ ML	▪ Plankalkül	▪ Snobol
▪ APL	▪ D	▪ Icon	▪ Modula	▪ PostScript	▪ SPARK
▪ ASP	▪ Delphi	▪ Inform	▪ Modula-2	▪ PowerBuilder	▪ Squeak

Los lenguajes de programación se pueden clasificar según varios criterios:

- NIVEL DE ABSTRACCIÓN: se refiere a que tanto se parece un lenguaje de programación a nuestro lenguaje natural. Mientras más similar a nuestro lenguaje de más nivel es el lenguaje, así se distinguen:
 - lenguaje máquina: cadenas de ceros y unos (01110110001...), es un lenguaje de muy bajo nivel
 - bajo nivel, ej.: lenguaje ensamblador
 - nivel medio, ej.: Cobol, C, C++, Pascal, entre otros.
 - alto nivel, ej.: visual basic, java, Delphi.
- PARADIGMA DE PROGRAMACIÓN: se refiere a como se organiza el código y al enfoque estructural que se la da al programa, sus variantes son:
 - No estructurados: lenguaje ensamblador.
 - Estructurados, ej.: Pascal y C.
 - Orientados a Objetos, ej.: C++, Java, php, entre otros.

Adicionalmente los lenguajes han sido desarrollados a través de 5 etapas o generaciones:

- 1era Generación: se crearon lenguajes de máquina.
- 2da Generación: se crearon los primeros lenguajes ensambladores.

- 3era Generación: se crearon los primeros lenguajes de edición y alto nivel. Ej.: C, Pascal, Cobol.
- 4ta Generación: se crearon los lenguajes capaces de generar código por sí solos, son los llamados RAD, con los cuales se pueden realizar aplicaciones sin ser un experto en el lenguaje. En esta etapa se han creado los lenguajes Orientados a Objetos, haciendo posible la reutilización de partes del código para otros programas. Ej. Visual, Natural, AdaBES.
- 5ta Generación: aquí se encuentran los lenguajes orientados a la inteligencia artificial. Estos lenguajes todavía están poco desarrollados. Ej. LISP.

El pseudolenguaje es parecido a un lenguaje de programación de alto nivel, en el sentido de que los términos a utilizar deben ser entendibles por sí mismos, porque deben ser nemotécnicos.

Dado que no necesita seguir ninguna regla específica, no es entendible por el computador, ya que no serán procesables por éste, como si puede serlo un programa creado en algún lenguaje de programación.

El pseudolenguaje tiene la ventaja de que se puede crear fácilmente usando lápiz y papel, o usando cualquier editor de texto en el computador. Además es muy usado para desarrollar la lógica algorítmica computacional o como estrategia para que se puedan entender programadores y no programadores.

5.2.- El Código Fuente y el Ejecutable

El CÓDIGO FUENTE es un conjunto de líneas que conforman un bloque de texto, escrito según las reglas sintácticas de algún lenguaje de programación destinado a ser legible por humanos, en consecuencia éste no es entendible por la computadora, solo por el programador.

Este código fuente es texto plano (sin formato), y puede ser escrito en un simple editor de texto (como el bloc de notas) o usando un programa especializado que reconoce el lenguaje.

```
(  
    // se leen las entradas del problema  
    cout << "Introduzca el numero de autos vendidos:";  
    cin >> nroAutos;  
    cout << "Introduzca el monto de las ventas:";  
    cin >> montoVentas;  
  
}  
  
void calcularSueldo(int nroAutos, float montoVentas, float &salarioNeto)  
{  
    // se hacen los calculos  
    salarioNeto=SALARIO_BASE + (nroAutos*200) + (montoVentas*0.15);  
}  
  
void mostrarSalidas(float salarioNeto)  
{  
    // se muestran las salidas  
    cout << "El salario neto del vendedor es " << salarioNeto << endl;  
    system("PAUSE");  
}  
  
int main(int argc, char *argv[]){  
    int nroAutos;  
    float salarioNeto,  
          montoVentas;  
  
    leerEntradas(nroAutos,montoVentas);  
    calcularSueldo(nroAutos,montoVentas,salarioNeto);  
    mostrarSalidas(salarioNeto);  
  
    return EXIT_SUCCESS;  
}
```

El código fuente está destinado a ser traducido a otro código, llamado código objeto.

En los lenguajes compilados, este proceso se denomina COMPILEACIÓN y permite la generación de archivos ejecutables en la arquitectura del computador para el que fue generado en lenguaje de máquina nativo. Durante el proceso de compilación el código fuente es verificado sintácticamente y semánticamente, si no cumple con las reglas del lenguaje no puede ser compilado.

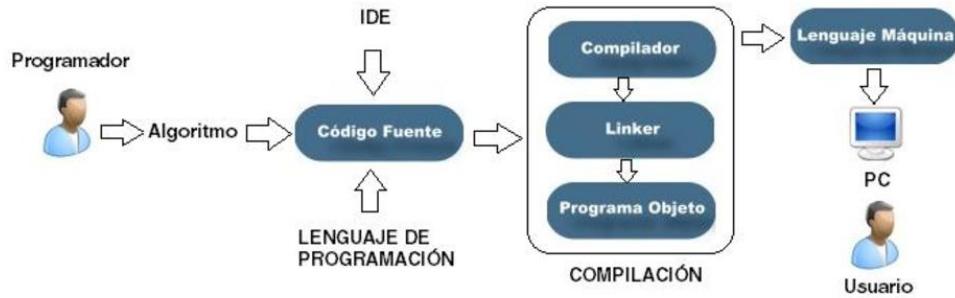
En los lenguajes interpretados, este proceso se denomina INTERPRETACIÓN y permite la generación en caliente de un código intermedio llamado bytecode, que puede ser ejecutado en distintas arquitecturas de computador, gracias a una pieza de software (ej: JVM para Java) que traduce una a una las líneas de bytecode a código de máquina.

Aquí se aprecian unas asociaciones importantes en cuanto al nivel del lenguaje y los tipos de código que acaban de verse:

Lenguajes de Bajo Nivel	Lenguaje Ensamblador	LEGIBLE
	Lenguaje de Máquina	ILEGIBLE

Tipo de código	Entendible por	Nivel del Lenguaje
Fuente	Humanos	Medio / Alto
Ejecutable	Dispositivo Programable	Muy Bajo Nivel

El siguiente esquema muestra el proceso desde la creación del algoritmo que resuelve el problema propuesto hasta que llega a la computadora para ser usado por el usuario final:



5.3.- Entornos de Desarrollo: IDE

Un ENTORNO DE DESARROLLO INTEGRADO conocido bajo el acrónimo IDE (Integrated Development Environment), es un tipo de software especializado utilizado por los programadores para escribir el código fuente en un lenguaje de programación particular.

Estas aplicaciones ofrecen una gama de herramientas que le permitirán al programador ser más eficiente y efectivo en su trabajo.

Los IDE pueden estar diseñados exclusivamente para un lenguaje de programación, lo que en ocasiones hace que se confunda el lenguaje con la herramienta, pero además, hay entornos de desarrollo en los cuales se puede programar para varios lenguajes de programación.

El IDE "Netbeans" se creó originalmente para programar en Java, pero actualmente se puede utilizar para programar también en Ruby, php y C++.

En este mismo orden de ideas, para crear programas bajo lenguaje C++ se pueden usar varios IDE: Dev-C++, Netbeans, TurboC++, entre otros. Así también, Java es un lenguaje de programación, para el que se puede programar empleando distintos IDE como Netbeans, JCreator, Eclipse, etc.

Pascal es un lenguaje de programación que ha evolucionado a través del tiempo que usado entornos de aprendizaje, para programar con este lenguaje se pueden usar los IDE TurboPascal y Delphi. En el IDE Visual Studio .Net de Microsoft se puede programar en C# y Visual Basic, entre otros.

Capítulo 6. PSEUDOCÓDIGO Y PSEINT

6.1.- Pseudocódigo

El PSEUCÓDIGO, es un término usado en el ámbito informático, y se refiere a un lenguaje artificial e informal usado por programadores para el desarrollo de algoritmos usando un Pseudolenguaje particular.

No es un lenguaje de programación verdadero por sé, por lo tanto, no puede ser traducido a lenguaje de máquina, ni ejecutado por el computador.

Existen algunos intérpretes que ejecutan pseudocódigos, pero especialmente con fines didácticos, por ejemplo el programa PSelnt, que permite desarrollar fácilmente la lógica de programación creando pseudocódigo en español.

El siguiente es un ejemplo de un algoritmo escrito en lenguaje natural, llevado a un pseudolenguaje particular:

ALGORITMO ESCRITO EN LENGUAJE NATURAL

- a) Solicitar los datos de un empleado (nombre, tarifa por hora, cantidad de horas trabajadas).
- b) Dependiendo de la cantidad de horas trabajadas a la semana:
 1. Si no es mayor a 40, calcular el sueldo sin horas extras.
 2. Si es mayor a 40, calcular el sueldo considerando que la hora extra se paga con 1.5% más sobre la tarifa por hora
- c) Mostrar el nombre del empleado y el sueldo que debe percibir

ALGORITMO ESCRITO EN PSEUDO LENGUAJE

```

Inicio
Haga Nombre=' ', TarifHr=0
Haga Hrs_Trab=0, Sueldo=0
Mostrar 'Ingrese los datos del empleado'
Capture Nombre, TarifHr,
           Hrs_Trab
Si Hrs_Trab<=40
entonces
    Sueldo=Hrs_Trab*tarifHr
sino
    Sueldo=40*tarifHr+(Hrs_Trab-
        40)*(1.5*tarifHr)
FinSi
Mostrar 'El sueldo de ', Nombre,
           'es ', Sueldo, 'Bs'
Fin

```

En este ejemplo se aprecia el pseudocódigo y el fluograma del algoritmo de hablar por teléfono:

Pseudocódigo:

```

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

Diagrama de flujos:

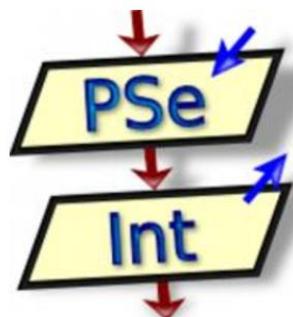


6.2.- Pseudo Intérprete

PSeInt es un software multiplataforma que implementa un pseudolenguaje que permite crear algoritmos en pseudocódigo, permitiendo desarrollar el pensamiento lógico-algorítmico y adquirir conocimiento en los fundamentos de la programación. El pseudocódigo se suele utilizar como primer contacto para introducir conceptos básicos de variables, así como el uso de expresiones, estructuras de control, etc... sin tener que lidiar con las particularidades de la sintaxis de un lenguaje de programación.

El nombre de este programa es un acrónimo de "PSeudo Intérprete" y es distribuido bajo la licencia GPL (software libre). Fue creado por el argentino Pablo Novara, quien lo mejora constantemente y su versión más reciente puede ser descargado desde el sitio web pseint.sourceforge.net.

PSeInt está en español y es muy intuitivo porque está pensado para asistir a las personas que se inician en la construcción de algoritmos computacionales, para luego crear programas. Por lo cual se ha constituido en una herramienta educativa muy popular y ampliamente utilizado en universidades de Latinoamérica y España.



El siguiente es un ejemplo de un pseudocódigo escrito en el pseudolenguaje de PseInt:

