INSTITUTO TECNOLÓGICO DE SONORA
Educar para Trascender

Pseudocódigo y PSEINT

Programa Educativo
Ingeniero en Software

Marzo 2016

INDICE

INTRODUCCIÓN

El siguiente manual muestra de manera sencilla como manejar el programa PSeint.

Cuando nos enfrentamos a un problema en la vida cotidiana, su resolución requiere que sigamos una serie de pasos; para tal fin. El conjunto ordenado de pasos seguidos con el fin de resolver un problema o lograr un objetivo es conocido como algoritmo.

Un algoritmo es un conjunto de instrucciones que especifica la secuencia de operaciones a realizar, en orden, para resolver un problema específico; en otras palabras, un algoritmo ***es una fórmula para la resolución de un problema***.

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

- **Entrada**: Información dada al algoritmo, o conjunto de instrucciones que generen los valores con que ha de trabajar**.**
- **Proceso**: Cálculos necesarios para que a partir de un dato de entrada se llegue a los resultados.
- **Salida**: Resultados finales o transformación que ha sufrido la información de entrada a través del proceso.

Cuando se formula un algoritmo el objetivo es ejecutar este en un computador, sin embargo, para que este entienda los pasos para llevar a cabo nuestro algoritmo debemos indicárselo siguiendo un conjunto de instrucciones y reglas que este entienda, y estas instrucciones son abstraídas en lo que conocemos como ***lenguaje de programación***.

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

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

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

Por otro lado, el **pseudocódigo** es un lenguaje de especificación (descripción) de algoritmos. El uso de tal lenguaje hace el paso de codificación final (traducción al

lenguaje de programación) relativamente fácil, por lo que este es considerado un primer borrador de la solución del programa.

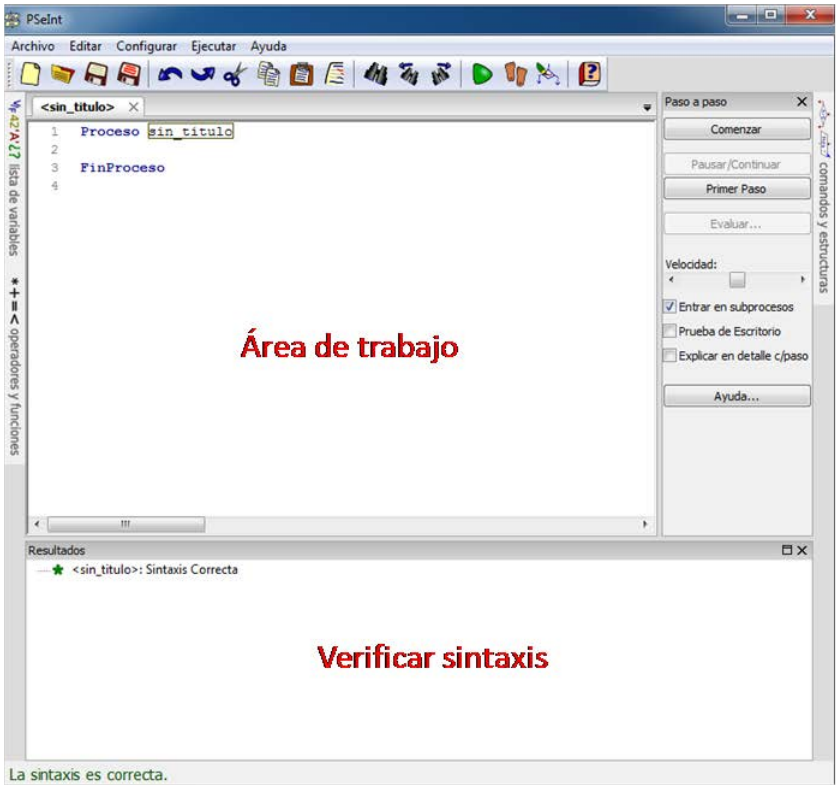
¿Qué es PSEINT?

PSeInt es principalmente un intérprete de pseudocódigo. El proyecto nació como trabajo final para la cátedra de *Programación I* de la carrera *Ingeniería en Informática* de la *Universidad nacional del Litoral*, razón por la cual el tipo de pseudocódigo que interpreta está basado en el pseudocódigo presentado en la cátedra de *Fundamentos de Programación* de dicha carrera. Actualmente incluye otras funcionalidades como editor y ayuda integrada, generación de diagramas de flujo o exportación a código C++ (en etapa experimental).

El proyecto se distribuye como software libre bajo licencia GPL.

Para descargarlo o conseguir actualizaciones visite **http://pseint.sourceforge.net**

La interfaz y el área de trabajo



← Barra de Menú
← Barra de Acceso rápido

Las funciones: botones

Abre un nuevo documento

Busca un fichero (archivo)

Guardar y guardar como

Deshacer y Rehacer respectivamente

Cortar

Copiar y pegar

Corregir indentado

Buscar

Ejecutar el algoritmo

Ejecutar paso a paso

Dibujar diagrama de flujo

Ayuda/contiene algunos ejemplos

El Pseudocódigo

Las características del este pseudolenguaje fueron propuestas en 2001 por el responsable de la asignatura Fundamentos de Programación (Horacio Loyarte) de la carrera de Ingeniería Informática de la FICH-UNL. Las premisas son:

- Sintaxis sencilla.
- Manejo de las estructuras básicas de control.
- Solo 3 tipos de datos básicos: numérico, carácter/cadenas de caracteres y lógico (verdadero/falso).
- Estructuras de datos: arreglos.

