

Manual de Pseint

PSeInt

1. ¿Qué es PSeInt?

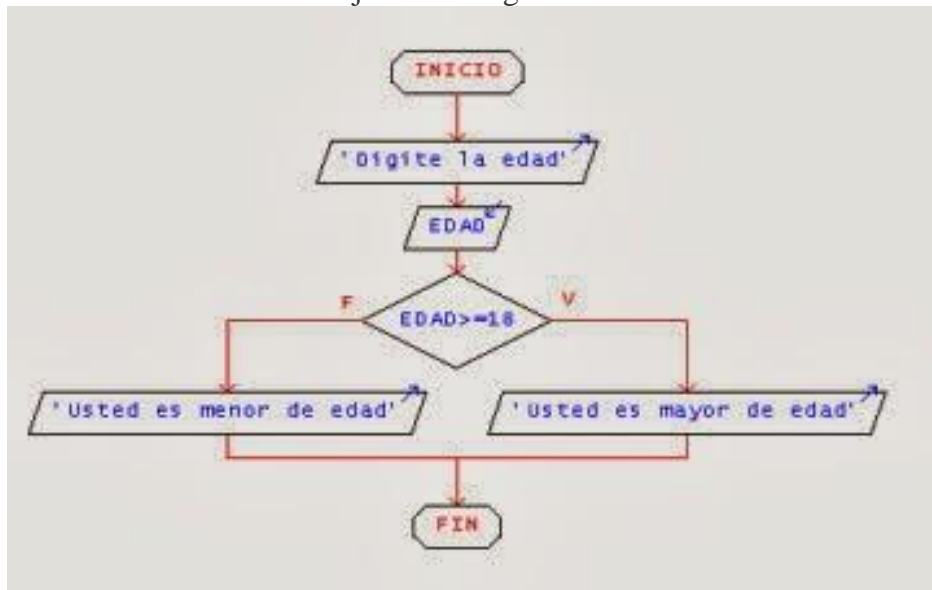
PSeInt está pensado para asistir a los estudiantes que se inician en la construcción de programas o algoritmos computacionales. El pseudocódigo se suele utilizar como primer contacto para introducir conceptos básicos como el uso de **estructuras de control**, **expresiones**, **variables**, **estructuras de datos**, etc, sin tener que lidiar con las particularidades de la sintaxis de un lenguaje real. Este software pretende facilitarle al principiante la tarea de escribir algoritmos en este **pseudolenguaje** presentando un conjunto de ayudas y asistencias, y brindarle además algunas herramientas adicionales que le ayuden a encontrar errores y comprender la lógica de los algoritmos.

Cuando se formula un algoritmo el objetivo es ejecutar este en una computadora, sin embargo, para que este entienda los pasos para llevar a cabo nuestro algoritmo debemos indicárselo siguiendo un conjunto de instrucciones y reglas que este entienda, y estas instrucciones son abstraídas en lo que conocemos como lenguaje de programación y luego son traducidas (compiladas) por el compilador del programa.

Un **algoritmo** codificado siguiendo un lenguaje de programación es conocido como programa. Antes de aprender un lenguaje de programación es necesario aprender la **metodología de programación**, es decir la estrategia necesaria para resolver problemas mediante programas.

Como punto de partida se aborda la manera como es representado un algoritmo. Básicamente analizamos dos formas, la representación usando **pseudocódigo** y la representación usando **diagramas de flujo**.

Un **diagrama de flujo** es un diagrama que utiliza símbolos (cajas) estándar y que tiene los pasos del algoritmo escritos en esas cajas unidas por flechas, denominadas líneas de flujo, que indican la secuencia que debe ejecutar el algoritmo.



Por otro lado, el pseudocódigo es un lenguaje de especificación (descripción) de algoritmos. El uso de tal lenguaje hace el paso de codificación final (traducción al lenguaje de programación) relativamente fácil, por lo que este es considerado un primer borrador de la solución del programa.

```

Proceso pseudocodigo
  Escribir "Digite la edad";
  Leer edad;
  Si edad > 18 Entonces
    Escribir "Ud es mayor de edad";
  Sino
    Escribir "Ud es menor de edad";
  Fin Si
FinProceso

```

Pseudocódigo en pocas palabras

Como habíamos dicho antes, el pseudocódigo es un lenguaje de descripción de algoritmos por lo que un primer paso consiste en familiarizarnos con este lenguaje. Como punto de partida tenemos que tener en cuenta lo siguiente:

- Anatomía de un algoritmo: Un algoritmo es **finito** por lo que tiene un principio y un fin. La siguiente plantilla muestra la forma básica de un algoritmo:

```

Algoritmo(nombre_algoritmo)
  Declaracion_de_variables
  Inicio
    Instrucción_1
    Instrucción_2

    Instrucción_N
  Fin_inicio
Fin(nombre_algoritmo)

```

- Las variables: Cuando nos referimos a variables nos referimos a **lugares de memoria en los cuales se almacena algún tipo de información**, por ejemplo el número de gallinas, la altura, la edad, el nombre y el peso.

VARIABLE: Es un espacio en memoria reservado para almacenar un valor, al cual se le reconoce con una etiqueta o nombre para de este modo tener acceso a la información que contiene.

- Existen diferentes tipos de datos como:

NUMERICO: Este tipo de variable contiene números decimales o enteros

CARÁCTER: Contiene cadenas de caracteres.

Como se muestra en la siguiente tabla

Tipo de dato	Descripción	Ejemplo
entero	Tipo de dato asociado a cantidades enteras. No poseen parte decimal. Ejemplo: 5, 6, -15, 199,...	Numero de vacas, edad.
real	Tipo de dato asociado a cantidades con parte decimal. Por ejemplo: 0.06, -3.4, 2.16, 1000.345,...	Estatura, peso, volumen.
lógicos	Se refiere a aquellos datos que pueden tomar solo dos posibles valores falso (F) o verdadero (T)	
alfanuméricos	Asociado a aquellos datos que contienen caracteres alfanuméricos (letras, número, signos de puntuación, etc).	Nombre, cedula, teléfono

- **Instrucción de asignación:** Escribe sobre una variable el valor de una expresión. En Pseint el operador de asignación es una flecha

variable ← expresión (en pseint)

Donde, una expresión es una combinación de valores, variables y operadores, los siguientes son algunos ejemplos de expresiones:

a ← 5
b ← c*d+(c-f)*m
z ← (x+y)/(w+s)
s ← (a/b)^3

Existen diferentes tipos de operadores. La siguiente tabla muestra los operadores aritméticos.

Operador	Significado
^	Potenciación
+	Suma
-	Resta
*	Multiplicación
/	División

- **Instrucciones de entrada y salida:** Para que un programa pueda interactuar con el usuario deben haber un conjunto de instrucciones que permitan especificar tal interacción, y estas son las instrucciones de entrada y salida.
- ✓ **Instrucciones de entrada:** Permite tomar uno o más datos de un medio externo (comúnmente el teclado) y asignarlos a una o más variables, su representación en pseudocódigo es:
Leer(var1, var2, ..., varN)
- ✓ **Instrucciones de salida:** Permite mostrar de variables y constante en un medio externo (comúnmente la pantalla). En pseudocódigo la instrucción asociada a la salida tiene la siguiente forma:
Escribir(var1,var2, ..., varN)

Ejemplo 1:

- ❖ Codifique un algoritmo que solicite el nombre y devuelva como salida el mensaje: Hola nombre_ingresado. Por ejemplo, si el usuario digita ramón, el mensaje desplegado será: Hola ramón.