```

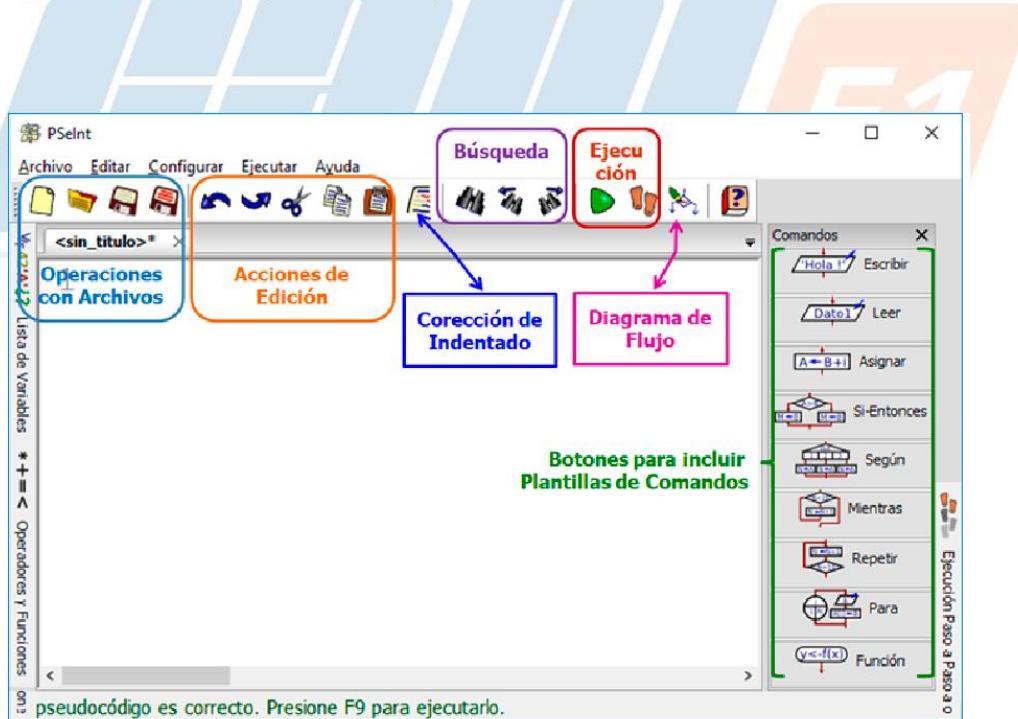
10 Proceso Hanoi
11
12 Dimension torres[3,10], cant_discos[3]
13
14 // pedir y validar cuantos discos colocar en la primer torre
15 Escribir "Ingrese el nro de discos (1-8):"
16 leer discos
17 mientras discos<1 O discos>8 Hacer
18   Escribir "El numero de discos debe ser mayor a 0 y menor a 5:"
19   leer discos
20 finmientras
21
22 // inicializar los datos
23 cant_discos[1]<-discos
24 cant_discos[2]<-0
25 cant_discos[3]<-0
26 Para i<-1 Hasta discos hacer
27   torres[1,i]<-discos-i+1
28 FinPara
29
30 // jugar!
31 cant_movs<-0
32 // mientras no esten todos los discos
33 Mientras cant_discos[3]<>discos Hacer
34
35   Limpiar Pantalla
36

```

Este software pretende facilitarle al principiante la tarea de escribir algoritmos brindándole un conjunto variado de funcionalidades que le ayuden a comprender la lógica de los algoritmos e identificar y corregir errores. Entre sus características más resaltantes están las siguientes:

- Lenguaje Autocompletado.
- Ayudas Emergentes.
- Plantillas de Comandos.
- Graficado, creación y edición de diagramas de flujo.
- Indentado Inteligente.
- Exportación a otros lenguajes tales como: C, C++, C#, Java, PHP, JavaScript, Visual Basic .NET, Python, MatLab.
- Soporta procedimientos y funciones.

En la siguiente lámina se aprecian los aspectos más resaltantes de su interfaz:



6.3.- Palabras Reservadas

Los lenguajes de programación poseen palabras especiales que tienen un significado particular para éste. A estas palabras se les denomina PALABRAS RESERVADAS.

Cada lenguaje de programación posee su propio conjunto de palabras reservadas, aunque algunas de ellas son usadas en casi todos los lenguajes tales como char, integer, include, etc. Estas palabras se deben usar en el código fuente solo para lo que están predefinidas, de lo contrario se producirá un error. Típicamente son palabras del idioma inglés, lo que inicialmente pudiera ser una barrera para algunos, pero luego de familiarizarse con ellas, no causa ningún inconveniente, debido a que son pocas las palabras que se deben aprender.

Uno de los problemas de usar pseudolenguaje es que no es un estándar, por lo tanto, se puede usar cualquier conjunto de palabras sin que ello cause errores. PSeInt como interpreta su pseudolenguaje, posee un conjunto de palabras reservadas escogidas del lenguaje coloquial, lo que ayuda al desarrollo intuitivo del pseudocódigo.

La siguiente es una lista de palabras reservadas que posee PSeInt y que son usadas para escribir algoritmos:

algoritmo
finalgoritmo
definir ...como
leer
escribir
mostrar
si...entonces...sino....finsi
para...finpara
repita ... hasta
mientras ...finmientras

6.4.- Los Comentarios

Al escribir el código fuente es posible que el programador escriba instrucciones cuyo propósito o la razón por las escribió de esa forma no sea fácil recordar posteriormente.

Este problema es mas serio cuando el código será revisado por otras personas o compartido en un equipo de desarrollo.

Para ayudar al programador a entender y documentar el código fuente que se está escribiendo, los lenguajes de programación permiten escribir texto especial no procesado computacionalmente llamado COMENTARIO, éste es ignorado por el verificador sintáctico, razón por la cual, existe libertad en cuanto a lo que se puede comentar. Naturalmente, los comentarios NO son instrucciones del algoritmo.

Cada lenguaje de programación tiene su forma de permitir los comentarios. En PSeInt, se agregan colocando 2 foreslash continuos (//), seguido del comentario.

El siguiente es un ejemplo de texto en comentarios en un algortimo en PSeInt:

```

10 Proceso Hanoi
11
12 Dimension torres[3,10], cant_discos[3]
13
14 // pedir y validar cuantos discos colocar en la primer torre
15 Escribir "Ingrese el nro de discos (1-8);"
16 leer discos
17 mientras discos<1 O discos>8 Hacer
18   Escribir "El numero de discos debe ser mayor a 0 y menor a 5;"
19   leer discos
20 finmientras
21
22 // inicializar los datos
23 cant_discos[1]<-discos
24 cant_discos[2]<-0
25 cant_discos[3]<-0
26 Para i<-1 Hasta discos hacer
27   torres[1,i]<-discos-i+1
28 FinPara
29
30 // jugar!
31 cant_movs<-0
32
33 // mientras no esten todos los discos
34 Mientras cant_discos[3]<>discos Hacer
35
36   Limpiar Pantalla

```

**EJEMPLOS DE
COMENTARIOS**

Capítulo 7. VARIABLES

7.1.- Dato y Variable

Un DATO es cualquier elemento de información que se usa a lo largo de diversas operaciones dentro de un programa o fragmentos de éste. Los datos son los elementos sobre los que se trabaja cuando se efectúa una operación en el computador, es decir, cuando se ejecuta una instrucción de un programa.

Una VARIABLE es básicamente un lugar en la memoria RAM del computador, donde se almacenará un dato temporalmente durante la ejecución del programa.

El valor de una variable puede cambiar durante la ejecución del programa. Es importante entender que por el hecho de que las variables se ubican en la memoria RAM del computador, la información que ellas almacenan es borrada cuando termina el programa o cuando se apaga la computadora.

7.2.- Tipos de Datos

Las variables deben tener asociado un tipo de dato específico, lo cual le indica al computador el rango de valores que dicho dato puede aceptar y las operaciones que sobre él se pueden realizar. El tipo de dato es una definición que agrupa los valores válidos para un conjunto de datos y las operaciones que sobre ellos se pueden realizar.

Las variables son instancias de un tipo de dato determinado. Antes de usar una variable, el programador debe definirla o declararla, indicando su nombre y el tipo de dato al que pertenece. La cantidad de memoria que ocupe la variable dependerá del tipo de dato asociado. Es por esto que se deben declarar las variables que realmente se necesitan y que sean del tipo de dato que más se ajuste a la naturaleza del mismo.

Los tipos de datos se clasifican en:

- PRIMITIVO: tipos básicos o simples predefinidos
- DEFINIDO POR EL USUARIO: agregados por el programador.

Los tipos de datos primitivos más importante son: numéricos, alfanuméricos y lógicos. A continuación se explica cada uno de ellos:

NUMÉRICO, pueden tener signo o no (+ o -) y se distinguen entre:

- Entero: Subconjunto finito matemático de los números enteros.
- Real: Subconjunto finito matemático de los números reales, es decir, aquellos que tienen parte decimal.

ALFANUMÉRICO: Letra(s), Número(s) o Caracter(es) Especial(es) y sus variantes son:

- Cadena (string): se refiere un conjunto de caracteres, por ej.: un nombre, una placa, un código, etc.
- Caracter (char): una letra, número o símbolo.

LÓGICOS O BOOLEANOS: es aquel que solo puede tomar uno de 2 valores: Verdadero o Falso (1 ó 0).

7.3.- Declaración de Variables

La mayoría de los lenguajes de programación exigen declarar las variables antes de usarlas. DECLARAR VARIABLES es indicarle al compilador qué variables se necesitan y de cual tipo de dato serán los valores que está deberá almacenar. En algunos lenguajes de programación el tipo de dato se determina en tiempo de ejecución.

PSelnt tiene una forma particular de declarar variables. El siguiente es un ejemplo de cómo se hacerlo:

```

1  Proceso sin_titulo
2      definir nombre Como Caracter
3      definir edad Como Entero
4      definir sueldo Como Real
5      definir sexo Como Caracter
6      definir esta_casado Como Logico
7
8
9  FinProceso

```

El nombre que se elija para una variable se denomina "identificador".

Al declarar una variable se deben tomar en cuenta los siguientes aspectos:

- Debe comenzar con letra
- No puede tener caracteres especiales (+,-,*,,/)
- No puede ser una palabra reservada ni una función del lenguaje (si, mostrar, entonces)
- No puede estar repetida.
- No puede tener espacios en blanco, y se puede usar el guión bajo (_) para conectar palabras que formen parte del identificador de la variable.

La mayoría de los lenguajes de programación no admiten el uso de acentos ni ninguna letra acompañadas de símbolos, ej.: ñ,á,é,í,ó,ú,ü.

Es muy importante, que los identificadores de las variables sean significativos, de manera que estén relacionados con el valor que van a almacenar, para que así el algoritmo sea más entendible. Éstos pueden ser abreviados, pero se debe tener cuidado que la abreviación no genere confusión.

En la práctica, las variables de un programa serán por lo menos las entradas y las salidas del problema. Existirán otras variables que se necesitaran solo para hacer cálculos aislados, las cuales solo ocuparan memoria temporalmente. Los siguientes son ejemplos de las variables que se usarían en el problema analizado en el capítulo 2 con sus respectivos tipos de dato:

```
1 Proceso agencia autos
2     definir nombre como caracter
3     definir monto_ventas,salario_neto Como Real
4     definir autos_vendidos Como Entero
5
6 |
7 FinProceso
```

Capítulo 8. INSTRUCCIONES BÁSICAS EN PSEUDOCÓDIGO

8.1.- Concepto

En la programación informática, una INSTRUCCIÓN se refiere cada paso del algoritmo en el código fuente. Una de las diferencias entre los lenguajes de programación de alto nivel y los de bajo nivel, es que en un lenguaje de bajo nivel se deben escribir muchas instrucciones para lograr una tarea simple, mientras que en un lenguaje de alto nivel, con una sola instrucción se pueden englobar muchas instrucciones de bajo nivel.

Las instrucciones se pueden clasificar en:

- 1) Declarativas.
- 2) De Entrada.
- 3) De Salida.
- 4) De Asignación.
- 5) De Control (condicionales y ciclos).

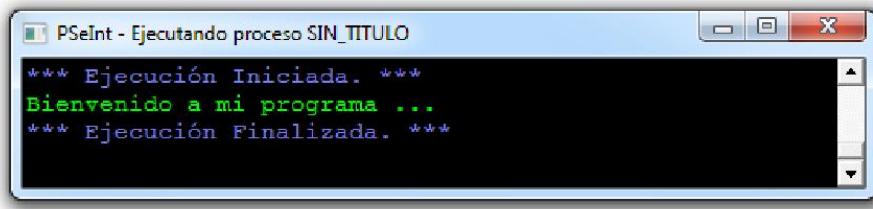
8.2.- Instrucción de Salida

Una instrucción de salida permite al programador mostrar información al usuario por algún dispositivo de salida, típicamente, el monitor. De esta forma se logra que el programa interactúe con el usuario final. Cada lenguaje de programación tiene su forma de mostrar salidas en pantalla. En Pselnt, la instrucción se llama "mostrar" aunque también se puede usar la palabra "escribir", seguido del mensaje que se desea que aparezca en pantalla. El mensaje debe estar encerrado entre comillas dobles. Por ejemplo:

```

1  Proceso sin_titulo
2      escribir "Bienvenido a mi programa ..."
3
4  FinProceso

```



En PSeInt, si se desea mostrar el contenido de una variable, se debe colocar el identificador de la variable, luego de la palabra "mostrar" o "escribir".

8.3.- Instrucción de Entrada

Las instrucciones de entrada le permiten al programador solicitar al usuario algún dato por medio de algún dispositivo de entrada, necesario para iniciar el procesamiento. El dato introducido por el usuario se almacena en una variable, para luego ser utilizado. El dispositivo de entrada que típicamente se utiliza es el teclado. Cada lenguaje de programación tiene sus instrucciones particulares para leer entradas. PSeInt posee una instrucción llamada "leer", seguido de la variable donde se almacenará el dato introducido por el usuario:

```
1 Proceso sin_titulo
2     definir nombre Como Caracter
3
4     mostrar "Introduzca el nombre:"
5     leer nombre
6
7     mostrar "El nombre leido fue ", nombre
8 FinProceso
```

Normalmente, antes de la instrucción de entrada se coloca una instrucción de salida, informando al usuario el dato que debe introducir. La ejecución del programa se detiene en una instrucción de entrada, esperando que el usuario introduzca el dato. Cuando el usuario escribe el dato, debe presionar la tecla "enter" para que el programa continúe con la siguiente instrucción.

El siguiente ejemplo muestra cómo leer las entradas y mostrar las salidas del ejercicio de la agencia de venta de vehículos que se ha venido desarrollando progresivamente:

```
1 Proceso agencia_autos
2     definir nombre como caracter
3     definir monto_ventas,salario_neto Como Real
4     definir autos_vendidos Como Entero
5
6     mostrar "Introduzca el nombre:"
7     leer nombre
8     mostrar "Introduzca el numero de autos:"
9     leer autos_vendidos
10    mostrar "Introduzca el monto de las ventas:"
11    leer monto_ventas
12
13    // se hacen los calculos (proceso)
14
15    mostrar "El sueldo neto es ", salario_neto
16 FinProceso
```

8.4.- Instrucciones Utilitarias

Los lenguajes de programación contienen instrucciones utilitarias que complementan los algoritmos y permiten al programador darle a sus programas algunos efectos que pueden hacer más amigable el uso de los mismos. En Psint, estas instrucciones tienen los siguientes nombres:

Limpiar Pantalla: borra lo que se encuentre en la pantalla de ejecución.

Esperar tecla: Iniciar una pausa hasta que el usuario pulsa cualquier tecla.

Esperar X segundos: Hace de pausa de X segundos y transcurrido el lapso de tiempo especificado, se prosigue con la ejecución.

Capítulo 9. OPERADORES ARITMÉTICOS

9.1.- Operadores Aritméticos

Un OPERADOR ARIMÉTICO es un símbolo que media entre uno o más valores llamados operandos. Ellos son usados en las matemáticas para realizar cálculos algebraicos. Los más comunes son los que permiten obtener resultados de las operaciones Suma (+), Resta (-), Multiplicación (*) y División (/).

Los operandos pueden ser variables o constantes. Los operadores y operandos se conjugan para crear expresiones aritméticas que se deberán plantear en el proceso de creación de algoritmos, para resolver los problemas propuestos.

Ejemplos de expresiones aritméticas son:

sueldobase - deducciones
 radio*3.14
 gananciatotal / 2
 sueldo + comision

El resultado de una expresión aritmética siempre es un número. Éste dependerá de los valores que tengan los operandos involucrados y del operador mismo.

El resultado de las expresiones aritméticas pueden ser mostradas en pantalla directamente en las instrucciones de salida. Por ejemplo:

```

1  Proceso sin_titulo
2    definir horas Como Entero
3    definir tarifa como real
4
5    Mostrar "ingrese la cantidad de horas"
6    Leer horas
7    Mostrar "ingrese la tarifa"
8    Leer tarifa
9
10   Mostrar "el salario es: " horas * tarifa
11  FinProceso
  
```

La potenciación es una operación matemática, donde media un operador aritmético que sirve para obtener la potencia de un número. Obsérvese los elementos de esta operación:

$$A^n = B$$

A → Base
n → Exponente
B → Potencia

Por ejemplo, si se necesita calcular cual es la potencia de 2 elevado a la 10, la expresión y su resultado sería el siguiente:

Exponente
10
Base
= 1024
Potencia

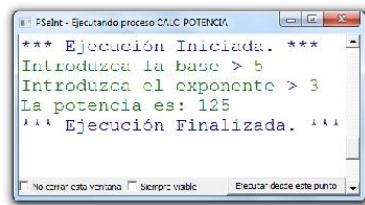
El operador aritmético para determinar una potencia puede cambiar de un lenguaje a otro; los más usados son: ^ (acento circunflejo) y ** (doble asterisco).

En Pselnt se usa el acento circunflejo, su empleo se ejemplifica en el siguiente algoritmo:

```

1 Algoritmo Calc_Potencia
2   definir a,n,potencia Como Entero
3
4   mostrar "Introduzca la base " Sin Saltar
5   leer a
6   mostrar "Introduzca el exponente " Sin Saltar
7   leer n
8
9   mostrar "La potencia es: " a^n
10 FinAlgoritmo

```



9.2.- Precedencia de Los Operadores

Cuando existen varios operadores aritméticos dentro de la misma expresión aritmética, el orden en que se evalúan los operadores dentro de la expresión influye directamente en el resultado que resulta de dicha expresión. Por ejemplo, obsérvese la siguiente expresión aritmética y las posibilidades de su evaluación:

$$2 + 3 * 2 + 3$$

Posibilidad 1: Evaluarla en el orden de aparición de los operandos y los operadores se tendría

$$(((2 + 3) * 2) + 3) = 13$$

Posibilidad 2: Aplicar el operador de la suma y luego el de la multiplicación se tendría >>

$$(2 + 3) * (2 + 3) = 25$$

Posibilidad 3: Evaluar primero la multiplicación y luego los operadores de suma se tendría:

$$2 + (3 * 2) + 3 = 11$$

Entonces...¿Cómo resolver esta situación potencialmente problemática?

La solución a una problemática como la planteada, es evaluar la prioridad entre los operadores, de modo que ante la posibilidad de usar varios operadores aritméticos en una misma expresión, siempre se aplicará primero el de mayor prioridad. Cada lenguaje de programación puede establecer sus propias reglas de prioridad o precedencia de operadores.

El orden de precedencia o importancia definido comúnmente para los operadores aritméticos es el que se aprecia en la imagen.

Además, es posible utilizar los paréntesis () para asociar aquellas operaciones que se desea priorizar, de esta manera se puede definir y cambiar el orden en que se evalúa una expresión aritmética dada.

<i>Prioridad de los operadores aritméticos (de mayor a menor) en pseudocódigo:</i>	
+ -	Signos más y menos
** ^	Potencia
* / div mod	Multiplicación, división real, división entera y módulo
+ -	Suma y resta

Entre dos operaciones que tienen la misma precedencia, para resolver la disyuntiva hay que usar la regla de la asociatividad. La más típica, es la asociatividad por la izquierda (evaluar primeramente lo que está más a la izquierda).

Considerando lo anterior, la expresión: $2+3 * 2 / 3 - 5^2$ asociativamente será evaluada así: $(2 + ((3 * 2) / 3)) - (5^2)$.

ACADEMIA DE SOFTWARE

En PSeInt

$$\begin{array}{l}
 2 + 3 * 2 / 3 - 5^2 \\
 2 + 3 * 2 / 3 - 25 \\
 \text{Asociatividad} \\
 \text{por la izquierda} \\
 2 + \underline{6} / 3 - 25 \\
 2 + 2 - 25 \\
 \underline{4} - 25 \\
 -21
 \end{array}$$

Proceso sin_titulo

mostrar 2+3*2/3-5^2

FinProceso

```

PSeInt - Ejecutando proceso SIN_TITULO
*** Ejecución Iniciada. ***
-21
*** Ejecución Finalizada. ***

```

9.3.- Expresiones no Lineales

Las expresiones que tradicionalmente se usan en las matemáticas, también son usadas en la programación, sólo que se deben escribirse de forma linealizada. Obsérvese este ejemplo de una expresión matemática fraccional con una variable "x" y como escribirla linealmente:

$$\frac{x+1}{2} + \frac{x+3}{4} + \frac{x}{5}$$

Es equivalente a:

$$((x+1)/2) + ((x+3)/4) + x/5$$

Si la expresión es más compleja, su equivalente en forma algorítmica depende del conocimiento de la precedencia de los operadores y del uso apropiado de los paréntesis. El siguiente es un ejemplo de una expresión matemática con coeficientes, donde "a" y "b" son variables (datos de entrada posiblemente):

¿Serán necesarios todos los paréntesis?

$$\frac{\frac{1}{6}a + \frac{5}{36}ab - \frac{1}{6}b}{\frac{1}{3}a + \frac{1}{2}b}$$

Es equivalente a:

$$(((1/6)*a)+((5/36)*a*b)- ((1/6)*b)) / (((1/3)*a) + ((1/2)*b))$$

Siguiendo el criterio explicado anteriormente, en las expresiones matemáticas que pueden incluir potencias, tanto en el numerador como en el denominador, debe usarse el operador para expresar la potencia aplicado a las sub-expresiones que corresponda. El siguiente es un ejemplo de una expresión con potenciación:

$$\frac{(x+2)^4 - a^4}{x^2 - 2ax + a^2}$$

Es equivalente a:

$$((x+2)^4 - a^4) / (x^2 - (2*a*x) + a^2)$$

Para linealizar una expresión radical, como la que se muestra a continuación se debe plantear la raíz como un exponente fraccionario, de numerador 1 y como denominador el índice de la raíz, siendo la base es la expresión dentro de la raíz.

$$\sqrt[4]{32x^4y^{21}z^{43}}$$

$$((32*x^4)*(y^{21})*(z^{43}))^{(1/4)}$$

Capítulo 10. EXPRESIONES EN CÁLCULOS COMUNES

10.1.- Introducción

Existen operaciones básicas que se realizan para resolver problemas típicos. Entre ellas están:

- promedios
- porcentajes
- aumentos
- descuentos

10.2.- Promedios

El PROMEDIO se obtiene al sumar todas las variables, cuyo resultado debe dividirse entre la cantidad de variables que se desean promediar. La forma general del cálculo es: $(var1 + var2 + var3 + \dots + varN) / N$

Donde "N" es el número de variables.

Por ejemplo, si se tienen 3 notas de un alumno en la misma escala la fórmula para calcular el promedio sería: $(nota_exam1 + nota_exam2 + nota_exam3) / 3$, y este sería el algoritmo:

```

1 Algoritmo Prom_Notas
2   Definir nota_exam1,nota_exam2,nota_exam3 como Real
3
4   Mostrar "Ingrese las notas de los 3 exámenes en la escala de 20 ptos..."
5   Leer nota_exam1,nota_exam2,nota_exam3
6
7   Mostrar "El promedio obtenido fue: " (nota_exam1+nota_exam2+nota_exam3)/3 "/20 ptos"
8 FinAlgoritmo

```

10.3.- Porcentajes

Los PORCENTAJES se obtienen al multiplicar un valor por el porcentaje en cuestión (expresado numéricamente), por ejemplo, el 12% del precio de un artículo se obtiene de la siguiente forma:

precio * 12 / 100. También se puede multiplicar por 0.12 (que es el resultado de 12/100).

Nótese que no se usa el símbolo de porcentaje (%) debido a que en algunos lenguajes de programación es un operador aritmético (mod) o lo usan para otros fines.

```

1 Algoritmo porcent_de_precio
2   Definir precio como real
3
4   Mostrar "Ingrese al precio al cual desea calcularle el 12%" Sin Saltar
5   Leer precio
6   // La primera forma de calcular el 12% es multiplicando por 12/100
7   Mostrar "El 12% del precio es: " precio*(12/100)
8   // La segunda forma de calcular el 12% es multiplicando por 0.12
9   Mostrar "El 12% del precio es: " precio*0.12
10 FinAlgoritmo

```

10.4.- Aumentos y Descuentos

Los cálculos de aumentos y descuentos son muy comunes. Estos aumentos y descuentos pueden depender de distintos valores:

- un valor (ej.: monto) constante
- un porcentaje del valor original

También es frecuente generar un valor aumentado por:

- un múltiplo del valor original (duplicar, triplicar)

Las expresiones de aumento se ejemplifican a continuación:

```

1 Algoritmo Aumentos_Sobre_Precio
2   Definir precio como Real
3
4   Mostrar "Ingrese el precio" Sin Saltar
5   Leer precio
6   Mostrar "---- Resultados ---"
7   // Aumento de una cantidad constante, por ej: 1000
8   Mostrar "Precio + 1000= " precio + 1000 " Bs."
9   // Duplicado del precio
10  Mostrar "Doble del precio: " precio * 2 " Bs."
11  // Triplicado del precio
12  Mostrar "Precio triplicado: " precio * 3 " Bs."
13  // Aumento de un porcentaje
14  Mostrar "Precio con un 25% de aumento: " precio + precio * 25/100 " Bs."
15  Mostrar "Precio con un 25% de aumento: " precio + precio * 0.25 " Bs."
16  Mostrar "Precio con un 25% de aumento: " precio * 1.25 " Bs."
17 FinAlgoritmo

```

```

PSeInt - Ejecutando proceso AUMENTOS_SOBRE_PRECIO
*** Ejecución Iniciada. ***
Ingrese el precio> 7500
--- Resultados ---
Precio + 1000= 8500 Bs.
Doble del precio: 15000 Bs.
Precio triplicado: 22500 Bs.
Precio con un 25% de aumento: 9375 Bs.
Precio con un 25% de aumento: 9375 Bs.
Precio con un 25% de aumento: 9375 Bs.
*** Ejecución Finalizada. ***

```

Las expresiones de descuento se ejemplifican a continuación:

```

1 Algoritmo Aumentos_Sobre_Precio
2   Definir precio como Real
3
4   Mostrar "Ingrese el precio" Sin Saltar
5   Leer precio
6   Mostrar "---- Resultados ---"
7   // Descuento de una cantidad constante, por ej: 1000
8   Mostrar "Precio - 1000= " precio - 1000 " Bs."
9   // Descuento de un porcentaje
10  Mostrar "Precio con un 25% de descuento: " precio - precio * 25/100 " Bs."
11  Mostrar "Precio con un 25% de descuento: " precio - precio * 0.25 " Bs."
12  Mostrar "Precio con un 25% de descuento: " precio * 0.75 " Bs."
13 FinAlgoritmo

```

```

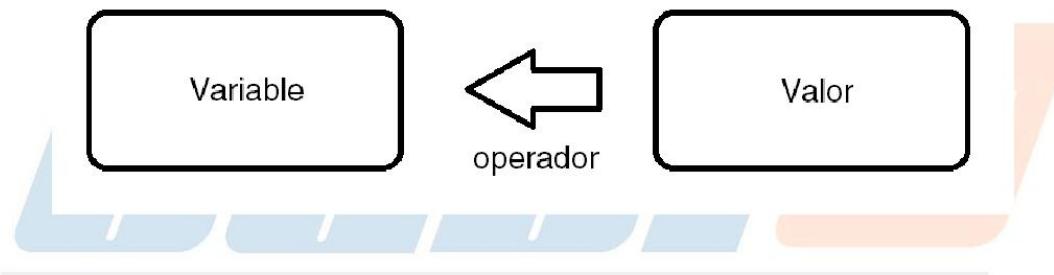
PSeInt - Ejecutando proceso AUMENTOS_SOBRE_PRECIO
*** Ejecución Iniciada. ***
Ingrese el precio> 6000
--- Resultados ---
Precio - 1000= 5000 Bs.
Precio con un 25% de descuento: 4500 Bs.
Precio con un 25% de descuento: 4500 Bs.
Precio con un 25% de descuento: 4500 Bs.
*** Ejecución Finalizada. ***

```

Capítulo 11. ASIGNACIONES

11.1.- Operador de Asignación

La operación de ASIGNACIÓN consiste en atribuir un valor a una variable. El operador de asignación puede variar de lenguaje en lenguaje. En la mayoría se utiliza el símbolo de igualdad (=). En PseInt se puede usar dicho signo, una flecha apuntando a la izquierda (<-) y la combinación dos puntos e igual (:=). En la siguiente imagen se demuestra lo que ocurre en una asignación, donde "Variable" representa la variable a la que se le asigna el valor dado en la expresión a la derecha del operador de asignación:



El valor asignado a una variable puede ser:

- un valor constante o literal.
- el valor de otra variable.
- una expresión, que pudiera incluir a la variable objeto de la asignación.

En una operación de Asignación deben tenerse en cuenta los siguientes aspectos:

- En la parte izquierda de la operación de asignación, solamente puede haber una variable.
- La variable a la que se asigna el valor pierde su valor anterior si lo tuviese (se sobreescribe su valor).
- El tipo de dato del valor de la parte derecha de una instrucción de asignación, tiene que ser del mismo tipo de dato o de un tipo compatible, con el del la variable a la cual se le va a asignar el resultado de la operación. Ej: la variable a la

izquierda del operador debe ser real, si los valores y/o variables en la expresión a la derecha del operador son de tipo entero o real.

- Si al lado derecho del operador hay una expresión, ésta se evalúa antes de realizar la asignación.
- Si la variable a la que se le asigna un valor (parte izquierda) participa en la expresión a evaluar (parte derecha), como por ejemplo en la sentencia:
 $x = x + 1$, entonces como primero se evalúa la expresión antes de realizar la asignación, el valor usado en la expresión es el que tenía la variable antes de la asignación. En éste ejemplo, si x tenía un valor igual a 5, entonces la evaluación de la expresión $x = x + 1$ sería $x = 5 + 1$, quedando x con un valor final de 6.

11.2.- Asignación de Valores Literales

Un VALOR LITERAL es una constante de un tipo de dato particular. Los literales pueden ser de varios tipos:

- Numérico: es un número cualquiera, por ejemplo: 55, 8600, 3.14 (No se utilizan puntos para separar miles debido a que el punto es el separador de decimales).
- Alfanumérico: es un carácter o una cadena de ellos y se debe encerrar entre comillas dobles o simples.
- Lógicos: es uno de los 2 valores: Verdadero o Falso.

ACADEMIA DE SOFTWARE

```

1  Proceso sin_titulo
2      definir nombre Como Caracter
3      definir edad Como Entero
4      definir sueldo Como Real
5      definir sexo Como Caracter
6      definir esta_casado Como Logico
7
8      nombre="jose luis"
9      edad = 35
10     sueldo = 7560.55
11     sexo="M"
12     esta_casado = Falso
13
14 FinProceso

```

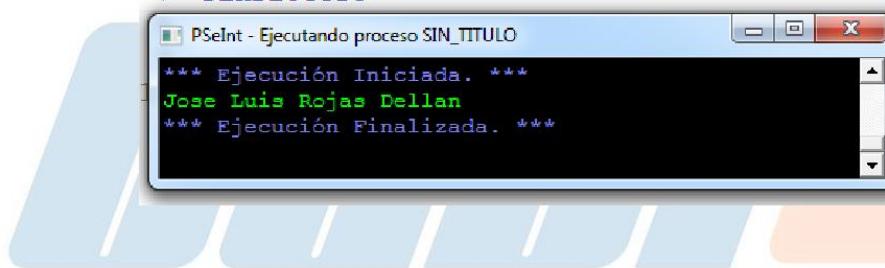
Para hacer asignaciones, PSeInt permite el uso tanto del símbolo "=" como la flecha "<-". En el ejemplo anterior se muestra cómo hacer asignaciones de valores literales a unas variables (previamente definidas).

En el caso de asignación de literales alfanuméricos, al mostrar la variable, lo que está entre las comillas aparecerá literalmente en la pantalla. El nombre de la variable NO debe estar encerrado entre las comillas. Por ejemplo:

```

1 Proceso sin_titulo
2     definir nombre como caracter
3
4     nombre = "Jose Luis Rojas Dellan"
5     mostrar nombre
6
7 FinProceso

```

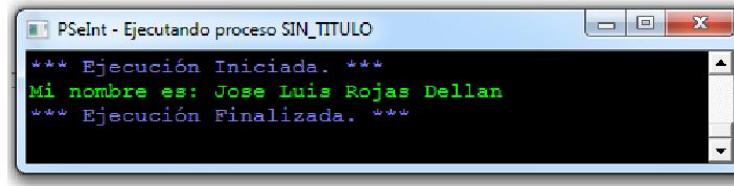


Es muy frecuente la necesidad de mostrar mensajes fijos combinados con variables. Lo cual se ejemplifica a continuación: La coma entre el literal alfanumérico y el nombre de la variable es un operador de concatenación.