Forma general de un algoritmo en Pseudocódigo

Todo algoritmo en pseudocódigo de Pseint tiene la siguiente estructura general:

```
Proceso SinTitulo
        accion 1;
        accion 1;
        .
        .
        .
        accion n;
FinProceso
```

Comienza con la palabra clave *Proceso* seguida del nombre del programa, luego le sigue una secuencia de instrucciones y finaliza con la palabra *FinProceso*. Una secuencia de instrucciones es una lista de una o más instrucciones, cada una terminada en punto y coma.

Las acciones incluyen operaciones de entrada y salida, asignaciones de variables, condicionales si-entonces o de selección múltiple y/o lazos mientras, repetir o para.

Tipos de datos

- Tipos Simples: Numérico, Lógico, Caracter.
- Estructuras de Datos: Arreglos.

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

Tipos de Datos Simples

Existen tres tipos de datos básicos:

- *Numérico*: números, tanto enteros como decimales. Para separar decimales se utiliza el punto. Ejemplos: 12 23 0 -2.3 3.14
- *Lógico*: solo puede tomar dos valores: VERDADERO o FALSO.
- *Carácter*: caracteres o cadenas de caracteres encerrados entre comillas (pueden ser dobles o simples). Ejemplos 'hola' "hola mundo" '123' 'FALSO' 'etc'

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

Estructuras de Datos: Arreglos

Los arreglos son estructuras de datos homogéneas (todos sus datos son del mismo tipo) que permiten almacenar un determinado número de datos bajo un mismo identificador, para luego referirse a los mismo utilizando uno o más subíndices. Los arreglos pueden pensarse como vectores, matrices, etc.

Para poder utilizar un arreglo, primero es obligatorio su dimensionamiento; es decir, definirlo declarando los rangos de sus subíndices, lo cual determina cuantos elementos se almacenarán y como se accederá a los mismos.

Dimensionamiento (Arreglos-Arrays)

La instrucción Dimensión permite definir un arreglo, indicando sus dimensiones.

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

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

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

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

Expresiones

- Operadores.
- Funciones.

Operadores

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

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

Operador	Significado	Ejemplo
Relacionales		
>	Mayor que	3>2
<	Menor que	'ABC'<'abc'
=	Igual que	4=3
<=	Menor o igual que	'a'<='b'
>=	Mayor o igual que	4>=5
<>	Distinto que	Var1<>var2
Lógicos		
& ó Y	Conjunción (y).	(7>4) & (2=1) //falso
| ó O	Disyunción (o).	(1=1 | 2=1) //verdadero
~ ó NO	Negación (no).	~(2<5) //falso
Algebraicos		
+	Suma	total <- cant1 + cant2
-	Resta	stock <- disp – venta
*	Multiplicación	area <- base * altura
/	División	porc <- 100 * parte / total
^	Potenciación	sup <- 3.41 * radio ^ 2
% ó MOD	Módulo (resto de la división entera)	resto <- num MOD div

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

Funciones matemática

Las funciones en el pseudocódigo se utilizan de forma similar a otros lenguajes. Se coloca su nombre seguido de los argumentos para la misma encerrados entre paréntesis (por ejemplo trunc(x)). Se pueden utilizar dentro de cualquier expresión, y cuando se evalúe la misma, se reemplazará por el resultado correspondiente. Actualmente, todas la funciones disponibles son matemáticas (es decir que

devolverán un resultado de tipo numérico) y reciben un sólo parámetro de tipo numérico. A continuación se listan las funciones integradas disponibles:

Función	*Significado*
RC(X)	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
TRUNC(X)	Parte entera de X
REDON(X)	Entero más cercano a X
AZAR(X)	Entero aleatorio entre 0 y x-1

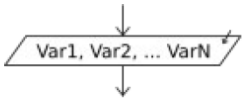
La función raíz cuadrada no debe recibir un argumento negativo.
La función exponencial no debe recibir un argumento menor o igual a cero.

Primitivas Secuenciales (Comandos de Entrada, Proceso y Salida)

- Lectura (Entrada).
- Asignación (Proceso).
- Escritura (Salida).

Lectura o entrada

La instrucción Leer permite ingresar información desde el ambiente.

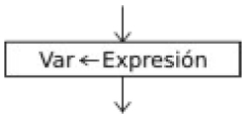


Leer <variablel> , <variable2> , ... , <variableN> ;

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

Asignación o proceso

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



<variable> <- <expresión> ;

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

Escritura o salida

La instrucción Escribir permite mostrar valores al ambiente.

Escribir <exprl> , <expr2> , ... , <exprN> ;

Esta instrucción imprime al ambiente (en este caso en la pantalla) los valores obtenidos de evaluar N expresiones. Dado que puede incluir una o más expresiones, mostrará uno o más valores.

