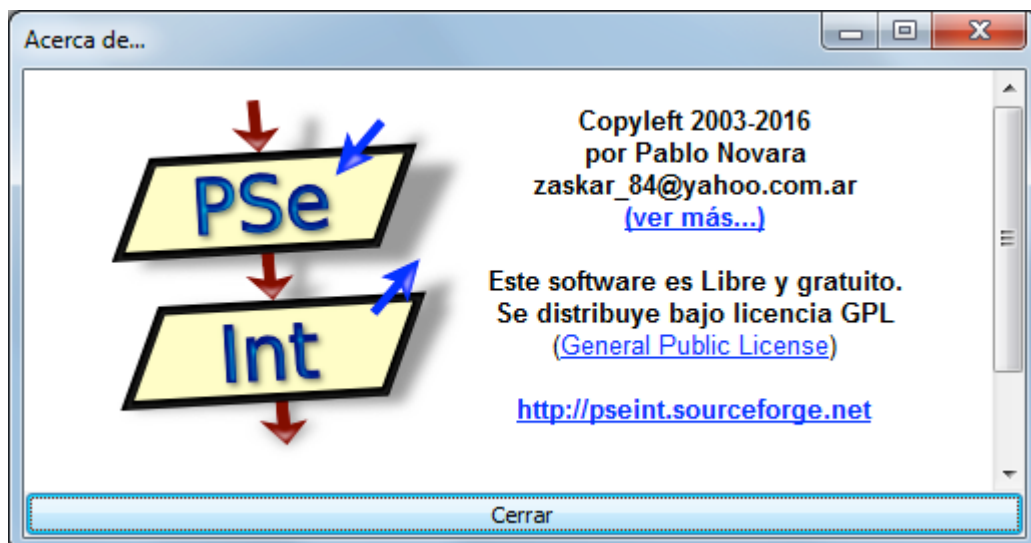


TECNOLOGÍAS DE LA INFORMACIÓN Y LA COMUNICACIÓN (TIC)

2º BACHILLERATO

Programación en:
PSeInt



Índice

Tema	Pág.
¿Qué es PSeInt?	3
Instalación	5
Apuntes preliminares	6
Entorno de PSeInt	7
<ul style="list-style-type: none"> • Botones y barras 	
Mi primer programa	15
Diagramas de flujo	20
<ul style="list-style-type: none"> • Editor de diagramas de flujo 	
Declarar variable	21
Operadores	22
Asignaciones y Operaciones matemáticas en un programa.	24
Instrucciones Condicionales	
<ul style="list-style-type: none"> • Si 	26
<ul style="list-style-type: none"> • Si anidado 	28
<ul style="list-style-type: none"> • Segun 	31
<ul style="list-style-type: none"> • Operador ó O 	22
<ul style="list-style-type: none"> • Operador & ó Y 	37
<ul style="list-style-type: none"> • Exportación a un lenguaje de programación real 	38
Instrucciones de ciclo	
<ul style="list-style-type: none"> • Ciclo Mientras 	38

• Ciclo Para	44
○ Ciclo Para con paso negativo	46
○ Ciclos Anidados	46
• Ciclo Repetir	49
SubProcesos	53
• Parámetros de entrada o valor	
• Parámetros de variable	
• SubProcesos con retorno o funciones	46
Dimensiones	64
• Dimension de una dimensión	
• Dimensiones bidimensionales	70
Ejecución Paso a Paso	
• Ejecución explicada	
Funciones predefinidas	
<i>Información teórica:</i>	
Registros o estructuras (<i>Información teórica</i>)	
• Dimensiones con registros	
Archivos de texto (<i>Información teórica</i>)	
<i>Información práctica:</i>	
Anexo:	
• Editor de diagramas de flujo	
• Desinstalar PSeInt	
• Editar código fuente	

¿Qué es PSeInt?

PSeInt es la abreviatura de *Pseudocode Interpreter*, en español, Intérprete de Pseudocódigo. Este programa fue creado como proyecto final para la materia *Programación 1* de la carrera *Ingeniería en Informática* de la [Facultad de Ingeniería y Ciencias Hídricas](#) de la [Universidad Nacional del Litoral](#), de Argentina del en aquel momento estudiante y hoy docente Pablo Novara.

Ha recibido varios reconocimientos, entre ellos fue el de Proyecto del Mes en SourceForge en dos oportunidades: 2015 y 2016 como se detalla en su sitio web oficial

El programa utiliza pseudocódigo, una descripción de un algoritmo computacional, cuya principal misión es que el programador pueda centrarse en los aspectos lógicos de la programación, dejando el apartado técnico para cuando el programador vea la sintaxis de un lenguaje de programación real.

PSeInt incluye en su editor diversas herramientas para que podamos crear y almacenar programas en este peculiar lenguaje, ejecutarlos directamente desde su interfaz, o incluso corregir posibles defectos que encontremos en su desarrollo.

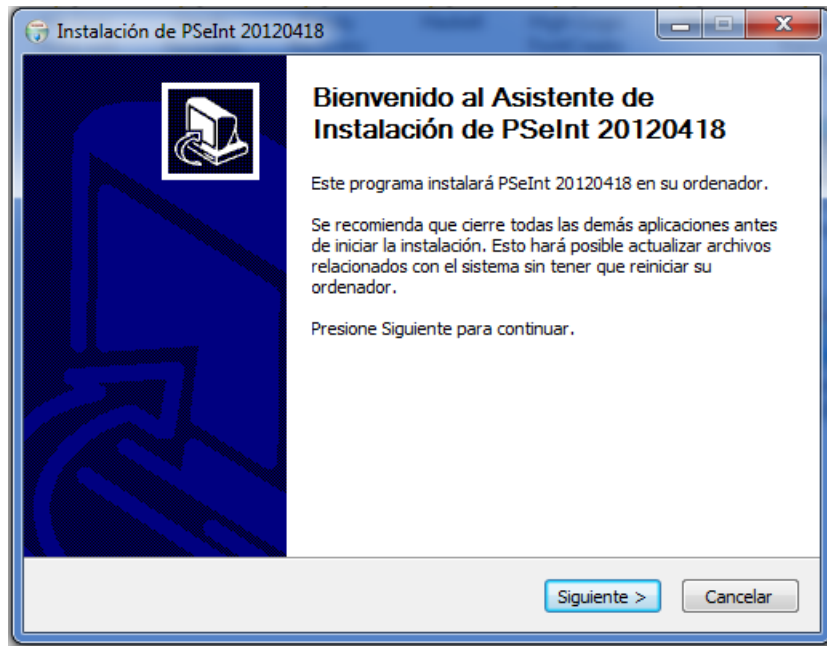
¿Por qué usar PSeInt y no otro intérprete o compilador de pseudocódigo?

- 1) Porque es software libre, sin necesidad de andar gastando dinero, haciendo giros, etc., violando los derechos de autor ni andar creando o consiguiendo cracks, que a veces sus links están inactivos y/o los programas no dejan craquearse.
- 2) Está constantemente atendido por su creador, a diferencia de los otros compiladores o intérpretes de pseudocódigo que están descontinuados.
- 3) Posee un foro para reportar errores y obtener ayuda, que está también constantemente atendido por su creador, esto ayuda a que los usuarios

colaboren para corregir y mejorar el programa y colaboren entre ellos.

- 4) Posee una extensa ayuda, que valga la redundancia ayuda a aprender a usarlo, y a aprender el lenguaje.
- 5) Está disponible su código fuente, y con instrucciones para ejecutarlo, de modo que si sabemos C++ podremos modificarlo, para personalizarlo y/o corregirlo.
- 6) Posee previsualización y exportación a C, C++, Java y otros lenguajes para que podamos ver el mismo código implementado en un lenguaje de programación real, lo que ayuda a aprender estos y otros lenguajes.
- 7) Se trata de un compilador que compila automáticamente cuando el usuario pulsa ejecutar, el algoritmo se guarda automáticamente en un archivo del disco duro, para su posterior ejecución, haciendo más cómodo su uso.
- 8) Es un software multiplataforma, está disponible para Windows (a partir de Windows XP), GNU/Linux y Mac, de modo que podemos seguir utilizándolo pese a que eventualmente tengamos que utilizar otro sistema operativo que no sea Windows. También hay un desarrollo independiente para Android, que usa la sintaxis estricta de PSeInt.

Instalación



Abrir el archivo " pseint-win-32-xxxxxxx.exe " (xxxx es número de la versión actual), el cual será proporcionado por la página del proyecto, al hacer doble clic se ejecuta el instalador.

Luego presionamos siguiente y nuevamente siguiente y así sucesivamente hasta instalarlo.

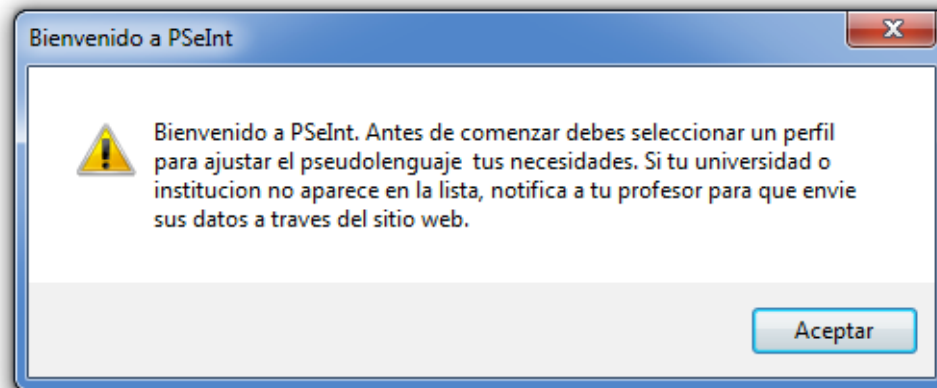
Al instalarlo, todos los archivos .psc (*Pseudocode*) que se generen quedarán automáticamente asociados a PSeInt.

Por otro lado, también se creará un ícono de acceso directo en el escritorio.

Apuntes preliminares

Antes de empezar a programar, es conveniente tener una idea del funcionamiento general de PSeInt.

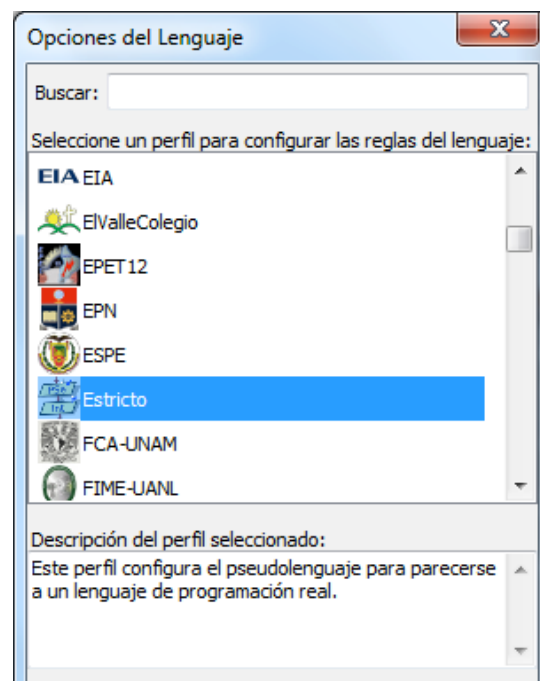
Cuando abrimos por primera vez PSeInt aparece un cartel preguntándonos que perfil deseamos utilizar, para evitar confusiones con el lenguaje.



Este manual se maneja con dos perfiles. El estricto, que es el más parecido a un lenguaje de programación real, se debe respetar al pie de la letra el formato del pseudocódigo propuesto por Novara. La sintaxis personalizada pero basada en la estricta la usaremos para ejecutar ciertos códigos que requieren un poco más flexibilidad a la hora de ejecutarse. A menos que se indique que se deba personalizar la sintaxis estricta, se utilizará la sintaxis estricta sin personalizaciones.

Nota: No confundir Sintaxis estricta con Perfil estricto

Vamos a Configurar → Opciones de Lenguaje → Elegimos Estricto y pulsamos aceptar.



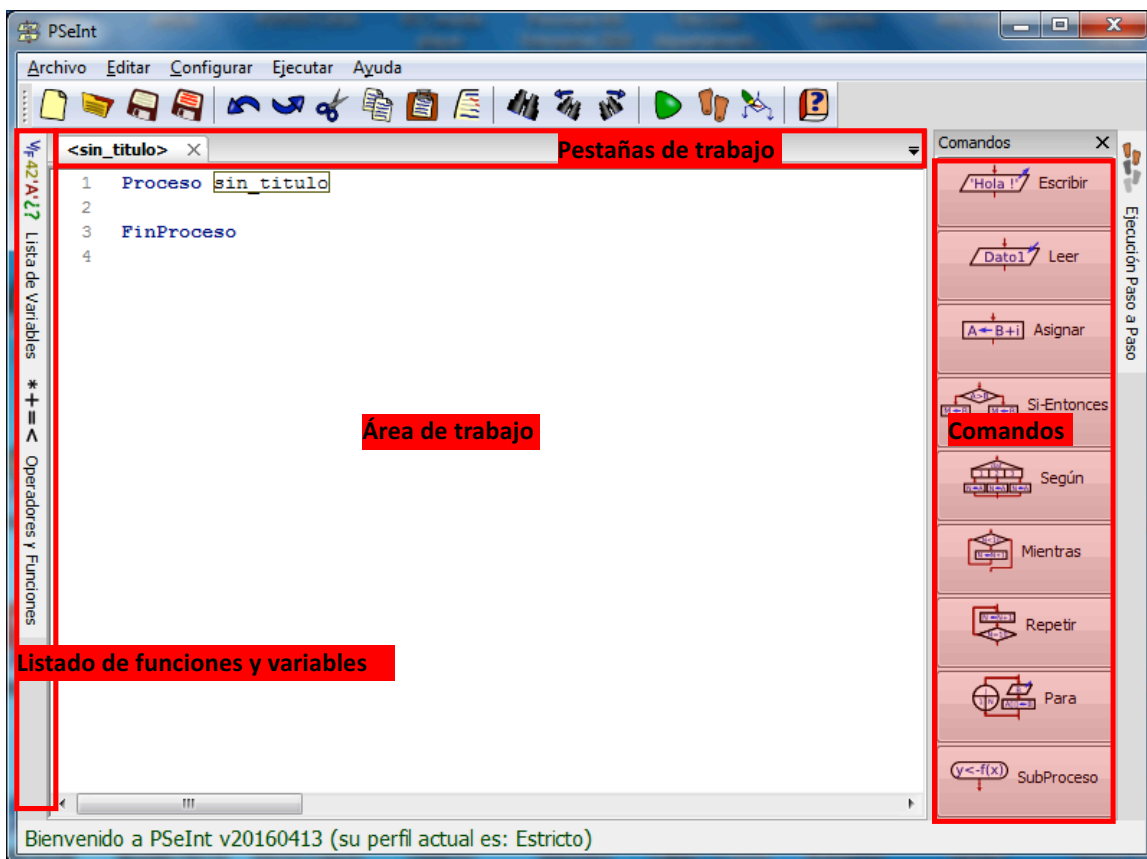
Abrir PSeInt

Para abrir PSeInt damos doble clic en el acceso directo PSeInt del escritorio y nos abre el programa.

Entorno de PSeInt

Ahora que abrimos y configuramos por primera vez PSeInt, pasamos a detallar el entorno de programación de PSeInt.

Esta esta captura se detallan los nombres de las partes que componen el entorno o interfaz del programa



Como se explica en los textos de esta captura, podemos dividir al entorno en cuatro secciones: la de los botones de comando, arriba la de las pestañas de trabajo y el rectángulo blanco que contiene a las palabras Proceso y FinProceso y

el listado de funciones y variables.

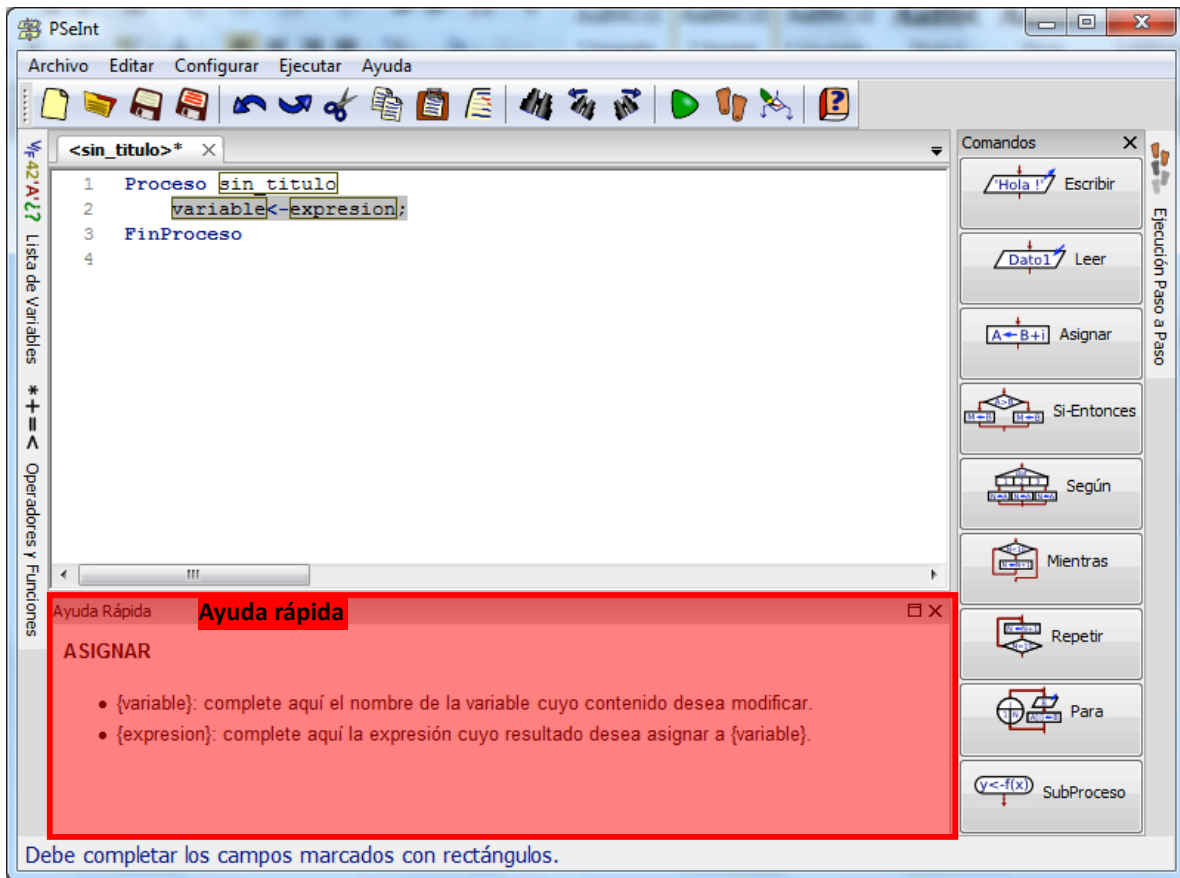
Pasamos a detallar cada una de sus partes de la siguiente forma:

Botones de comando

PSeInt, al ser una herramienta didáctica y orientada a personas con poco o sin ninguna experiencia en programación, presupone que dicho usuario no conoce todavía la sintaxis válida en PSeInt. A tales efectos, como se ve en la captura, en este caso resaltados con rojo a la derecha del entorno, este dispone a su lado botones etiquetados con las estructuras usadas en este programa, que de ser presionadas escriben en el editor de texto dicha sintaxis válida de PSeInt, sirviendo de ayuda al programador o usuario:

Esta sintaxis del pseudocódigo en PSeInt se escribe entre las líneas Proceso y FinProceso, excepto la estructura subprocesso que se escribe fuera del proceso principal ya que un proceso no puede contener ni uno ni más de un subprocesso.

Como parte también de la ayuda, cualquier estructura que se escriba en el área de trabajo abajo muestra su correspondiente descripción que especifica cómo se maneja la estructura citada, lo que en la captura de abajo se especifica como Ayuda rápida.

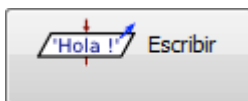


Pasamos a detallar los comandos:

Acciones secuenciales:

Botón Escribir

Dibujo:



Función del botón:

Escribir: Nos permite mostrar en pantalla algún tipo de dato, o varios separados por una coma (,) y esos datos deben estar entre comillas ("")

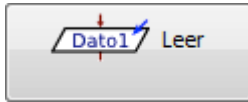
Nota: La variable nunca va entre paréntesis, de lo contrario marca error

Ejemplo de uso:

Ejemplos: **Escribir** "hola mundo"; **Escribir** "hola mundo, hola, 2, c";

Botón Leer

Dibujo:



Función del botón:

Leer: nos permite recibir valores por teclado y guardarlos en variables.

Ejemplo de uso:

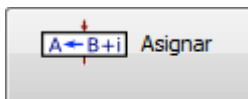
Leer a; //recibe el valor y lo almacena en a.

Leer a, b, c; //recibe 3 valores y los guarda en la variable que a, b y c respectivamente

Nota: La variable nunca va entre paréntesis

Botón Asignar:

Dibujo:



Función del botón:

Asignación: nos permite guardar un valor en una variable.

Ejemplo de uso:

c <- 2; por lo tanto c=2 (c tiene el valor dos), que es lo mismo decir c tiene el valor 2.

Acciones selectivas o interrogativas:

Botón Si

Dibujo:



Función del botón:

Nos permite evaluar la propiedad de una variable, y en función de esta, realizar una acción determinada

Ejemplo de uso:

```
Si cant_numeros != 0 Entonces
    //sentencias
FinSi
```

Botón Segun

Dibujo:



Función del botón:

Nos permite evaluar la propiedad de una variable, y después de comparar una a una las salidas con la evaluación, realizar la acción determinada

Ejemplo de uso:

```
Proceso primer_programa
    Segun num_dia_sem Hacer
        1: Escribir "Lunes";
        2: Escribir "Martes";
        3: Escribir "Miércoles";
        4: Escribir "Jueves";
        5: Escribir "Viernes";
        6: Escribir "Sábado";
        7: Escribir "Domingo";
    De Otro Modo:
        Escribir "No es un día de la semana";
    FinSegun
FinProceso
```

Acciones repetitivas o interactivas:

Botón Mientras

Dibujo:



Función del botón:

Permite realizar cierta acción determinada por la condición del Mientras

Ejemplo de uso:

```
Mientras num != 0 Hacer
    Leer num;
```

```

cant_num<-cant_num+1;
FinMientras

```

Botón Repetir

Dibujo:



Función del botón:

Repite una serie de acciones hasta que se de cierta condición. Dicha acción la determina el operador =, que hace que salga del bucle

Ejemplo de uso:

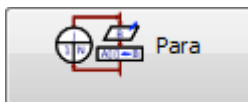
```

Repetir
  Leer num;
  cant_num<-cant_num+1;
Hasta Que num = 0;

```

Botón Para

Dibujo:



Función del botón:

Presenta un cierto rango de valores, y para ellos realiza una determinada acción

Ejemplo de uso:

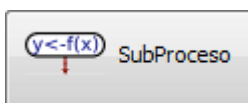
```

Para i <- 1 Hasta 10 Con Paso 1 Hacer
  Escribir i;
FinPara

```

Funciones y SubProcesos

Dibujo:



Función del botón:

SubProcesos: Permite añadir Funciones/SubProcesos al programa

Ejemplo de uso:

SubProcesos que no devuelven valor, solo realizan una tarea específica

```
SubProceso ImprimirResultado(x)
    Escribir "El resultado es: ", x;
FinSubProceso
```

SubProcesos que devuelven valores (Funciones)

```
SubProceso x <- LeerDato(cosa)
    Definir x Como Entero;
    Escribir "Ingrese ", cosa, ": ";
    Leer x;
FinSubProceso
```

Nota: En el caso que nos moleste o que ya no necesitemos este panel podemos cerrarlo con el botón cerrar ubicado a la derecha superior del mismo

Nota 2: Todas las estructuras tanto selectivas como repetitivas, así como también el uso de subprocesos se explican al detalle en cada sub apartado

Área de trabajo

El lugar donde escribimos el código del pseudocódigo.

Los números a la izquierda indican el número de línea de código del programa.

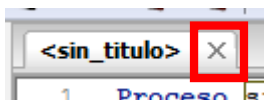
Pestañas de trabajo

Sobre la parte superior del área de trabajo vemos una pestaña que por defecto dice <sin_titulo>

La pestaña activa se corresponde al área de trabajo actual. En caso de que guardemos el archivo en pseudocódigo, la pestaña tomará el nombre del nombre del archivo en pseudocódigo que hayamos guardado.