```

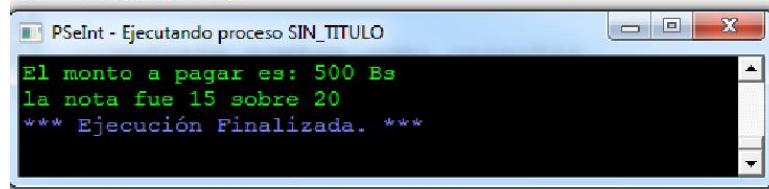
1 Proceso sin_titulo
2     definir nombre como caracter
3
4     nombre="Jose Luis Rojas Dellan"
5     escribir "Mi nombre es: ", nombre
6
7 FinProceso

```



Para mostrar el valor de variables se pueden combinar varios mensajes fijos y varias variables en una misma instrucción (línea 9). Por ejemplo:

```
1 Proceso sin_titulo
2     definir monto Como Real
3     definir nota, maximo como entero
4
5     monto = 500
6     nota = 15
7     maximo = 20
8     escribir "El monto a pagar es: ", monto, " Bs"
9     escribir "la nota fue ", nota, " sobre ", maximo
10 FinProceso
```



Es muy importante cuando se asignan literales que exista coincidencia del tipo de literal, con el tipo de dato de la variable objeto de la asignación, de lo contrario, ocurrirá un error.

IMPORTANTE: El hecho de que una variable tome un valor literal, no implica que no pueda cambiar. Por eso se llama variable.

El siguiente es un ejemplo de asignación incorrecta de literales:

```

1  Proceso sin_titulo
2      definir nombre Como Caracter
3      definir edad Como Entero
4      definir sueldo Como Real
5      definir sexo Como Caracter
6      definir esta_casado Como Logico
7
8 +     nombre=jose_luis
9     edad = 35.55
10 +    sueldo = "7560.55"
11    sexo=M
12 +    esta_casado = "Verdadero"
13
14  FinProceso

```

Los errores cometidos en el ejemplo anterior son los siguientes:

- En la línea 8 la asignación es incorrecta debido a que el valor no está entre comillas.
- En la línea 9, la variable "edad" está declarada como entero y se le está asignando un valor real (con decimales).
- En la línea 10, la variable sueldo está declarada como real y se le está asignando un valor entre comillas, lo que indica que es un alfanumérico.
- En la línea 11, se haciendo referencia a una variable llamada M que no existe. Si lo que se desea es asignar el valor "M" debe estar entre comillas.
- En la línea 12, a una variable de tipo Lógico se le está tratando de asignar el literal alfanumérico Verdadero ya está entre comillas, lo cual incorrecto.

PSelInt no subraya todos los errores, porque en algunos casos como el de la línea 9, internamente sólo se queda con la parte entera del número asignado y en el caso de la línea 11, porque asume que M es una variable auto-definida.

11.3.- Asignación de Variables

Otra forma de hacer asignaciones es usar una variable para darle el valor de otra variable. Es esencial que las variables involucradas en una expresión de asignación sean del mismo tipo de dato o de un tipo compatible, como pasa con los números. El

siguiente es un ejemplo de cómo hacer asignaciones tomando el valor de otras variables:

```

1  Proceso sin_titulo
2      definir nombre_1,nombre_2,ganador Como Caracter
3      definir sueldo,sueldo_anterior Como Real
4
5      nombre_1="jose luis"
6      nombre_2="maria laura"
7      ganador=nombre_1
8
9      sueldo=7500
10     sueldo_anterior=sueldo
11     sueldo=8500
12
13 FinProceso

```

En la línea 7, la variable ganador está recibiendo el valor de la variable nombre_1. Ganador tomará el mismo valor que tiene la variable nombre_1, lo que hará que momentáneamente ambas contengan el mismo valor. Si posteriormente en el algoritmo, alguna de las 2 variables involucradas en la asignación cambian de valor, la otra no se ve afectada.

Asimismo, se pueden cometer errores asignando una variable a otra si no son del mismo tipo o son incompatibles. Los siguientes son ejemplos de asignaciones erradas:

```

1  Proceso sin_titulo
2      definir nombre_1,nombre_2,ganador Como Caracter
3      definir sueldo,sueldo_anterior Como Real
4
5      nombre_1="jose luis"
6      nombre_2=maria
7      ganador="nombre_1"
8
9      sueldo=7500
10 +     sueldo_anterior=ganador
11     sueldo=sueldo
12
13 FinProceso

```

Los errores que se cometieron en el ejemplo son:

- En la línea 6, la variable maria no existe. Si lo que se desea es que tome el valor "maria", se debe colocar entre comillas.
- En la línea 7, se colocó la variable nombre_1 entre comillas. Si lo que se desea hacer es que la variable ganador tome el valor que contiene la variable nombre_1, no debe colocarse las comillas.
- En la línea 10, las variables "sueldo_anterior" y "ganador" no son del mismo tipo de dato.
- En la línea 11, la variable "sueldo" es asignada a ella misma, lo cual es totalmente innecesario, debido a que una variable mantiene su valor hasta que no haya una instrucción que haga lo contrario.

11.4.- Asignación de Expresiones

A una variable le puede ser asignada el resultado de una expresión. En este caso, la expresión se resuelve usando los criterios de precedencia de operadores y/o paréntesis, luego el resultado de la expresión se guarda en la variable. Esto es especialmente útil cuando se necesitan hacer cálculos y almacenarlos para usarlos varias veces posteriormente. En el siguiente ejemplo, las variables bono1 y bono2 almacenan el resultado de unas expresiones matemáticas, porque se necesita usar esos resultados para calcular el sueldo final, para luego ser mostrados en pantalla:

```
ACADEMIA DE SOFTWARE

bono1 = monto_ventas * 10 /100
bono2 = nro_autos * 200000
sueldo_final = 500000 + bono1 + bono2

mostrar "El bono por ventas fue ", bono1
mostrar "El bono por autos fue ", bono2
mostrar "El sueldo final fue de ", sueldo_final
```

Por otra parte, también es posible que en una expresión esté involucrada la misma variable que va a recibir el valor como resultado de la asignación. En ese caso, se utiliza en la expresión el antiguo valor de la variable, se evalúa el resultado y luego se efectúa la asignación, modificando así su valor. Por ejemplo:

```
// se aumentan 100 Bs al precio  
precio = precio + 100  
// se duplica el precio  
precio = precio * 2  
// se aumenta el precio 10%  
precio = precio + precio*0.1
```



Capítulo 12. RESOLUCIÓN DE PROBLEMAS Y ALGORITMOS

12.1.- Problema y su Resolución

Conceptualmente un PROBLEMA es:



Los pasos para resolver un problema están destacados en la siguiente imagen:

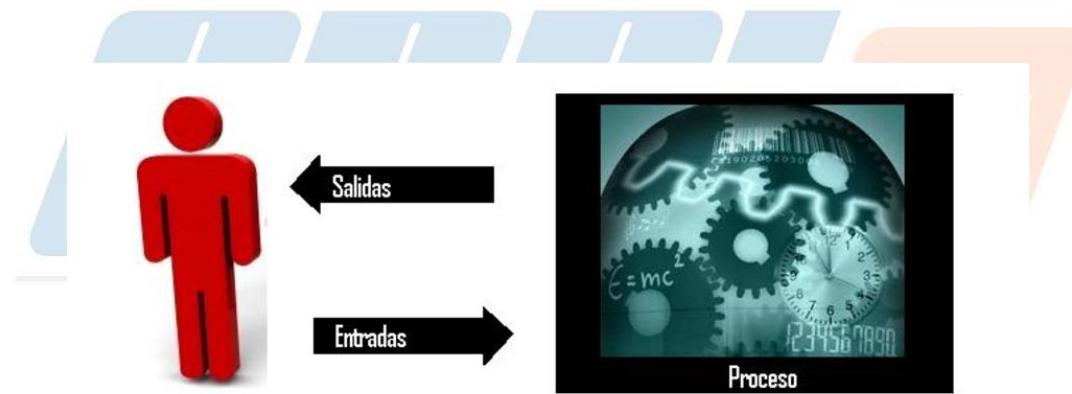


En el caso de un problema que puede ser resuelto computacionalmente, el plan a diseñar un algoritmo, que debe ser seguido, para luego, examinar si éste resuelve la situación planteada como problema o si debe ser corregido.

12.2.- Entradas, Proceso y Salidas

Para resolver problemas mediante un dispositivo de cómputo, debe visualizarse en función del comportamiento de un algoritmo, que luego se traducirá en programas de computadoras.

Los elementos esenciales en la interacción con un programa o sistema informático son los siguientes: Entradas, Proceso y Salidas. En la imagen se destaca la figura del usuario, sin embargo, puede presentarse interacción entre sistemas informáticos, sin la intervención del usuario.



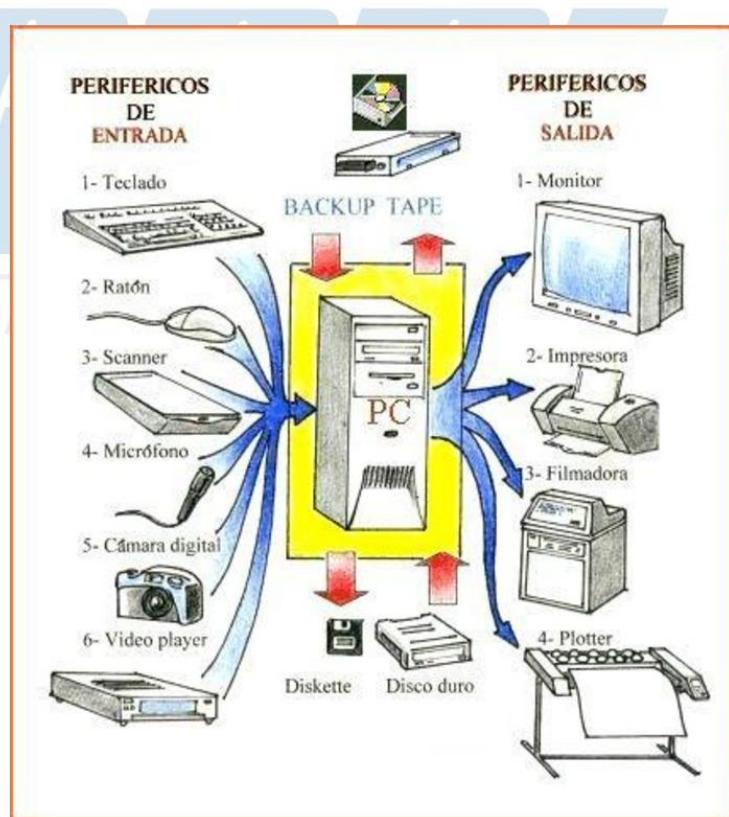
Desde la perspectiva computacional, estos elementos se definen de la siguiente manera:

- **ENTRADAS:** son los datos que el usuario o sistema le provee al programa para iniciar el algoritmo que fue implementado en el código fuente.
- **PROCESO:** son los pasos, tareas u acciones que se deben realizar para usar las Entradas y transformarlas en información útil para el usuario o el sistema que amerita un procesamiento de datos.

- **SALIDAS:** son el resultado del procesamiento realizado en el paso anterior, y que de alguna manera deben ser utilizados para mostrarlos al usuario o provisionarlos a un sistema solicitante.

Las entradas requeridas por un programa de computadoras que dependen del usuario pueden ser provistas mediante los Periféricos de Entrada:

- teclado,
- ratón,
- scanner,
- webcam,
- micrófono,
- lectores digitales: código de barras, huella dactilar, entre otros.



Las salidas se ponen a disposición del usuario a través de los periféricos de Salida:

- monitor,
- cornetas / audífonos,
- impresora, entre otros.

Para efectos del curso solo se usará como dispositivo de entrada el teclado, el ratón y como dispositivo de salida el monitor.

12.3.- Fases de la Resolución de Problemas

Como se ha citado previamente, un algoritmo es una fórmula para obtener la solución de un problema, esta fórmula puede ser planteada de diferentes formas y cada una de estas ellas puede efectivamente resolver el problema planteado.

Dado lo anterior, el diseño de un algoritmo es un proceso creativo personalísimo, ya que no existe un conjunto de reglas que indiquen expresamente como formular un único algoritmo correcto, por el contrario, pueden existir varias maneras de resolver un problema; en consecuencia pueden crearse varios algoritmos que implementen una solución correcta. Sin embargo, hay una serie de fases que permiten resolver un problema de la forma más conveniente, estos son:

- Análisis del problema (Entender el problema).
- Diseño del algoritmo (Trazar el plan).
- Verificación del algoritmo (Ejecutar el plan y revisar sus resultados).

En la fase del ANÁLISIS DEL PROBLEMA el objetivo es entenderlo plenamente. Para lo cual éste debe ser examinado cuidadosamente para responder a las siguientes interrogantes:

- ¿qué tipo de información se necesita producir como salida? (S)
- ¿qué datos e información se necesita para encontrar la solución? (E)
- ¿qué debe hacerse para procesar aquello de lo que se dispone para generar lo que se espera? (P)

En otras palabras, es importante considerar lo que se está pidiendo satisfacer en el planteamiento del problema, con que elementos se cuenta y cuales se deben solicitar para establecer el camino a seguir para la solución del problema.

Luego de identificar las entradas, las acciones de procesos a realizar, y las salidas a generar, se procede a cumplir con la fase de DISEÑO DEL ALGORITMO.

Una vez finalizado, el algoritmo de "correrse" en frío, lo cual debe hacerse varias veces, para cumplir con la VERIFICACIÓN del mismo, para determinar si no falta nada y si se está hace todo lo necesario.

En resumen, en el Análisis del Problema básicamente buscaremos detectar tres (3) elementos:

- las entradas,
- las salidas,
- el proceso necesario para convertir las entradas en salidas.

La identificación de estos tres (3) elementos es fundamental para la correctitud lógica en la creación del algoritmo y su posterior codificación de un programa de computadora.

Para detectarlos no existe una secuencia estándar, en esencia, se espera que el analista emplee una estrategia estructurada de análisis de los problemas, que espera convierta en algoritmos que puedan ser ejecutados por el computador. Sin embargo, de lo referido anteriormente se puede inferir una estrategia intuitiva asociada con la identificación de estos elementos siguiendo la siguiente secuencia: S -> E -> P

El siguiente es un ejemplo de un planteamiento de problema, que es objeto de análisis para realizar identificación de los 3 elementos (entrada, proceso y salida):

Una agencia de carros paga a su personal de ventas un salario base de 500.000 Bs. más una comisión de 200.000 Bs. por cada automóvil vendido. Adicional se le paga a cada vendedor el 10% del valor total de las ventas. Si se tiene como entrada el nombre del vendedor, cantidad de autos vendidos y el monto total de sus ventas. Se pide mostrar el nombre del vendedor y el sueldo final a percibir.

Al analizar el problema se identifican:

- Entradas: nombre del vendedor, cant. de autos vendidos y monto de las ventas
- Salidas: Nombre del Vendedor y su Sueldo Final
- Proceso: El sueldo final se puede obtener procesando la siguiente expresión aritmética:

$$500000 + \text{AutosVend} * 200000 + \text{MontoVtas} * 0.10$$

12.4.- Tabla de Variables

Para complementar el análisis se recomienda la construcción de la Tabla de Variables, a través de la cual se facilitará la identificación de todas las variables que deberán utilizarse en el algoritmo que se elaborará para dar solución a un problema:

Identificador de la Variable	Tipo de Dato asociado	¿Dónde se usa la variable?		
		E	P	S

Para aumentar el grado de certidumbre del análisis se recomienda construir la Tabla de Variables, recordando que los identificadores asignados a las variables deben ser significativos:

Identificador de la Variable	Tipo de Dato asociado	¿Dónde se usa la variable?		
		E	P	S
Nomb_Vend	Caracter	X		X
Aut_Vend	Entero	X	X	
Mto_Vtas	Real	X	X	
Sueldo_Final	Real		X	X



Capítulo 13. ALGORITMOS SECUENCIALES

13.1.- Introducción

Hasta ahora se han visto en progresión, todos los elementos que contiene un programa de computadora:

- Variables y sus declaraciones,
- Tipos de Instrucciones,
- Operadores / Operandos Básicos y
- Expresiones Aritméticas.

Por ello, es el momento de hacer un programa completo.

Es importante tener presente que al diseñar algoritmos se aplica un concepto muy propio de la programación: Flujo de Control de un Programa. El cual se refiere al orden en que se ejecutan las sentencias que lo componen.

En lo que se verá a continuación, ese flujo de control es secuencial, porque cada instrucción se ejecuta siempre que se corra el programa.

13.2.- Construcción Progresiva: Algoritmo y Programa

Como se presentó con anterioridad, el 1er paso previo al diseño del algoritmo es realizar el análisis del problema con el objeto de identificar cuales son las variables de entrada, salida y proceso, que deben ser usadas e identificar las acciones de proceso que deben ser realizadas.

Luego, se puede Diseñar el Algoritmo, y al escribirlo en un lenguaje de programación particular (también aplicable al pseucódigo), lo más recomendable es construirlo en la siguiente secuencia:

- 1) Definir las variables.
- 2) Leer las entradas.
- 3) Colocar las acciones de proceso.
- 4) Mostrar las salidas.

En la construcción de cualquier programa (luego de diseñar el algoritmo), el primero paso es la DECLARACIÓN DE VARIABLES que se van a utilizar. En algunos lenguajes de programación, no es necesario declarar las variables, porque las variables son autodefinidas.

Luego se escriben las líneas de código relacionadas con la OBTENCIÓN DE LAS ENTRADAS, luego el cálculo o determinación de las salidas, que son LAS ACCIONES DE PROCESO y al final se colocan las INSTRUCCIONES DE SALIDA.

Este orden no es estrictamente obligatorio, pero aplicarlo es lo más recomendable. Así como también es recomendable diferenciar bien dentro de cual bloque va cada instrucción:

- Declarativo
- Entradas
- Proceso
- Salidas

13.3.- Ejemplo de Algoritmo Secuencial

Repasemos el enunciado del caso de la Agencia de Autos que se analizo en el capítulo anterior: "Una agencia de venta de autos paga a su personal de ventas un salario base de 500.000 Bs. más una comisión de 200.000 Bs. por cada automóvil vendido, más el 10% del valor total de sus ventas. Si se tiene como entrada el nombre del vendedor, el número de autos vendidos y el valor total de las ventas realizadas por éste. Se pide calcular y mostrar el sueldo a percibir por el vendedor.

Del análisis se tiene identificados los 3 elementos claves:

ENTRADAS: nombre del vendedor, cant. de autos vendidos y monto de sus ventas.

SALIDAS: Nombre del Vendedor y su Sueldo Final

PROCESO: El sueldo final se puede obtener procesando la siguiente instrucción de asignación: $\text{Sueldo_Final}=500000+\text{Aut_Vend}*200000+\text{Mto_Vtas}*0.10$

Veamos la progresión en la construcción de algoritmo de la Agencia de Autos.

Lo primero que debe hacerse es la definición de las variables.

```
// BLOQUE DECLARATIVO
Definir Nomb_Vend Como Caracter // Variables de E y S
Definir Aut_Vend Como Entero // Variables de E y P
Definir Mto_Vtas Como Real // Variables de E y P
Definir Sueldo_Final Como Real // Variables de P y S
```

El bloque de Entradas sería el siguiente:

```
// BLOQUE DE ENTRADAS
Mostrar "Ingrese la información requerida a continuación"
Mostrar "Nombre del Vendedor" Sin Saltar
Leer Nomb_Vend
Mostrar "Cantidad de autos vendida" Sin Saltar
Leer Aut_Vend
Mostrar "Monto total de ventas" Sin Saltar
Leer Mto_Vtas
```

El Bloque de Proceso es bastante simple, ya que sólo consta de una instrucción de asignación, a la cual se le asignará el resultado de la expresión aritmética que genera el resultado del cálculo requerido:

```
// BLOQUE DE PROCESO
Sueldo_Final=500000+Aut_Vend*200000+Mto_Vtas*0.10
```

El Bloque de Salidas de nuestro algoritmo, también consta de una sola sentencia, en la cual se combinan de forma alternada, literales de texto con variables:

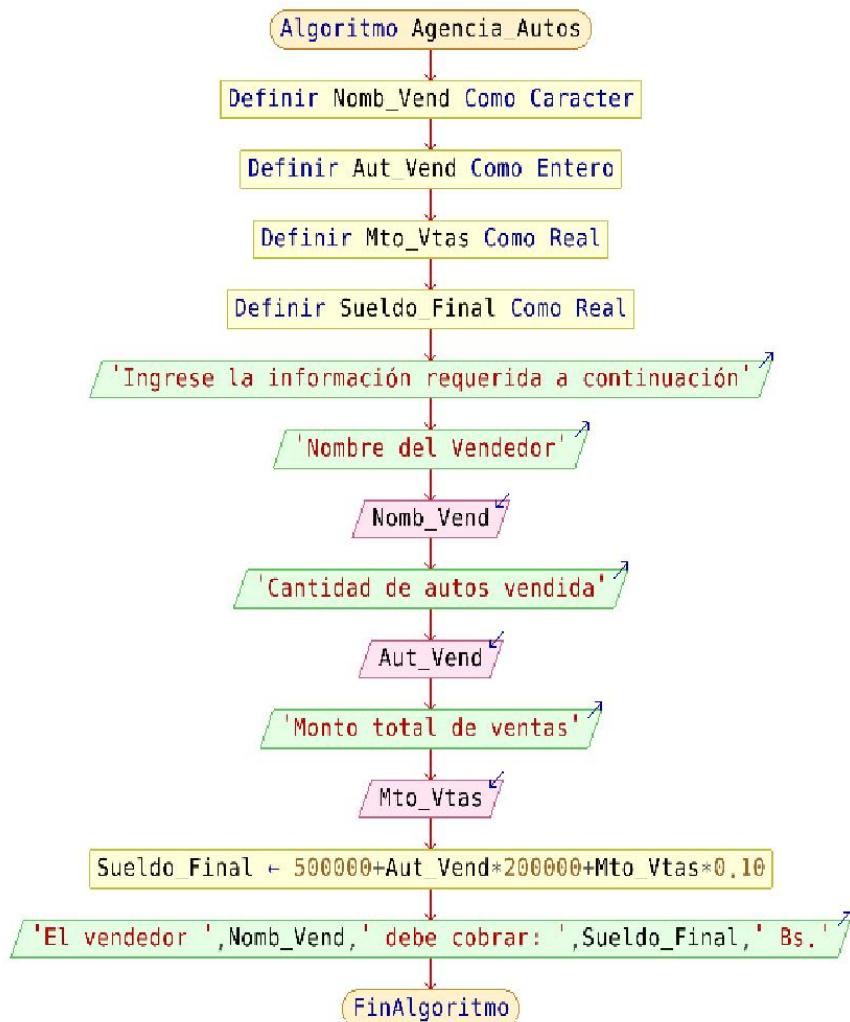
```
// BLOQUE DE SALIDAS
Mostrar "El vendedor " Nomb_Vend " debe cobrar: " Sueldo_Final " Bs."
```

El algoritmo completo para la Agencia de Autos queda así:

```
1 Algoritmo Agencia_Autos
2   // BLOQUE DECLARATIVO
3   Definir Nomb_Vend Como Caracter // Variables de E y S
4   Definir Aut_Vend Como Entero // Variables de E y P
5   Definir Mto_Vtas Como Real // Variables de E y P
6   Definir Sueldo_Final Como Real // Variables de P y S
7
8   // BLOQUE DE ENTRADAS
9   Mostrar "Ingrese la información requerida a continuación"
10  Mostrar "Nombre del Vendedor" Sin Saltar
11  Leer Nomb_Vend
12  Mostrar "Cantidad de autos vendida" Sin Saltar
13  Leer Aut_Vend
14  Mostrar "Monto total de ventas" Sin Saltar
15  Leer Mto_Vtas
16
17   // BLOQUE DE PROCESO
18   Sueldo_Final=500000+Aut_Vend*200000+Mto_Vtas*0,10
19
20   // BLOQUE DE SALIDAS
21   Mostrar "El vendedor " Nomb_Vend " debe cobrar: " Sueldo_Final " Bs."
22 FinAlgoritmo
```

Apreciemos el Flujograma:

ACADEMIA DE SOFTWARE



Capítulo 14. FUNCIONES INTERNAS

14.1.- Concepto

Cuando se programa, en muchas ocasiones es necesario realizar cálculos que incluyan operaciones matemáticas que van más allá de una simple suma o multiplicación, por ejemplo, calcular raíces cuadradas, valor absoluto, senos, etc, o pueden necesitar en vez de mostrar un nombre, mostrar sólo la inicial del nombre en mayúsculas. Para ello, los lenguajes de programación implementan bloques de código listos para usar para facilitarle el desarrollo a los programadores. A estos bloques de instrucciones se les llama FUNCIONES, que no son más que estructuras se invocan con un nombre, que devuelven un solo valor, aplicado a algo que se le especifica entre paréntesis. Éstas pueden ser de 2 tipos:

- 1) Predefinidas o Internas: forman parte del lenguaje de programación.
- 2) Definidas por usuario o externas: son creadas por el programador según su necesidad. Este tipo de función se estudiará y crearán en el siguiente nivel del curso.

Las funciones pueden ser clasificadas atendiendo a la naturaleza de las operaciones que realizan, a saber:

- Funciones Matemáticas.
- Funciones Trigonométricas.
- Funciones de Texto o Cadena.
- Funciones de Fecha y Hora.
- Funciones Lógicas.
- Funciones Financieras.
- Funciones Estadísticas, etc.

En PseInt existen funciones comúnmente predefinidas en los lenguajes de programación, quizás con otros nombres, entre ellas están:

Función	Significado
RC(X) o RAIZ(X)	Raiz Cuadrada de X
ABS(X)	Valor Absoluto de X
LN(X)	Logaritmo Natural de X
EXP(X)	Función Exponencial de X
SEN(X)	Seno de X
COS(X)	Coseno de X
TAN(X)	Tangente de X
ASEN(X)	Arcoseno de X
ACOS(X)	Arcocoseno de X
ATAN(X)	Arcotangente de X
TRUNC(X)	Parte entera de X
REDON(X)	Entero más cercano a X
AZAR(X)	Entero aleatorio en el rango [0;x-1]
ALEATORIO(A,B)	Entero aleatorio en el rango [A:B]
LONGITUD(S)	Cantidad de caracteres de la cadena S
MAYUSCULAS(S)	Retorna una copia de la cadena S con todos sus caracteres en mayúsculas
MINUSCULAS(S)	Retorna una copia de la cadena S con todos sus caracteres en minúsculas
SUBCADENA(S,X,Y)	Retorna una nueva cadena que consiste en la parte de la cadena S que va desde la posición X hasta la posición Y (incluyendo ambos extremos). Las posiciones utilizan la misma base que los arreglos, por lo que la primer letra será la 0 o la 1 de acuerdo al perfil del lenguaje utilizado.
CONCATENAR(S1,S2)	Retorna una nueva cadena resulta de unir las cadenas S1 y S2.
CONVERTIRNUMERO(X)	Recibe una cadena de caracteres que contiene un número y devuelve una variable numérica con el mismo.
CONVERTIRTEXTO(S)	Recibe un real y devuelve una variable numérica con la representación como cadena de caracteres de dicho real.

14.2.- Funciones Con Números

La raíz cuadrada de un número es un valor especial que, cuando se lo multiplica por sí mismo, dá el número dentro de la raíz (radicando). Por ejemplo, la raíz cuadrada de 16 es 4, ya que al multiplicar 4 por 4, se obtiene 16.

Es importante tener en cuenta que esta función aplica a los enteros positivos y no necesariamente resultará en un número entero.

El nombre que típicamente se utiliza para esta función en la mayoría de los lenguajes de programación es SQRT. En PSeInt la función se puede invocar con RC y RAIZ.



El siguiente ejemplo muestra como llevar una expresión con raíz cuadrada a una expresión algorítmica:

$$\sqrt{a + b}$$

Es equivalente a:

$$rc(a + b)$$

Las expresiones pueden incluir divisiones, multiplicaciones, potencias y/o cualquier combinación de ellas, para linealizarla y llevarlas a llevar a expresiones algorítmicas se debe aplicar el mismo criterio:

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Es equivalente a:

$$(-1 * b + rc(b^2 - (4 * a * c))) / (2 * a)$$

El siguiente ejemplo muestra cómo escribir un algoritmo que utiliza la función para obtener la raíz cuadrada en Pselnt:

```

1 Proceso sin_titulo
2     definir a,b,c,x como entero
3
4     a=5
5     b=10
6     c=2
7
8     x = (-1*b + raiz(b^2 - 4*a*c)) / 2*a
9     mostrar "La raiz es " x
10 FinProceso

```

Otra función de números utilizada frecuentemente es la generación de números aleatorios. Los números aleatorios son generados al azar según rango referencial que representa los topes inferior y superior para los posibles valores que puede ser generados. Para generar este tipo de números enteros en Pselnt se puede utilizar las funciones identificadas con las palabras AZAR o ALEATORIO. En los lenguajes de programación el término asociado es "random". El dato sobre el cual opera la función AZAR es un número que establece un tope para la variable. Por ejemplo: AZAR(7) generará un número entre 0 y 6. El siguiente ejemplo genera 3 números aleatorios y realiza un cálculo usando esos valores:

```

1 Proceso sin_titulo
2     definir a,b,c,x como entero
3
4     a=azar(8)
5     b=azar(20)
6     c=azar(5)
7
8     x = (b + raiz(b^2 - 4*a*c)) / 2*a
9     mostrar "La raiz es " x
10 FinProceso

```

14.3.- Funciones de Cadena

En Pselnt, hay otra función que es muy útil que se denomina SUBCADENA, que sirve para extraer porciones de una cadena y se invoca de la siguiente manera:

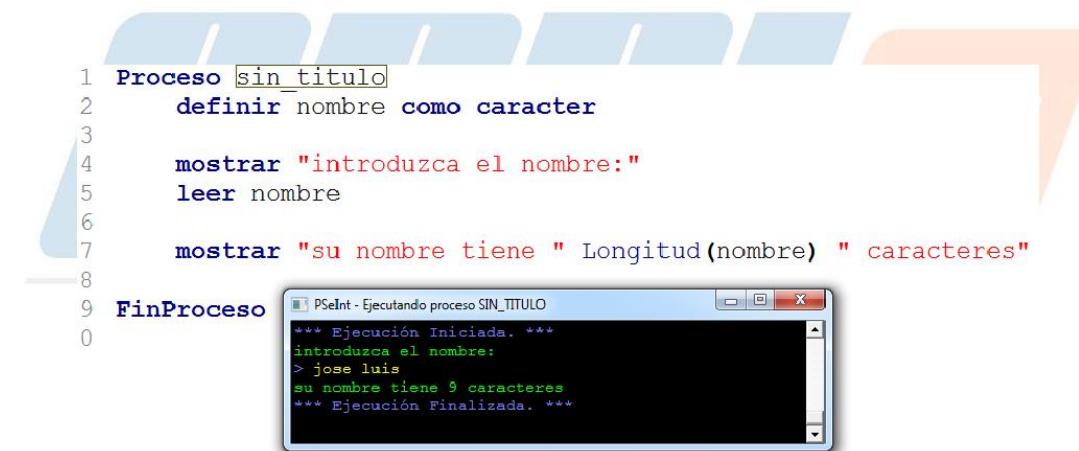
SUBCADENA(Cadena,Posic_Inic;Posic_Final)

Donde cada dato es:

Cadena: es la cadena de caracteres dentro de la cual se hará la extracción.

Posic_Inic: es un número que identifica la posición del 1º carácter a extraer.

Posic_Final: es un número que identifica la posición del último carácter a extraer.