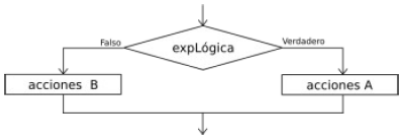
Estructuras de Control (Proceso)

- Condicionales
  - Si-Entonces
  - Selección Múltiple

- Repetitivas
  - Mientras
  - Repetir
  - Para

Condicionales

Si-Entonces (If-Then)



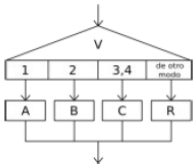
La secuencia de instrucciones ejecutadas por la instrucción Si-Entonces-Sino depende del valor de una condición lógica.

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

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

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

Selección Múltiple (Select If)



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

```
Segun <variable> Hacer
        <número1>: <instrucciones>
        <número2>,<número3>: <instrucciones>
        <...>
```

```
        De Otro Modo: <instrucciones>
    FinSegun
```

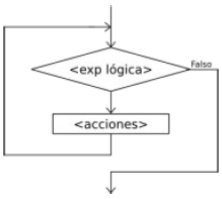
Esta instrucción permite ejecutar opcionalmente varias acciones posibles, dependiendo del valor almacenado en una variable de tipo numérico. Al ejecutarse, se evalúa el contenido de la variable y se ejecuta la secuencia de instrucciones asociada con dicho valor.

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

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

Repetitivas

Mientras Hacer (while)



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

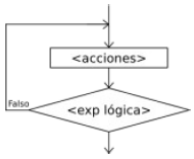
```
    Mientras <condición> Hacer
        <instrucciones>
    FinMientras
```

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

Note que las instrucciones del cuerpo del ciclo pueden no ejecutarse nunca, si al evaluar por primera vez la condición resulta ser falsa.
Si la condición siempre es verdadera, al ejecutar esta instrucción se produce un ciclo infinito. A fin de evitarlo, las instrucciones del cuerpo del ciclo deben contener alguna instrucción que modifique la o las variables involucradas en la condición,

de modo que ésta sea falsificada en algún momento y así finalice la ejecución del ciclo.

Repetir Hasta Que (do-while)



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

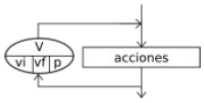
```
Repetir
        <instrucciones>
Hasta Que <condición>
```

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

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

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

Para (for)



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

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

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

Ejecución Paso a Paso

La ejecución paso a paso permite realizar un seguimiento más detallado de la ejecución del algoritmo. Es decir, permite observar en tiempo real qué instrucciones y en qué orden se ejecutan, como así también observar el contenido de variables o expresiones durante el proceso.
Para acceder al panel de ejecución paso a paso puede o bien utilizar la opción "Mostrar Panel de Ejecución Paso a Paso" del menú "Configuración", o bien hacer click sobre el botón de ejecución paso a paso en la barra accesos rápidos (ubicado entre los botones para ejecutar y dibujar diagrama de flujo).

El botón "**Comenzar**" del panel sirve para iniciar la ejecución automática. Cuando lo utilice, el algoritmo comenzará a ejecutarse lentamente y cada instrucción que se vaya ejecutando según el flujo del programa se irá seleccionando en el código de dicho algoritmo. La velocidad con que avance la ejecución del algoritmo, inicialmente depende de la seleccionada en el menú "Configuración", aunque mientras la ejecución paso a paso está en marcha, puede variarla desplazando el control rotulado como "**Velocidad**" en el panel.

Otra forma de comenzar la ejecución paso a paso es utilizar el botón "**Primer Paso**" del mismo panel. Este botón iniciará la ejecución, pero a diferencia de "Comenzar" no avanzará de forma automática, sino que se parará sobre la primer línea del programa y esperará a que el usuario avance manualmente cada paso con el mismo botón (que pasará a llamarse "Avanzar un Paso".

El botón "**Pausar/Continuar**" sirve para detener momentáneamente la ejecución del algoritmo y reanudarla nuevamente después. Detener el algoritmo puede servir para analizar el código fuente, o para verificar qué valor tiene asignado una variable o cuanto valdría una determinada expresión en ese punto.

Para determinar el valor de una variable o expresión, una vez pausada la ejecución paso a paso, utilice el botón "**Evaluar...**". Aparecerá una ventana donde podrá introducir cualquier nombre de variable o expresión arbitraria (incluyendo funciones y operadores), para luego observar su valor.

Finalmente, la forma más completa para analizar la ejecución es la denominada Prueba de Escritorio.

Antes de comenzar la ejecución, puede seleccionar qué variables o expresiones desea visualizar durante la ejecución. Para ello utilice el botón "**Prueba de Esc.**" y modifique la lista. Cuando la ejecución comience, por cada línea ejecutada, se añadirá un renglón en la tabla de la prueba de escritorio (se mostrará en la parte inferior de la ventana como un panel acoplable) indicando el número de línea y los valores de todas la variables y expresiones especificadas.

Algunas Observaciones

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

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

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

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

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

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

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

- Las constantes lógicas son *Verdadero* y *Falso*.

- Actualmente este pseudolenguaje no contempla la creación de nuevas funciones o subprocesos.

Ejemplos de Algoritmos

PSeInt incluye un conjunto de algoritmos de diferentes niveles de dificultad para ejemplificar la sintaxis y el uso del pseudocódigo. A continuación se describen los ejemplos disponibles:

1. **AdivinaNumero**: Sencillo juego en el que el usuario debe adivinar un número aleatorio.

```
// Juego simple que pide al usuario que adivine un numero en 10 intentos
Proceso Adivina_Numero
	intentos<-9;
	num_secreto <- azar(100)+1;

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

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

2. **Mayores**: Busca los dos mayores de una lista de N datos.