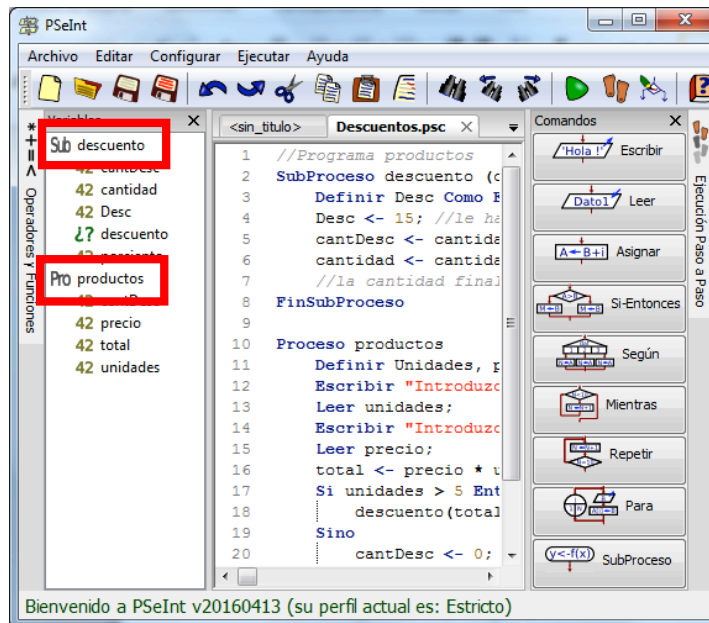
Por otro lado, PSeInt puede abrir varios archivos en pseudocódigo a la vez, mostrándose en las pestañas de trabajo.

Se pueden cerrar el proyecto con el botón cerrar de la pestaña.



Listado de funciones y variables

A la izquierda vemos dos pestañas. La de más arriba, como su nombre lo indica, muestra la lista de variables. Los signos que aparecen antes del texto (V/F de verdadero/falso, números, letras y signos de interrogación) ofician de íconos del texto. Si hacemos clic izquierdo sobre el texto, se abre una solapa que detalla el Proceso y los SubProcesos que eventualmente puedan existir en el pseudocódigo.



En el panel, como se observa en la imagen, el Proceso se marca con el ícono Pro seguido por el nombre del Proceso. Por su parte, el SubProceso se marca con el ícono Sub seguido del nombre del SubProceso. Nótese que subordinado al ícono Sub y el nombre del SubProceso aparece el número 42 seguido de las variables. Estas variables son los argumentos de la función o la variable de retorno. Este 42 significa los potenciales tipos de datos que determina el intérprete en caso de que el tipo de variable pueda deducirse antes de ejecutar el algoritmo.

En caso de tratarse del proceso principal, estos textos que se muestran son las variables que se usan en ese proceso principal.

Nota: Como bien menciona Pablo Novara en este programa no puede hacer

instrucciones fuera ni de los Procesos ni de los SubProcesos, por lo tanto no es posible declarar variables globales.

Nota2: Si ponemos el cursor sobre el 42 o la variable aparece un cartel con la ayuda de correspondiente

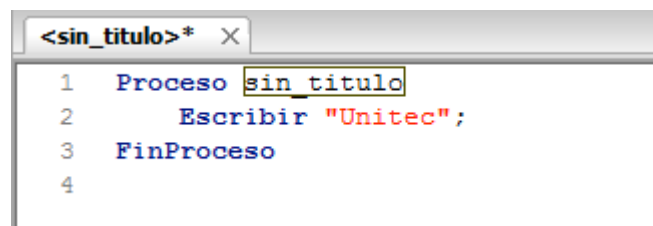
Si hacemos clic sobre el ícono Sub, PSeInt nos marca con azul en que parte del pseudocódigo se encuentra el SubProceso. Lo mismo sucede hacemos clic sobre las variables. Eso es útil cuando tenemos múltiples SubProcesos, variables o parámetros y se nos hace difícil saber dónde está cada una de ellas/ellos.

Ayuda

En esta área, que se encuentra en la parte inferior de programa, como se le informa al programador los errores que puedan tener el programa y una descripción para que el programador entienda como corregir el error en caso de que el programador no sepa corregirlos.

Escribir mi primer programa

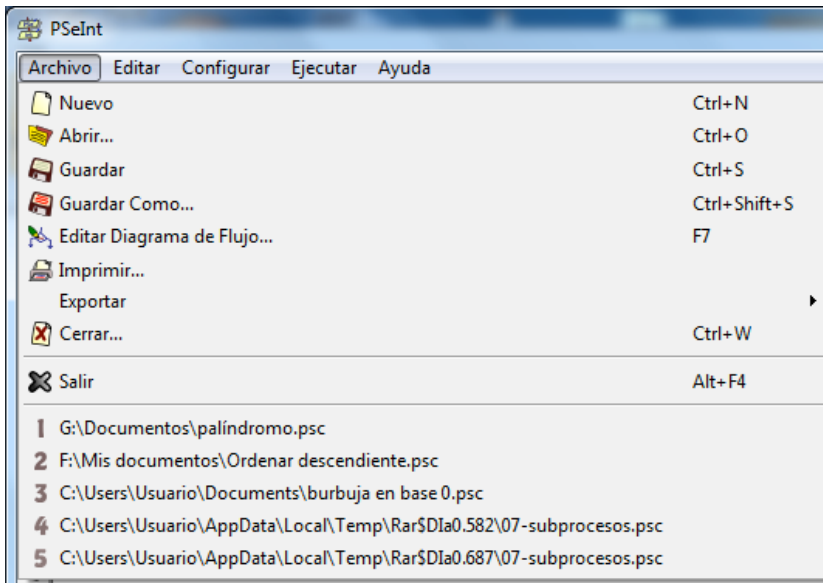
Ya analizado el entorno y conociendo sus partes, pasamos a escribir nuestro primer programa. Ya abierto PSeInt y habiendo configurado sintaxis estricta, este nos presenta el área de trabajo con dos palabras claves Proceso seguido del nombre sin_titulo y FinProceso, entre estas dos líneas escribiremos nuestro primer programa:



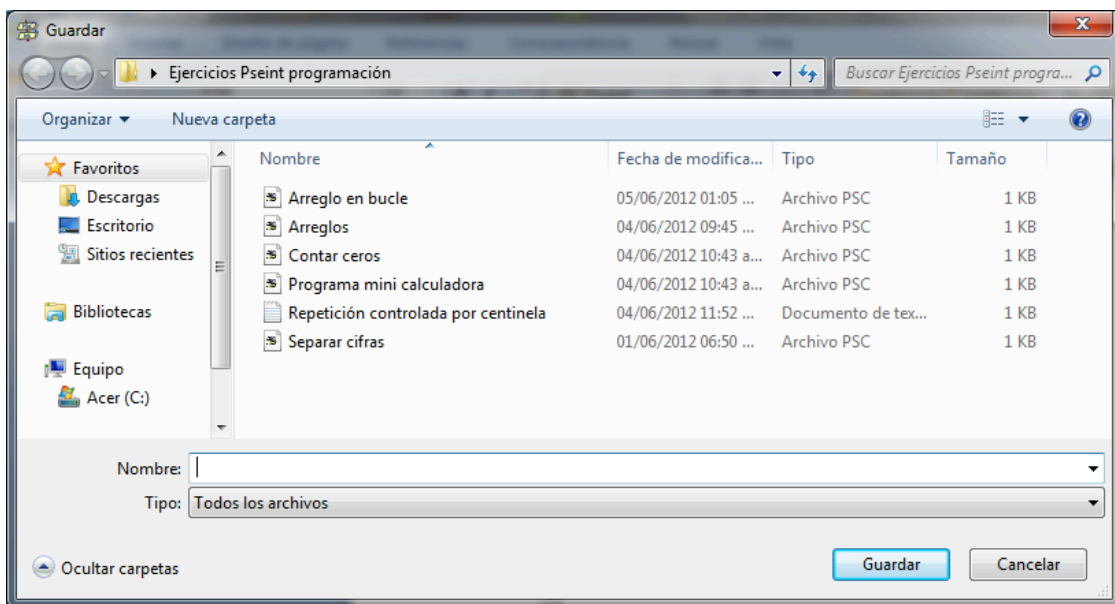
```

<sin_titulo>*
1  Proceso sin_titulo
2      Escribir "Unitec";
3  FinProceso
4
  
```


Luego lo guardamos



Escribimos el nombre del programa en la ventana que nos aparece y luego presionamos Guardar Como... .



Ahora que los hemos guardado necesitamos que nuestro programa funcione y escriba en la pantalla Unitec, pero aunque PSeInt subraya con rojo los errores de sintaxis, es bueno verificar sintaxis para ver los errores. Para ello vamos a

ejecutar, → verificar sintaxis. De todos modos, si tuviéramos errores él nos subrayaría la frase donde se encuentre el error, luego lo corregimos y lo volvemos a ejecutar, hasta que no aparezca nada subrayado con rojo.

Luego que el programa no tiene errores de compilación, es decir no aparecen líneas subrayadas con rojo, vamos al menú ejecutar, luego seleccionamos opción ejecutar, y en la pantalla aparecerá la palabra Unitec que es la salida del programa, también para ejecutar el programa puede usar el ícono de ejecutar o pulsar F9:



Si la ejecución se realizó con éxito correcta al final aparecerá un mensaje diciendo que el programa se ejecutó correctamente.

Siempre que queremos escribir un programa en PSeInt iniciamos debajo de la palabra

Proceso sin_titulo

//escribimos el cuerpo del programa;

FinProceso

Y el proceso principal se cierra con las palabras claves FinProceso que indica el final del programa principal.

Combine asignarle un nombre al programa, sustituyendo sin_titulo por el nombre que queramos darle. Recordar que nombre del pseudocódigo en ninguna sintaxis puede tener espacios y en sintaxis estricta tampoco caracteres acentuados. No confundir el nombre del proceso con el del archivo en pseudocódigo.

La palabra reservada **Escribir** *escribe en la pantalla lo que está encerrado entre comillas*. En sintaxis flexible también podemos utilizar la palabra **Imprimir** o **Mostrar**. *Reitero, a menos que se indique personalizar la sintaxis estricta, se utilizará siempre sintaxis Estricta.*

Número de línea

A la derecha, al igual que ocurre en la mayoría de los IDEs, a la izquierda del editor de código se encuentran una serie de números en forma ascendente hacia abajo.

```

1  E
2
3
4
5
6
7
8  E
9
10 S
11
12
13
14
15
16
17
18 E
19

```

Esto significa el número de línea de la instrucción. Esto es particularmente importante en programas largos, dónde, en el caso de producirse un error en el programa, PSeInt marcará la línea dónde está el error, haciendo que sea más fácil identificar la línea dónde se encuentra dicho error.

En el caso de que se produzca un error lógico, también va a marcar la línea dónde está el error.

Ventana de ejecución

PSeInt dispone de dos ventanas de ejecución. Una es la propia y la otra es la terminal del sistema. La que vimos arriba es la terminal propia y que aparece por defecto en PSeInt, que a su vez por defecto es de fondo blanco, aunque también podamos utilizar la terminal del sistema.

Esta ventana tiene dos particularidades que pueden ser una ventaja a la hora de ejecutar nuestro pseudocódigo: permite reiniciar el programa una vez finalizado sin tener que cerrar la ventana y ejecutar el programa de nuevo. Esto tiene la ventaja de ser más cómodo su uso y poder testear más eficiente nuestro programa.

Por otro lado, la otra particularidad, mientras ejecutamos nuestro algoritmo con esta ventana, nos va marcando el número de línea que el programa va ejecutando.



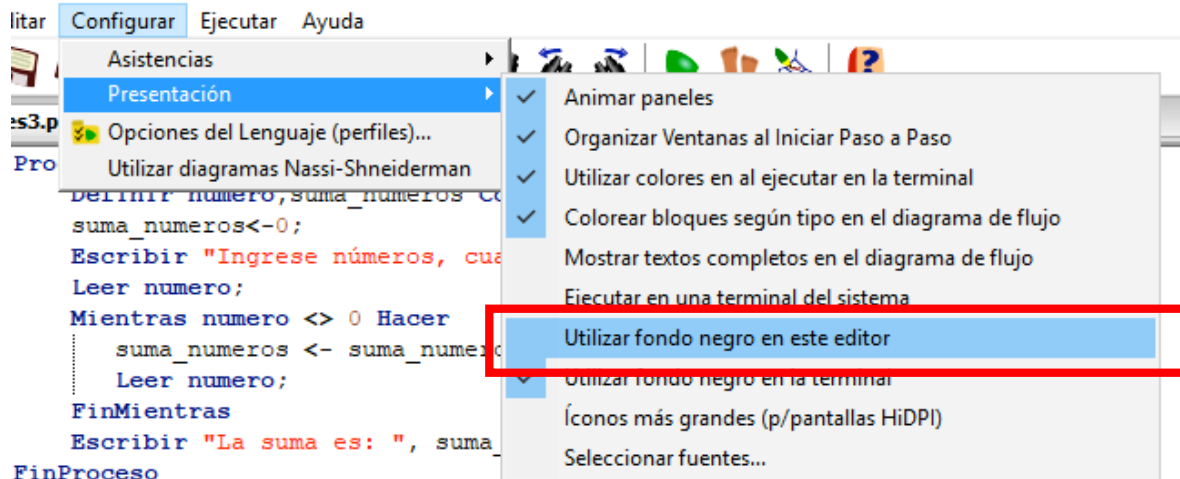
Esto significa que se está ejecutando la línea 8 del pseudocódigo.

Cuando nuestro programa en pseudocódigo finaliza, se nos da la posibilidad de reiniciar el programa desde el botón reiniciar, ubicado en el costado inferior derecho de la ventana.

El programa empezará de nuevo.

Nota: Por defecto, la terminal se muestra con colores. Podemos cambiarlo yendo a *Configurar → Presentación → Utilizar colores en al ejecutar en la terminal*.

También hay una opción para cambiar el color de fondo yendo nuevamente a *Presentación → Utilizar fondo negro en este editor*



Nota2: Podemos utilizar la ventana terminal del sistema yendo a Configurar → Asistencias → Utilizar fondo negro en la terminal

Concatenar texto

Yendo nuevamente al código, podemos concatenar texto para que se muestre o escriba en pantalla de una determinada forma u otra como detallaremos a continuación.

```
Proceso primer_programa
    Escribir "Mi primer programa";
    Escribir " en PSeInt ";
FinProceso
```

La salida del programa es

Mi primer programa

En PSeInt

Esto porque aunque se escriba otro texto, y se coloque la palabra clave Escribir, el texto sigue escribiéndose al final del otro, ahora si quisiéramos escribir:

Mi primer programa En PSeInt

El programa es de esta forma ejemplo

```
Proceso primer_programa
    Escribir "Mi primer programa " Sin Saltar;
    Escribir "En PSeInt ";
```

FinProceso

Con esto deducimos que la instrucción Sin Saltar concatena, en este caso, el contenido de una cadena de texto, y el contenido del próximo Escribir se escribe a continuación del anterior y no en la línea de abajo.

Recordar que en sintaxis estricta la colocación del punto y coma al final de las sentencias es obligatoria, en flexible es opcional.

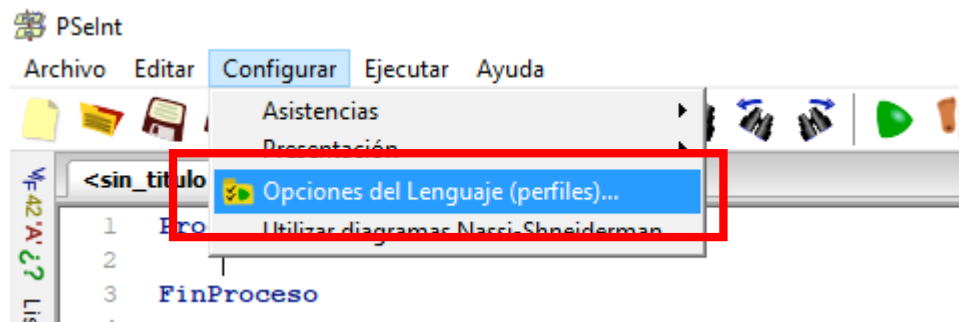
Nota: Las comillas deben ser siempre simples y nunca tipográficas pues estas últimas son símbolos gráficos que ningún lenguaje de programación hasta el momento puede interpretar. Siempre por defecto en los editores de texto de los IDEs se escriben comillas simples, pero cuando se importa o se formatea pseudocódigo traído de afuera, hay que corregir el encomillado, de no hacerlo provocaría un error de compilación.

Nota 2: PSeInt no es case sensitive, por lo tanto colocar Escribir con mayúsculas y minúsculas es lo mismo y no genera errores de ningún tipo, pero por respeto a la sintaxis mostrada por los botones se debe escribir con mayúscula inicial, evitando así errores de formato.

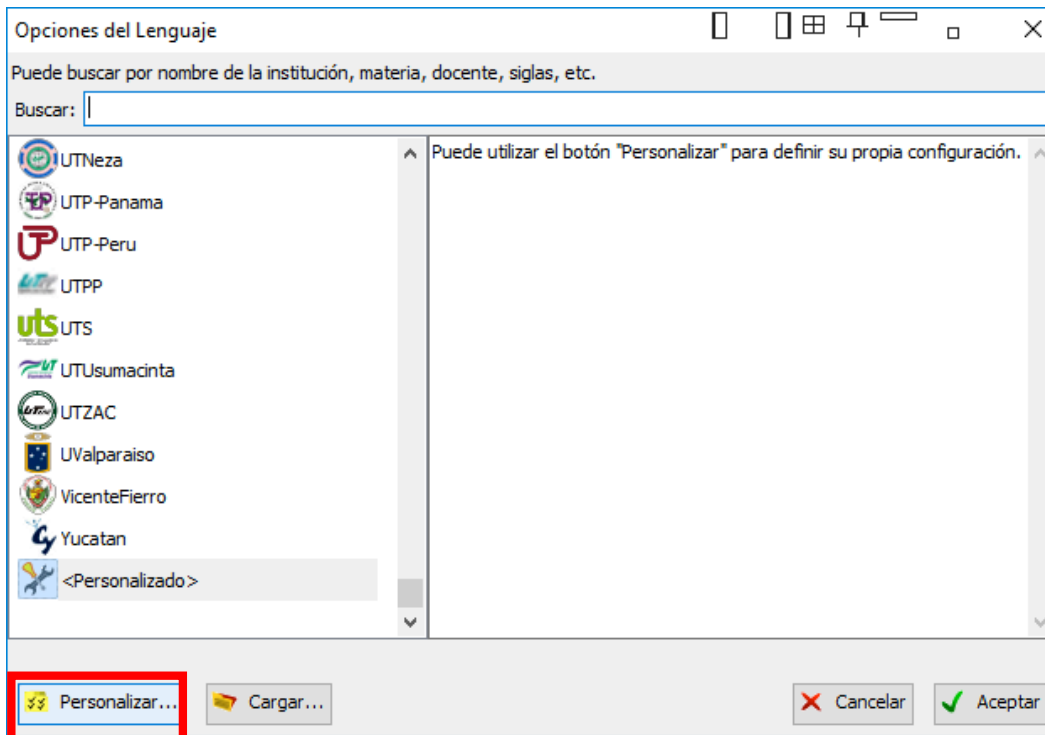
Nota 3: En sintaxis estricta, las sentencias siempre finalizan en punto y coma.

Nota4: El Proceso principal debe cerrarse con la palabra FinProceso y, en sintaxis estricta, el nombre del proceso y los eventuales subprocesos que haya en el programa no deben tener letras acentuadas.

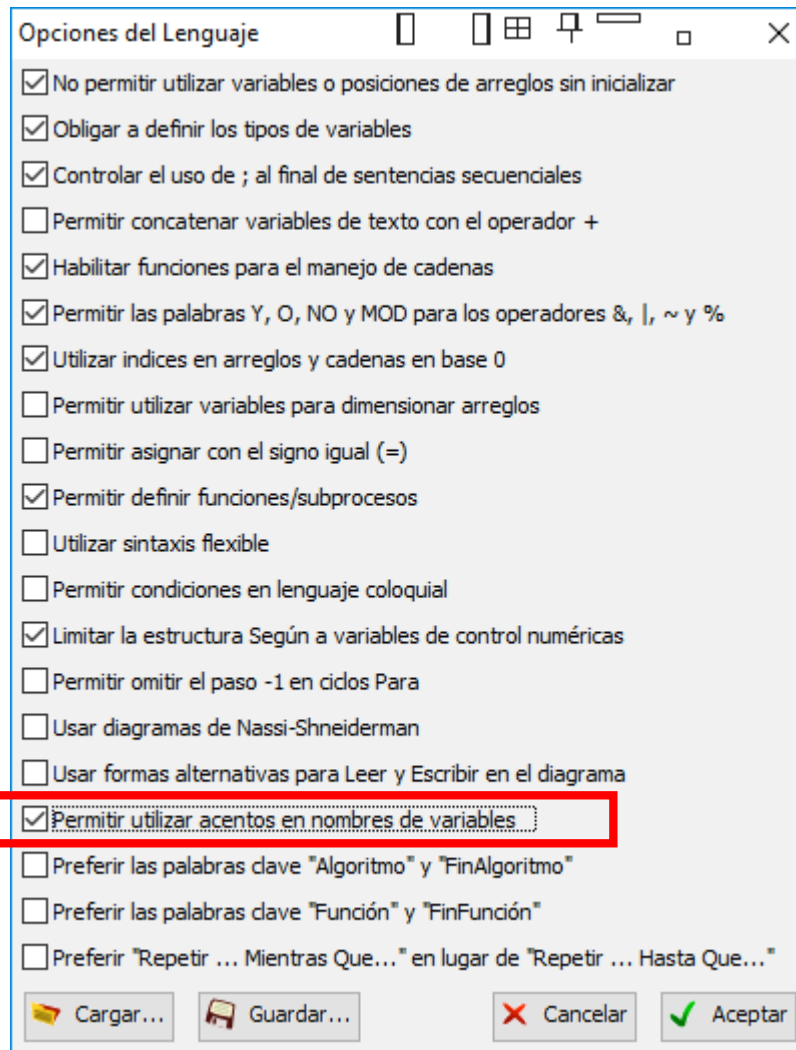
De necesitar variables con letras acentuadas, vamos a Configurar → Opciones de lenguaje (perfiles)



En el cuadro que se nos presenta vamos a Personalizar




Se nos presentará este cuadro:



Tildamos la opción Permitir utilizar acentos en nombres de variables. Seguidamente pulsamos en Aceptar. Y aceptar nuevamente para cerrar el cuadro de perfiles.

Diagramas de flujo

PSeInt es capaz de interpretar los pseudocódigos y transformarlos a diagrama de flujo, para eso dispone de un visualizador y editor de diagramas de flujo. Esto es útil si queremos analizar el pseudocódigo desde un punto de vista gráfico.

Se accede pulsando el ícono  de la barra de tareas. PSeInt no solo es capaz de visualizarlo, sino también editarlos.

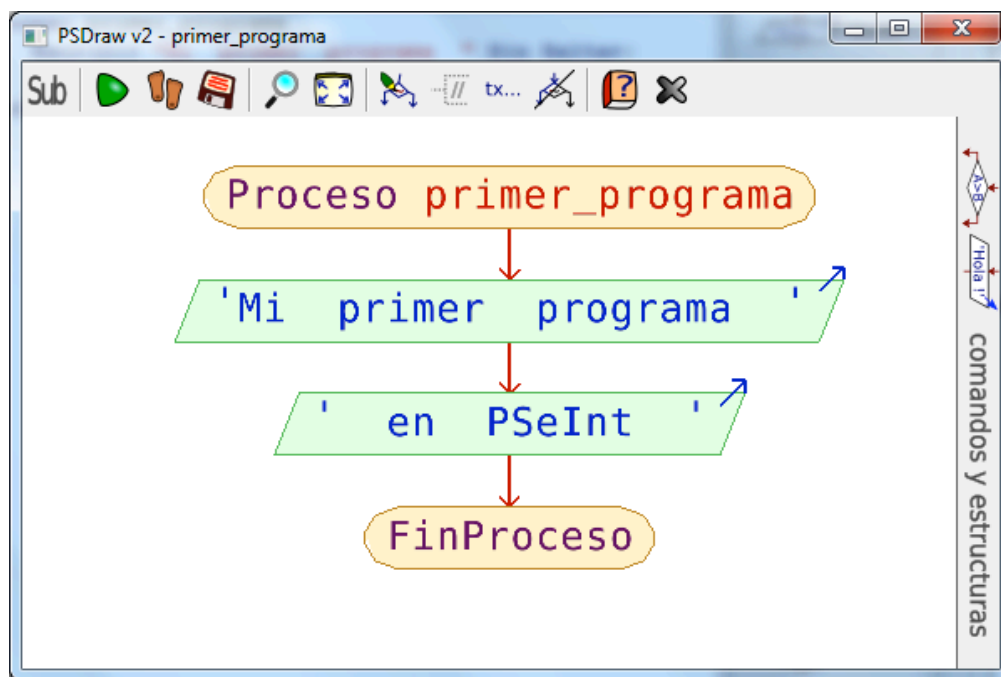


Ejemplo:

Considera el siguiente programa

```
Proceso primer_programa
    Escribir "Mi primer programa " Sin Saltar;
    Escribir " en PSeInt ";
FinProceso
```

Su representación en diagrama de flujo es la siguiente:



Aquí vemos el inicio del proceso representado como una elipse, pues es donde comienza el programa, la sentencia escribir representada en un rectángulo, pues es una entrada o salida, pero en este caso es salida, por pantalla (obsérvese la flecha de salida) y abajo nuevamente una elipse que representa el fin del proceso.