En lenguaje natural seria:

Pedir el nombre

Mostrar el nombre

En pseudocódigo seria:

Algoritmo (nombre)

Variables:

alfanumerica: nom

INICIO

 ESCRIBA("Digite el nombre")

 LEA(nom)

 ESCRIBA("Hola ", nom)

FIN_INICIO

Fin(nombre)

Proceso nombre

Definir nom **Como** Caracter;

Escribir "Digite el nombre";

Leer nom;

Escribir "Hola " nom;

FinProceso

Ejemplo 2:

- ❖ Realice un algoritmo que pida dos números enteros, realice su suma y muestre el resultado.

En lenguaje natural seria:

Pedir los dos números

Sumar ambos números

Mostrar la suma

En pseudocódigo seria:

Algoritmo (suma)

Variables:

entero: a, b, c

INICIO

 ESCRIBA("Digite el primer numero (entero)")

 LEA(a)

 ESCRIBA("Digite el segundo numero (entero)")

 LEA(b)

 c = a + b

 ESCRIBA("La suma es: ", c)

FIN_INICIO

Fin(sumar)

Proceso suma

Definir a,b,c **Como** Entero;

Escribir "Digite el primer número";

Leer a;

Escribir "Digite el segundo numero";

Leer b;

c<-a+b;

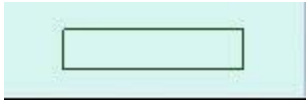
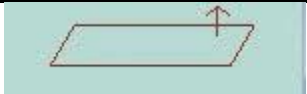
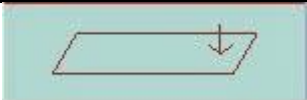
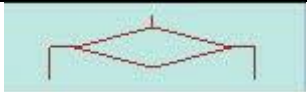



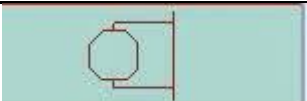
Escribir "Suma es: " c;

FinProceso

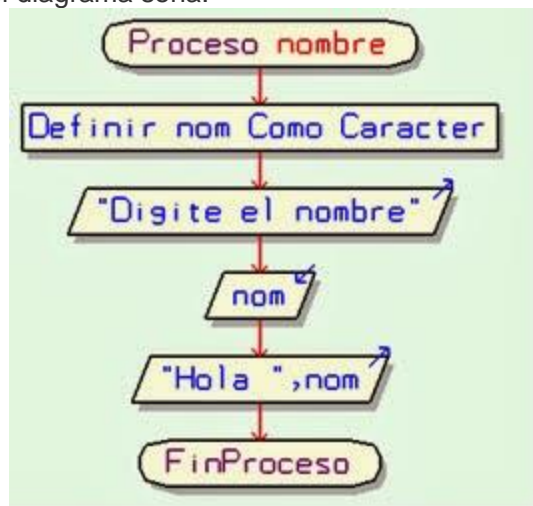
Nota: la declaración de variables es necesaria pues así se trabaja en un lenguaje de programación.

Diagramas de flujo

El diagrama de flujo es la representación grafica del algoritmo, de modo que lo único que es necesario es conocer la equivalencia de este con el pseudocódigo. La siguiente tabla resume esto:

Instrucción	Representación en PSeint	Representación en Diagrama de flujo
Definición de variables	Definir var1 Como Entero	
Escribir en la salida	Escribir var1	
Leer en la entrada	Leer var1	
Si - Entonces	Si edad > 18 Entonces Escribir "Ud es mayor de edad"; Sino Escribir "Ud es menor de edad"; Fin Si	
Segun	Segun edad Hacer 16: Escribir "Ud es menor de edad"; 18: Escribir "Ud es mayor de edad"; De Otro Modo: Escribir "Ud debe dar una edad numerica"; Fin Segun	
Mientras	edad<-1; Mientras edad<18 Hacer Escribir "Ud es menor de edad"; edad<-edad + 1; Fin Mientras	
Repetir – Hasta que	edad<-1; Repetir Escribir "Ud es menor de edad"; edad<-edad + 1; Hasta Que edad >18	
Para	Para edad<-1 Hasta 18 Con Paso 1 Hacer Escribir "Ud es menor de edad"; Fin Para	

❖ Para el ejemplo 1, pedir el nombre, el diagrama seria:



❖ Para el Ejemplo 2, sumar dos números enteros, el diagrama seria:



❖ Ejemplo3

Realizar un algoritmo que calcule el perímetro y el área de un rectángulo dado la base y la altura del mismo.

En pseudocódigo seria:

Pedir la base

Pedir la altura

Calcular el area como $\text{base} \times \text{altura} / 2$

Mostrar el area

Codificación en PSEINT

Proceso area

Definir base, altura, area **Como** Real;

Escribir "De la base";

Leer base;

Escribir "De la altura";

Leer altura;

area<-(base * altura)/2;

Escribir "Area es " area;

FinProceso

Ya se tiene el pseudocódigo del programa codificado en nuestro pseudocódigo por convención sin embargo la idea es probarlo para ver cómo funciona. La asignación en PSeInt no es con igual (=) sino con flecha (<-) y al final va punto y coma (;).

Entrada de datos

Leer a,b,c;

La instrucción de entrada en PSeInt se llama Leer no hace uso de paréntesis y termina con signo de punto y coma (;).

Salida de datos

Escribir "Hola", nombre;

La instrucción de entrada en PSeInt se llama Escribir no hace uso de paréntesis y termina con signo de punto y coma (;). Y se utiliza comillas dobles para los carteles.

Ejecución del PSeInt

Abra el programa PSEINT (Inicio > todos los Programas > PSeInt). Una vez ejecutado, se debe abrir una ventana como la mostrada en la figura 1:

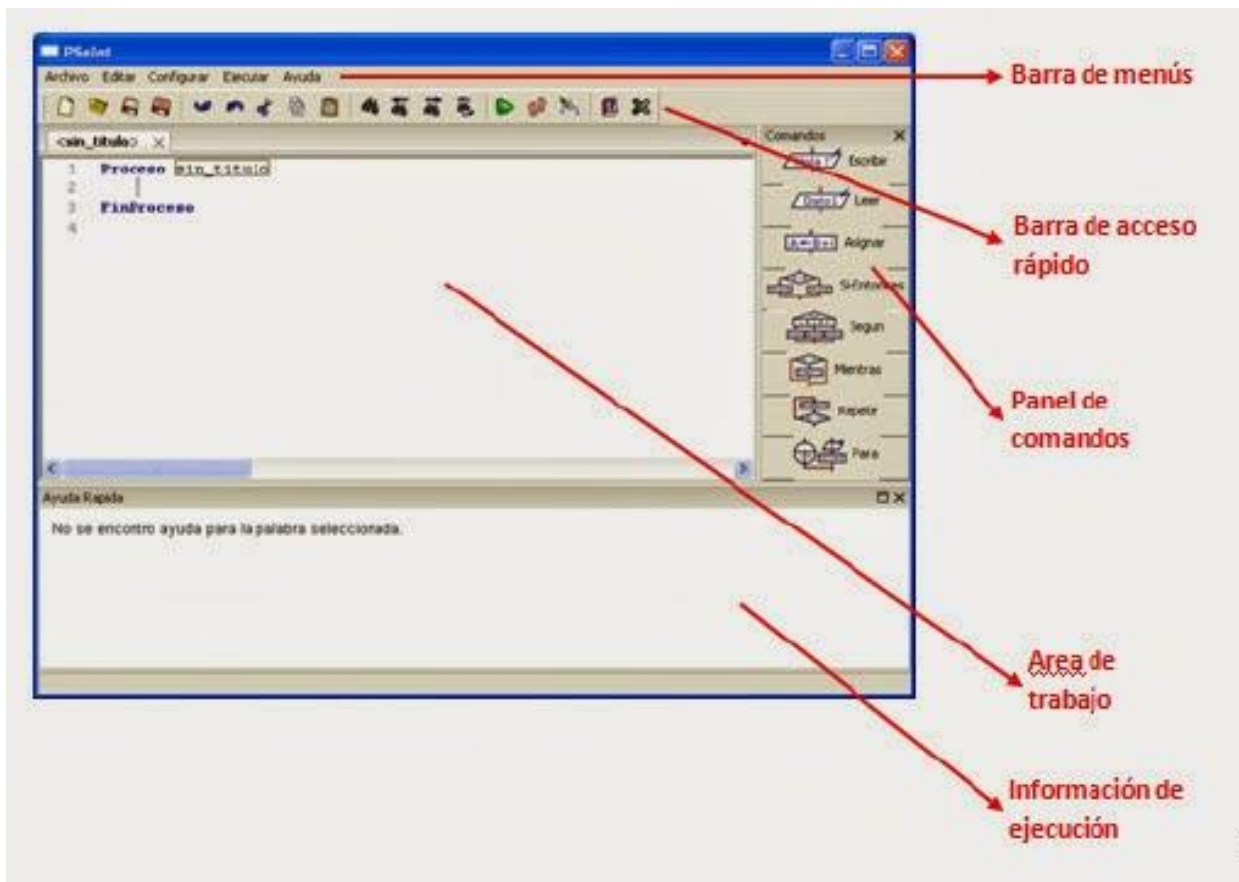


Figura 1. Ventana principal del programa PSeInt.

Codificación del algoritmo

Al explorar la herramienta dando click en los diferentes botones del panel de comando, observe el efecto en el área de trabajo, una vez que se haya familiarizado un poco con la herramienta intente adaptar el código mostrado en el pseudocódigo del problema anterior en el Pselnt:

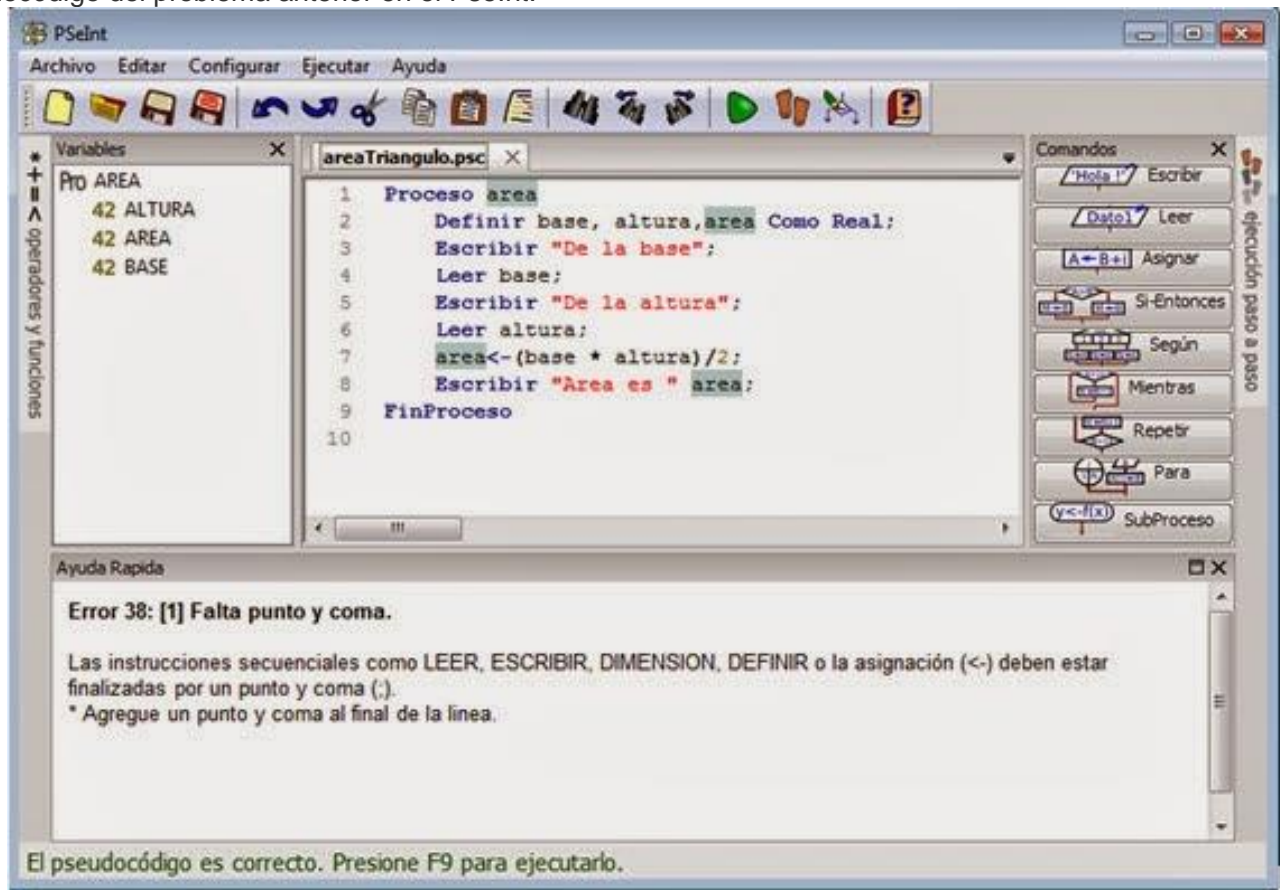


Figura 2. Adaptación del Pseudocódigo a Pselnt.

Una vez codificado el pseudocódigo (ayudado de los botones del panel de comandos) en el área de trabajo guarde el archivo en una ruta conocida.

Una vez realizado lo anterior obtenga el diagrama de flujo asociado al pseudocódigo para ello presione el dibujar diagrama de flujo.



Figura 3. Botón para obtener el diagrama de flujo.

Si lo anterior está bien, se generará un diagrama como el mostrado en la siguiente figura:

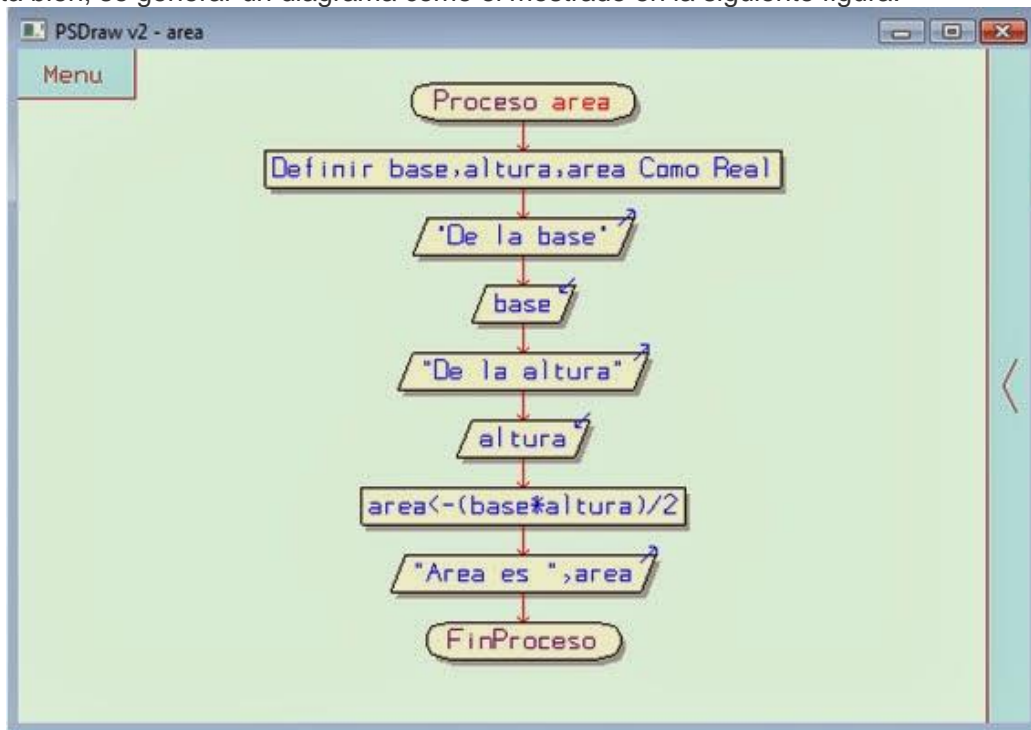


Figura 4. Diagrama de flujo del algoritmo del problema.

Guarde el diagrama de flujo anterior como una imagen jpg (puede serle útil después, por ejemplo para un informe).



Figura 5. Guardada de la imagen.

Ejecución del algoritmo

Una vez guardado el programa anterior, proceda a realizar la prueba del algoritmo presionando el botón ejecutar.



Deberá aparecer una ventana como la siguiente asociada al programa:

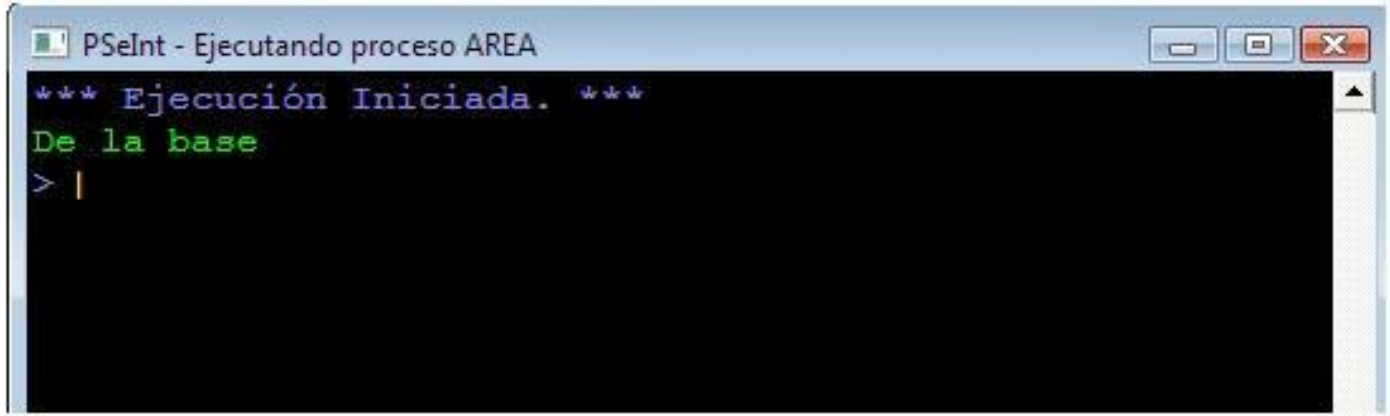


Figura 7. Ejecución del programa asociado al algoritmo (antes de ingresar el valor solicitado por teclado).

Lo anterior se debe a la instrucción Escribir "De la base";

Si lo nota el cursor se queda titilando esperando a que sean introducidos los valores para la altura y la base, esto debido a la instrucción Leer base;

Introduzca el valor de 2 como valor para la base y 3 como valor para la altura.

Note que cada vez que introduce un valor por teclado debe presionar enter. Una vez que presione el enter después de digitar el segundo valor aparece algo como lo siguiente:

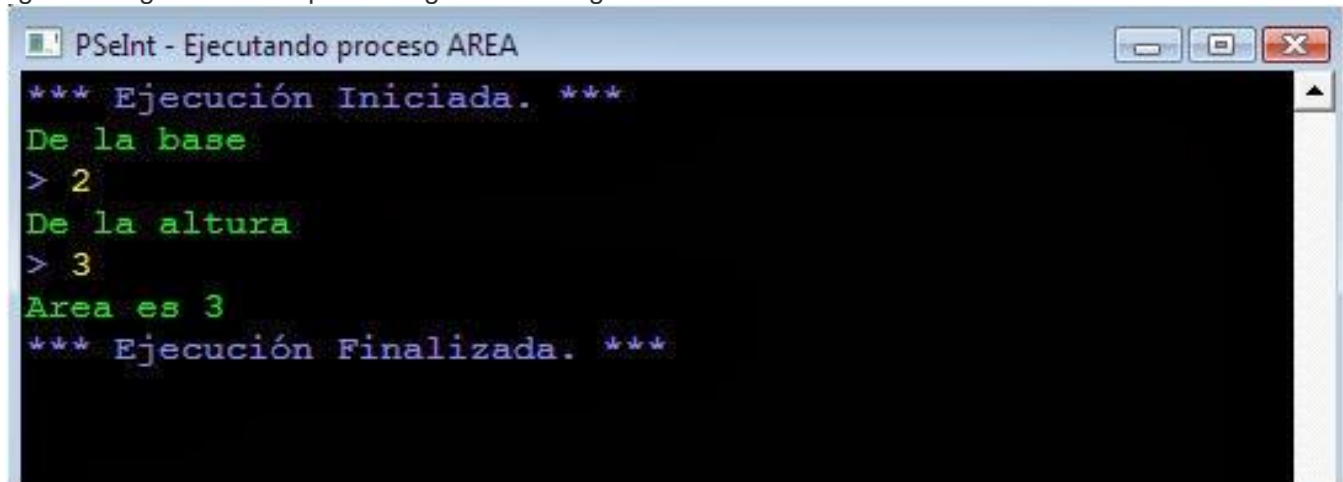


Figura 8. Ejecución del programa asociado al algoritmo (después de ingresar el valor solicitado por teclado).

Después de que aparece la ventana anterior si damos enter esta se cierra. Intente nuevamente ejecutar el algoritmo pero esta vez de 6 como valor para la base y 7 como valor para la altura.

COMENTARIO

Un comentario es una opción que permite un lenguaje de programación para establecer algunas oraciones y párrafos y así comprender mejor el código que estamos programando. Cada lenguaje contiene una serie de caracteres especiales que indican al compilador que lo que está contenido en ellos no debe ejecutarse.

COMENTARIOS EN PSEINT

Comentarios por párrafo: el carácter que indica los comentarios por párrafo es: `(/*)` para abrir y `(*/)` para cerrar. Por ejemplo

```
/*Esto es un  
Comentario de dos líneas*/
```

Comentarios por línea: el carácter que indica los comentarios por línea es `(//)`.