```
// Busca los dos mayores de una lista de N datos
Proceso Mayores
	Dimension datos[200];
	Escribir "Ingrese la cantidad de datos:";
	Leer n;
	Para i<-1 Hasta n Hacer
		Escribir "Ingrese el dato ",i,":";
		Leer datos[i];
	FinPara
```

```
        Si datos[1]>datos[2] Entonces
                may1<-datos[1];
                may2<-datos[2];
        Sino
                may1<-datos[2];
                may2<-datos[1];
        FinSi

        Para i<-3 Hasta n Hacer
                Si datos[i]>may1 Entonces
                        may2<-may1;
                        may1<-datos[i];
                Sino
                        Si datos[i]>may2 Entonces
                                may2<-datos[i];
                        FinSi
                FinSi
        FinPara

        Escribir "El mayor es: ",may1;
        Escribir "El segundo mayor es: ",may2;
FinProceso
```

3. **Triángulo**: Este algoritmo determina a partir de las longitudes de tres lados de un triángulo si corresponden a un triángulo rectángulo (para utiliza la relación de Pitágoras, tomando los dos lados de menor longitud como catetos), y en caso afirmativo informa el área del mismo. Lee los tres lados de un triángulo rectángulo, determina si corresponden (por Pitágoras) y en caso afirmativo calcula el área Proceso TrianguloRectangulo

```
        // cargar datos
        Escribir "Ingrese el lado 1:";
        Leer l1;
        Escribir "Ingrese el lado 2:";
        Leer l2;
        Escribir "Ingrese el lado 3:";
        Leer l3;

        // encontrar la hipotenusa (mayor lado)
        Si l1>l2 Entonces
                cat1<-l2;
                Si l1>l3 Entonces
                        hip<-l1;
                        cat2<-l3;
                Sino
```

```
                hip<-l3;
                cat2<-l1;
            FinSi
        Sino
            cat1<-l1;
            Si l2>l3 Entonces
                hip<-l2;
                cat2<-l3;
            Sino
                hip<-l3;
                cat2<-l2;
            FinSi
        FinSi

// ver si cumple con Pitágoras
Si hip^2 = cat1^2 + cat2^2 Entonces
        // calcular área
        area<-(cat1*cat2)/2;
        Escribir "El área es: ",area;
Sino
        Escribir "No es un triángulo rectángulo.";
FinSi
FinProceso
```

4. **OrdenaLista**: Este ejemplo almacena una lista de nombres en un arreglo y luego los ordena alfabéticamente. El método de ordenamiento es relativamente simple. Para la entrada de datos se utiliza una estructura MIENTRAS, sin saber a priori la cantidad de datos que se ingresarán. Se ingresa una lista de nombres (la lista termina cuando se ingresa un nombre en blanco) no permitiendo ingresar repetidos y luego se ordena y muestra.

```
Proceso OrdenaLista Dimension lista[200];

Escribir "Ingrese los nombres (enter en blanco para terminar):";

// leer la lista
cant<-0;
Leer nombre;
Mientras nombre<>"" Hacer
        cant<-cant+1;
        lista[cant]<-nombre;
        Repetir // leer un nombre y ver que no esté ya en la lista
            Leer nombre;
            se_repite<-Falso;
            Para i<-1 Hasta cant Hacer
                Si nombre=lista[i] Entonces
```

```
                    se_repite<-Verdadero;
                FinSi
            FinPara
        Hasta Que NO se_repite
FinMientras

// ordenar
Para i<-1 Hasta cant-1 Hacer
        // busca el menor entre i y cant
        pos_menor<-i;
        Para j<-i+1 Hasta cant Hacer
            Si lista[j]<lista[pos_menor] Entonces
                pos_menor<-j;
            FinSi
        FinPara

        // intercambia el que estaba en i con el menor que encontro
        aux<-lista[i];
        lista[i]<-lista[pos_menor];
        lista[pos_menor]<-aux;
FinPara

// mostrar cómo queda la lista
Escribir "La lista ordenada es:";
Para i<-1 Hasta cant Hacer
        Escribir " ",lista[i];
FinPara
FinProceso
```

5. **Promedio**: Ejemplo básico de uso de un acumulador y la estructura de control PARA para calcular el promedio de un conjunto de valores.

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

Proceso Promedio

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

        acum<-0;

        Para i<-1 Hasta n Hacer
            Escribir "Ingrese el dato ",i,":";
            Leer dato;
            acum<-acum+dato;
        FinPara
```