Nota 1: En la página

<http://pseint.sourceforge.net/index.php?page=pseudocodigo.php> del sitio oficial de PSeInt se explica la estructura de los diagramas de flujo.

Nota 2: En el anexo se explica cómo editar diagramas de flujo.

Declarar variables

En sintaxis estricta, siempre que necesitemos hacer un programa, tendremos que declarar variables para poder guardar la información que introduzcamos al programa.

Los tipos de datos básicos soportados son los siguientes:

1. Entero: solo números enteros.
2. Real: números con cifras decimales.
3. Caracter: cuando queremos guardar un carácter.
4. Logico: cuando necesitamos guardar una expresión lógica (verdadero o falso)
5. Cadena: cuando queremos guardar cadenas de caracteres.

Nota 1: Cadena y Caracter son términos equivalentes, no genera error que las escribamos indistintamente.

Nota 2: El plural de Caracter es Caracteres o Cadena

Ejemplos de declaración de variables:

Si queremos declarar una variable de tipo entero sería así:

```
Definir numero Como Entero;
```

Numero se convierte en una variable de tipo entero

Nota 3: En sintaxis estricta, ni los nombres de variables ni palabras claves pueden tener caracteres acentuados

Si queremos declarar una variable tipo Cadena para guardar el nombre sería así

```
Dimension nombre[25];
Definir nombre Como Cadena;
```

Nombre sería una variable que guardaría solo 25 caracteres aunque podemos escribir más de 25 letras, él en la memoria solo guardara los primeros 25 caracteres.

Nota 4: Ver el apartado Dimensiones para más detalles.

Nota 5: Aunque esto no genere errores en tiempo de ejecución, si se declaran varias variables a la vez, para evitar un error de formato – concordancia, de debe pluralizar el tipo de variable. Ej.: Definir a, b, c Como Enteros;

Nota 6: la capacidad de la dimensión debe ir pegada su nombre, si se deja un espacio marca error.

Operadores

PSeInt proporciona los siguientes operadores:

Operador Función

()	Agrupar expresiones
^	Operador para exponenciación
*	Operador de multiplicación
/	Operador de división
% ó mod	Operador de cálculo de residuo
trunc(valor1 / valor2);	Sintaxis de división entera
& ó y	Operador lógica y
+	Operador de suma
-	Operador de Resta
ó o	Operador lógico o

Nota: Tanto en sintaxis flexible como estricta, podemos utilizar también los operadores & | y mod como y o y % respectivamente.

Leer valores y almacenarlos en las variables

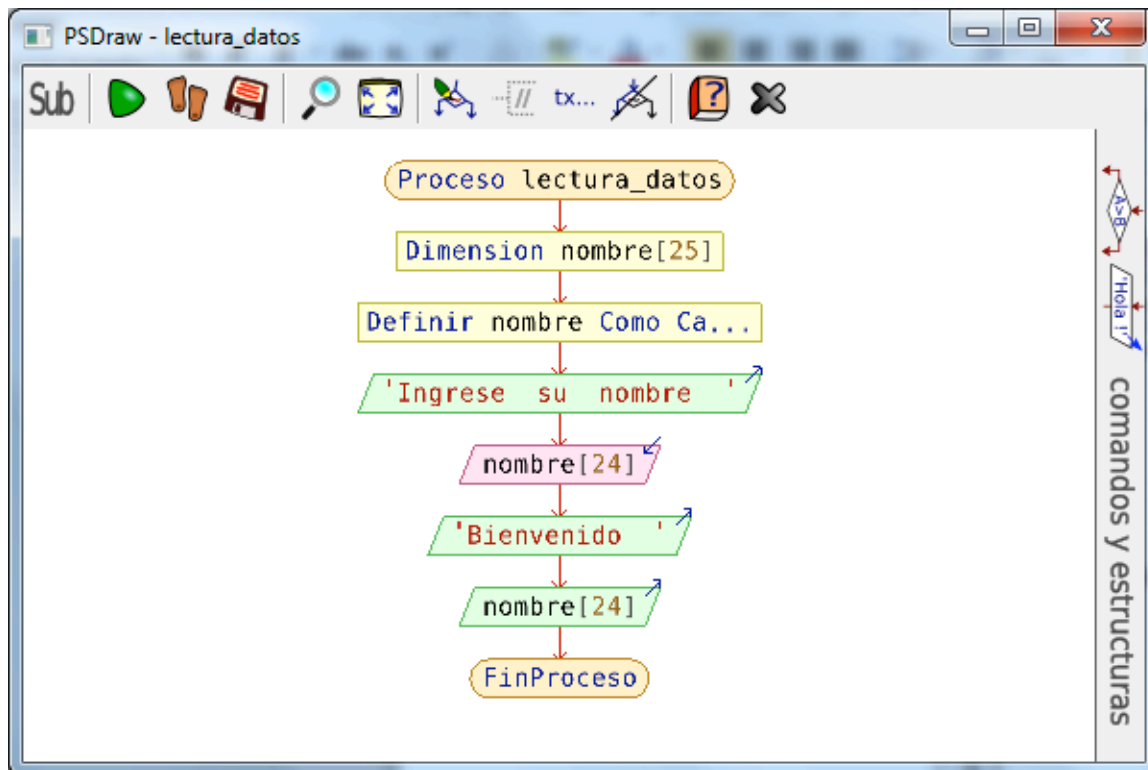
Cuando nosotros queremos leer un valor y almacenarlo en una variable usaremos la palabra **Leer < variable>**; y en sintaxis estricta, cuando queremos asignar un valor o una operación matemática, en sintaxis estricta, usaremos <- que es el símbolo de < más - o := (símbolo : más =).

Ejemplo sobre lectura de datos

```
Proceso lectura_datos
    Dimension nombre[25];
    Definir nombre Como Cadena;
    Escribir "Ingrese su nombre ";
    Leer nombre[24];
    Escribir "Bienvenido ";
    Escribir nombre[24];
FinProceso
```

El programa declara una variable para el nombre, que guarda 25 caracteres máximo, ingresa el nombre y luego escribe en la pantalla Bienvenido el nombre que se ingresó. Algo importante es que cuando se quiere presentar el valor de la variable esta no se escribe entre comillas.

Su diagrama de flujo:



En la tabla se nos muestra como se pudo sustituir un bloque del programa que nos daría el mismo resultado

Caso 1	Caso 2
<pre> Escribir "Bienvenido "; Escribir nombre; </pre>	<pre> Escribir "bienvenido " Sin Saltar , nombre; </pre>

Nota: No es necesario indicar de cuantos caracteres es la cadena que PSeInt debe leer, pero si se debe indicar si declaramos a la dimensión como un vector de caracteres.

Asignaciones y Operaciones matemáticas en un programa.

En sintaxis estricta, el símbolo <- lo usaremos para asignar valores a las

variables ejemplo **Sueldo<-500**; Con esta instrucción estamos asignando el valor de 500 a la variable sueldo que pudo declararse como entero o real

Nombre<-"juan"; con esta instrucción asignamos la cadena "Juan " a la variable nombre que es una variable de tipo cadena

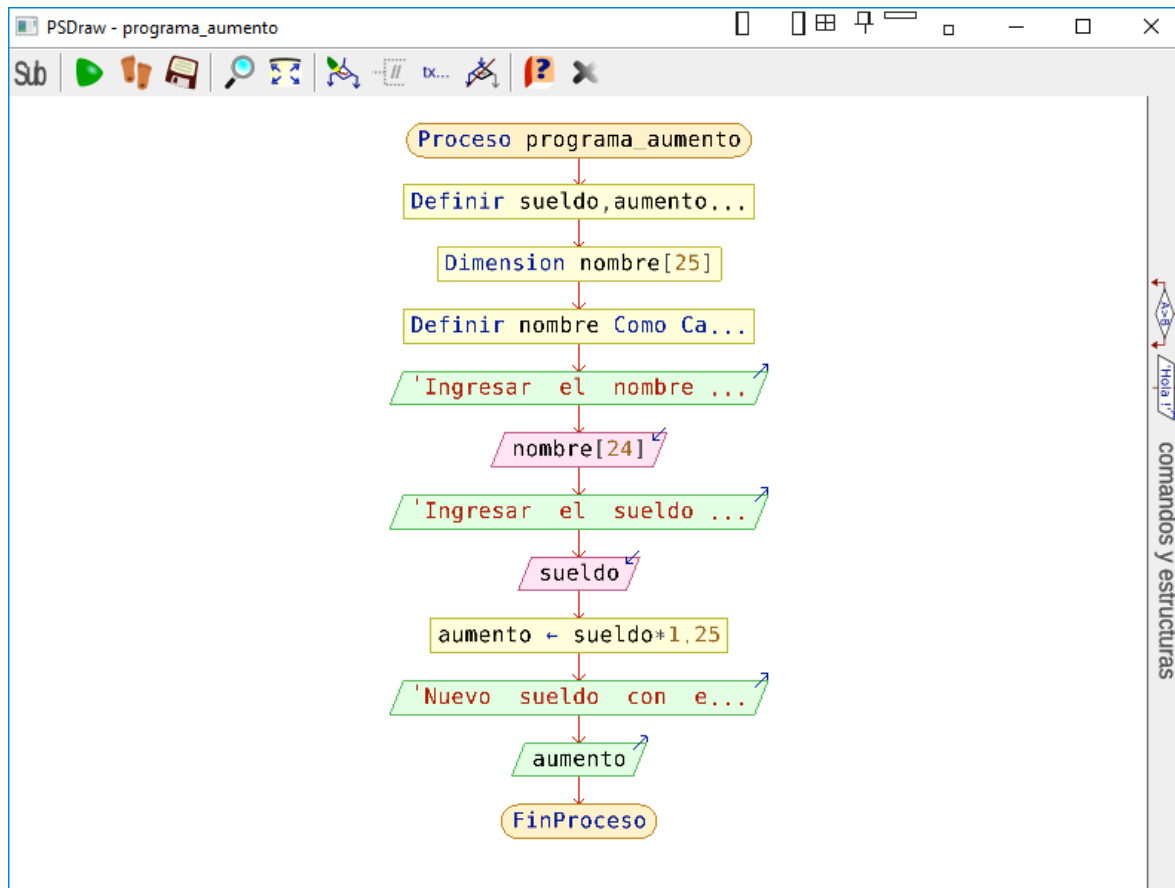
Nota: En sintaxis estricta, también se puede utilizar **:=** para asignar variables

Ejemplo sobre asignaciones de valores a las variables

```
Proceso programa_aumento
  Definir sueldo Como Entero;
  Definir aumento Como Real;
  Dimension nombre[25];
  Definir nombre Como Cadena;
  Escribir "Ingresar el nombre del empleado";
  Leer nombre[24];
  Escribir "Ingresar el sueldo del empleado";
  Leer sueldo;
  Aumento<-sueldo*1.25;
  Escribir "Nuevo sueldo con el 25% de aumento";
  Escribir aumento;
FinProceso
```

El programa pide el nombre y el sueldo del empleado luego calcula el 25% de aumento de sueldo y lo guarda en la variable aumento y luego presenta el nuevo sueldo.

Diagrama de flujo:



Ejemplo sobre suma de cadenas

Nota: En sintaxis flexible también se puede concatenar textos y arreglos con el operador +

```

Proceso suma_de_cadenas
  Dimension nombre[25], apellido[25], completo[25];
  Definir nombre, apellido, completo Como Cadenas;
  Escribir "Su Nombre";
  Leer nombre[24];
  Escribir "Apellido";
  Leer apellido[24];
  Completo[24] <- concatenar(concatenar(nombre[24], "
"), apellido[24]);
  Escribir "Nombre completo ", completo[24];
FinProceso
  
```

La variable completo toma el valor del nombre más un espacio en blanco más el apellido y lo guardamos en una variable donde ahora tenemos el nombre y el apellido.

Usamos la función predefinida concatenar para concatenar ambas cadenas

Nota: No es estrictamente necesario dimensionar cadenas de caracteres. Véase la página que trata el tema de dimensiones.

Instrucciones condicionales

Anteriormente hemos estado haciendo programas que solo hacen cálculos, pero la programación es más interesante cuando nuestros programas toman sus propias decisiones, en PSeInt existen instrucciones condicionales que se describen a continuación:

Instrucción Si:

sintaxis

```
Si condición Entonces instrucciones;
FinSi
```

ó

```
Si condición Entonces instrucciones;
Sino
    instrucciones;
FinSi
```

Ejemplo sobre decisiones

Ingresar un número y si el número es mayor a 100, escribir en la pantalla el

número es mayor a 100.

```
Proceso decision
    Definir num como Entero;
    Escribir "Ingresar un número";
    Leer num;
    Si num > 100 Entonces
```

En programa solo escribirá que el número fue mayor a 100 cuando cumpla con la condición **num > 100** sino cumple con la condición no hace nada.

Ejemplo sobre decisiones

Ingresar el nombre del empleado, las horas trabajadas, luego Calcular pago bruto (50 lps la hora) IHSS y total a pagar, presentar los resultados del programa

Nota: *el seguro social es 84 si el sueldo es mayor 2400 sino es el 3.5% del sueldo del empleado.*

```
Proceso empleados
    Definir horas como Entero;
    Definir Pbruto,ihss,tp como Reales;
    Dimension nombre[25];
    Definir nombre Como Cadena;
    Escribir "Ingresar el nombre";
    Leer nombre[24];
    Escribir "Ingresar las horas trabajadas";
    Leer horas;
    Pbruto<-horas*50;
    Si pbruto > 2400 Entonces
        Ihss<-84;
    Sino
        Ihss<-0.035*pbruto;
    FinSi
    Tp<-pbruto-ihss;
    Escribir "Pago bruto " , pbruto;
    Escribir "Seguro Social " , ihss;
    Escribir "Total a pagar " , tp;
FinProceso
```

En este programa se usó en el cálculo del ihss una decisión que tiene dos salidas una cuando se cumple la condición que es el entonces y la otra cuando no se cumple la condición que es el sino, ahora esto nos ayuda a que nuestros programas puedan tomar una decisión cuando la condición se cumple y otra cuando no se cumple.

Ahora en el siguiente ejercicio que se presenta, ya no hay dos soluciones a la condición sino tres, cuando sucede esto se usan condiciones anidadas.

Sintaxis de una condición anidada:

```

Si condición 1 Entonces Instrucciones;
    Sino Si condición 2 Entonces Instrucciones;
        Sino Si condición 2 Entonces Instrucciones;
            Sino
                Instrucciones;
        FinSi
    FinSi
FinSi

```

Ejemplo sobre decisiones anidadas

Ingresa el nombre del empleado, la zona de trabajo, las ventas del empleado, luego calcular su comisión en base a un porcentaje basado en la zona de trabajo, luego determinar el IHSS y el total a pagar, presentar los datos.

Tabla para el cálculo de la comisión

Zona	Porcentaje de Comisión
A	6%
B	8%
C	9%

```

Proceso Comision
    Definir zona como Caracter;
    Dimension nombre[25];
    Definir nombre Como Cadena;
    Definir ventas, comis, ihss, tp Como Reales;

```

```

Escribir "Ingresar el nombre del empleado ";
Leer nombre[24];
Escribir "Ingresar las ventas del empleado ";
Leer ventas;
Escribir "Ingresar la zona de trabajo ";
Leer zona;
Si zona ='A' Entonces
    comis<- 0.06 * ventas;
Sino Si zona='B' Entonces
    comis<- 0.08 * ventas;
    Sino Si zona='C' Entonces
        comis<- 0.09 * ventas;
    Sino
        comis<- 0;
    FinSi
FinSi
FinSi
Si comis > 2400 Entonces
    ihss<-84;
Sino
    ihss<-0.035*comis;
    tp<-comis-ihss;
FinSi
Escribir "Comisión ganada ", comis;
Escribir "Seguro Social ", ihss;
Escribir "Total a pagar ", tp;
FinProceso

```

En este programa usamos decisiones anidadas para el cálculo de la comisión del empleado, esto porque se tenían varias opciones de la cuales elegir.

El ultimo sino donde la comisión es 0 se hace porque no estamos seguros de que la persona que opera el programa introduzca correctamente la zona, si se ingresó otra zona de las permitidas la comisión es cero.

Estructura Segun

Esta se usa como sustituto en algunos casos del si anidado, por ser más práctico al aplicarlo en la evaluación de algunas condiciones.

Sintaxis

Segun variable **Hacer**

```
valor1, valor2, valor3, ... : instrucciones;
valor1, valor2, valor3, ... : instrucciones;
.
.
[ De Otro Modo : instrucciones;]
```

FinSegun

Los valores a evaluar se separan por comas si hay varios, tal como aparece en la sintaxis valor1, valor2 etc., también se puede usar el De Otro Modo que nos indica, que en caso de no seleccionar ninguna de las instrucciones anteriores se ejecutan.

Nota importante: En sintaxis estricta las opciones del Segun deben ser siempre del tipo numérico. Para poder evaluar opciones del tipo texto se debe personalizar el lenguaje utilizando sintaxis flexible, o personalizando el lenguaje des tildando. Limitar la estructura Según a variables de control numéricas en el editor o en su defecto utilizar otro perfil, como el perfil taller de informática o el flexible.

Ejemplo sobre la aplicación de la estructura Segun

En el ejercicio anterior usamos decisiones anidadas para determinar la comisión, ahora usaremos una estructura Según.

Para realizar el ejercicio de abajo debemos configurar PSeInt para que acepte opciones del Segun tipo Carácter o Cadena.

Hay varias formas de configurar PSeInt para lograr esto, como se detalla más arriba, pero en este manual lo hacemos personalizando el perfil y des tildando la opción Limitar la estructura Según a variables de control numéricas.

Ahora si estamos en condiciones de ejecutar el código que se encuentra más abajo:

```
Proceso ejemplo_caso
  Definir zona Como Carácter;
  Dimension nombre[25];
  Definir nombre Como Cadena;
  Definir ventas,comis,ihss,tp Como Reales;
  Escribir "Ingresar el nombre del empleado ";
  Leer nombre[24];
  Escribir "Ingresar las ventas del empleado ";
  Leer ventas;
  Escribir "Ingresar la zona de trabajo ";
  Leer zona;
  Segun Zona Hacer
    'a','A' :      comis<- 0.06 * ventas;
    'b','B' :      comis<- 0.08 * ventas;
    'c','C' :      comis<- 0.09 * ventas;
  De Otro Modo :
    comis<- 0;
  FinSegun
  Si comis > 2400 Entonces
    ihss<- 84;
  Sino
    ihss<-0.035*comis;
```

```

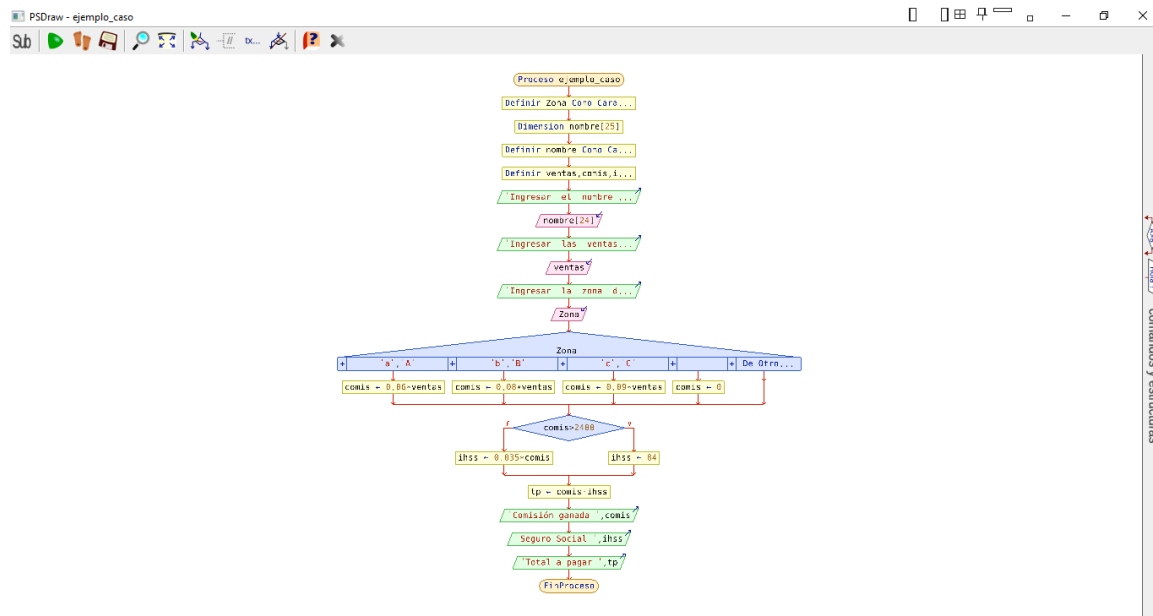
FinSi
tp<-comis - ihss;
Escribir "Comisión ganada ", comis;
Escribir "Seguro Social ", ihss;
Escribir "Total a pagar ", tp;
FinProceso

```

Ahora nuestro programa reconoce las mayúsculas y minúsculas en la evaluación de la zona

Nota: Para evitar ambigüedades en el lenguaje, las estructuras de selección deben cerrarse siempre, no es posible que hayan “Estructuras abiertas”.

Su representación en diagrama de flujo:



Uso del operador | ú o

El operador | (O) se utiliza cuando estamos evaluando dos o más condiciones y queremos que la condición se cumpla cuando una de las condiciones que estamos evaluando se hacen verdadera. Ejemplo

Cuando se introduce la zona en el ejercicio con la estructura Si solo evaluábamos una opción que la zona sea igual a la letra A y si el usuario escribía una a

minúscula no se tomaba en cuenta pero esto se puede corregir de esta manera:

```

Si  zona  ='A'      |      zona  ='a'      Entonces
                                comis<- 0.06 * ventas;
Sino Si zona='B' | zona='b' Entonces
                                comis<- 0.08 * ventas;
Sino Si zona='C' | zona='c' Entonces
                                comis<- 0.09 * ventas;
Sino
                                comis<- 0;
FinSi
FinSi
FinSi

```

Ahora la condición dice, **si zona es igual a la letra A o es igual a la letra a**, cualquiera que sea la zona a o A en ambos casos la condición es verdadera, ahora el usuario puede usar mayúsculas y minúsculas y el resultado será el mismo.

Ejemplo sobre el operador |

Ingresar el nombre del cliente, luego la cantidad del producto, precio y tipo de cliente, calcular el subtotal, descuento, impuesto s/v, total a pagar, presentar los datos.