Por ejemplo `//Esto es un comentario de una línea`

Como se programa con Pseint

1. Definición de un algoritmo
2. Tipos de datos
3. Representaciones de un algoritmo Lenguaje natural Pseudocódigo Diagrama de flujo
4. Lenguaje de programación Pseint -
5. Tipos de datos -
6. Expresiones -
7. Acciones secuenciales -
8. Estructuras de control -
9. Arreglos -
10. Notaciones Ejercicios

1. DEFINICIÓN DE UN ALGORITMO

- Un algoritmo es una lista de operaciones o acciones (instrucciones) para poder encontrar la solución a un problema. Estas instrucciones deben estar ordenadas, estar bien definidas (no ser ambiguas, que cada instrucción tenga un solo significado), finitas (un numero específico de las mismas para poder finalizar la tarea)
- Está compuesto por operaciones, métodos y variables.
- Entre las operaciones se conocen las matemáticas: +, -, /, *, funciones trigonométricas, raíz cuadrada, etc.
- Los métodos son las funciones disponibles: Mostrar en pantalla. Pedir un dato
- Las variables pueden ser de diversos tipos: por ejemplo la variable entera de nombre dato puede contener los valores -1, 2, 3, etc. La variable booleana (lógica) llamada nombre puede contener los valores V o F (Verdadero o Falso). Una variable de tipo cadena o string llamada palabra puede contener los valores "aaaaaa", "Hola Mundo", etc. La variable de tipo caracter llamada vocal puede contener los valores 'a', 'b', '\$', etc

2. TIPOS DE DATOS

- ❖ **Numérico:** representa un numero, dentro de este tipo de dato pueden existir:
 - los enteros: numeros como el 12, 1, -5, etc
 - los reales (con coma decimal), como el 1.3, 5.4, -1.33, etc
- ❖ **Booleano:** representa un valor logico que solo puede ser verdadero o falso
- ❖ **Caracter:** representa a una sola letra del sistema alfanumerico ascii, como por ejemplo 'a', '@', '\$', '3', etc
- ❖ **String** o cadena de caracteres: representa una palabra entera, como ser: "Hola Mundo", "Jose", "a", etc

Tipos de Datos Simples

Existen tres tipos de datos básicos:

Numérico: números, tanto enteros como reales. Para separar decimales se utiliza el punto. Ejemplos: 12 23 0 -2.3 3.14

Lógico: solo puede tomar dos valores: VERDADERO o FALSO.

Caracter: caracteres o cadenas de caracteres encerrados entre comillas (pueden ser dobles o simples). Ejemplos 'hola' "hola mundo" '123' 'FALSO' 'etc'

Los tipos de datos simples se determinan automáticamente cuando se crean las variables. Las dos acciones que pueden crear una variable son la lectura(LEER) y la asignación(<-). Por ejemplo, la asignación "A<-0;" está indicando implícitamente que la variable A será una variable numérica. Una vez determinado el tipo de dato, deberá permanecer constante durante toda la ejecución del proceso; en caso contrario el proceso será interrumpido.

Opcionalmente, se puede declarar una variable numérica como entera con la instrucción DEFINIR. En este caso, todo valor no entero que se lea o asigne a la misma será truncado

Definición de variables

La instrucción definir permite explicitar el tipo de una o más variables. Esta definición puede ser opcional u obligatoria dependiendo de la configuración del lenguaje. La sintaxis es:

Definir <var1> , <var2> , ... , <varN> Como [REAL/ENTERO/LOGICO/CARACTER];

Una variable debe definirse antes de ser utilizada por primera vez. Los arreglos, se definen utilizando su identificador (sin subíndices ni dimensiones) antes o después de dimensionarlos, y el tipo aplica para todos los elementos del mismo (ya que se trata de una estructura de datos homogénea).

Los tipos posibles son NUMERO, NUMERICO, REAL, ENTERO, LOGICO, CARACTER, TEXTO, CADENA.

NUMERO, NUMERICO y REAL son sinónimos para el tipo de datos numérico básico, que puede almacenar tanto números reales como enteros. El tipo ENTERO es una especialización que sólo permite almacenar valores enteros; cualquier valor no entero que se lea o asigne en una variable de este tipo será truncado.

Una variable de tipo LOGICO sólo puede tomar los valores VERDADERO y FALSO, pero cuando se lee una variable ya definida como lógica, el usuario puede ingresar también las abreviaciones V y F, o 0 y 1.

CARACTER, TEXTO y CADENA son sinónimos para definir variables de tipo caracter. Estas pueden contener cero, uno o más caracteres arbitrarios y no tienen una longitud máxima. Si se declara una variable de este tipo y en una lectura el usuario ingresa un número o un valor lógico, se asignará una cadena que contiene el texto ingresado (ejemplo: "1", "VERDADERO", etc).

Si se intenta asignar a una variable ya definida un dato de un tipo incorrecto se producirá un error en tiempo de ejecución.

2. Arreglos

Los arreglos son estructuras de datos homogéneas (todos sus datos son del mismo tipo) que permiten almacenar un determinado número de datos bajo un mismo identificador, para luego referirse a los mismos utilizando uno o más subíndices. Los arreglos pueden pensarse como vectores, matrices, etc. Para poder utilizar un arreglo, primero es obligatorio su dimensionamiento; es decir, declarar los rangos de sus subíndices, lo cual determina cuantos elementos se almacenarán y como se accederá a los mismos.

Dimensionamiento

La instrucción **Dimension** permite declarar un arreglo, indicando sus dimensiones.

Dimension <identificador> (<max1>, ..., <maxN>);

Esta instrucción define un arreglo con el nombre indicado en <identificador> y N dimensiones. Los N parámetros indican la cantidad de dimensiones y el valor máximo de cada una de ellas. La cantidad de dimensiones puede ser una o más, y la máxima cantidad de elementos debe ser una expresión numérica positiva.

Si se utilizan arreglos en base 0, al declarar un arreglo de, por ejemplo 15 elementos, los índices de elementos válidos van de 0 a 14; mientras que si se utilizan arreglos en base 1 los índices válidos van de 1 a 15. Esto se configura en el cuadro de Opciones del Pseudocódigo.

Se pueden declarar más de un arreglo en una misma instrucción, separándolos con una coma (,).

Dimension <ident1> (<max11>, ..., <max1N>), ..., <identM> (<maxM1>, ..., <maxMN>)

Por ejemplo declarar un arreglo de 10 posiciones:

Dimension A [10]

Ejemplo de un arreglo de 10 enteros que son cargados en el mismo y luego mostrados:

Proceso Arreglo

Dimension A[10];

Definir A Como Entero;

Definir i Como Entero;

Para i<-1 Hasta 10 Con Paso 1 Hacer

 Escribir "De un numero";

 Leer A[i];

FinPara

Para i<-1 Hasta 10 Con Paso 1 Hacer

 Escribir "El numero en la posicion: " i " es: " A[i];

FinPara

FinProceso

3. REPRESENTACIONES DE UN ALGORITMO

Todo algoritmo puede ser representado por:

- Lenguaje natural
- Pseudocódigo
- Diagramas de flujo
- Lenguajes de programación

4. LENGUAJE NATURAL

Problema : Sumar 2 números.

Representación mediante Lenguaje natural:

Inicio Suma

Ingresar primer número

Guardar número en variable a

Ingresar segundo número

Guardar número en variable b

Sumar a y b

Guardar resultado en R

Mostrar R

Fin

Desventajas: Ambiguo y Extenso

5. PSEUDOCÓDIGO

Es una forma de representar un algoritmo, que se acerca a los lenguajes de programación y con elementos del lenguaje natural.

El pseudocódigo se compone de:

- Cabecera
- Declaraciones
- Cuerpo

La cabecera es la parte del algoritmo que posee el nombre de éste.

Las declaraciones son las variables y constantes que utilizará el algoritmo para resolver el problema.

El cuerpo son el conjunto de instrucciones o acciones que están entre el Inicio y el Fin.