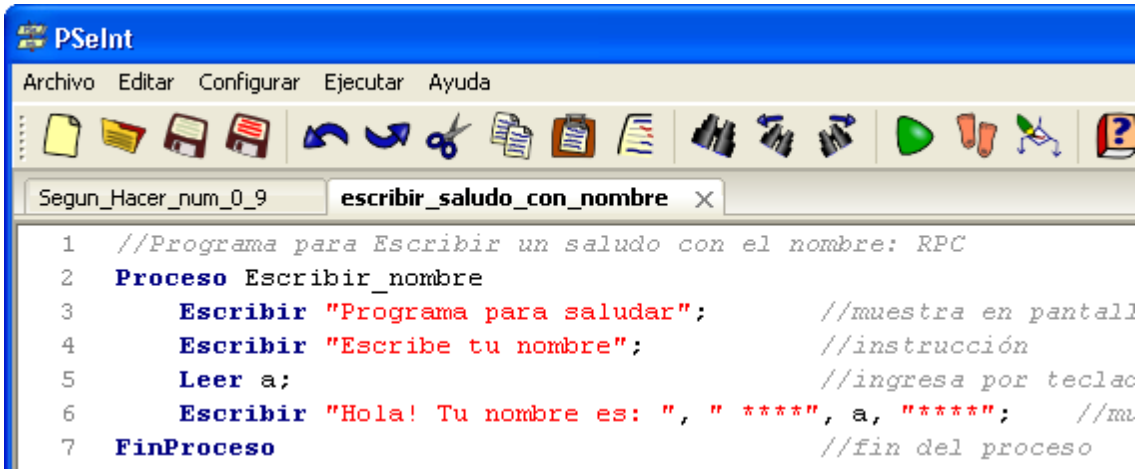
```
        prom<-acum/n;

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

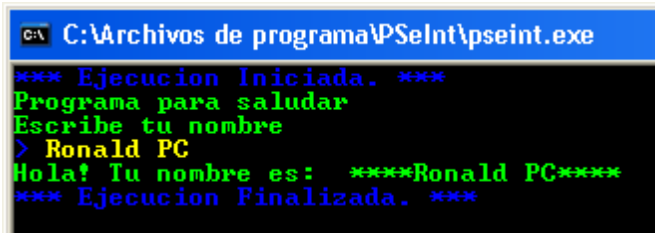
Ejercicios Resueltos utilizando PSeint

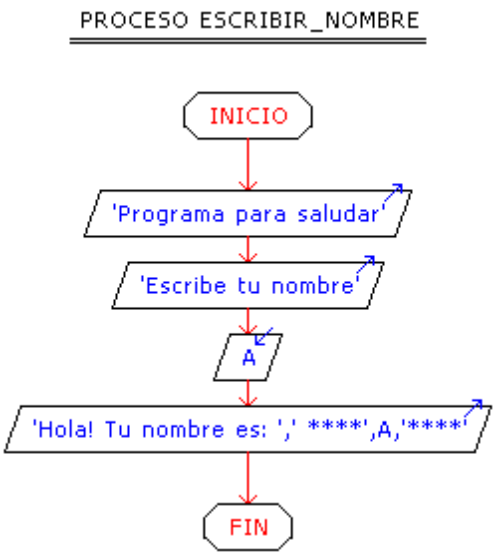
1. Escribir un nombre y saludar

//Programa para Escribir un saludo con el nombre: RPC

```
Proceso Escribir_nombre
    Escribir "Programa para saludar"; //muestra en pantalla:
"Progr...saludar"
    Escribir "Escribe tu nombre"; //instrucción
    Leer a; //ingresa por teclado un texto
    Escribir "Hola! Tu nombre es: ", " ****", a, "****"; //muestra un saludo
    con el nombre escrito
FinProceso //fin del proceso
```



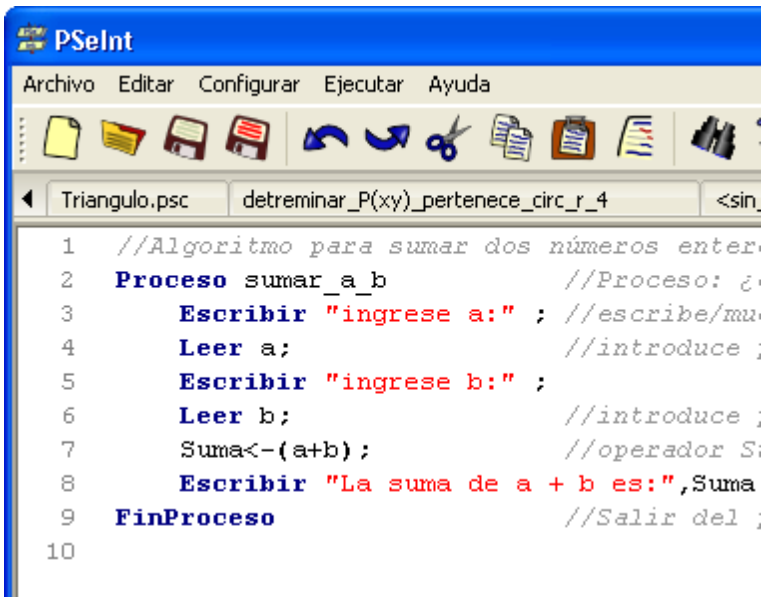


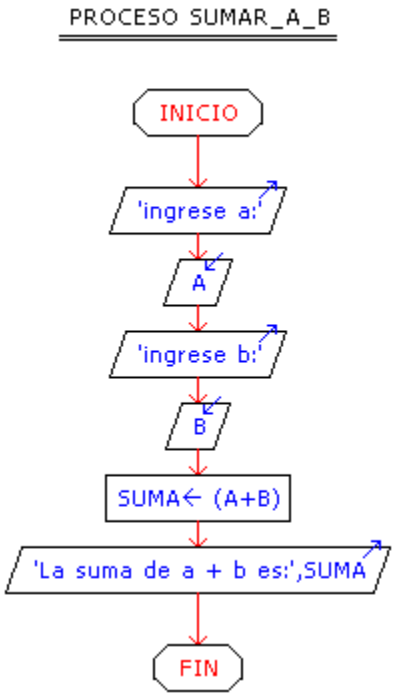


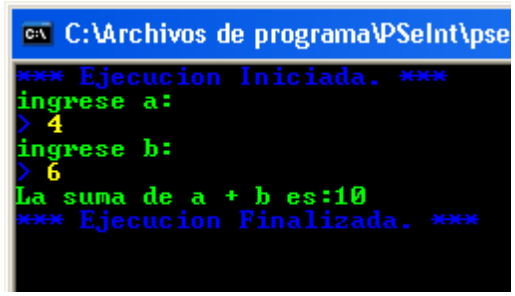
2. Sumar dos números ‘a’ y ‘b’

//Algoritmo para sumar dos números enteros 'a' y 'b' desarrollado por RPC