El descuento es del 10% si el cliente es de tipo A o la cantidad de cualquier producto es mayor a 100 sino es de 5%.

```

Proceso descuento
    Definir precio,st,des,tp,sv Como Reales;
    Dimension nombre[25];
    Definir nombre Como Cadena;
    Definir tipoM Como Caracter;
    Definir cant Como Entero;
    Escribir "Nombre del cliente";
    Leer nombre[24];
    Escribir "Ingresar el Tipo de cliente";
    Leer tipoM;
    Escribir "Ingresar el precio del producto";
    Leer precio;
    Escribir "Ingresar la cantidad ";
    Leer cant;
    St<- precio*cant;
    Si tipoM ='a' | tipoM='A' | cant>100 Entonces

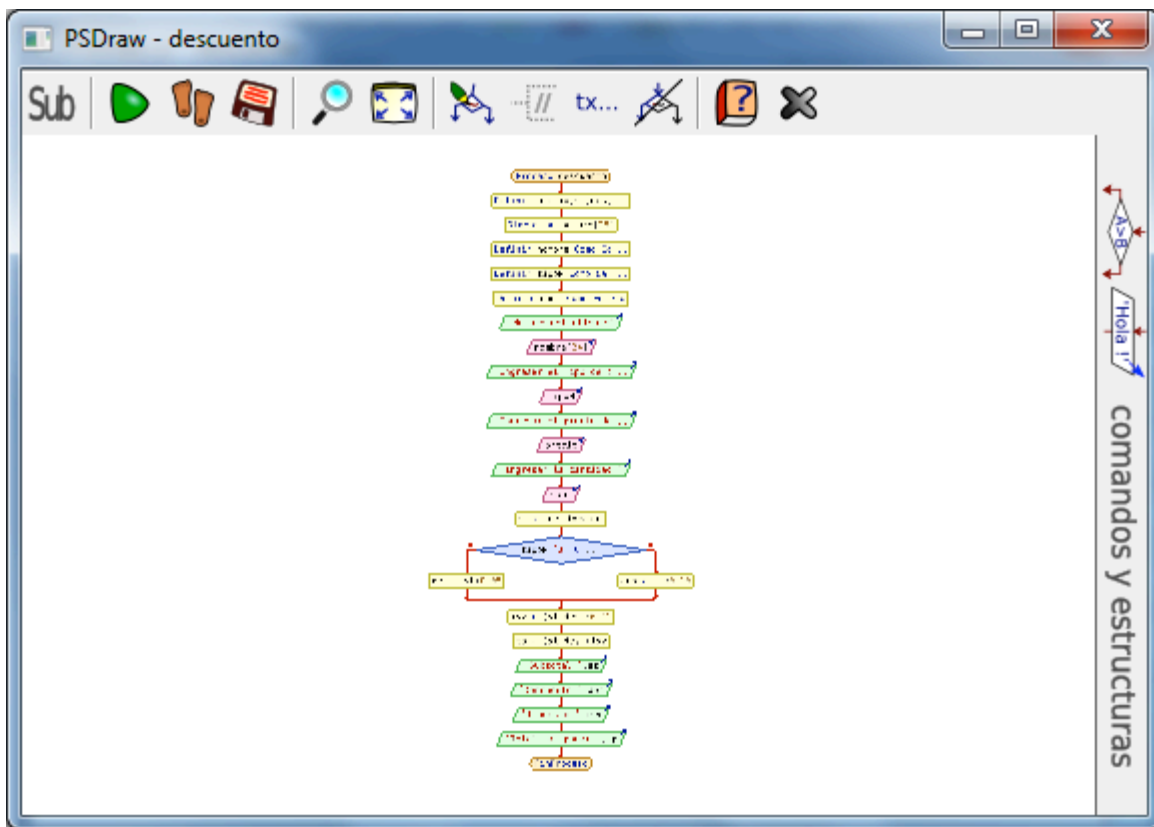
```

```

Des<-st*0.10;
Sino
Des<-st*0.05;
FinSi
Isv<-(st-des)*0.12;
Tp<-(st-des)+isv;
Escribir "Subtotal ", st;
Escribir "Descuento ", des;
Escribir "Impuesto ", isv;
Escribir "Total a pagar ",tp;
FinProceso

```

Su representación en diagrama de flujo:



Como vemos, el proceso es tan largo, que aparece con la letra muy chica, para que se vea más grande movemos el scroll para que se agrande.

Uso del operador & ó y

El operador Y (&) se utiliza cuando estamos evaluando dos o más condiciones y

queremos que la condición se cumpla cuando las dos condiciones que estamos evaluando se hacen verdadera. Ejemplo

Ejemplo sobre el operador &

Se ingresa un número y se desea saber si dicho número está entre 50 y 100.

```
Proceso ejemplo_operador_y
  Definir num Como Entero;
  Escribir "Número a evaluar";
  Leer num;
  Si num >=50 & num<=100 Entonces
    Escribir "El número está entre 50 y 100";
  Sino
    Escribir "Fuera del rango 50 y 100";
  FinSi
FinProceso
```

Exportación a un lenguaje de programación real

PSeInt puede exportar el programa el algoritmo a C, C++, C#, Java y otros lenguajes. Por ejemplo, en el caso de exportar a C, genera un archivo con la extensión .c. No es necesario guardar previamente el archivo en pseudocódigo para que se exporte a un lenguaje de programación real.

Simplemente vamos a Archivo → Exportar y seleccionamos Convertir el código a C (c).

También podemos ver la vista previa yendo a archivo → Exportar → Vista previa. Esta vista previa también puede ser guardada con el ícono del disquete.

Nota: Puede que el código generado por el interpretador no sea del todo correcto, esto se va a ir solucionando en las próximas versiones de PSeInt

Instrucciones de ciclo

Hemos hecho programas que solo se repiten una vez, pero en la programación necesitamos que los programas corran varias veces y que nos presenten información al final de correr varias veces, en estos casos usaremos ciclos, que son estructuras de repetición, que se repiten hasta cumplir con una condición o simplemente indicamos cuantas veces se van a repetir.

Nota: Al igual que las estructuras de selección, las estructuras de repetición también deben cerrarse siempre, en cualquier sintaxis, no es posible que hayan “Ciclos abiertos”.

Ciclo Mientras:

Sintaxis

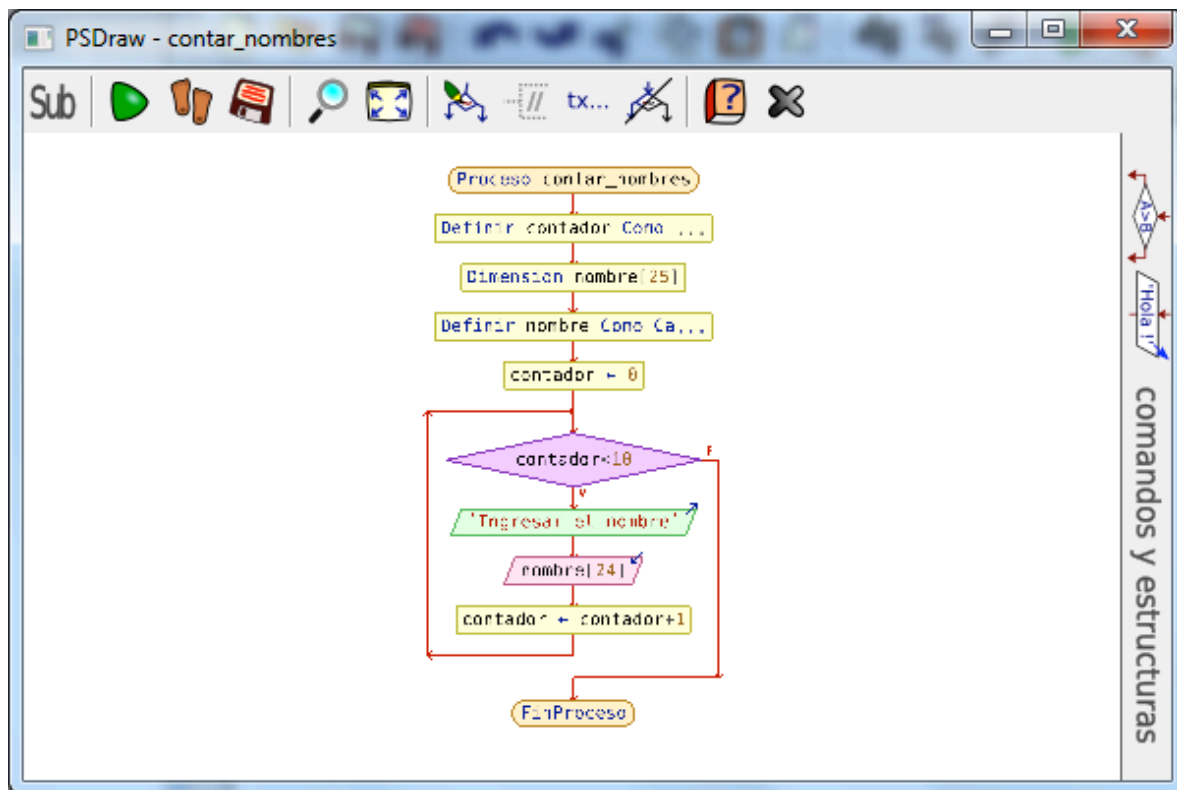
```
Mientras condición Hacer instrucciones;
FinMientras
```

El ciclo mientras se utiliza cuando se quiere ejecutar repetidamente un bloque instrucciones basado en una condición, el ciclo se repite mientras la condición se cumple.

Ejemplo sobre el ciclo Mientras usando un contador

Ingresar 10 nombres

```
Proceso contar_nombres
  Definir contador Como Entero;
  Dimension nombre[25];
  Definir nombre Como Cadena;
  Contador<-0;
  Mientras contador<10 Hacer
    Escribir "Ingresar el nombre";
    Leer nombre[24];
    contador<-contador+1;
  FinMientras
FinProceso
```



En este programa introducimos el concepto de contador, que es una variable que se incrementa su valor en 1 y de esta manera contamos cuantos nombres se van ingresando para parar cuando ingresemos 10, esto nos dice que la condición ya no se cumple porque cuando el contador vale 10 la condición de $\text{contador} < 10$ ya no se cumple porque es igual y el ciclo termina.

Ejemplo sobre el ciclo Mientras usando acumuladores

Ingresar 10 números y al final presentar la suma de los números.

```

Proceso acumuladores
  Definir Contador, Suma, Num Como Enteros;
  Contador ← 0;
  Suma ← 0;
  Mientras contador < 10 Hacer
    Escribir "Ingresar un número";
    Leer Num;
    Contador ← Contador + 1;
    Suma ← Num + Suma;
  FinMientras

```

```

Escribir "Suma de los 10 números ", Suma;
FinProceso

```

Nota: Para evitar ambigüedades, los números se deben ingresar de a uno pulsando enter sucesivamente. Ingresarlos en una fila separados por espacios provocaría un error de no coincidencia de tipos ya que se toma el espacio como un tipo de dato de ingreso más y un espacio no es un dato de tipo numérico.

El ciclo recorre 10 veces y pide los 10 números, pero la línea `suma<- suma + num`, hace que la variable suma, incremente su valor con el número que se introduce en ese momento, a diferencia del contador, un acumulador se incrementa con una variable, acumulando su valor hasta que el ciclo termine, al final se presenta la suma, solo en ese momento se debe de presentar un acumulador, porque antes no reflejaría la suma de todos los números.

Siempre que usemos un contador o acumulador debemos darle un valor inicial de generalmente será 0.

Ejemplo sobre el ciclo mientras usando una respuesta para controlar la salida del ciclo.

Ingresar el nombre del cliente, el precio del producto, cantidad y luego calcular el subtotal, isv y total a pagar, presentar los datos luego preguntar si desea continuar, al final presentar el monto global de la factura.

```

Proceso producto
  Definir Resp Como Caracter;
  Dimension nombre[25];
  Definir nombre Como Cadena;
  Definir Precio, cantidad, totalglobal, st, isv, tp Como Reales;
  Totalglobal<-0;
  Resp<-'S';
  Mientras resp <>'N' Hacer
    Escribir "Nombre del cliente";
    Leer nombre[24];
    Escribir "Ingresar la cantidad del producto ";
    Leer cantidad;
    Escribir "Ingresar el precio de producto ";

```

```

Leer precio;
St<- precio*cantidad;
Isv<-st*0.012;
Tp<-st-isv;
Totalglobal<-totalglobal+st;
Escribir "Subtotal ", st;
Escribir "Impuesto sobre venta ", isv;
Escribir "Total a pagar ", tp;
Escribir "Desea continuar S/N";
Leer Resp;
FinMientras
Escribir "Total de la venta ", totalglobal;
FinProceso

```

En este ejercicio, observamos que el ciclo lo controla una respuesta que se pide al final S para seguir o N para terminar, pero daría el mismo resultado si escribe cualquier letra distinta a S, aunque no sea N siempre seguiría funcionando el programa, la validación de los datos de entrada lo estudiaremos más adelante.

Ejemplo sobre estructuras de condición dentro del ciclo Mientras.

Ingresar el nombre del alumno, la nota examen y nota acumulada, luego calcular la nota final, y presentar la nota final y la observación del alumno.

Preguntar si desea continuar, al final presentar el número de aprobados y reprobados.

```

Proceso producto
Definir Resp Como Caracter;
Dimension nombre[25];
Definir nombre Como Cadena;
Definir Precio, cantidad, totalglobal, st, isv, tp Como Reales;
Totalglobal<-0;
Resp<-'S';
Mientras resp <>'N' Hacer
Escribir "Nombre del cliente";
Leer nombre[24];
Escribir "Ingresar la cantidad del producto ";
Leer cantidad;
Escribir "Ingresar el precio de producto ";
Leer precio;
St<- precio*cantidad;
Isv<-st*0.012;
Tp<-st-isv;
Totalglobal<-totalglobal+st;
Escribir "Subtotal ", st;
Escribir "Impuesto sobre venta ", isv;

```

```

    Escribir "Total a pagar ", tp;
    Escribir "Desea continuar S/N";
    Leer Resp;
FinMientras
Escribir "Total de la venta ", totalglobal;
FinProceso

```

Nota: Las variables no pueden declararse inicializadas o con valores, se declaran primero y se inicializan o se le asigna un valor por teclado después.

Como podemos observar en las líneas del programa, usamos dentro del ciclo mientras, decisiones para poder contar los reprobados y aprobados que resulten del ingreso de los alumnos, si la nota es mayor a 60 escribe aprobado e incrementa el contador y sino hace lo contrario, escribir reprobado e incrementar el contador.

Ciclo Para

Sintaxis

```
Para variable <- valor_inicial Hasta valor_final Con Paso Paso Hacer
    instrucciones
FinPara
```

Descripción

El ciclo Para se utiliza generalmente para ejecutar un conjunto de instrucciones que se repiten un número de veces, establecido antes de ejecutar el ciclo.

Variable: es de tipo entero

Valor_inicial: este puede ser un número entero o una variable entera.

Valor_final: este puede ser un número entero o una variable entera.

Paso: este puede ser un número entero o una variable entera.

Nota: la expresión “Con Paso 1” puede omitirse, tanto en sintaxis estricta como flexible

Ejemplo: presentar los números del 1 al 10 en la pantalla.

```
Proceso ciclo_Para
    Definir I Como Entero;
    Para I<-1 Hasta 10 Con Paso 1 Hacer
        Escribir I;
    FinPara
FinProceso
```

El programa el ciclo para establece el número de veces que se repetirá el ciclo indicando **1 hasta 10** luego la variable I toma el valor 1 a 10 según el ciclo se va ejecutando, es por eso que al escribir la I la primera vez escribe 1 la segunda vez 2 y así hasta llegar al final que es 10.

Ejemplo: sobre el uso de variables en el rango del ciclo Para.

```

Proceso ciclo_Para_2
  Definir I, final Como Enteros;
  Escribir "Ingresar el número de veces a repetir el ciclo ";
  Leer final;
  Para I<-1 Hasta final Con Paso 1 Hacer
    Escribir I;
  FinPara
FinProceso

```

Ahora el programa se vuelve más dinámico, nosotros podemos indicar el número de veces que se repetirá el ciclo, usando una variable entera para indicar el final del ciclo.

Ejemplo uso del ciclo Para, en el cálculo del factorial de un número.

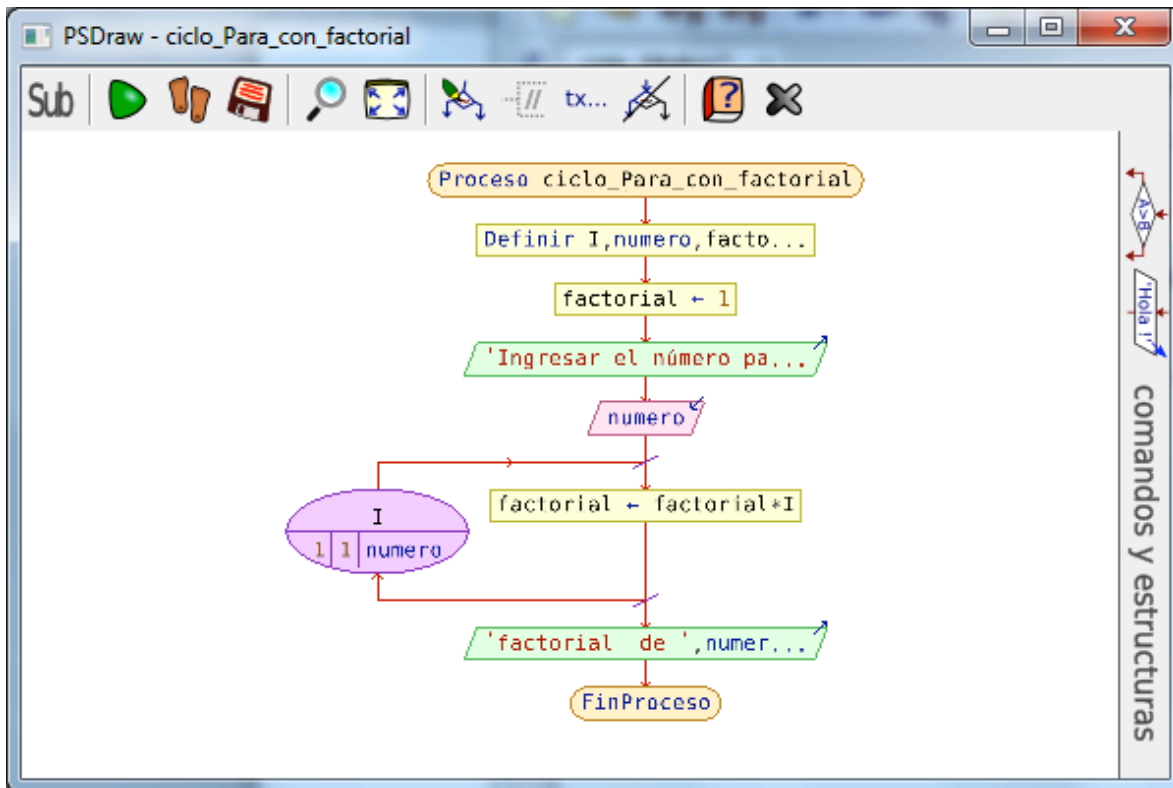
```

Proceso ciclo_Para_con_factorial
  Definir I, numero, factorial Como Enteros;
  factorial<-1;
  Escribir "Ingresar el número para determinar su factorial ";
  Leer numero;
  Para I<-1 hasta numero Con Paso 1 Hacer
    factorial<- factorial*I;
  FinPara
  Escribir "factorial de ", numero , " es ", factorial;
FinProceso

```

En este ejercicio se inicia el factorial en 1 porque acumulara una multiplicación y si lo iniciamos en cero nos daría el resultado cero, si nosotros ingresar 3, el ciclo se ejecutara 3 veces, el factorial tomaría el valor de 1x2x3.

Diagrama de flujo:



Ciclos con paso negativo

PSeInt también puede realizar ciclos inversos para mostrar, por ejemplo secuencias de mayor a menor, solamente invirtiendo el orden de los números del ejercicio anterior y colocando como Paso -1

```

Proceso ciclo_Para_negativo
  Definir I Como Entero;
  Para I<-10 Hasta 1 Con Paso -1 Hacer
    Escribir I;
  FinPara
FinProceso

```

Nota: Puede omitirse la expresión “Con Paso -1” en el ciclo Configurando las opciones del lenguaje, tildando la opción Permitir omitir el paso -1 en los ciclos Para.

Ciclos anidados

Cuando un ciclo se encuentra dentro de otro ciclo se le llama ciclo anidado.

Ejemplo de un ciclo anidado

Producir la siguiente salida en la pantalla

11111

22222

33333

44444

```
Proceso ciclo_Para_anidado
  Definir I,k Como Enteros;
  Para I <- 1 Hasta 4 Hacer
    Para K <-1 Hasta 5 Hacer
      Escribir I Sin Bajar;
    FinPara
    Escribir " ";
  FinPara
FinProceso
```

Cuando usamos ciclos anidados, las variables para manejar los ciclos para deben de ser diferentes pues cada una de ellas toma un valor diferente, en este ejercicio necesitamos que se haga 5 veces el ciclo que está dentro, que es el que presenta 4 veces el valor de la I, luego salta una línea, para que aparezcan los grupos de números en cada línea.

Ejemplo de un ciclo anidado

Ingresar 5 números y calcular el factorial para c/u de los números.

En este ejercicio necesitamos ingresar 5 números pero cada vez que ingresemos un numero debemos de calcular el factorial, entonces necesitaremos una variable para el caculo del factorial, que forzosamente tiene que inicializarse en 1 cada vez que el ciclo que calcula el factorial inicie, des esta manera estaremos seguro que la variable no ha acumulado el valor del factorial anterior.

Ahora con lo anterior deducimos que necesitamos un ciclo para pedir los datos y otro para calcular el factorial.

```
Proceso factorial
  Definir I,k,fac,num Como Enteros;
  Para I <- 1 Hasta 5 Con Paso 1 Hacer
    Escribir " ingresar un número ";
    Leer Num;
    fac<-1;
    Para k <-1 Hasta num Hacer
      fac<-fac*K;
    FinPara
    Escribir "factorial de ", num , " es ",fac;
  FinPara
FinProceso
```

Ciclo Repetir

Sintaxis:

```
Repetir
    //Instrucciones
Hasta Que condición
```

Descripción

El ciclo Repetir es lo contrario al ciclo Mientras, en éste la ejecución se lleva a cabo hasta que se cumple la condición impuesta.

La diferencia con el ciclo Mientras radica en que este evalúa la condición desde el principio, y si está no se cumple, el código que está encerrado dentro del cuerpo del mientras no se ejecuta.

En cambio, el Repetir - Mientras Que evalúa la condición para seguir ejecutándose luego de haber ejecutado el código dentro de su cuerpo, es decir siempre se ejecuta por lo menos una vez el código.

Nota: En perfil flexible, habilitando sintaxis flexible o en personalizar también es posible usar la estructura

```
Hacer
    //Instrucciones
Mientras Que condición
```

O

```
Repetir
    //Instrucciones
Mientras Que condición
```

como alternativa a **Repetir – Mientras Que** correspondiente a la sintaxis estricta. Recordar que en este caso la condición sale por el distinto, a diferencia del **Repetir** que sale por el igual.

Ejemplo del Repetir

Ingresar el nombre del alumno, la nota, luego preguntar si desea continuar, al final presentar el número de aprobados y reprobados.

```
Proceso ejemplo_Repetir
    Definir resp Como Caracter;
    Definir nota Como Real;
    Definir ca,cr Como Enteros;
    Dimension nombre[25];
    Definir nombre como Cadena;
    ca<-0;
    cr<-0;
    Repetir
        Escribir "ingresar el nombre del alumno ";
        Leer nombre[24];
        Escribir "ingresar la nota del alumno ";
        Leer nota;
        Si nota >= 60 Entonces
            ca<-ca+1;
        Sino
            cr<-cr+1;
        FinSi

        Escribir "Desea continuar S/N";
        Leer resp;
    Hasta Que resp='n' | resp='N';
    Escribir "Aprobados ",ca;
    Escribir "Reprobados ",cr;
FinProceso
```

si comparamos este programa con los hechos con el ciclo mientras notaremos que la variable Resp le damos un valor inicial de 'S', para que sea distinta de N, ya que la condición se verifica al inicio del ciclo, pero ahora con el ciclo repita ya no es necesario pues el primer valor de resp lo toma dentro del ciclo, que es la pregunta que hacemos si desea continuar, y luego verificamos la condición.

Algo importante del ciclo **Repetir** es, como ya se dijo, que se ejecuta por lo menos una vez, antes de validar la condición de salida del ciclo, es por esto, que siempre que escribamos un programa que verifique la condición antes de entrar

ciclo se debe de usar el ciclo Mientras.