6. PSEUDOCÓDIGO

La estructura del pseudocódigo es la siguiente:

Proceso SinTitulo

 accion1;

 accion2;

 accionN;

FinProceso

La sección “Proceso SinTitulo” es la cabecera del algoritmo, aquí debe estar el nombre del algoritmo

La sección “acción 1, acción 1,...” es el cuerpo del algoritmo, es la lista de intrucciones ordenadas y separadas por el punto y coma. Donde termina un punto y coma termina una instrucción (sentencia) y comienza la siguiente.

En este caso como utilizaremos el Pseint la sección de declaraciones del algoritmo no se toma en cuenta, ya que el software se encarga de asignarle el tipo de dato a cada variable dependiendo del uso que se le dé.

El ejemplo anterior de sumar dos números, en Pseint se vería como:

Proceso suma

 Escribir "Ingrese primer numero" ;

 Leer a;

 Escribir "Ingrese segundo numero" ;

 Leer b;

 c <- a + b;

 Escribir "La suma es: " c;

FinProceso

La Asignación

La instrucción de asignación permite almacenar una valor en una variable.

<variable > <- <expresión> ;

Al ejecutarse la asignación, primero se evalúa la expresión de la derecha y luego se asigna el resultado a la variable de la izquierda. El tipo de la variable y el de la expresión deben coincidir.

Si la variable de la izquierda no existía previamente a la asignación, se crea. Si la variable existía se pierde su valor anterior y toma el valor nuevo, razón por la cual se dice que la asignación es "destructiva" (destruye el valor que tenía la variable de la izquierda). Los contenidos de las variables que intervienen en la expresión de la derecha no se modifican.

Existen dos operadores de asignación alternativos que pueden utilizarse indistintamente en cualquier caso, pero la habilitación del segundo (=) depende del perfil de lenguaje seleccionado.

```
<variable> := <expresión> ;  
<variable> = <expresión> ;
```

Lectura

La instrucción Leer permite ingresar información desde el ambiente.

```
Leer <variable> , <variable2> , ... , <variableN> ;
```

Esta instrucción toma N valores desde el ambiente (en este caso el teclado) y los asigna a las N variables mencionadas. Pueden incluirse una o más variables, por lo tanto el comando leerá uno o más valores.

Si una variable donde se debe guardar el valor leído no existe, se crea durante la lectura. Si la variable existe se pierde su valor anterior ya que tomará el valor nuevo, razón por la cual se dice que la lectura es "destructiva" (destruye el valor que tenía previamente la variable).

Si se utiliza sintaxis flexible se permite opcionalmente separar las variables a leer simplemente con espacios en lugar de comas. Esto se configura en el cuadro de Opciones del Pseudocódigo.

El ejemplo Suma muestra un programa muy simple que lee dos números y calcula y muestra la suma de los mismos.

Escritura

La instrucción Escribir permite mostrar valores al ambiente.

```
Escribir <expr1> , <expr2> , ... , <exprN> ;
```

Esta instrucción informa al ambiente (en este caso escribiendo en pantalla) los valores obtenidos de evaluar N expresiones. Dado que puede incluir una o más expresiones, mostrará uno o más valores. Si hay más de una expresión, se escriben una a continuación de la otra sin separación, por lo que el algoritmo debe explicitar los espacios necesarios para diferenciar dos resultados si así lo requiere.

Si en algún punto de la línea se encuentran las palabras clave "SIN SALTAR" o "SIN BAJAR" los valores se muestran en la pantalla, pero no se avanza a la línea siguiente, de modo que la próxima acción de lectura o escritura continuará en la misma línea. En caso contrario, se añade un salto de línea luego de las expresiones mostradas.

```
Escribir Sin Saltar <expr1> , ... , <exprN>;  
Escribir <expr1> , ... , <exprN> Sin Saltar;
```

Puede utilizarse indistintamente las palabras Imprimir y Mostrar en lugar de Escribir si su perfil de lenguaje permite sintaxis flexible. Además, en este caso se permite opcionalmente separar las expresiones a mostrar simplemente con espacios en lugar de comas. Esto se configura en el cuadro de Opciones del Pseudocódigo.

El ejemplo Suma muestra un programa muy simple que lee dos números mostrando con la instrucción Escribir las indicaciones para el usuario y el resultado de sumar los mismos.

Consideraciones

Todas las acciones (instrucciones) deben estar entre **Proceso** y **FinProceso**. Como se deben sumar dos números entonces necesitamos de dos variables que se llamarán a y b respectivamente de tipo numérico. Para mostrar una leyenda pidiendo que se ingrese un número se utiliza la instrucción Escribir, que permite mostrar información en la pantalla. Para ingresar un dato se utiliza la instrucción Leer.

Y finalmente para asignar la suma de a y b a otra variable c se utiliza la instrucción de asignación <- que permite almacenar un valor en una variable.

Se pueden introducir comentarios luego de una instrucción, o en líneas separadas, mediante el uso de la doble barra (//). Todo lo que precede a //, hasta el fin de la línea, no será tomado en cuenta al interpretar el algoritmo. No es válido introducir comentario con /* y */.

No puede haber instrucciones fuera del proceso (antes de PROCESO, o después de FINPROCESO), aunque sí comentarios.

Las estructuras no secuenciales pueden anidarse. Es decir, pueden contener otras adentro, pero la estructura contenida debe comenzar y finalizar dentro de la contenedora.

Los identificadores, o nombres de variables, deben constar sólo de letras, números y/o guión_bajo (_), comenzando siempre con una letra.

Los tipos de datos de las variables no se declaran explícitamente, sino que se infieren a partir de su utilización.

Las constantes de tipo carácter se escriben entre comillas (").

En las constantes numéricas, el punto (.) es el separador decimal.

Las constantes lógicas son Verdadero y Falso.

Operadores

Este pseudolenguaje dispone de un conjunto básico de operadores que pueden ser utilizados para la construcción de expresiones más o menos complejas.

Las siguientes tablas exhiben la totalidad de los operadores de este lenguaje reducido:

Operador	Significado	Ejemplo
Relacionales		
>	Mayor que	3>2
<	Menor que	'ABC'<'abc'
=	Igual que	4=3
<=	Menor o igual que	'a'<='b'
>=	Mayor o igual que	4>=5
<>	Distinto que	'a'<>'b'
Logicos		
& ó Y	Conjunción (y).	(7>4) & (2=1) //falso
ó O	Disyunción (o).	(1=1 2=1) //verdadero
~ ó NO	Negación (no).	~(2<5) //falso
Algebraicos		
+	Suma	total <- cant1 + cant2
-	Resta	stock <- disp - venta
*	Multiplicación	area <- base * altura
/	División	porc <- 100 * parte / total
^	Potenciación	sup <- 3.41 * radio ^ 2
% ó MOD	Módulo (resto de la división entera)	resto <- num MOD div

La jerarquía de los operadores matemáticos es igual a la del álgebra, aunque puede alterarse mediante el uso de paréntesis.