```
Proceso sumar_a_b //Proceso: ¿qué desea hacer el 'programa'?: sumar a y b
    Escribir "ingrese a:" ; //escribe/muestra en pantalla
    Leer a; //introduce por teclado el valor de 'a'
    Escribir "ingrese b:" ;
    Leer b; //introduce por teclado el valor de 'b'
    Suma<-(a+b); //operador Suma=a+b
    Escribir "La suma de a + b es:",Suma ; //escribe/muestra en pantalla + el valor Suma
FinProceso
```



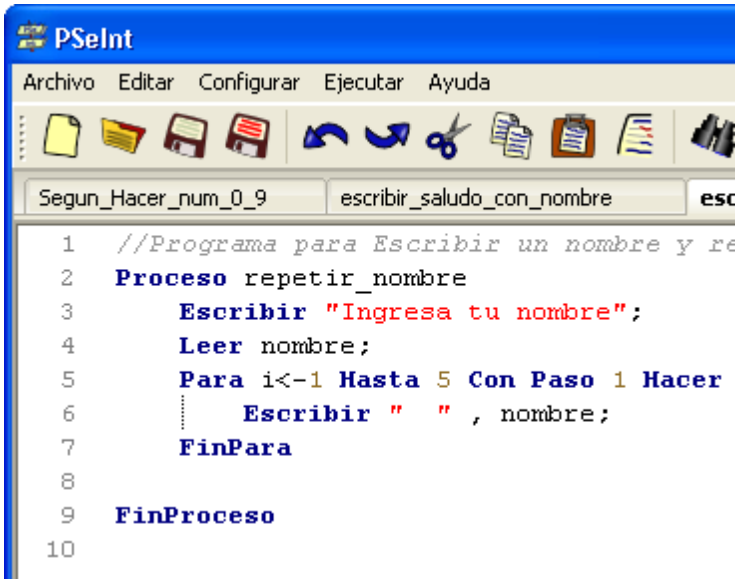


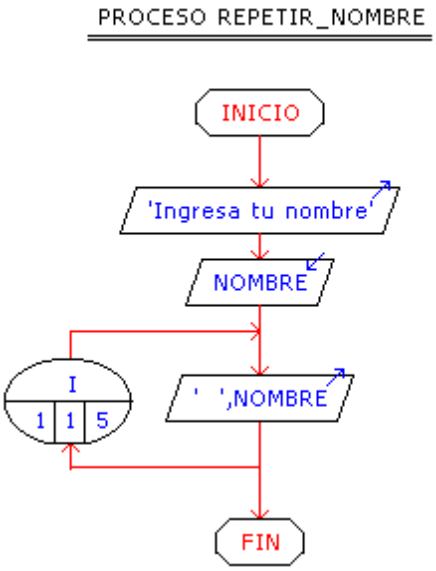


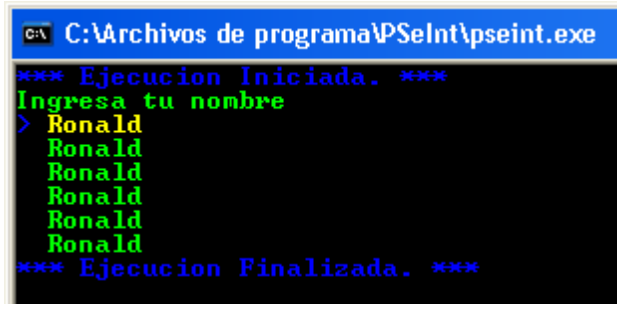
3. Escribir un nombre 5 veces

//Programa para Escribir un nombre y repetir 5 veces: RPC