El programa anterior no es la versión final, puesto que debemos hacer que el usuario solo ingrese S o N cuando responda si desea continuar, esto nos lleva a escribir un ciclo repetir dentro del ciclo repetir, para pedir la respuesta y hacer que se salga del ciclo solo cuando responda S o N, de esta manera estaremos seguros de que la respuesta es correcta.

```
Proceso ejemplo_Repetir
  Definir resp Como Caracter;
  Definir nota Como Real;
  Definir ca,cr Como Enteros;
  Dimension nombre[25];
  Definir nombre como Cadena;
  ca<-0;
  cr<-0;
  Repetir
    Escribir "Ingresar el nombre del alumno ";
    Leer nombre[24];
    Escribir "Ingresar la nota del alumno ";
    Leer nota;

    Si nota >= 60 Entonces
      ca<-ca+1;
    Sino
      cr<-cr+1;
    FinSi
  Repetir
    Escribir "Desea continuar S/N";
    Leer resp;
    Hasta Que resp='N' | resp='S'
  Hasta Que resp='N';
  Escribir "Aprobados ",ca;
  Escribir "Reprobados ",cr;
FinProceso
```

SubProcesos

Un subproceso es un subprograma o procedimiento que realiza una tarea específica y que puede ser definido mediante 0, 1 o más parámetros. Tanto en entrada de información al subproceso como la devolución de resultados desde el subproceso se realiza mediante parámetros, el cual nos sirve para introducir o modificar información del programa principal.

Sintaxis

```
SubProceso  NombreSubProceso
```

```
// ...hacer algo con los argumentos
```

```
FinSubProceso
```

Los subprocesos pueden o no tener retorno. En este caso, este subproceso no devuelve nada, los subprocesos que retornan argumentos, llamados también funciones, los veremos más adelante.

Los parámetros de un SubProceso pueden ser de distinto tipo de datos pero dichos parámetros deben ser pasados en el mismo orden en que están colocados en el subproceso.

Nota: Las variables pierden su valor cuando se llega a la palabra **FinSubProceso**, a no ser que pasen y/o entren por referencia o valor desde otro subproceso

Ejemplo: elaborar un subproceso que presente 5 asteriscos en una línea horizontal.

```
SubProceso asteriscos
  Definir I Como Entero;
  Para i <- 1 Hasta 5 Hacer
    Escribir "*" Sin Bajar;
  FinPara
FinSubProceso

Proceso Principal
  Dimension nombre[25];
  Definir nombre como Cadena;
  Escribir "Ingresar el nombre ..:";
  Leer nombre[24];
  asteriscos;
  Escribir "";
  Escribir nombre[24];
  Escribir "";
  asteriscos;
  Escribir "";
FinProceso
```

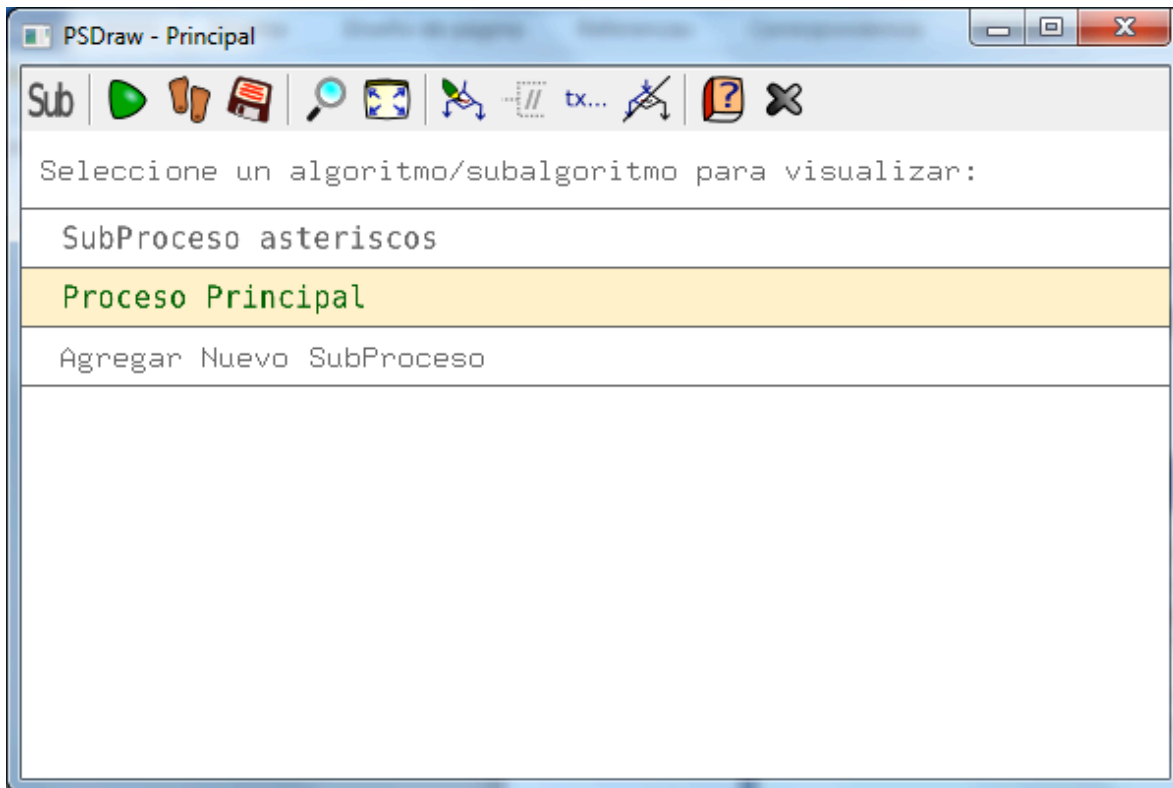
En este programa usamos un subproceso (función -palabra equivalente, PSeInt también la toma-, o procedimiento) para escribir 5 asteriscos, si no lo hubiéramos hecho de esta manera donde se encuentra la instrucción asteriscos; tendríamos que escribir el ciclo, y lo haríamos dos veces, de la forma en que lo escribimos es más estructurado, pues se divide ese proceso en un subprograma, que cuando necesitamos una línea de 5 asteriscos solo llamamos el procedimiento.

Nota: Los subprocesos sin parámetros se llaman desde el proceso principal simplemente por su nombre sin más argumentos, se pueden abrir y cerrar paréntesis, pero esto es opcional.

Ahora en el programa anterior usa un procedimiento estático, siempre escribirá 5 asteriscos, ahora lo podemos hacer dinámico usando parámetros para indicar cuantos asteriscos queremos presentar en la línea.

Visualizador de diagramas de flujo

Los subprocesos en el diagrama de flujo se muestran de la siguiente manera:



Una lista con los SubProcesos marcados con rojo:

Se elige a cuál subproceso entrar pulsando sobre el subproceso. Como dice la captura, también es posible agregar nuevos SubProcesos.

Parámetros de valor

Este tipo de parámetro se le conoce con el nombre de **parámetro de valor**, que se debe especificar si es por valor o por referencia, por defecto es por valor, este último tipo de parámetro aunque durante el procedimiento su valor cambie el valor no será asignado a la variable del programa principal, por ejemplo si la variable num del programa que presentamos abajo se le asigna otro valor diferente al 10, este cambio se reflejaría en la variable num, y por esto en el programa principal, es este tipo de parámetros que se le conoce como parámetros de valor.

Ejemplo Subproceso con valor

SubProceso asteriscos

```

Definir num, I Como Enteros;
num <- 10;
Para i <- 0 Hasta num-1 Con Paso 1 Hacer
    Escribir "*" Sin Bajar;
FinPara
Escribir "";
FinSubProceso

```

```

Proceso principal
    Dimension nombre[25];
    Definir nombre Como Cadena;
    Definir num Como Entero;
    num<-10;
    Escribir "Ingresar el nombre ...: ";
    Leer nombre[24];
    asteriscos;
    Escribir "";
    Escribir nombre[24];
    Escribir "";
    asteriscos;
FinProceso

```

En la línea **num <-10** estamos asignando al parámetro num de asteriscos el valor de 10, esto hace que el ciclo recorra 10 veces, luego más abajo del programa en la instrucción **asteriscos**; se pasó una variable como parámetro asignando el valor de num-1 a número, el cual número en el programa principal tiene un valor de 10 el cual se le asigna a numero en el paso del valor de parámetro.

Parámetros de variable

El siguiente programa, nos enseña el uso de los parámetros de variable o referencia, los cuales se le pospone las palabras reservada Por Referencia para indicar que esa variable será un parámetro de referencia o variable, esto nos indica que cualquier cambio que sufra la variable del procedimiento, la variable del programa principal también lo sufrirá, de esta manera podemos enviar información modificarla y enviar resultados al programa principal.

La sintaxis es la siguiente:

Ejemplo parámetros de variable o referencia.

Elaborar un programa donde se ingrese el nombre y el apellido usando un procedimiento y luego presentar los datos.

```
SubProceso Pedir_datos (nombre Por Referencia, apellido Por Valor)
  Escribir "Ingresar el nombre ";
  Leer nombre;
  Escribir "Ingresar el apellido ";
  Leer apellido;
FinSubProceso

Proceso Principal
  Definir nombre, apellido Como Cadenas;
  nombre<-"No hay cambios en nombre";
  apellido<-"No hay cambios en apellido";
  Pedir_datos(nombre,apellido);
  Escribir "Nombre completo ", nombre," ", apellido;
FinProceso
```

Nota: En caso de que la variable se deba pasar por referencia siempre se debe indicar. En cambio, si se pasa por valor, la indicación de pase puede omitirse. Siempre por defecto se pasa por valor.

En el programa anterior, se inician las variables de apellido y nombre, luego se pasan como parámetros al subproceso, el nombre por referencia y el apellido por valor, luego escribimos los valores y solo el nombre presentara el cambio que sufrió en el subproceso y el apellido seguirá escribiendo el mismo valor que tenía al empezar el programa esto porque no se pasó como parámetro de variable (por referencia) sino como de valor y no se le permitió sufrir alguna modificación.

Para mejorar el programa anterior el procedimiento tendría que escribirse así:

```
SubProceso pedir_datos (nombre por Referencia, apellido Por Referencia)
  Escribir "Ingresar el nombre";
  Leer nombre;
  Escribir "Ingresar el apellido";
  Leer apellido;
FinSubProceso
```

Ejemplo

Ingresar la base y el exponente y luego calcular la potencia.

En este programa usaremos un subproceso para el ingreso de los datos y otro para calcular la potencia.

```
SubProceso Ingreso (base Por Referencia, expo Por Referencia)
  Escribir "Ingresar la base ";
  Leer base;
  Escribir "Ingresar el exponente ";
  Leer expo;
FinSubProceso

SubProceso pot <- Potencia (base, expo, pot Por Referencia)
  Definir I Como Entero;
  pot<-1;
  Para I <- 1 Hasta expo Con Paso 1 Hacer
    pot<-pot*base;
  FinPara
FinSubProceso

Proceso principal
  Definir base,expo,pot como Enteros;
  Ingreso(base,expo);
  Escribir "Potencia es ",Potencia(base,expo,pot);
FinProceso
```

En el subproceso de ingreso los dos datos, exponente y base son de tipo entero y parámetros de variable, esto porque necesitamos que el procedimiento nos devuelva los valores para luego introducirlos en el procedimiento potencia pero aquí, base, expo son de tipo valor, esto porque no necesitamos modificar o leer su valor como anteriormente los hicimos en el procedimiento de ingreso , luego la variable pot si se pasa como parámetro de variable debido a que necesitamos modificar su valor y presentarlo en el programa principal.

Nota: Los subprocesos y/o funciones no se pueden llamar igual que las variables que se declaran en el programa.

SubProcesos que devuelven valor o con retorno (Funciones)

Las SubProcesos también pueden devolver un valor, pero solo uno. Este tipo de SubProceso recibe el nombre de función.

Sintaxis

Sintaxis

```
SubProceso valor_de_retorno <- nombre_SubProceso [( parámetros ) ]
    //[variables locales]

    //instrucciones
```

FinSubProceso

Si notamos en la sintaxis de la función observamos que hay dos variables entre una flecha que apunta a la izquierda, esta está apuntado a la variable "retorno" la cual devuelve un valor.

Nota: En sintaxis estricta también se puede usar indistintamente la palabra *funcion* en lugar de *subproceso*. En PSeInt, son términos equivalentes.

Ejemplo: cálculo de la potencia

Usaremos el mismo ejercicio que usamos para los subprocesos, para hacer una demostración de cómo cambiaría el programa usando un subproceso sin retorno para el cálculo de la potencia.

```
SubProceso resp <- potencia (base, expo Por Referencia)
    Definir I, resp Como Enteros;
    resp<-1;
    Para I <- 1 Hasta expo Con Paso 1 Hacer
        resp<-resp*base;
    FinPara
FinSubProceso

SubProceso Ingreso (base Por Referencia, expo Por Referencia)
    Escribir "Ingresar la base ";
    Leer base;
    Escribir "Ingresar el exponente ";
    Leer expo;
FinSubProceso
```

```

Proceso principal
  Definir base, expo, pot Como Enteros;
  Ingreso(base,expo);
  pot<-Potencia(base,expo);
  Escribir "Potencia es ", pot;
FinProceso

```

Nota: Como se ve en el ejemplo anterior, cuando se llama a funciones, además de anteponer también la palabra reservada *Escribir*, cuando se coloca el nombre de la función a llamar los argumentos de dicha función deben estar pegados al nombre de la función. De lo contrario aparecerá un cartel que dice “Los argumentos para invocar a un subproceso deben ir entre paréntesis”

Ahora veremos cómo dibuja el diagrama de flujo el intérprete de diagramas de flujo:

Diagrama de flujo del procedimiento ingreso:

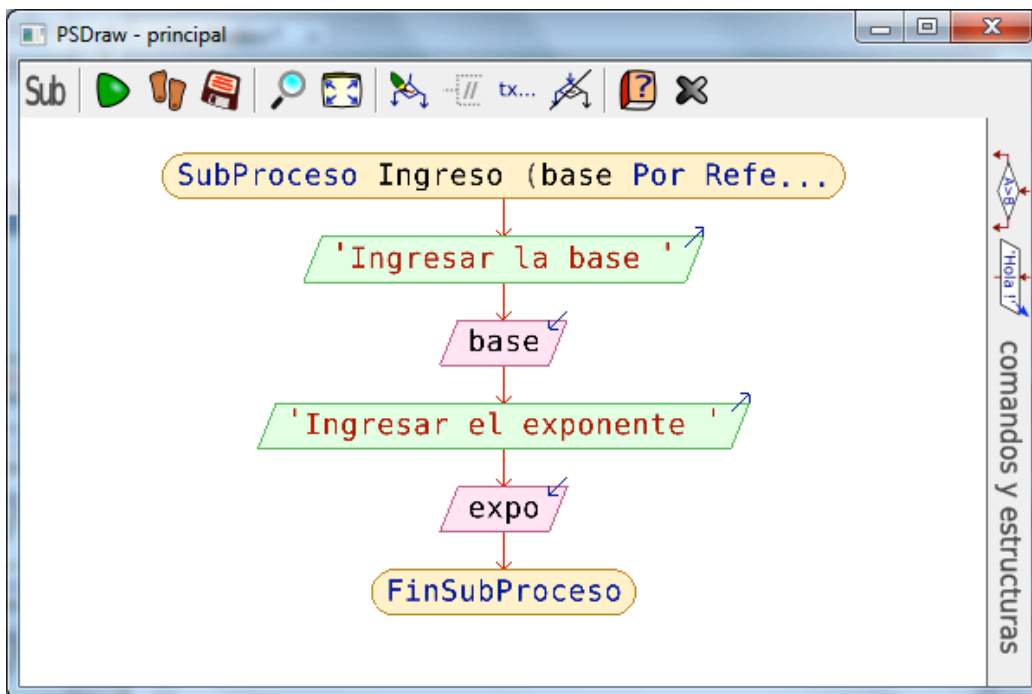
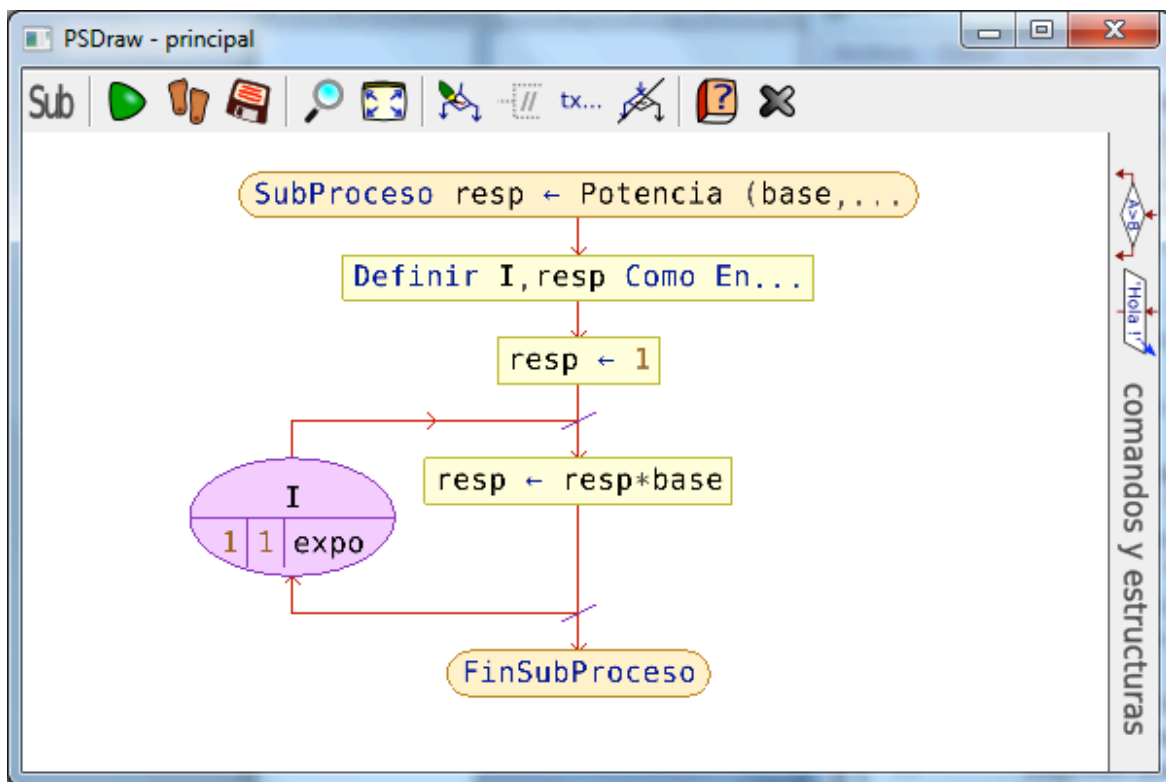


Diagrama de flujo del procedimiento potencia:



Si miramos este diagrama de flujo o el pseudocódigo, observamos que en la función **Potencia** se declaran una variable **I** que es para el ciclo y la otra **resp** que es para el cálculo de la potencia, la cual usaremos como acumulador de la multiplicación de la potencia, y después de la variable **resp**, a su vez después de la palabra clave **subproceso**, que es lo que nos devuelve el valor, y lo asigna en a la variable **pot** en el programa principal, cuando usamos la instrucción **pot<-potencia(base Por Referencia, expo Por Referencia);**.

En conclusión, las funciones siempre nos retornaran un valor que es producto de uno o más cálculos, y se devuelve el valor de la variable que pusimos después de la palabra clave **SubProceso**.

Ejemplo de planilla (SubProcesos con y sin retorno)

Se ingresan el nombre, las ventas y la zona del empleado usando un

procedimiento, luego se calcula la comisión en base a la zona de trabajo, ihss y total a pagar, luego se presentan los datos.

Nota:

- *se deberá de usar un subproceso con retorno para los cálculos y la presentación de los datos.*
- *Usar un subproceso con retorno para el cálculo del ihss.*
- *Usar un subproceso con retorno para el cálculo de la comisión.*

Subproceso de ingreso

En este subproceso sin retorno se ingresan los datos, validando que la zona solo sea A,B,C

Subproceso de cálculo

Se calcula la comisión e ihss usando los subprocesos sin retorno declarados anteriormente, luego el total a pagar, algo que debemos de notar es que las ventas y la zona se pasan como parámetros de valor y las demás ihss, comis y tp como parámetros de variable porque necesitamos modificar su valor

SubProceso presentar

Presentamos los cálculos y pasamos las variables como parámetros de valor, porque solo las necesitamos presentar

```
SubProceso vihss <- seguro(comis)
  Definir Vihss Como Real;
  Si comis >2400 Entonces
    vihss<-84;
```



```

    Sino
        vihss<-0.035*comis;
    FinSi
FinSubProceso

SubProceso vcomis <- comision(zona,ventas)
    Definir vcomis como Real;
    Segun zona Hacer
        'A' : vcomis<-0.05*ventas;
        'B' : vcomis<-0.06*ventas;
        'C' : vcomis<-0.09*ventas;
    FinSegun
FinSubProceso

SubProceso ingreso (nombre Por Referencia, zona Por Referencia, ventas
Por Referencia)
    Escribir "Ingresar el nombre ";
    Leer nombre;
    Escribir "Ventas mensuales ";
    Leer ventas;
    Repetir
        Escribir "Zona A,B,C ";
        Leer zona;
    Hasta Que zona ='B' | zona ='C' | zona ='A'
FinSubProceso

SubProceso calculos (zona, ventas, comis Por Referencia, ihss Por
Referencia, tp Por Referencia)
    comis<-comision(zona,ventas);
    ihss<-seguro(comis);
    tp<-comis-ihss;
FinSubProceso

Subproceso presentar (comis,ihss,tp)
    Escribir "Comisión ",comis;
    Escribir "Seguro Social ", ihss;
    Escribir "Total a pagar ", tp;
FinSubProceso

Proceso principal
    Definir ventas,comis,ihss,tp Como Reales;
    Definir nombre Como Cadena;
    Definir zona Como Caracter;
    Ingreso(nombre,zona,ventas);
    Calculos(zona,ventas,comis,ihss,tp);
    Presentar(comis,ihss,tp);
FinProceso

```

En este caso los subprocesos con retorno los declaremos antes de los subprocesos sin retorno solo porque estas se usarán en el subproceso sin retorno cálculos, y es más legible al momento de leer un programa, pero, a los efectos de la ejecución, PSeInt, no tiene en cuenta el orden del proceso y de los subprocesos.

Nota: En sintaxis estricta, la variable de retorno debe ser declarada

Dimensiones

Es una colección de datos del mismo tipo, que se almacenan en posiciones consecutivas de memoria y reciben un nombre común.

Y para referirse a un determinado elemento tendremos de acceder usando un índice para especificar la posición que queremos extraer o modificar su valor. Las dimensiones pueden ser:

1-Unidimensionales: solo tiene una sola dimensión una fila y una columna

2-Bidimensionales: tablas o matrices.

3-Multidimensionales: de 3 o más dimensiones.

Dimension de I Capacidad (Unidimensionales)

Declaración:

Dimension <Nombre de la dimension> [**<capacidad>**];

Definir <Nombre de la variable de la dimension> **Como** <tipo de la variable>;

Capacidad: es el tamaño de la dimension, es un número entero con el cual indicamos el número de elementos que queremos guardar con el mismo tipo.

Nombre de la variable: es el nombre con el cual vamos a ser referencia en el programa principal

Tipo de datos: es el tipo de datos que queremos que sea la colección, puede ser entero, real, cadena, carácter o una estructura.

Nota: *En sintaxis estricta, se debe definir siempre la variable antes o después de dimensionarla. A diferencia de otros lenguajes de programación, dimensionar una variable no implica declararla.*

Ejemplo:

```
Dimension numero [9];
```

Con esta declaración estamos creando una colección de 10 números enteros

3	5	7	8	3	6	9	2	45	67
0	1	2	3	4	5	6	7	8	9

Nota: *Al igual que en los lenguajes de programación reales, en sintaxis estricta, la base de la dimensión es 0, pero en otras sintaxis, entre ellas sintaxis flexible o perfil flexible, es base 1. Para utilizar dimensiones variables debemos habilitar la opción, permitir utilizar variables para dimensionar arreglos en las opciones de personalización.*

Siempre que nosotros queremos hacer referencia a uno de los elementos de la dimension, tendremos que indicar la posición, con un número entero que este dentro del rango.

Seguidamente definimos el tipo de dimensión:

```
Definir numero Como Entero;
```

Si que queremos escribir el valor de posición 7 tendremos que escribir:

```
Escribir numero[7]; // de esta indicamos escribir la posición 7
```

0

```
I<- 7 //asignamos un valor a una variable de tipo entero
```

```
Escribir numero[ 0 ]; // luego usamos la variable I para indicar la posición  
que queremos presentar.
```

```
// Si deseamos asignar valores a un elemento de la dimension  
lo podremos  
// hacer:
```

```
Leer numero[2]; // indicamos directamente la posición que  
queremos leer
```

```
I<-6 // Asignamos un valor a una variable entero y luego la  
usamos
```

```
Leer numero[ i ]; // para indicar la lectura de elemento que  
queremos leer
```

Ejemplo

Ingresar 10 números a una dimension de 10 elementos y luego presentar los números.