Funciones

Las funciones en el pseudocódigo se utilizan de forma similar a otros lenguajes. Se coloca su nombre seguido de los argumentos para la misma encerrados entre paréntesis (por ejemplo `trunc(x)`). Se pueden utilizar dentro de cualquier expresión, y cuando se evalúe la misma, se reemplazará por el resultado correspondiente. Actualmente, todas las funciones disponibles son matemáticas (es decir que devolverán un resultado de tipo numérico) y reciben un sólo parámetro de tipo numérico. A continuación se listan las funciones integradas disponibles:

Función	Significado
RC(X) o RAIZ(X)	Raíz 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 entre 0 y x-1

Las estructuras en Pseint se dividen en condiciones y repetitivas:

Estructuras Condicionales

Condicional Si-Entonces

La secuencia de instrucciones ejecutadas por la instrucción Si-Entonces-Sino depende del valor de una condición lógica.

```
Si <condición>
    Entonces
        <instrucciones>
    Sino
        <instrucciones>
FinSi
```

Al ejecutarse esta instrucción, se evalúa la condición y se ejecutan las instrucciones que correspondan: las instrucciones que le siguen al Entonces si la condición es verdadera, o las instrucciones que le siguen al Sino si la condición es falsa. La condición debe ser una expresión lógica, que al ser evaluada retorna Verdadero o Falso.

La cláusula Entonces debe aparecer siempre, pero la cláusula Sino puede no estar. En ese caso, si la condición es falsa no se ejecuta ninguna instrucción y la ejecución del programa continúa con la instrucción siguiente.

- Ejemplo: realice un algoritmo que pida dos números y busque el mayor de ambos

Proceso ElMayor

```

Definir a,b Como Real;
Escribir "De el primer lado del triangulo";
Leer a;
Escribir "De el segundo lado del triangulo";
Leer b;

Si a < b Entonces
    Escribir "a es menor que b" a "menor que " b;
Sino
    Escribir "b es menor que a" b "menor que " a;
Fin Si
FinProceso

```

Selección Multiple

La secuencia de instrucciones ejecutada por una instrucción Segun depende del valor de una variable numérica.

```

Segun <variable> Hacer
    <número1>: <instrucciones>
    <número2>,<número3>: <instrucciones>
    <...>
    De Otro Modo: <instrucciones>
FinSegun

```

Esta instrucción permite ejecutar opcionalmente varias acciones posibles, dependiendo del valor almacenado en una variable de tipo numérico. Al ejecutarse, se evalúa el contenido de la variable y se ejecuta la secuencia de instrucciones asociada con dicho valor.

Cada opción está formada por uno o más números separados por comas, dos puntos y una secuencia de instrucciones. Si una opción incluye varios números, la secuencia de instrucciones asociada se debe ejecutar cuando el valor de la variable es uno de esos números.

Opcionalmente, se puede agregar una opción final, denominada De Otro Modo, cuya secuencia de instrucciones asociada se ejecutará sólo si el valor almacenado en la variable no coincide con ninguna de las opciones anteriores.

Ejemplo: realizar un programa que muestre un menú de opciones dependiendo de la opción elegida un numero de 1 a 5 que significa 1 Literatura, 2 Cine, 3 Música, 4 videojuegos 5 Salir, si es un número mayor a 5 debe decir opción no valida.

// Muestra como hacer un menú simple con las estructuras Repetir-Hasta Que y Según

Proceso Menu

Repetir

// mostrar menu

Escribir "Menú de recomendaciones";

Escribir " 1. Literatura";

Escribir " 2. Cine";

Escribir " 3. Música";

Escribir " 4. Videojuegos";

Escribir " 5. Salir";

// ingresar una opcion

Escribir "Elija una opción (1-5): ";

Leer OP;

// procesar esa opción

Segun OP Hacer

1:

Escribir "Lecturas recomendables:";

Escribir " + Esperándolo a Tito y otros cuentos de fútbol (Eduardo Sacheri)";

Escribir " + El juego de Ender (Orson Scott Card)";

Escribir " + El sueño de los héroes (Adolfo Bioy Casares)";

2:

Escribir "Películas recomendables:";

Escribir " + Matrix (1999)";

Escribir " + El último samuray (2003)";

Escribir " + Cars (2006)";

3:

Escribir "Discos recomendables:";

Escribir " + Despedazado por mil partes (La Renga, 1996)";

Escribir " + Búfalo (La Mississippi, 2008)";

Escribir " + Gaia (Mägo de Oz, 2003)";

4:

Escribir "Videojuegos clásicos recomendables:";

Escribir " + Día del tentáculo (LucasArts, 1993)";

Escribir " + Terminal Velocity (Terminal Reality/3D Realms, 1995)";

Escribir " + Death Rally (Remedy/Apogee, 1996)";

5:

Escribir "Gracias, vuelva pronto";

De otro modo:

Escribir "Opción no válida";

FinSegun

Escribir "Presione enter para continuar";

Esperar Tecla;

Hasta Que OP=5

FinProceso

Estructuras Repetitivas

Mientras

La instrucción Mientras ejecuta una secuencia de instrucciones mientras una condición sea verdadera.

```
Mientras <condición> Hacer
  <instrucciones>
FinMientras
```

Al ejecutarse esta instrucción, la condición es evaluada. Si la condición resulta verdadera, se ejecuta una vez la secuencia de instrucciones que forman el cuerpo del ciclo. Al finalizar la ejecución del cuerpo del ciclo se vuelve a evaluar la condición y, si es verdadera, la ejecución se repite. Estos pasos se repiten mientras la condición sea verdadera.

Note que las instrucciones del cuerpo del ciclo pueden no ejecutarse nunca, si al evaluar por primera vez la condición resulta ser falsa.

Si la condición siempre es verdadera, al ejecutar esta instrucción se produce un ciclo infinito. A fin de evitarlo, las instrucciones del cuerpo del ciclo deben contener alguna instrucción que modifique la o las variables involucradas en la condición, de modo que ésta sea falsificada en algún momento y así finalice la ejecución del ciclo.

El ejemplo AdivinaNumero le da al usuario 10 intentos para adivinar un número generado aleatoriamente, utilizando esta estructura para verificar si el usuario acierta el número o si se agotan los intentos.

```
// Juego simple que pide al usuario que adivine un numero en 10 intentos
```

Proceso Adivina_Numero

```
intentos<-10;
num_secreto <- azar(100)+1;

Escribir "Adivine el numero (de 1 a 100):";
Leer num_ingresado;
Mientras num_secreto<>num_ingresado Y intentos>0 Hacer
  Si num_secreto>num_ingresado Entonces
    Escribir "Muy bajo";
  Sino
    Escribir "Muy alto";
  FinSi
  intentos <- intentos-1;
  Escribir "Le quedan ",intentos," intentos:";
  Leer num_ingresado;
FinMientras

Si intentos=0 Entonces
  Escribir "El numero era: ",num_secreto;
Sino
  Escribir "Exacto! Usted adivino en ",11-intentos," intentos.";
FinSi
```

```
FinProceso
```

Repetir

La instrucción Repetir-Hasta Que ejecuta una secuencia de instrucciones hasta que la condición sea verdadera.

```
Repetir
  <instrucciones>
Hasta Que <condición>
```

Al ejecutarse esta instrucción, la secuencia de instrucciones que forma el cuerpo del ciclo se ejecuta una vez y luego se evalúa la condición. Si la condición es falsa, el cuerpo del ciclo se ejecuta nuevamente y se vuelve a evaluar la condición. Esto se repite hasta que la condición sea verdadera.

Note que, dado que la condición se evalúa al final, las instrucciones del cuerpo del ciclo serán ejecutadas al menos una vez.

Además, a fin de evitar ciclos infinitos, el cuerpo del ciclo debe contener alguna instrucción que modifique la o las variables involucradas en la condición de modo que en algún momento la condición sea verdadera y se finalice la ejecución del ciclo.