```
Proceso repetir_nombre
    Escribir "Ingresa tu nombre"; //muestra en teclado: ingresa ti nombre
    leer nombre; //leer/ingresar por teclado el nombre
    Para i<-1 Hasta 5 Con Paso 1 Hacer //para: use la opción del menú de la derecha
        Escribir " " , nombre; // escribe el nombre 5 veces, las comillas le dan espacio
    FinPara //fin del comando "Para"
FinProceso //fin del proceso
```



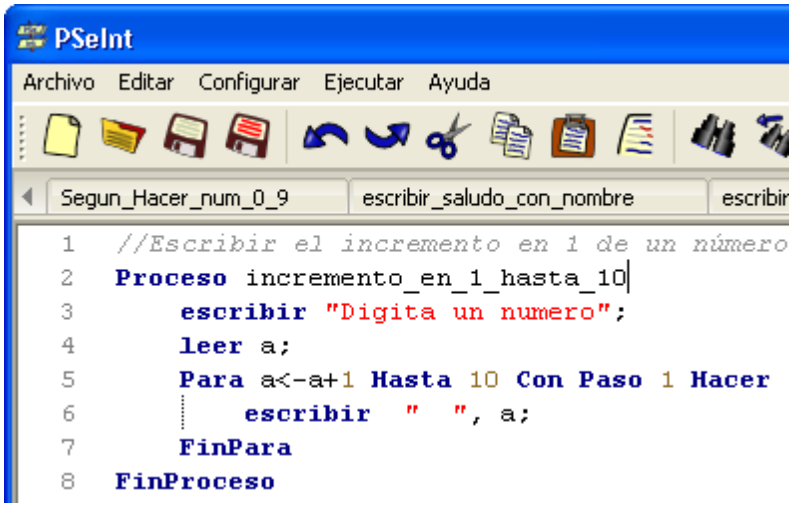


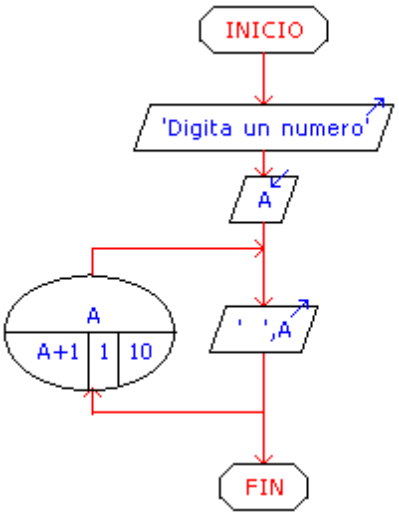


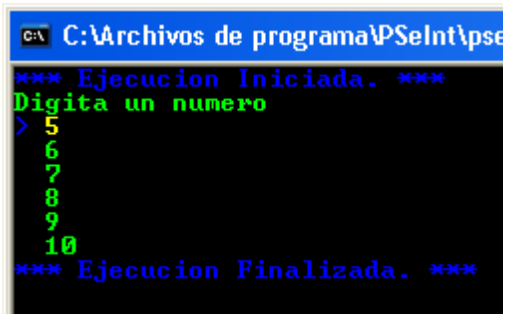
4. Escribir un el incremento en 1 de un nº menor a 10 hasta 10

//Escribir el incremento en 1 de un número menor a 10 hasta 10: RPC

```
Proceso sin_titulo
    escribir "Digita un numero"; //Muestra en pantalla la instrucción
    leer a; //ingresa la variable 'a' (número menor a 10)
    Para a<-a+1 Hasta 10 Con Paso 1 Hacer //Comando Para: está al
    final derecha de este IDE
        escribir " ", a; //El espacio entre comillas ("") solo ajusta el
        texto debajo de la variable ingresada
    FinPara //Fin del comando Para
FinProceso //Fin del proceso
```



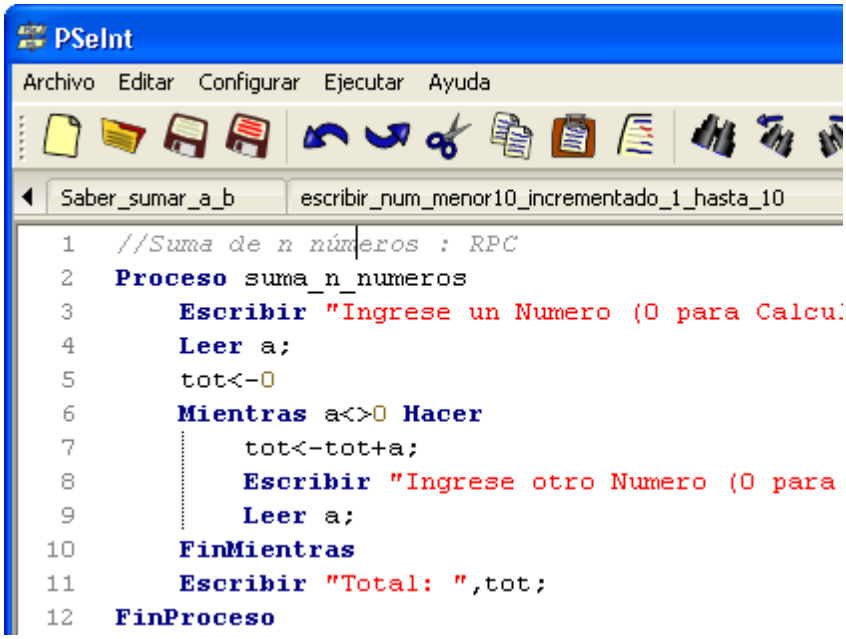


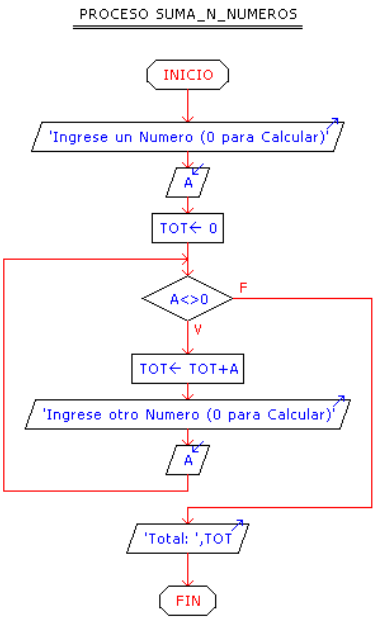


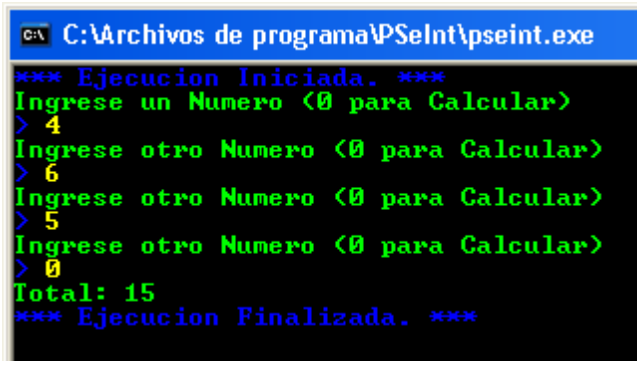
5. Sumar n números utilizando MIENTRAS