En este programa tendremos que usar un ciclo que la variable I tome un valor de 0..9, para leer los valores o asignar valores a la dimension, luego usaremos otro ciclo para presentar los datos.

Cuando guardamos los datos en una dimension, sus valores son almacenados en la memoria y no se borran después al leer el siguiente número, como en los programas anteriores, cuando usábamos una variable para ingresar 10 números, pero la variable al final del ingreso solo guardaba el último número que se introdujo, ahora con los arreglos se guardan los 10 números en la memoria.

Nota: Si PSeInt está configurado para trabajar en base 0, se define una dimensión, y por ejemplo, se recorre una dimensión con un Para que comience en 1 y finalice con el número de elementos que se declaró a la dimensión, por ejemplo para llenar el vector de números, el último elemento ingresado no va a tener posición de memoria en la dimension ingresada. Esto lo podemos cambiar definiendo el Para desde base 0 y el número de la dimensión-1, personalizando el perfil para que use arreglos o dimensiones en base 1 o en su defecto utilizando sintaxis flexible.

```
// programa de ingreso de 10 números a una dimensión
Proceso dimension_10
    Dimension numero[10];
    Definir numero Como Entero;
    Definir I Como Entero;

    Para I <- 0 Hasta 9 Con Paso 1 Hacer
        Escribir "Ingrese el número de la pos# ", I , "....:";
        Leer numero[I];
    FinPara

    Para I <- 0 Hasta 9 Con Paso 1 Hacer
        Escribir numero[I];
    FinPara
FinProceso
```

Ejemplo

Ingresar el nombre del empleado en una dimension y el sueldo en otra

dimension, luego de ingresar los datos determinar el ihss, el total a pagar para cada uno de los empleados.

En este programa se almacena el nombre del empleado y el sueldo en dos arreglos diferentes el nombre en un arreglo de cadena y el sueldo en una dimension de tipo real, primero se ingresa los datos en la dimension luego se calculan los datos en otro ciclo con el fin de enfatizar que los arreglos guardan los datos en la memoria durante el programa funciona y los podemos usar después de ingresados los datos, lo que antes no podíamos hacer pues al ingresar el elemento 10 en la variable solo ese podíamos guardar, es por ello que los cálculos se hacían en el mismo ciclo.

```
Proceso dimension_empleado
  Dimension nombre[5];
  Definir nombre Como Cadena;
  Dimension sueldo[5];
  Definir sueldo como Entero;
  Definir ihss,tp Como Reales;
  Definir I Como Entero;

  Para I <- 0 Hasta 4 Hacer
    Escribir "Nombre del empleado [",i+1,"]...:";
    Leer nombre[i];
    Escribir "Sueldo del empleado ...:";
    Leer sueldo[i];
  FinPara
  Para I <- 0 Hasta 4 Con Paso 1 Hacer
    Si sueldo[i]>2400 Entonces
      ihss<-84;
    Sino
      ihss<-0.035*sueldo[i];
    FinSi
    tp<-sueldo[i]-ihss;
    Escribir "Nombre ...:", nombre[i];
    Escribir "Sueldo ...:",sueldo[i];
    Escribir "Ihss ...:",ihss;
    Escribir "Total pagar...:",tp;
  FinPara
FinProceso
```

Dimensiones dimensionadas dinámicamente

Nota: Para poder utilizar dimensiones de dimensión variable (es decir dimensionadas con variables) es necesario habilitar Permitir utilizar variables para

dimensionar arreglos o en su defecto utilizar perfil flexible.

Puede ocurrir que el usuario del programa necesite poder dimensionar una dimensión o arreglo en tiempo de ejecución, es decir, poder leer una variable y usarla para “topear” una dimensión.

Para ejemplificar esto, tomemos el siguiente ejemplo, extraído del sitio web oficial de PSeInt (<http://pseint.sourceforge.net/index.php?page=perfiles.php>)

El código modificado en perfil estricto quedaría así:

```
Proceso ejemplo_arreglo_dimensionado_por_usuario
  Definir VAL_MAYOR, ARREGLO, CANT, I Como Enteros;
  Escribir "Ingrese la cantidad de números:";
  Leer CANT;
  Dimension ARREGLO[CANT];
  Para I<-0 Hasta CANT-1 Hacer
    Escribir "Ingrese un numero:";
    Leer ARREGLO[I];
  FinPara
  VAL_MAYOR<-0;
  Para I<-0 Hasta CANT-1 Hacer
    Si ARREGLO[I]>VAL_MAYOR Entonces
      VAL_MAYOR<-ARREGLO[I];
    FinSi
  FinPara
  Si VAL_MAYOR % 2 = 0 Entonces
    Escribir "El mayor es ", VAL_MAYOR, " y es par";
  Sino
    Escribir "El mayor es ", VAL_MAYOR, " y es impar";
  FinSi
FinProceso
```

Se lee la variable CANT y el valor leído pasa a ser el tope de la Dimension. Ejemplo: Si el usuario ingresa un 5, la dimensión pasa a ser de 5 posiciones (de 0 hasta 4), en el caso de estar configurado en base 0.

Seguidamente, se recorre el arreglo de dimension establecida por el usuario para que el usuario lo rellene con los elementos que considere.

En el segundo Para se calcula el elemento ingresado que es el mayor del que existe en el arreglo.

En el Si, se evalúa si dicho elemento que se determinó como mayor es par o impar preguntando si el resto da 0. En caso de ser par, se lo muestra diciendo que es par y de lo contrario también mostrará el número pero dirá impar

Nota: Los arreglos de dimensión variable se especifican después de la instrucción Leer, como aparece en el ejemplo. Sin embargo, la definición de la variable puede hacerse tanto antes de definir la dimensión como después.

Uso de arreglos o dimensiones como parámetros en los subprocesos y funciones

En el ejemplo que se presenta se demuestra el uso de los arreglos o dimensiones en los subprocesos y parámetros, el ejemplo muestra un subproceso sin retorno para el ingreso de datos a una dimension de 5 números enteros, luego se usa una función de mayor que nos devuelve el número de la dimension.

```
SubProceso nummayor <- mayor (num)
  Definir nummayor, i Como Enteros;
  nummayor<-0;
  Para i <-0 Hasta 4 Con Paso 1 Hacer
    Si num[i]>nummayor Entonces
      nummayor<-num[i];
    FinSi
  FinPara
FinSubProceso

SubProceso ingreso (num)
  Definir i como Entero;
  Para i <- 0 Hasta 4 Con Paso 1 Hacer
    Escribir "Ingresar un número ";
    Leer num[i];
  FinPara
FinSubProceso

Proceso Principal
  Dimension num[5];
  Definir num, max Como Enteros;
  Ingreso(num);
  Max<-mayor(num);
  Escribir "Mayor ", max;
FinProceso
```


Nota: Por defecto, los arreglos siempre se pasan por referencia. Si intentamos pasarlo por valor provocaremos un error.

Función mayor

En esta función se determina el número mayor comparando los números que se ingresan, cuando se inicia la función nummayor vale cero pero supongamos que ingresamos en el arreglos 3-5-4-2-8

Cuando el elemento uno de la dimension se compara con 3, hay una nueva asignación para nummayor que es 3, cuando se compara con 5 el 3 es menor al 5 hay una nueva asignación a nummayor es 5, cuando se compara con 4 el 5 no es menor al cuatro, así que nummayor no se asigna ningún valor y se queda con el 5 ahora cuando se compara con 8 nummayor se le asigna el 8 porque el 5 es menor a 8.

Num	Nummayor
cuando num[0] es 3	Entonces vale 3
cuando num[0] es 5	Entonces vale 5
cuando num[0] es 4	No hay cambio y sigue valiendo 5
cuando num[0] es 2	No hay cambio y sigue valiendo 5
cuando num[0] es 8	Entonces vale 8

Dimension de II capacidad (de doble capacidad o bidimensionales)

Declaración:

Dimension <Nombre de la variable> [<Líneas>, <Columnas>];

También se les denomina matrices o tablas. Una dimension bidimensional es una tabla que ahora tiene líneas y columnas, donde las líneas indican la primera dimensión y las columnas la segunda dimensión.

	0	1	2	3
0				
1				
2				
3				
4				

La tabla que se muestra nos representa un dimension de 2 dimensiones con 5 líneas (4 posiciones) y 4 columnas (3 posiciones), el código para declarar este dimension sería:

```
Dimension numero[4,3];
```

La referencia a un determinado elemento de la matriz requiere el empleo de un primero subíndice que indica la fila y el segundo que indica la columna. Ambos subíndices deberán de ser de tipo entero.

Por ejemplo, si quisiéramos guardar el valor de 30 en la línea 4 columna 3 el código en PSeInt sería:

```
Numero[3,2]<-30;
```

El siguiente ejemplo nos muestra como ingresar datos a una dimension de 5 líneas y 4 columnas para luego presentar los datos en la pantalla:

```
Proceso dimension_5_lineas
    Dimension numero[5,4];
    Definir numero Como Entero;
    Definir L, C Como Enteros;

    Para L <- 0 Hasta 4 Con Paso 1 Hacer
        Para C <- 0 Hasta 3 Con Paso 1 Hacer
            Escribir "Numero[" , L , " , " , C , " ] ";
            Leer numero[L,C];
        FinPara
    FinPara

    Limpiar pantalla;

    Para L <- 0 Hasta 4 con Paso 1 Hacer
        Para C <- 0 Hasta 3 Con Paso 1 Hacer
            Escribir numero[L,C] , " " Sin Bajar;
        FinPara
        Escribir "";
    FinPara
FinProceso
```

En este programa usamos dos variables enteras L que se usa para las líneas y C que se usa para las columnas, usamos ciclos anidados porque necesitas recorrer por cada línea, todas las columnas, esto sucede así:

Cuando la L tiene el valor de 0 la C toma el valor de 0 a 3 esto hace que se puede leer el elemento Numero [0,1], Numero [0,2], Numero [0,3], luego cuando la L tiene el valor de 2 entonces la L vuelve a iniciar de 0 a 3 haciendo lo mismo 4 veces que es el número de las líneas.

Suma de líneas y columnas de un dimension bidimensional

El programa que se presenta ingresa los datos y los presenta usando un subproceso sin retorno.

```

SubProceso sum <- SumaLinea (numero, linea)
  Definir sum, C Como Enteros;
  sum<-0;
  Si linea>=0 | linea<=4 Entonces
    Para C<-0 Hasta 3 Con Paso 1 Hacer
      sum<-sum + numero [linea,C];
    FinPara
  FinSi
FinSubProceso

SubProceso sum <- SumaColumna (numero, col)
  Definir sum, L Como Entero;
  sum<-0;
  Si col>=0 | col<=3 Entonces
    Para L <- 0 Hasta 4 Con Paso 1 Hacer
      sum<-sum + numero [L,col];
    FinPara
  FinSi
FinSubProceso

SubProceso ingreso(numero)
  Definir L,C Como Enteros;
  Para L <- 0 Hasta 4 Con Paso 1 Hacer
    Para C <- 0 Hasta 3 Con Paso 1 Hacer
      Escribir "Ingresar un número .:.";
      Leer numero[L,C];
    FinPara
  FinPara
  Escribir "";
FinSubProceso

SubProceso presentar (numero)
  Definir L, C Como Enteros;
  Limpiar pantalla;
  Para L <- 0 Hasta 4 Con Paso 1 Hacer
    Para C <- 0 Hasta 3 Con Paso 1 Hacer
      Escribir numero[L,C], " " Sin Bajar;
    FinPara
  FinPara
  Escribir "";
FinSubProceso

Proceso principal
  Dimension numero[5,4];
  Definir numero Como Entero;
  Definir linea,col,sumaC,sumaL Como Enteros;
  Ingreso(numero);
  Presentar(numero);
  Escribir "Línea a sumar";
  Leer linea;
  Escribir "Columna a sumar";

```

```

Leer col;
sumaL<-sumaLinea(numero,linea);
sumaC<-sumaColumna(numero,col);
Escribir "Suma de la columna ", col, " es ", sumaC;
Escribir "Suma de la línea ", linea, " es ", sumaL;
FinProceso

```

Nota 1: Los arreglos que son parte de los parámetros de los subprocesos y/o funciones no se declaran dentro de dichos subprocesos, porque siempre son por referencia. De hacerse, aparecerá un cartel que dice No se debe redefinir el tipo de argumento.

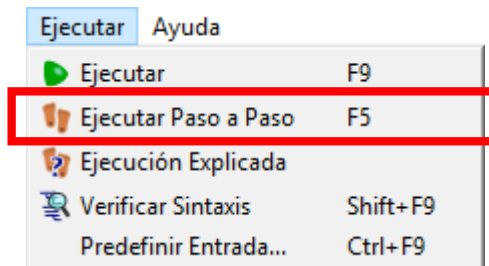
Nota 2: Los arreglos o dimensiones que son parámetros de SubProcesos/Funciones, tanto cuando se pasan a otro subproceso, como cuando se hace referencia al subproceso desde el proceso principal no se especifica su capacidad.

Ejecución paso a paso

A la hora de ejecutar algoritmos, puede pasar que necesitemos saber exactamente a que subproceso entra nuestro programa o en que condición entra a los bucles. Esto puede ser importante, cuando nuestro programa presenta errores lógicos y nos cuesta detectar dónde está el error.

Para este y otros casos existe lo que Novara denomina Ejecución paso a paso, que consiste en ejecutar el programa pero que PSeInt marque exactamente qué línea se está ejecutando, para saber si el programa está entrado donde debe.

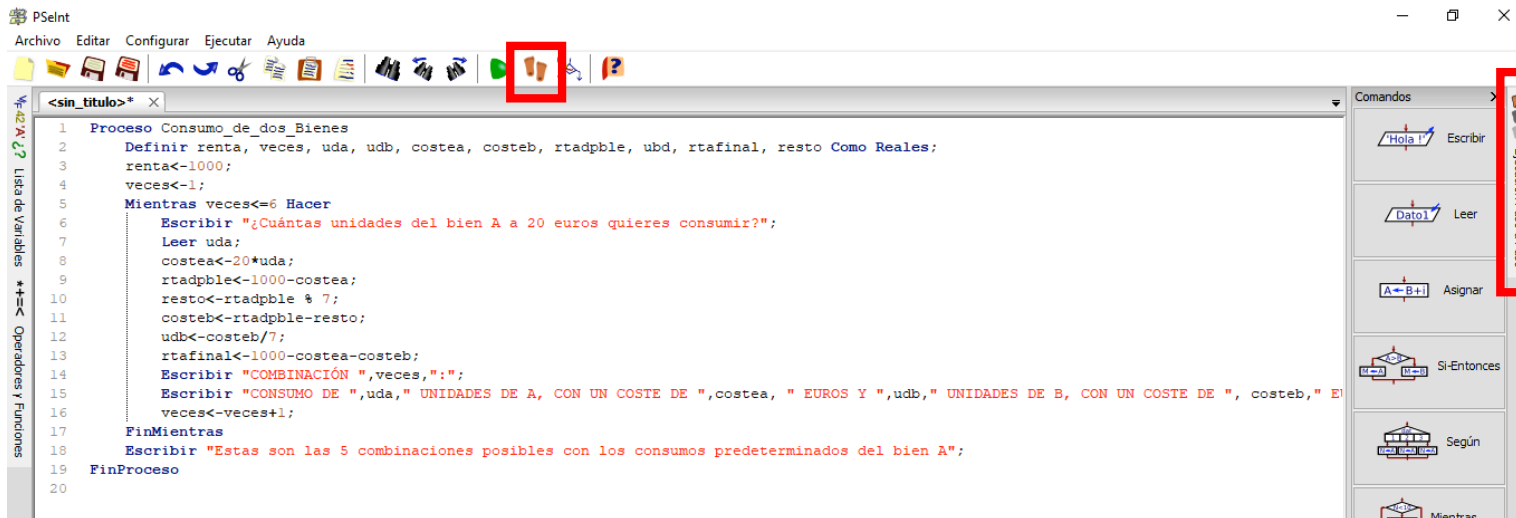
Para realizar una ejecución paso a paso vamos a Ejecutar → Ejecutar Paso a Paso o en su defecto Pulsar F5.



Nota: Para iniciar la ejecución paso a paso también podemos bien hacer clic en el icono de ejecución paso a paso que aparece en la barra de herramientas, arriba a la derecha, representado por dos pies.

Otra posibilidad es hacer clic en la pestaña que aparece en la parte lateral derecha de la ventana de PSeInt, junto a los botones de los comandos.

La imagen siguiente muestra algunas de estas posibilidades:



Para explicar con claridad cómo funciona la Ejecución paso a paso, vamos a utilizar un programa de ejemplo:

```

Proceso Consumo_de_dos_Bienes
  Definir renta, veces, uda, udb, costea, costeb, rtadpble, udb, rtafinal, resto Como Reales;
  renta<-1000;
  veces<-1;
  Mientras veces<=6 Hacer
    Escribir "¿Cuántas unidades del bien A a 20 euros quieres consumir?";
    Leer uda;
    costea<-20*uda;
    rtadpble<-1000-costea;
    resto<-rtadpble % 7;
    costeb<-rtadpble-resto;
    udb<-costeb/7;
    rtafinal<-1000-costea-costeb;
    Escribir "COMBINACIÓN ",veces,":";
    Escribir "CONSUMO DE ",uda," UNIDADES DE A, CON UN COSTE DE ",costea, " EUROS Y ",udb," UNIDADES DE B, CON UN COSTE DE ", costeb," EUROS. TE SOBRAN ",rtafinal," EUROS";
    veces<-veces+1;
  FinMientras
  Escribir "Estas son las 5 combinaciones posibles con los consumos predeterminados del bien A";
FinProceso
  
```

Con la ejecución paso a paso comprobaremos cómo el depurador nos indica por dónde va el flujo del programa en cada momento, circulando a través de sus bucles de repetición según las condiciones impuestas por el programador.

En una ventana se muestra el programa, la secuencia de órdenes y el flujo, dirección y orden de ejecución, en otra se va ejecutando propiamente el programa, mostrando los valores y resultados buscados, y a la derecha tenemos el panel "paso a paso" donde podemos elegir varias opciones que te ofrece el depurador. Así, si marcamos la opción que nos muestra el detalle, abajo se nos abre otra ventana donde se explica qué está haciendo el programa en cada momento, donde guarda valores, en qué variables, cuáles son los valores que guarda, etc. Vamos a verlo de forma más detallada.

Inmediatamente después de iniciada la ejecución paso a paso, el algoritmo comenzará a ejecutarse paso a paso, mostrando la ventana de ejecución del programa y marcando sobre el código fuente, en la ventana principal de PSeInt, qué instrucción se está ejecutando en cada momento.

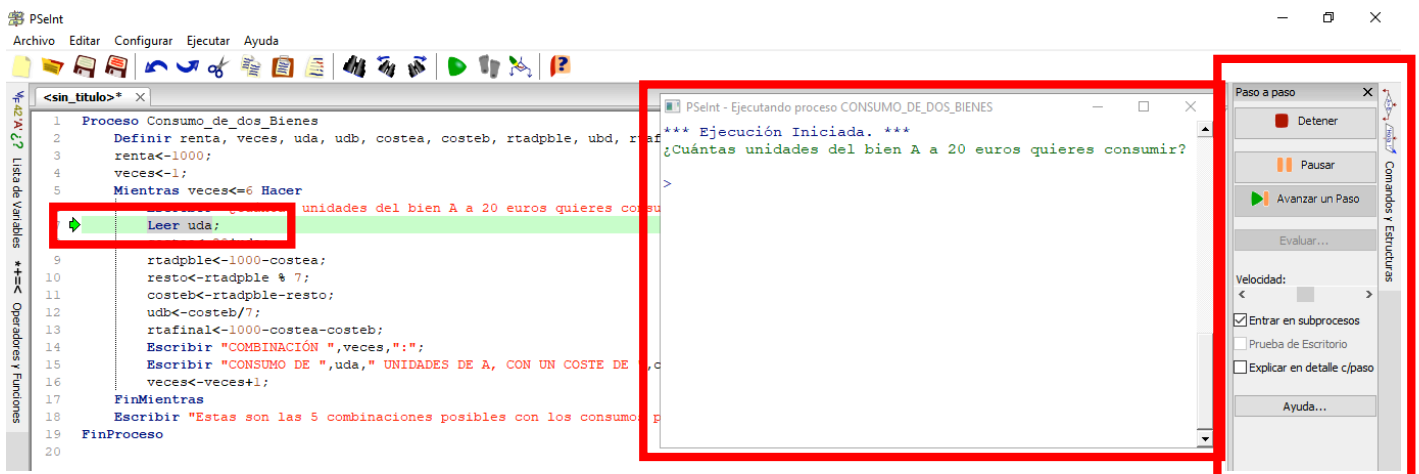
También verás en la parte derecha de la ventana de PSeInt un panel lateral con distintas opciones que podrás utilizar para el proceso de ejecución paso a paso. Entre esas opciones encontrarás:

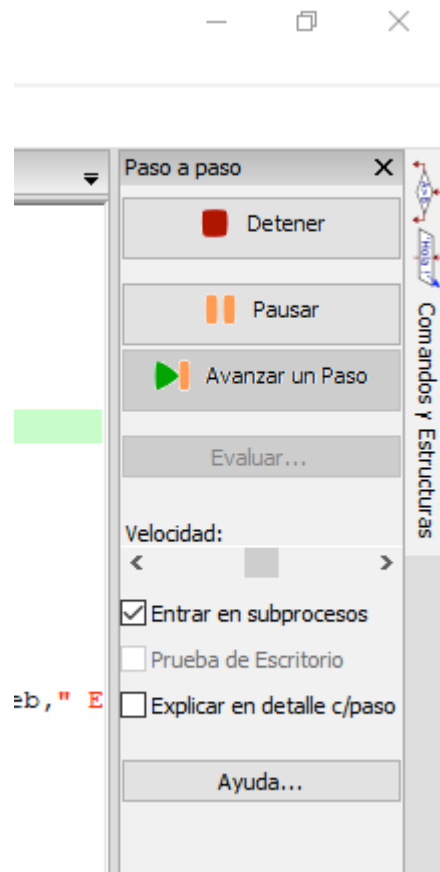
- Un primer botón para detener la ejecución paso a paso por completo.
- Un botón que será utilizado para hacer pausas durante la ejecución paso a paso.
- Un botón para poder avanzar instrucción a instrucción.
- Otro botón que permitirá evaluar expresiones cuando se haga una pausa en la ejecución paso a paso. Este botón estará habilitado solo cuando la ejecución esté pausada.
- Una barra horizontal que marcará la velocidad a la que se irá ejecutando cada paso. Desplazando esta barra hacia la derecha configurarás PSeInt para que se haga un tiempo de retardo menor entre cada instrucción ejecutada. Por

el contrario, desplazándola hacia la izquierda PSeInt tardará más tiempo en pasar de una instrucción a otra.

- Una opción que configurará PSeInt para que la ejecución paso a paso sea también a nivel de subprocesos (funciones) o solo del programa principal. Es decir, qué si está marcada esta opción, los subprocesos también se ejecutarán paso a paso. En caso contrario, los subprocesos se ejecutarán sin hacer paradas o pausas.
- Otra opción llamada Prueba de Escritorio que mostrará una nueva ventana en la parte inferior de la pantalla en la que se pueden agregar variables o expresiones para seguir los valores que van tomando durante la ejecución.
- También aparecerá una opción que permitirá mostrar de forma detallada cada paso durante la ejecución.
- Por último, aparece un botón para acceder a la ayuda de PSeInt.

Inmediatamente después, el algoritmo comenzará a ejecutarse paso a paso, mostrando la ventana de ejecución del programa y marcando sobre el código fuente, en la ventana principal de PSeInt, qué instrucción se está ejecutando en cada momento.





Funciones predefinidas

Además de las funciones que puede definir el programador, PSeInt también dispone de funciones ya predefinidas por el programa.

Ellas son:

<i>Función</i>	<i>Significado</i>
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 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.
CONVERTIRANUMERO(X)	Recibe una cadena de caracteres que contiene un número y devuelve una variable numérica con el mismo.
CONVERTIRATEXTO(S)	Recibe un real y devuelve una variable numérica con la representación como cadena de caracteres de dicho real.