Si se utiliza sintaxis flexible (ver Opciones del Pseudocódigo) se permite opcionalmente utilizar Mientras Que en lugar de Hasta Que, de modo que el conjunto de acciones contenidas en el bucle se ejecuta mientras que la condición sea verdadera. Notar que la palabra Que es la que diferencia el uso de la palabra Mientras en la estructura repetir de la estructura Mientras. Es decir, si se omite la palabra que se considera como el comienzo de un bucle Mientras en lugar de el final de un bucle Repetir.

Para

La instrucción Para ejecuta una secuencia de instrucciones un número determinado de veces.

```
Para <variable> <- <inicial> Hasta <final> Con Paso <paso> Hacer
  <instrucciones>
FinPara
```

Al ingresar al bloque, la variable <variable> recibe el valor <inicial> y se ejecuta la secuencia de instrucciones que forma el cuerpo del ciclo. Luego se incrementa la variable <variable> en <paso> unidades y se evalúa si el valor almacenado en <variable> superó al valor <final>. Si esto es falso se repite hasta que <variable> supere a <final>. Si se omite la cláusula Con Paso <paso>, la variable <variable> se incrementará en 1.

Si se habilita la sintaxis flexible en las configuración del lenguaje se pueden utilizar dos alternativas. La primer variante consiste en reemplazar el operador de asignación por la palabra clave Desde:

```
Para <variable> Desde <inicial> Hasta <final> Con Paso <paso> Hacer ...
```

De esta forma, la lectura de la sentencia resulta más obvia. Además, con sintaxis flexible, si no se especifica el paso pero el valor final es menor al inicial, el bucle recorrerá los valores en orden inverso como si el paso fuera -1. La segunda variante solo sirve para recorrer arreglos de una o más dimensiones. Se introduce con la construcción Para Cada seguida de un identificador, la palabra clave De y otro identificador:

```
Para Cada <elemento> De <Arreglo> Hacer ...
```

El segundo identificador debe corresponder a un arreglo. El primero será el que irá variando en cada iteración. El ciclo realizará tantas iteraciones como elementos contenga el arreglo y en cada uno el primer identificador servirá para referirse al elemento del arreglo en cuestión.

El ejemplo Promedio utiliza un bucle de este tipo para leer N valores numéricos con los cuales calcula un promedio.

```
// Calcula el promedio de una lista de N datos
```

Proceso Promedio

```
    Escribir "Ingrese la cantidad de datos:";  
    Leer n;
```

```
    acum<-0;
```

```
    Para i<-1 Hasta n Hacer
```

```
        Escribir "Ingrese el dato ",i,":";
```

```
        Leer dato;
```

```
        acum<-acum+dato;
```

```
    FinPara
```

```
    prom<-acum/n;
```

```
    Escribir "El promedio es: ",prom;
```

FinProceso

El ejemplo Para1 utiliza las tres variantes de este tipo de bucles para recorrer un arreglo.

```
// Para poder ejecutar correctamente este ejemplo debe tener  
// habilitada la sintaxis flexible en su perfil de lenguaje
```

Proceso Para1

```
    // declara un arreglo de 10 elementos
```

```
    Dimension A[10];
```

```
    // recorre los 10 elementos y va asignandoles enteros aleatorios
```

```
    para cada elemento de A Hacer
```

```
        // elemento toma el contenido de cada posicion del arreglo
```

```
        // y si se modifica elemento se modifica el arreglo
```

```
        elemento <- azar(100);
```

```
    FinPara
```

```
    Escribir "Los elementos del arreglo son:";
```

```
    // recorre los 10 elementos utilizando subindices y los muestra en pantalla
```

```
    para i desde 1 hasta 10 Hacer
```

```
        escribir "Posición " i ": " A[i];
```

```
    FinPara
```

```
    Escribir ""; // deja una linea en blanco
```

```
    Escribir "En orden inverso:";
```

```
    // recorre los 10 elementos en orden inverso y los muestra en una misma linea
```

```
    para i desde 10 hasta 1 Hacer
```

```
        escribir sin bajar A[i] " ";
```

```
    FinPara
```

FinProceso

Matrices en pseint

Tal como en excel que existen filas y columnas por cada celda que contiene un valor, también en pseint se puede crear una matriz, para lo cual se deben crear dos índices para saber cual es la fila y columna donde se ubicara un valor.

Proceso Matriz

```
//Primero se crea la matriz de 2 x 2 y se define su tipo, asi como los dos indices para la fila y columna
```

```
Dimension M [2,2];
```

```
Definir M,I,J como Entero;
```

```
// Leer valores, aqui se cargan los valores en cada celda
```

```
Para J<-1 Hasta 2 Hacer
```

```
    Para I<-1 Hasta 2 Hacer
```

```
        Escribir 'Ingrese valor para indice ',I,', en columna ',J','
```

```
        Leer M[I,J]
```

```
    FinPara
```

```
FinPara
```

```
// Mostrar valores de cada celda
```

```
Para I<-1 Hasta 2 Hacer
```

```
    Para J<-1 Hasta 2 Hacer
```

```
        Escribir 'El valor para indice ',I,', en columna ',J,'es :' M[I,J];
```

```
    FinPara
```

```
FinPara
```

FinProceso

Condiciones Lógicas

Hay que recordar que toda expresión lógica utiliza las tablas de verdad de los conectivos lógicos, los cuales son:

& ó Y	Conjunción (y)	(7>4) & (2=1) //falso
ó O	Disyunción (o).	(1=1 2=1) //verdadero
~ ó NO	Negación (no).	~(2<5) //falso

El resultado de una condición lógica puede ser verdadera o falsa, se utilizan los conectivos lógicos para crear condiciones complejas en las cuales su valor de verdad depende del conector lógico

Recordar que para la conjunción solo es verdad cuando ambas condiciones son verdad

Recordar que para la disyunción es verdad cuando una de las dos condiciones es verdad o ambas son verdad

Un ejemplo es un programa que averigüe de tres números cual es el menor y que ordene los tres números de menor a mayor, fíjate el programa Ordenar

Proceso ordenar

```
Definir a,b,c Como Real;
Escribir "De el primer numero";
Leer a;
Escribir "De el segundo numero";
Leer b;
Escribir "De el tercer numero";
Leer c;
Si a < b Y a < c Entonces
    Si b < c Entonces
        Escribir "a es el menor y el ordenamiento es";
        Escribir "a: " a " < " b: " b " < " c: " c;
    Sino
        Escribir "a: " a " < " c: " c " < " b: " b;
    Fin Si
Fin Si

Si b < a Y b < c Entonces
    Si a < c Entonces
        Escribir "b es el menor y el ordenamiento es";
        Escribir "b: " b " < " a: " a " < " c: " c;
    Sino
        Escribir "b: " b " < " c: " c " < " a: " a;
    Fin Si
Fin Si

Si c < a Y c < b Entonces
    Si a < b Entonces
        Escribir "c es el menor y el ordenamiento es";
        Escribir "c: " c " < " a: " a " < " b: " b;
    Sino
        Escribir "c: " c " < " b: " b " < " a: " a;
    Fin Si
Fin Si

Fin Proceso
```


Ejemplo. Pedir dos números y realizar la división entera, el segundo número no puede ser cero o negativo. El primer número tampoco puede ser negativo

Proceso dividir

Definir a,b,c Como Real;

Escribir "De el primer número";

Leer a;

Escribir "De el segundo numero";

Leer b;

//Se realiza la division entera entre positivos

Si ((b < 0) O (b = 0)) Y ~(a < 0) Entonces

Escribir "No se puede dividir por 0 ni por negativo";

Sino

c<-a / b;

Escribir "Resultado es " c;

Fin Si

FinProceso