```

1 Proceso sin_titulo
2     definir nombre como caracter
3
4     mostrar "introduzca el nombre:"
5     leer nombre
6
7     mostrar "su nombre tiene " Longitud(nombre) " caracteres"
8
9 FinProceso
0

```

The screenshot shows the Pselnt IDE interface. On the left, there is a code editor window with the above pseudocode. On the right, there is a terminal window titled "Pselnt - Ejecutando proceso SIN_TITULO". The terminal displays the following output:

```

*** Ejecución Iniciada ***
introduzca el nombre:
> jose luis
su nombre tiene 9 caracteres
*** Ejecución Finalizada ***

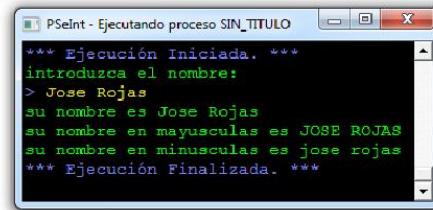
```

En muchas ocasiones es necesario convertir una cadena de caracteres que contiene letras a sus correspondientes mayúsculas o minúsculas. Para esto, se utilizan funciones que realizan esta operación. Pselnt incluye 2 funciones a tales efectos, llamadas: "mayuscula" y "minuscula" (sin acentos). A continuación se muestra un ejemplo de cómo usarlas y el resultado que arroja:

```

1 Proceso sin_titulo
2   definir nombre como caracter
3   definir nombre_may como caracter
4   definir nombre_min como caracter
5
6   mostrar "introduzca el nombre:"
7   leer nombre
8
9   nombre_may= Mayusculas(nombre)
10  nombre_min= Minusculas(nombre)
11
12  mostrar "su nombre es " nombre
13  mostrar "su nombre en mayusculas es " nombre_may
14  mostrar "su nombre en minusculas es " nombre_min
15 FinProceso
16

```



Otra tarea común al procesar cadenas de texto (string) es CONCATENAR. Este término muy particular de la programación, se refiere a juntar 2 o más cadenas de texto para crear una. En algunos lenguajes de programación se utiliza el mismo operador de suma algebraica (+), pero en muchos lenguajes se utiliza un operador particular para esta operación. Para entender más claramente el concepto, se muestra la diferencia entre sumar números y concatenar cadena:

Suma algebraica tradicional:

$15 + 50 \Rightarrow 65$

La concatenación de cadenas:

"15" + "50" \Rightarrow "1550"

PSeInt posee una función llamada "concatenar", que opera con sólo para concatenar 2 cadenas, pero también permite utilizar el operador "+" para concatenar porque es más simple e intuitivo de usar.

A continuación se muestra un ejemplo de cómo concatenar:

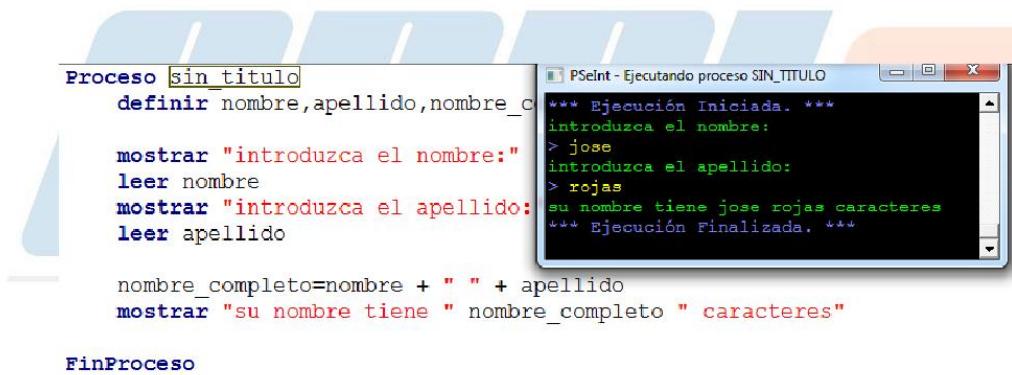
```

Proceso sin_titulo
    definir nombre,apellido,nombre_completo
    mostrar "introduzca el nombre:"
    leer nombre
    mostrar "introduzca el apellido:"
    leer apellido
    nombre_completo=nombre + apellido
    mostrar "su nombre tiene " nombre_completo " caracteres"

FinProceso

```

En el ejemplo anterior se puede notar cómo las 2 cadenas quedan "pegadas" una después de la otra, lo cual no es deseable. Dado que se pueden concatenar varias cadenas a la vez... El siguiente ejemplo muestra como corregir el problema citado, añadiendo entre el nombre y el apellido una cadena que contiene un espacio en blanco:



The screenshot shows the PSeInt interface. On the left, the code editor displays a process named 'sin_titulo' with the following pseudocode:

```

Proceso sin_titulo
    definir nombre,apellido,nombre_completo
    mostrar "introduzca el nombre:"
    leer nombre
    mostrar "introduzca el apellido:"
    leer apellido
    nombre_completo=nombre + " " + apellido
    mostrar "su nombre tiene " nombre_completo " caracteres"

FinProceso

```

On the right, the execution window titled 'PSeInt - Ejecutando proceso SIN_TITULO' shows the interaction with the user and the resulting output:

```

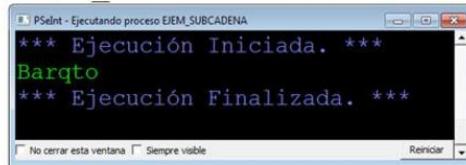
*** Ejecución Iniciada. ***
introduzca el nombre:
> jose
introduzca el apellido:
> rojas
su nombre tiene jose rojas caracteres
*** Ejecución Finalizada. ***

```

En Pselnt, hay otra función que es muy útil que se denomina SUBCADENA, que sirve para extraer porciones de una cadena y se invoca de la siguiente manera: SUBCADENA(Cadena,Pos_Inic,Pos_Final), donde cada dato es:

- Cadena: es la cadena de caracteres dentro de la cual se hará la extracción.
- Pos_Inic: es un número que identifica la posición del 1º carácter a extraer.
- Pos_Final: es un número que identifica la posición del último carácter a extraer.

```
1 Algoritmo Ejem_Subcadena
2     Ciudad="Barquisimeto"
3     SubCad1=SUBCADENA(Ciudad,1,4)
4     SubCad2=SUBCADENA(Ciudad,11,12)
5     Ciudad_Abrev=SubCad1+SubCad2
6     Mostrar Ciudad_Abrev
7 FinAlgoritmo
8
```



Capítulo 15. DEPURACIÓN DE PROGRAMAS

15.1.- Concepto

La DEPURACIÓN, consiste en recorrer un algoritmo en búsqueda de errores, en pro de resolverlos. Cuando se programa se pueden presentar 3 tipos de errores:

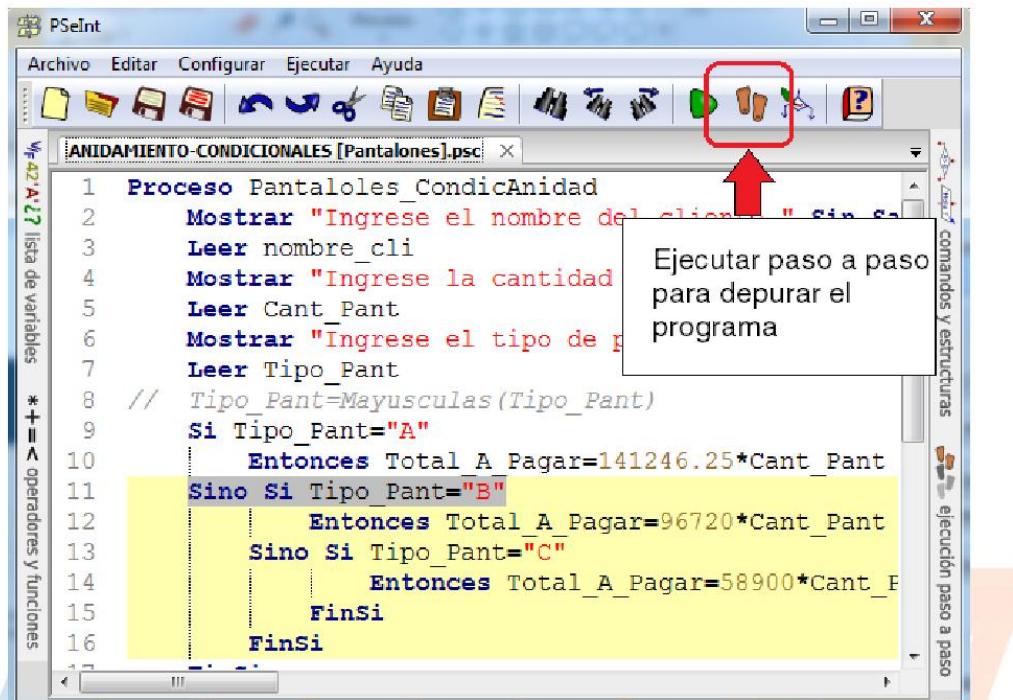
- De Sintaxis >> cuando no se cumple con las reglas del lenguaje. Son indicados por el verificador sintáctico.
- Lógicos >> ocurren cuando las operaciones aritméticas o lógicas no son correctas. Este tipo de errores son difíciles de detectar, debido a que no son mostrados por el verificador sintáctico.
- De Ejecución >> aquellos que ocurren durante la ejecución del programa. Pueden ser causados por un error lógico.

Encontrar errores lógicos puede ser muy difícil en algunos casos. Por lo tanto, es necesario utilizar una herramienta que facilite encontrarlos recorriendo el código paulatinamente.

15.2.- Ejecución Paso a Paso

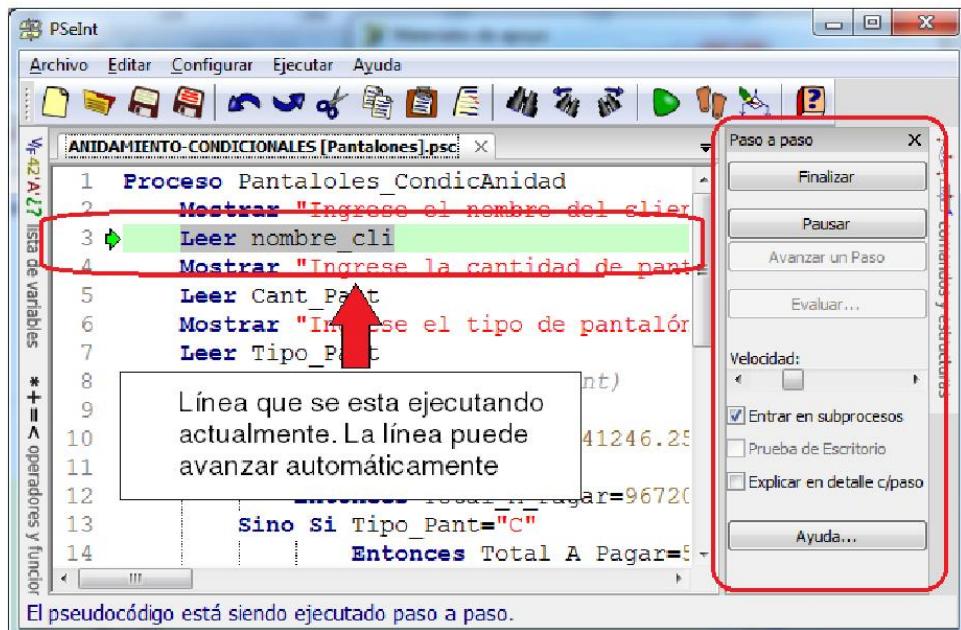
La depuración de los programas consiste en ejecutar el código fuente línea por línea, también conocido como "paso a paso". Esto permite al programador evaluar cada instrucción del código y facilitar la detección. Los entornos de desarrollo (IDE) poseen herramientas para depurar el código.

PseInt posee una herramienta de ejecución paso a paso que facilita la depuración de los programas. Esta se consigue al hacer clic en el ícono que tiene 2 pies en la barra de herramientas (como se muestra en la imagen).



Al hacer clic en este botón, el programa arranca la ejecución paso a paso. Al iniciarla, se muestra una flecha verde que destaca indica la línea del código que se está ejecutando. Esta flecha puede avanzar automáticamente o el programador puede hacer que vaya avanzando al ritmo que sea necesario para ir verificando cada instrucción.

En modo "2paso a paso" se activa también un panel localizado a la derecha, que permite pausar el programa para controlar de forma más granular su ejecución.



15.3.- Evaluación de Variables

En ocasiones es necesario evaluar el valor que tiene una variable durante la ejecución del programa, para poder verificar si el valor que ha obtenido es el apropiado. En caso contrario, verificar las instrucciones que le asignan valor y luego corregir el error.

En el panel "Paso a Paso", PSeInt incluye una opción llamada "Evaluar", al pulsarla aparece una ventana donde se coloca el valor de la variable que se desea evaluar y al presionar Enter aparecerá el valor que contiene la variable en ese momento.