Funciones raíz cuadrada y logaritmo natural

Para ejemplificar las funciones raíz cuadrada y logaritmo natural pongamos un ejemplo:

Tenemos el siguiente código:

```

Proceso raiz_cuadrada
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su raíz
cuadrada";
    Leer num;
    Escribir "La raíz cuadrada de ",num, " es "; Sin Bajar
    CalcularRaiz (num);
FinProceso

SubProceso CalcularRaiz ( num )
    Escribir RC (num);
FinSubProceso

```

Como se ve, este programa calcula la raíz cuadrada de un número que el usuario ingresa vía teclado, usando un procedimiento.

Si bien este código no tiene errores de compilación (no hay líneas marcadas en rojo) va a tirar error si, por ejemplo, el usuario ingresa un número negativo, pues no existen raíces cuadradas de números negativos en el campo de los números reales.

Para corregir este error vamos a usar otra función predefinida llamada ABS (valor absoluto)

Ahora el código nos quedaría de la siguiente forma:

```

Proceso programa_raiz_cuadrada_con_complejos
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su raíz
cuadrada";
    Leer num;
    Escribir "La raíz cuadrada de ",num, " es "; Sin Bajar
    CalcularRaiz (num);
FinProceso

SubProceso CalcularRaiz ( num )
    Si num < 0 Entonces
        Escribir RC (ABS (num)) , "i (resultado complejo)";
    Sino
        Escribir RC (num);
    FinSi
FinSubProceso

```

Ahora si maneja valores complejos, y el programa no va a tirar error porque se le

ingrese un número negativo, sino que va a mostrar el resultado seguido de una i ya que el resultado es complejo, seguido de una leyenda que indica que la naturaleza del resultado.

Pero supongamos que queremos extender la funcionalidad de este programa y queremos que calcule también el logaritmo de un número.

La idea sería mostrar un menú que permita al usuario elegir si desea calcular la raíz cuadrada o el logaritmo de un número.

El menú sería este:

```
Proceso programa_raiz_cuadrada_y_logaritmo
  Definir opcion Como Entero;
  Escribir "          Menú de opciones";
  Escribir "Presione 1 para calcular la raíz de un número";
  Escribir "Presione 2 para calcular el logaritmo de un número";
  Escribir "Presione 0 para salir";
  Leer opcion;
  Segun opcion Hacer
    1:
      CalcularRaiz();
    2:
      CalcularLogaritmo();
    0:
      Escribir "Saliendo...";
  De Otro Modo:
      Escribir "Opción incorrecta";
  FinSegun
FinProceso
```

Presentamos al usuario un menú que le pregunta al usuario si quiere calcular raíz cuadrada o logaritmo de un número. Si desea salir presiona la tecla 0.

Ahora creamos la función calcular logaritmo, que es parecida a la función raíz cuadrada, lo único que al número se le aplica la función LN (logaritmo natural en base N) en lugar de raíz cuadrada.

El procedimiento calcular logaritmo sería este:

```
SubProceso CalcularLogaritmo()
  Definir num, Resultado Como Reales;
```

```

    Escribir "Ingrese un número al que le desea calcular su logaritmo
natural";
    Leer num;
    Si num <= 0 Entonces
        Escribir "El logaritmo de ",num, " es ", LN(ABS(num)), "i
(resultado complejo)";
    Sino
        Escribir "El logaritmo de ",num, " es ",LN(num);
    FinSi
FinSubProceso

```

Finalmente, el programa completo quedaría así:

```

Proceso programa_raiz_cuadrada_y_logaritmo
    Definir opción Como Entero;
    Escribir "          Menú de opciones";
    Escribir "Presione 1 para calcular la raíz de un número";
    Escribir "Presione 2 para calcular el logaritmo de un número";
    Escribir "Presione 0 para salir";
    Leer opcion;
    Segun opcion Hacer
        1:
            CalcularRaiz();
        2:
            CalcularLogaritmo();
        0:
            Escribir "Saliendo...";
    De Otro Modo:
        Escribir "Opción incorrecta";
    FinSegun
FinProceso

SubProceso CalcularRaiz()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su raíz
cuadrada";
    Leer num;
    Si num < 0 Entonces
        Escribir "La raíz cuadrada de ",num, " es ", RC(ABS(num)), "i
(resultado complejo)";
    Sino
        Escribir "La raíz cuadrada de ",num, " es ",RC(num);
    FinSi
FinSubProceso

SubProceso CalcularLogaritmo()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su logaritmo
natural";
    Leer num;
    Si num <= 0 Entonces
        Escribir "El logaritmo de ",num, " es ", LN(ABS(num)), "i
(resultado complejo)";
    Sino

```

```

    Escribir "El logaritmo de ", num, " es ", LN(num);
FinSi
FinSubProceso

```

Funciones trigonométricas

PSeInt también puede manejar funciones trigonométricas: Seno, coseno, tangente, aseno, acoseo y atangente de un número.

Nota: Los resultados de las funciones trigonométricas se expresan en radianes.

Nota2: El tipo de dato de las variables para escribir resultados deben estar expresadas en reales, pues los resultados son siempre decimales, y si están definidos como enteros nos va a tirar error en tiempo de ejecución pues una variable entera estaría recibiendo un número que no es entero.

Empecemos por el menú:

```

Proceso funciones_tigonometricas
    Definir opcion Como Entero;
    Escribir "          Menú de opciones";
    Escribir "Presione 1 para calcular el seno de un número";
    Escribir "Presione 2 para calcular el coseno de un número";
    Escribir "Presione 3 para calcular la tangente de un número";
    Escribir "Presione 4 para calcular el arco seno de un número";
    Escribir "Presione 5 para calcular el arco coseno de un número";
    Escribir "Presione 6 para calcular el arco tangente de un número";
    Escribir "Presione 0 para salir";
    Leer opcion;
    Segun opcion Hacer
        1:
            CalcularSeno();
        2:
            CalcularCoseno();
        3:
            CalcularTangente();
        4:
            CalcularArcoSeno();
        5:
            CalcularArcoCoseno();
        6:
            CalcularArcoTangente();
        0:
            Escribir "Saliendo...";
    De Otro Modo:
        Escribir "Opción incorrecta";
    FinSegun
FinProceso

```

Ahora pasamos a definir las funciones correspondientes a cada función

```
SubProceso CalcularSeno()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su seno";
    Leer num;
    Escribir "El seno de ", num, " es ", Sen(num), " radianes";
FinSubProceso

SubProceso CalcularCoseno()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su coseno";
    Leer num;
    Escribir "El coseno de ", num, " es ", Cos(num), " radianes";
FinSubProceso

SubProceso CalcularTangente()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su tangente";
    Leer num;
    Escribir "La tangente de ", num, " es ", TAN(num), " radianes";
FinSubProceso
```

Ahora pasamos a los arcos:

Nota: Los valores de los arcos seno, arco coseno y arco tangente deben estar entre 1 y -1

Usamos la misma forma que el anterior, lo único que previamente debemos chequear que el valor dado por el usuario debe estar entre -1 y 1.

Quedaría así

```
SubProceso CalcularArcoSeno()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su arcoseno";
    Leer num;
    Si num > 1 | num < -1 Entonces
        Escribir "No, los números deben estar comprendidos entre 1 y -1";
    Sino
        Escribir "El arcoseno de ", num, " es ", ASen(num), " radianes";
    FinSi
FinSubProceso
```

Con la misma forma hacemos eso para los restantes subprocesos.

```

SubProceso CalcularArcoSeno()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su arcoseno";
    Leer num;
    Si num > 1 | num < -1 Entonces
        Escribir "No, los números deben estar comprendidos entre 1 y -1";
    Sino
        Escribir "El arcoseno de ", num, " es ", ASen(num), " radianes";
    FinSi
FinSubProceso

SubProceso CalcularArcoCoseno()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su arcocoseno";
    Leer num;
    Si num > 1 | num < -1 Entonces
        Escribir "No, los números deben estar comprendidos entre 1 y -1";
    Sino
        Escribir "El arcoseno de ", num, " es ", ACos(num), " radianes";
    FinSi
FinSubProceso

SubProceso CalcularArcoTangente()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su arcotangente";
    Leer num;
    Si num > 1 | num < -1 Entonces
        Escribir "No, los números deben estar comprendidos entre 1 y -1";
    Sino
        Escribir "El arcoseno de ", num, " es ", ATan(num), " radianes";
    FinSi
FinSubProceso

```

El programa completo quedaría así:

```

Proceso funciones_tigonometricas
    Definir opcion Como Entero;
    Escribir "          Menú de opciones";
    Escribir "Presione 1 para calcular el seno de un número";
    Escribir "Presione 2 para calcular el coseno de un número";
    Escribir "Presione 3 para calcular la tangente de un número";
    Escribir "Presione 4 para calcular el arco seno de un número";
    Escribir "Presione 5 para calcular el arco coseno de un número";
    Escribir "Presione 6 para calcular el arco tangente de un número";
    Escribir "Presione 0 para salir";
    Leer opcion;
    Segun opcion Hacer
        1:
            CalcularSeno();
        2:

```



```

        CalcularCoseno();
3:    CalcularTangente();
4:    CalcularArcoSeno();
5:    CalcularArcoCoseno();
6:    CalcularArcoTangente();
0:
    Escribir "Saliendo...";
De Otro Modo:
    Escribir "Opción incorrecta";
FinSegun
FinProceso

SubProceso CalcularSeno()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su seno";
    Leer num;
    Escribir "El seno de ", num, " es ", Sen(num), " radianes";
FinSubProceso

SubProceso CalcularCoseno()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su coseno";
    Leer num;
    Escribir "El coseno de ", num, " es ", Cos(num), " radianes";
FinSubProceso

SubProceso CalcularTangente()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su tangente";
    Leer num;
    Escribir "La tangente de ", num, " es ", TAN(num), " radianes";
FinSubProceso

SubProceso CalcularArcoSeno()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su arcoseno";
    Leer num;
    Si num > 1 | num < -1 Entonces
        Escribir "No, los números deben estar comprendidos entre 1 y -1";
    Sino
        Escribir "El arcoseno de ", num, " es ", ASen(num), " radianes";
    FinSi
FinSubProceso

SubProceso CalcularArcoCoseno()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su arcocoseno";
    Leer num;
    Si num > 1 | num < -1 Entonces
        Escribir "No, los números deben estar comprendidos entre 1 y -1";
    Sino

```

```

        Escribir "El arcoseno de ",num, " es ",ACos(num), " radianes";
    FinSi
FinSubProceso

SubProceso CalcularArcoTangente()
    Definir num, Resultado Como Reales;
    Escribir "Ingrese un número al que le desea calcular su
arcotangente";
    Leer num;
    Si num > 1 | num < -1 Entonces
        Escribir "No, los números deben estar comprendidos entre 1 y -1
";
    Sino
        Escribir "El arcoseno de ",num, " es ",ATan(num), " radianes";
    FinSi
FinSubProceso

```

Funciones trunc y redon

Trunc devuelve el *número entero* del argumento z, esto es, suprime la parte decimal de z: $\text{trunc}(z) = z - \text{parte decimal de } (z)$. Así, z puede ser cualquier expresión algebraica que evalúe a un número real. Ejemplo: $\text{trunc}(\pi) = 3$.

Por su parte, la función redon devuelve el entero más cercano a x.

Nota: el redondeo es hacia el entero mayor más cercano a x. Ejemplo: redon de 3.5 va a devolver 4 y no 3.

Para ejemplificar ambas funciones tomamos este programa de ejemplo:

```

Proceso redondeaAdosDecimales
    Definir numero Como Real;
    Definir numeroEntero, numeroRedondeado Como Enteros;
    Escribir "Escribe un número con decimales";
    Leer numero;
    numeroEntero <- trunc(numero);
    numeroRedondeado <- redon(numero);
    Escribir "Tu número truncado es ",numeroEntero;
    Escribir "Tu número redondeado es ",numeroRedondeado;
FinProceso

```

En primer lugar, se declara una variable numero de tipo real, ya que puede recibir números con punto (decimales)

Por otro lado, se declaran dos variables más: numeroEntero y

numeroRedondeado de tipo Entero ya que van a recibir un número que va a ser entero.

Acto seguido el programa pide un número al usuario un número (no importa si es de tipo entero o real), y ocurren dos cosas:

- En primero lugar se trunca el número con la palabra trunc y el resultado se guarda en la palabra numeroEntero
- En segundo lugar se redondea el número con la palabra redon y el resultado se guarda en la palabra numeroRedondeado

Por último, se muestran ambos resultados, con las sentencias Escribir

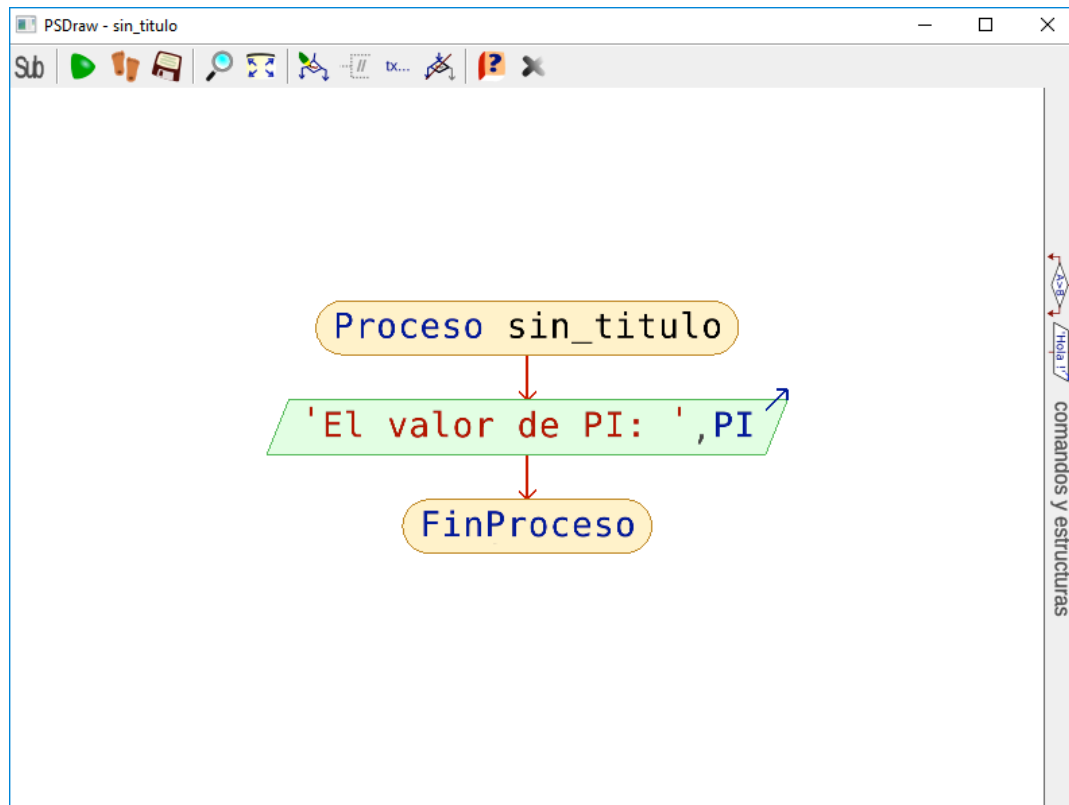
Constante PI

Otra función predefinida que tiene PSeInt es el número PI, o sea la relación entre la longitud de una circunferencia y su diámetro.

Para averiguar su valor basta con escribir la siguiente línea de código:

```
Proceso sin_titulo
    Escribir "El valor de PI: ",PI;
FinProceso
```

Diagrama de flujo



Si somos observadores, vemos que la palabra PI aparece en azul, esto porque además de ser una constante también es una palabra clave.

Concatenar y convertir a número

En sintaxis estricta, para concatenar dos cadenas se utiliza la palabra concatenar.

Concatenar es unir dos cadenas formando una más larga

Para ejemplificar estas palabras claves vamos a considerar lo siguiente.

Supongamos que queremos escribir en PSeInt un programa que despliegue el siguiente triángulo.

Por ejemplo, para altura = 10 debe desplegar

```

1
12
123
1234
12345
123456
1234567
12345678
  
```

```
123456789
12345678910
```

Hay otra forma de hacerlo, pero la que nos interesa a nosotros es esta por manejar estas dos palabras claves que mencionamos anteriormente.

Aquí está el código:

```
Proceso triangulo_numerico
  Definir piso Como Caracter;
  Definir x, altura Como Enteros;
  piso <- "";
  Escribir Sin Saltar "Ingrese la altura de la pirámide: ";
  Leer altura;
  Para x <- 1 Hasta altura Con Paso 1 Hacer
    piso <- Concatenar(piso, ConvertirATexto(x));
    Escribir piso;
  FinPara
FinProceso
```

Se recorre el bloque Para desde 1 hasta la altura (que es la que elige el usuario), y seguidamente la variable piso se la carga con la concatenación de 2 cadenas, la primera la variable piso y la conversión a texto de la variable x que a su vez va tomando los valores de 1,2... hasta altura. Al mismo tiempo se va escribiendo la variable piso a medida que se va incrementando el valor de x hasta que el valor de x coincida con el de altura (los valores sean iguales).

Nota: La variable piso, pesar de ser una variable de tipo caracter, debe inicializarse al igual que las variables numéricas, ya que en este caso el usuario no le carga ningún valor. Se le asigna asignándole vacío ("") a la variable, como se muestra el código.

Como bien se aclaró más arriba, en otras sintaxis que no sean la estricta, también se puede concatenar con el signo de más pero recordar que estamos utilizando esta última sintaxis, por lo tanto debemos usar la palabra concatenar.

Subcadena y convertir a número

Una subcadena es un segmento, o fragmento de una cadena más larga. Como dice la ayuda que incorpora PSeInt, la sintaxis de esta función es la siguiente:

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).

Nota: 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. En el caso, ya que estamos con perfil estricto, será 0

Un ejemplo para subcadena se muestra en el código de más abajo

```
// Lee un entero positivo e imprime el mismo número expresado
// en base 16.
// (c) jsegovia@cnc.una.py
// (Modificado)
Proceso de_dec_a_hexa
    Definir n, k, pos Como Enteros;
    Definir hex Como Caracter;
    Definir DIG_HEX Como Cadena;
    n <- 0;
    hex <- "";
    k <- 0;
    pos <- 0;
    DIG_HEX <- "0123456789ABCDEF";
    Escribir "Ingrese un entero positivo:";
    Leer n;
    Repetir
        pos <- n % 16 + 1;
        hex <- concatenar(subcadena(DIG_HEX, pos, pos), hex);
        n <- trunc(n/16);
    Hasta Que n = 0
    Escribir "Hexadecimal= ", hex;
FinProceso
```

El programa lo que hace es convertir un número a hexadecimal. Para hacer esto lo que hace después de pedir el número al usuario en decimal, es en primer lugar ir calculando la posición de hacer el resto más uno de dividir el número entre 16. Después se extrae la pos de cadena DIG_HEX y después se concatena con el valor de hex. Se calcula n como la división entera de n entre 16

Mayúsculas y minúsculas

Estas funciones lo que hacen es devolver una cadena. En el caso de mayúsculas devuelve la cadena convertida a mayúsculas y minúsculas a minúsculas.

A modo de ejemplo, supongamos esta consigna:

Crear un programa que pida su nombre al usuario y lo escriba alternando letras mayúsculas y minúsculas (por ejemplo, "Nacho" se mostraría como "NaChO").

Código que responde a la consigna:

```
Proceso alternandolettras
    Definir r1 Como Cadena;
    Escribir "Ingrese una palabra";
    Leer r1;
    alternarmayusculas( r1 );
FinProceso

SubProceso alternarmayusculas ( r1 )
    Definir p Como Cadena;
    Definir j Como Entero;
    Para j<-0 Hasta longitud(r1)-1 Hacer
        p<-subcadena(r1,j,j);
        Si j%2=0 Entonces
            Escribir Minusculas(p) Sin Bajar;
        Sino
            Escribir Mayusculas(p) Sin Bajar;
        FinSi
    FinPara
    Escribir "";
FinSubProceso
```

Se le pide al usuario que ingrese una palabra. Dicha palabra se guardará en la variable r1. Seguidamente, se la llama al procedimiento alternarmayusculas(r1). Dentro de dicho proceso, se recorre la cadena desde 0 (que es la base del arreglo) hasta la longitud -1 de la cadena ingresada por el usuario. Seguidamente, la subcadena r1 desde j hasta j y se guarda dicha subcadena en la variable p.

Si el valor de la variable j es par ($j\%2=0$, es decir el resultado de dividir el valor de j entre 2) En el caso de que lo sea, Escribir el carácter en mayúscula sino el minúscula. El ciclo se repite hasta que se llegue al final de la cadena, preguntado y colocando alternadamente en cada posición de la cadena.

Información teórica

Estructuras o registros

Nota: La información de estructuras se toma como teórica. De momento, PSeInt no soporta estructuras o registros.

Una estructura o registro es un dato estructurado, formado por elementos lógicamente relacionados, que pueden ser del mismo o de distintos tipos, a los que se les denomina campos. Los campos de una estructura podrían ser de los tipos previamente definidos por PSeInt (Entero, Real etc.) o bien por una estructura definida anteriormente.

Ejemplo: demostración de estructuras

En este programa usaremos una estructura para guardar la información del alumno usando una estructura que se llama Alum.

Siempre que queremos acceder a una estructura se hace

Estructura.Variable;

Entonces si queremos acceder a nombre escribiríamos Alum.nombre;

Alum porque así se llama la variable que es de tipo estructura re_alumno.

```

Registro regAlum
    Dimension nombre[30];
    Definir nombre Como Cadena;
    Dimension carrera[30];
    Definir carrera Como Cadena;
    Definir cuenta Como Entero;
FinRegistro

Proceso principal
    Definir Alum Como reg_alum;
    Escribir "El nombre del Alumno ";
    Leer Alum.nombre;
    Escribir "Cuenta del Alumno ";
    Leer Alum.cuenta;
    Escribir "Carrera que estudia ";
    Leer Alum.carrera;
    Escribir "El alumno ", Alum.nombre;
    Escribir "Con cuenta ", Alum.cuenta, " Estudia ", Alum.carrera;
FinProceso
  
```


Ahora lo más importante es que podamos usar estructuras o registros como parámetros en los subprocesos con retorno y sin retorno para hacer más fácil el pasar información como parámetro.

Ejemplo estructuras o registros con subprocesos

Se desea elaborar un programa donde se ingrese el nombre del alumno, la nota acumulada, examen, nota final y observación, luego que se determine la nota final y observación.

Usaremos una estructura para guardar la información, un subproceso sin retorno para el ingreso de datos, otro para calcular la nota final y la observación (se usará una función para el cálculo de la observación).

Siempre debemos de tomar en cuenta cuales son los parámetros de variable y de valor, en este programa usa en los subprocesos ingreso y cálculo de variable y en presentar de valor porque no se modifica ninguna variable.

```
// declaración de la estructura
```

```
Registro reg_alum
  Dimension nombre[30];
  Definir nombre Como Cadena;
  Definir na,ne,nf Como Reales;
  Dimension obs[10];
  Definir obs Como Cadena;
FinRegistro

Definir Alum Como RegAlum;

SubProceso vobs <- observacion (nota)
  Definir vobs Como Cadena;
  Si nota >= 60 Entonces
    vobs <- "aprobado";
  Sino
    vobs <- "reprobado";
  FinSi
FinProceso

SubProceso ingreso(alum Por Referencia)
```

```

Definir Alum Como reg_alum;
Escribir "Ingresar el nombre ";
Leer Alum.nombre;
Escribir "Ingresar la nota examen ";
Leer Alum.ne;
Escribir "Ingresar la nota acumulada ";
Leer Alum.na;
FinSubProceso

SubProceso calculo(alum Por Referencia)
    Alum.nf<-Alum.na+Alum.ne;
    Alum.obs<-observacion(Alum.nf);
FinSubProceso

SubProceso presentar(alum)
    Escribir "Nota Final ",Alum.nf;
    Escribir "Observación ",Alum.obs;
FinSubProceso

Proceso principal
    Definir I Como Entero;
    Para I<- 0 Hasta 4 Con Paso 1 Hacer
        ingreso(Alum);
        calculo(Alum);
        presentar(Alum);
    FinPara
FinProceso

```

Dimensiones con estructuras

Nota: Información teórica

Hasta ahora nuestros arreglos solo han guardado un solo datos ya sea real, entero cadena o caracter, cuando se quiere guardar más de un dato en una casilla de la dimension se declara una estructura y la dimension se declara que es del tipo estructura que declaramos.

Ejemplo:

```

Registro emple
    Definir codigo Como Entero;
    Dimension nombre[30];
    Definir nombre Como Cadena;
FinRegistro

Dimension empleado[5];
Definir empleado Como emple;

```

Código	Código	Código	Código	Código
Nombre	Nombre	Nombre	Nombre	Nombre
0	1	2	3	4

Declaramos la estructura o registro empleado y luego declaramos la dimension que será de tipo empleado ahora para acceder a la dimension:

Lectura de datos

```

Escribir "Ingresar Nombre del Empleado ";
Leer emple[3].nombre;
Escribir "Ingresar el código de estructura ";
Leer emple[3].codigo;

```

Al momento de leer, se tiene que especificar la posición de la dimension que deseo leer emple(3).nombre nos indica que se leerá de posición 3 el nombre.

Escribir datos

```

Escribir "Nombre del Empleado ", emple[3].nombre;
Escribir "Código de estructura ", emple[3].codigo;

```

Igual que al leer los datos para escribir especificamos el elemento de la dimension, del cual queremos presentar los datos de la estructura

Ejemplo de dimensiones con registro.

En este ejemplo declaramos la estructura, luego se declara la dimension de tipo

estructura, se elabora un subproceso sin retorno para el ingreso de los datos de la dimension y otro para presentar las estructuras de la dimension.

Cuando declaramos `Dimension emple[5];` y después `Definir emple Como Empleado;` en el subproceso de ingreso nos referimos a que tenemos una dimension de 5 elementos que es de tipo empleado (la estructura) y que la variable se llama emple.

En ambos subprocesos se recorre la dimension y luego por cada una de las posiciones de la dimension se lee el nombre y el código.

```

Registro emple <- Empleado
    Definir codigo Como Entero;
    Dimension nombre[30];
    Definir nombre Como Cadena;
FinRegistro

SubProceso Ingreso (emple)
    Definir i Como Entero;
    Para i <- 0 Hasta 4 Con Paso 1 Hacer
        Escribir "Ingresar Nombre del Empleado ";
        Leer emple.nombre;
        Escribir "Ingresar el código de estructura ";
        Leer emple.codigo;
    FinPara
FinSubProceso

SubProceso Presentar (emple)
    Definir i Como Entero;
    Limpiar Pantalla;
    Para i <- 0 Hasta 4 Con Paso 1 Hacer
        Escribir "Nombre del Empleado ",emple.nombre;
        Escribir "Código de estructura ", emple.codigo;
    FinPara
FinSubProceso

Proceso principal
    Dimension emple[5];
    Definir emple Como Empleado;
    Ingreso(emple);
    Presentar(emple);
FinProceso

```

Ejemplo de dimensiones con registro.

En este ejemplo declaramos la estructura o registro luego, se declara la dimension de tipo de tipo estructura alumno, luego usamos una función para determinar la observación, no se introduce todo el registro porque solo se ocupa un dato, para determinar la observación, luego en el procedimiento de cálculo al momento de enviar la nota para usar la observación indicamos el elemento de la dimension y la parte de la estructura que queremos enviar que es la nota:

```

alum[i].obs<-observacion(alum[i].nf);

// declaración del registro
Registro alum <- reg_alumno
  Dimension nombre[30];
  Definir nombre Como Caracter;
  Definir na,ne,nf Como Reales;
  Dimension obs[10];
  Definir obs Como Cadena;
FinRegistro

SubProceso vobs <- observacion (nota)
  Dimension vobs[10];
  Definir vobs Como Cadena;
  Si nota>= 60 Entonces
    vobs<-"aprobado";
  Sino
    vobs<-"reprobado";
  FinSi
FinSubProceso

SubProceso ingreso(alum Por Referencia)
  Definir i Como Entero;
  Para i <- 0 Hasta 4 Con Paso 1 Hacer
    Escribir "Ingresar el nombre ";
    Leer alum[i].nombre;
    Escribir "Ingresar la nota examen ";
    Leer alum[i].ne;
    Escribir "Ingresar la nota acumulada ";
    Leer alum[i].na;
  FinPara
FinSubProceso

SubProceso calculo(alum Por Referencia)
  Definir I Como Entero;
  Para i <- 0 Hasta 4 Con Paso 1 Hacer
    alum[i].nf<-alum[i].na + alum[i].ne alum[i].obs<-
observacion(alum[i].nf);
  FinPara
FinSubProceso

SubProceso presentar (alum)

```

```

Definir i Como Entero;
Para i <- 0 Hasta 4 Con Paso 1 Hacer
    Escribir "Nombre del alumno ", alum[i].nombre;
    Escribir "Nota Final ", alum[i].nf;
    Escribir "Observación ", alum[i].obs;
FinPara
FinSubProceso

Proceso Principal
    // declaración del arreglo de tipo registro
    Dimension alum[5];
    Definir alum Como reg_alumno;
    Ingreso(alum);
    Calculo(alum);
    Presentar(alum);
FinProceso

```

Ejemplo arreglos con estructura.

Se declara una estructura con las variables de nombre ventas, comisión ihss y total a pagar, se laboran una función para el seguro social, luego se elabora un procedimiento de ingreso de datos donde se el nombre y las ventas, después el procedimiento de cálculo, donde se determina la comisión que es el 5% de las ventas, el seguro usando la función del Seguro y el total a pagar, luego se presentan los datos usando un procedimiento.

```

Registro emple <- Empleado
    Dimension nombre[30];
    Definir nombre Como Cadena;
    Definir ventas, comis, ihss, tp Como Reales;
FinRegistro

SubProceso retorno <- seguro(sueldo)
    Definir retorno Como Reales;
    Si sueldo > 2400 Entonces
        retorno <- 84;
    Sino
        retorno <- 0.035*sueldo;
    FinSi
FinSubProceso

SubProceso Ingreso (emplea)
    Definir i Como Entero;
    Para i <- 0 Hasta 1 Con Paso 1 Hacer
        Escribir "ingresar Nombre del Empleado ";
        Leer emple[i].nombre;
        Escribir "Ingresar las ventas ";
        Leer emple[i].ventas;
    FinPara

```

FinSubProceso

```

SubProceso Calculo(emple)
  Dimension empleado[5];
  Definir emple Como Empleado;
  Definir i Como Entero;
  Para i <- 0 Hasta 2 Con Paso 1 Hacer
    emple[i].comis<-emple[i].ventas*0.05;
    emple[i].ihss<-seguro(emple[i].comis);
    emple[i].tp<-emple[i].comis-emple[i].ihss;
  FinPara
FinSubProceso

SubProceso Presentar (emple)
  Dimension empleado[5];
  Definir emple Como Empleado;
  Definir i Como Entero;
  Para i <- 0 Hasta 1 Hacer
    Escribir "Empleado ",emple[i].nombre;
    Escribir "Comisión ..:", emple[i].comis;
    Escribir "Seguro Social..:", emple[i].ihss;
    Escribir "Total a Pagar ..:", emple[i].tp;
    Escribir "";
    Escribir "";
  FinPara
FinSubProceso

Proceso principal
  Ingreso(emple);
  Calculo(emple);
  Presentar(emple);
FinProceso

```

Manejo De Archivos De Texto

Nota: Información teórica

Hasta esta parte, todos los resultados de los programas se borran de la memoria al terminar el programa, en este capítulo aprenderemos de forma teórica como guardaríamos la información en un archivo de texto para su posterior utilización.

Sintaxis

Declarar un tipo archivo

Declarar un tipo archivo secuencial es necesario para , declarar variable de este tipo ejemplo :

```
Tipo Arch Es Archivo Secuencial;
```

Abrir un archivo

Sintaxis

Abrir nombre_archivo como variable [para lectura, escritura]

ejemplo:

```
Abrir "empleados.txt" Como Archemple Para Lectura;
```

Descripción

Esta instrucción sirve para abrir el archivo. Las operaciones permitidas para el archivo son lectura, escritura o ambas. En la sintaxis variable se refiere a variable de tipo archivo que se usará para referenciar el archivo.

Cerrar un archivo

Sintaxis

Cerrar variable de tipo archivo

Ejemplo :

```
Cerrar archemple;
```


Descripción

Esta instrucción sirve para cerrar un archivo. Variable

Leer de un archivo

Sintaxis

Leer variable_archivo, variable_datos ejemplo :

```
Leer   archemple, emple.nombre;
```

Descripción

Esta instrucción lee una variable desde un archivo. La primera variable de la instrucción debe ser de tipo archivo, la segunda puede ser de cualquier tipo, eso dependerá del tipo de archivo.

Escribir en un archivo

Sintaxis

Escribir variable_archivo, variable_datos; ejemplo:

```
Escribir archemple,  emple.nombre;
```

Descripción

Esta instrucción escribe una variable en un archivo. La primera variable de la

instrucción debe ser de tipo archivo, la segunda puede ser de cualquier tipo, eso dependerá del tipo de archivo.

Ejemplo Ingreso de datos a un archivo secuencial (texto).

Lo primero que tenemos que hacer es crear con Windows un archivo de texto, con el notepad, y lo salvamos con el nombre de empleados, en el mismo directorio donde salvaremos el programa de ingreso de datos.

Declaramos el tipo de archivo secuencial

```
Tipo Arch es archivo secuencial;
```

luego la estructura o registro que usaremos para ingresar los datos

```
Estructura Empleado
    Dimension nombre[50];
    Definir nombre Como Cadena;
    Definir sueldo Como Real;
    Definir sexo Como Caracter;
FinEstructura
```

luego declaramos la variable para manejar el archivo de texto, que de tipo arch y la variable de tipo estructura

```
Definir emplea Como Empleado;
Definir ArchEmple Como Arch;
Definir resp Como Caracter;
```

Luego en el programa lo primero que se hace es abrir el archivo para escritura, luego se piden los datos y se salvar en el archivo, al final se cierra el archivo de texto, ahora si nosotros queremos saber si guardo los datos, podremos abrir empleados con el notepad y veremos los datos que se salvaron en el archivo.

```

Tipo Arch Es Archivo Secuencial;

Estructura emple <- Empleado
  Dimension nombre[50];
  Definir nombre Como Cadena;
  Definir sueldo Como Real;
  Definir sexo como Caracter;
FinEstructura

Proceso principal
  Definir emple Como Empleado;
  Definir ArchEmple Como Arch;
  Definir resp como Caracater;
  Abrir "empleados.txt" Como archemple Para Escritura;
  Repetir
    Escribir "Nombre del emnpleado...";
    Leer emple.nombre;
    Escribir "Sueldo del empleado...";
    Leer emple.sueldo;
    Escribir "Sexo ...";
    Leer emple.sexo;
    Escribir archemple, emple.nombre;
    Escribir archemple, emple.sueldo;
    Escribir archemple, emple.sexo;
    Escribir "Desea Continuar ...";
    Leer resp;
  Hasta Que resp="S" | resp="N";
  Hasta Que resp='N';
  Cerrar archemple;
FinProceso

```

Ejemplo Listar el contenido de un archivo secuencial (texto).

Se declara el tipo del archivo, el registro y las variables para usar la estructura y el archivo de texto, luego se abre el archivo para lectura y se hace un ciclo mientras no sea fin de archivo, esto se logra con la función FDA que nos devuelve verdadero cuando se encuentra al final del archivo y falso cuando no lo está.

Se usa la instrucción Leer, para recuperar los valores que se guardaron en el archivo de texto, luego usando un procedimiento se escriben los valores de la estructura en la pantalla

```

Estructura emple <- Empleado
  Dimension nombre[50];

```

```

    Definir nombre Como Cadena;
    Definir sueldo Como Real;
    Definir sexo Como Caracter;
FinEstructura

Proceso principal
    Definir emple Como Empleado;
    Definir ArchEmple Como Arch;
    Definir resp Como Caracter;
    Tipo Arch Es Archivo Secuencial;
    Abrir "empleados.txt" Como archemple Para Escritura;
        Repetir
            Escribir "Nombre del emnpleado..:";
            Leer emple.nombre;
            Escribir "Sueldo del empleado...:";
            Leer emple.sueldo;
            Escribir "Sexo ..:";
            Leer emple.sexo;
            Escribir archemple, emple.nombre;
            Escribir archemple, emple.sueldo;
            Escribir archemple, emple.sexo;
            Escribir "Desea Continuar ..:";
            Leer resp;
        Hasta Que resp="S" | resp="N";
    Hasta Que resp='N';
    Cerrar archemple;
FinProceso

```

Anexo:

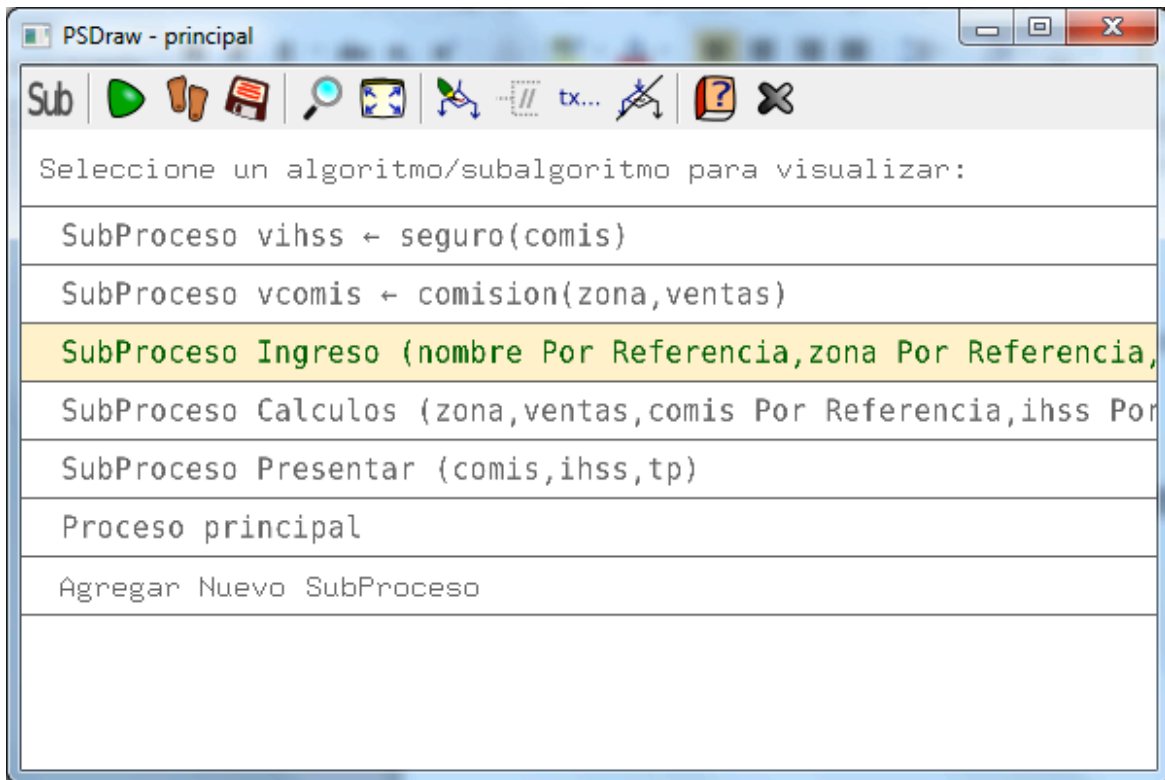
Nota: Información práctica

Editar diagramas de flujo

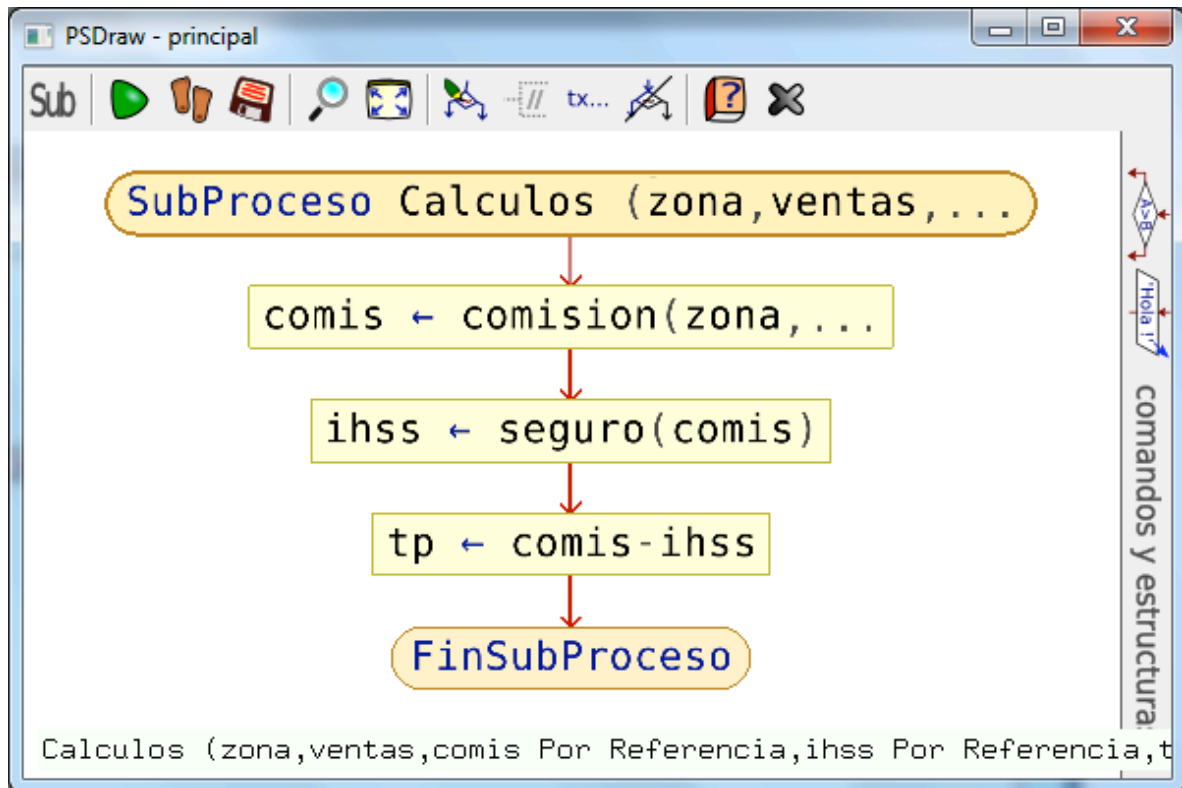
PSeInt permite editar el diagrama de flujo, luego editar los cambios, para que pueda ser ejecutado desde pseudocódigo.

Accedemos al editor de diagramas de flujo yendo a Archivo → Editar diagramas de flujo:

Elegimos un subproceso y hacemos clic en uno de ellos



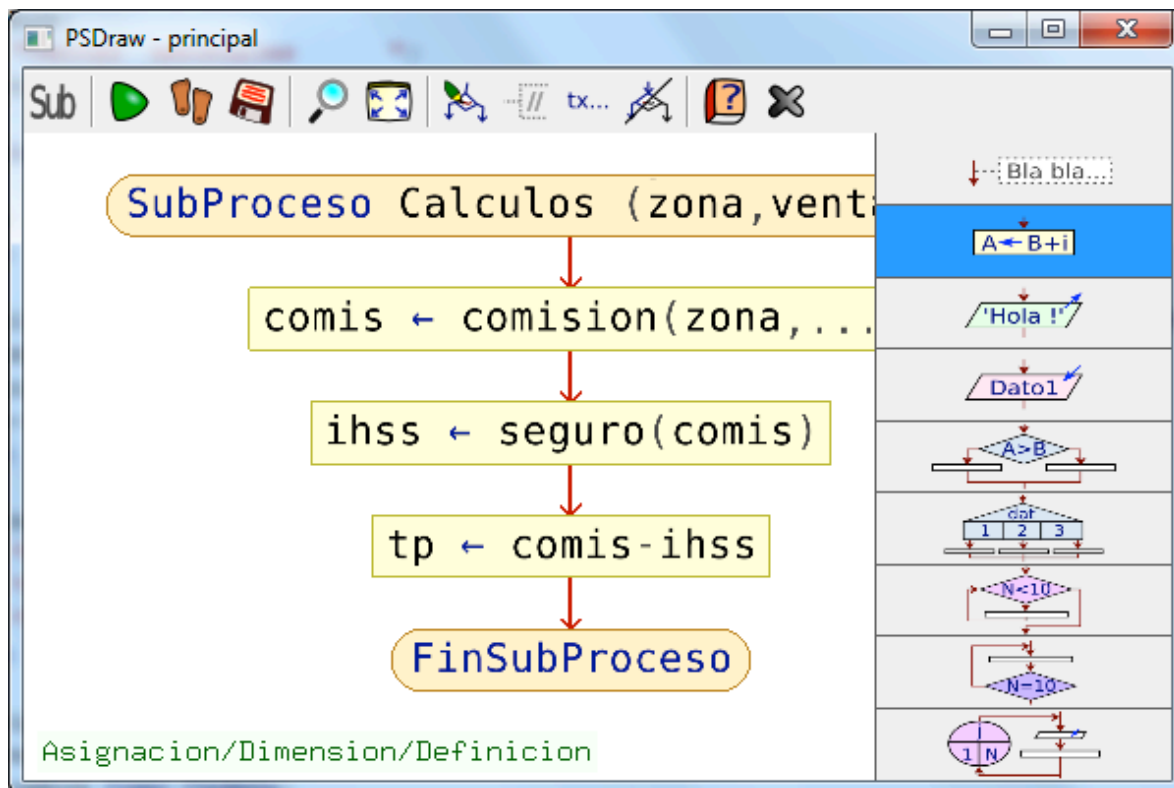
Hacemos clic en el proceso principal o cualquiera de los subprocesos, en este caso el subproceso Ingreso.



Se nos presenta una pantalla mostrando el diagrama de flujo correspondiente al subproceso que estamos ejecutando.

A la derecha encontramos una pestaña que aparecen dos iconos y al costado el título comandos y estructuras

Pasamos el mouse sobre la misma.



Se nos presenta un dibujo con las estructuras usadas, y al costado izquierdo inferior aparece su nombre.

Si queremos añadir un nuevo bloque al diagrama de flujo, lo que hacemos es clicar en un bloque y sin soltar el botón izquierdo del mouse arrastrarlo hasta el diagrama de flujo. Para fijar el bloque, presionamos la tecla escape.

En las sentencias escribir, el texto se debe poner entre comillas.

Guardar cambios

Para guardar los cambios, vamos al botón que se encuentra al costado izquierdo superior y hacemos clic en guardar cambios.

No se ejecutan diagramas de flujo que no sean guardados.

Nota: También se pueden crear diagramas sin necesidad de escribir su pseudocódigo correspondiente.

Nota: Por la forma de trabajar del intérprete de diagrama de flujo, si se guardan

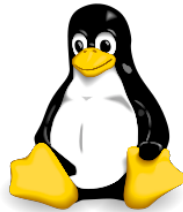
los cambios desde el editor de diagrama de flujo, hay modificaciones en el pseudocódigo, por ejemplo, pasado de comillas a apóstrofes, etc. Estos errores se pueden ir resolviendo a medida que salgan nuevas versiones de PSeInt.

Desinstalar PSeInt

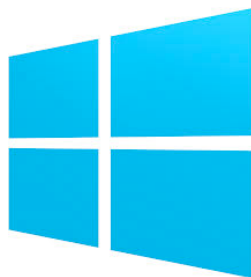
PSeInt dispone de un des instalador, que se accede desde agregar o quitar programas. Se desinstala como cualquier otro programa.

Abrir el código fuente

En estos blogs se explica cómo lo que debemos hacer para abrir el código fuente del programa:



[Bajo Linux](#)



[Bajo Windows](#)

Bibliografía

- <http://studylib.es/doc/380831/funciones>
- <http://www.slideshare.net/juanrobyn/manual-de-pseint>
- <http://slideplayer.es/slide/9449505/>
- <http://odelys2003.files.wordpress.com/2011/10/pseint.pptx>

Sitio Web oficial

- <http://pseint.sourceforge.net/>

Funcionalidades y comparaciones con el diagrama de flujo

- <http://pseint.sourceforge.net/index.php?page=pseudocodigo.php>

Bibliografía implementada en el manual de PSeInt (con modificaciones)

Entorno de PSeInt y explicación de sus partes

- <http://studylib.es/doc/88005/expresiones-pseint>

Ejecución paso a paso

- <http://agrega.juntadeandalucia.es/repositorio/20022017/d7/es-an 2017022012 9123947/descripcin de la tarea.html>