```
//Suma de n números : RPC

    Proceso suma_n_numeros
```

```
        Escribir "Ingrese un Número (0 para Calcular)";
        Leer a;
        tot<-0
        Mientras a<>0 Hacer
            tot<-tot+a;
            Escribir "Ingrese otro Numero (0 para Calcular)";
            Leer a;
        FinMientras
        Escribir "Total: ",tot;
FinProceso
```





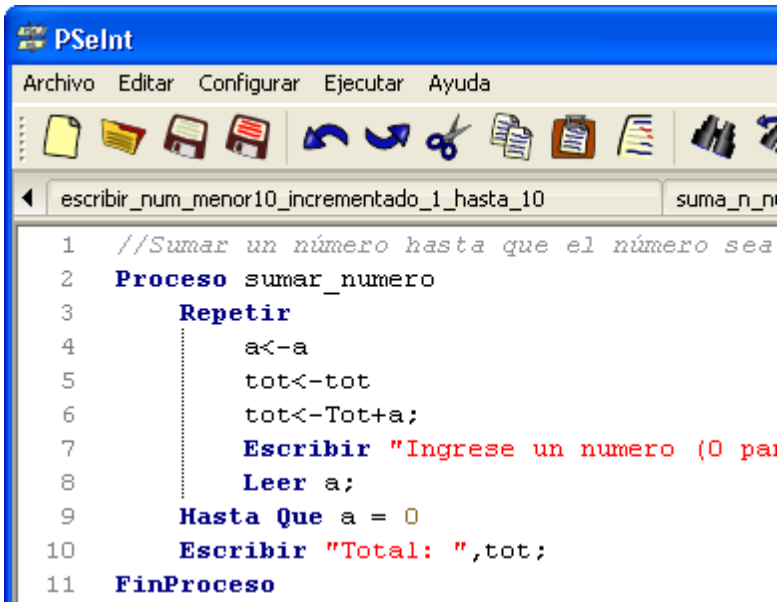


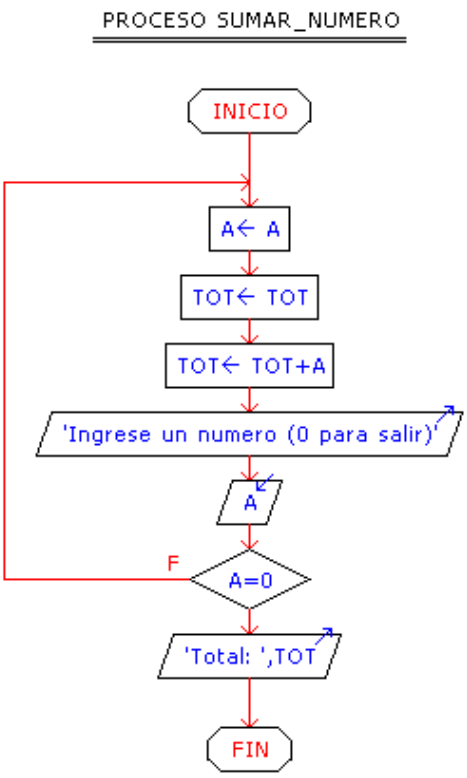
6. Sumar n números utilizando REPETIR

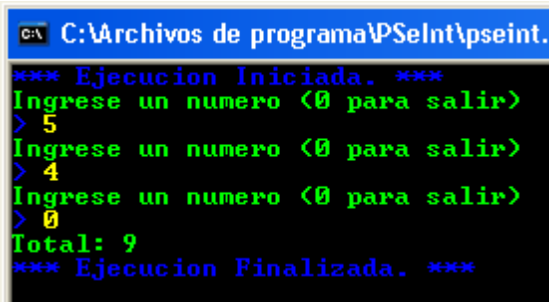
//Sumar un número hasta que el número sea a=0

```
Proceso sumar_numero
    Repetir
```

```
            a<-a
            tot<-tot
            tot<-Tot+a;
            Escribir "Ingrese un número (0 para salir)";
            Leer a;
        Hasta Que a = 0
        Escribir "Total: ",tot;
FinProceso
```





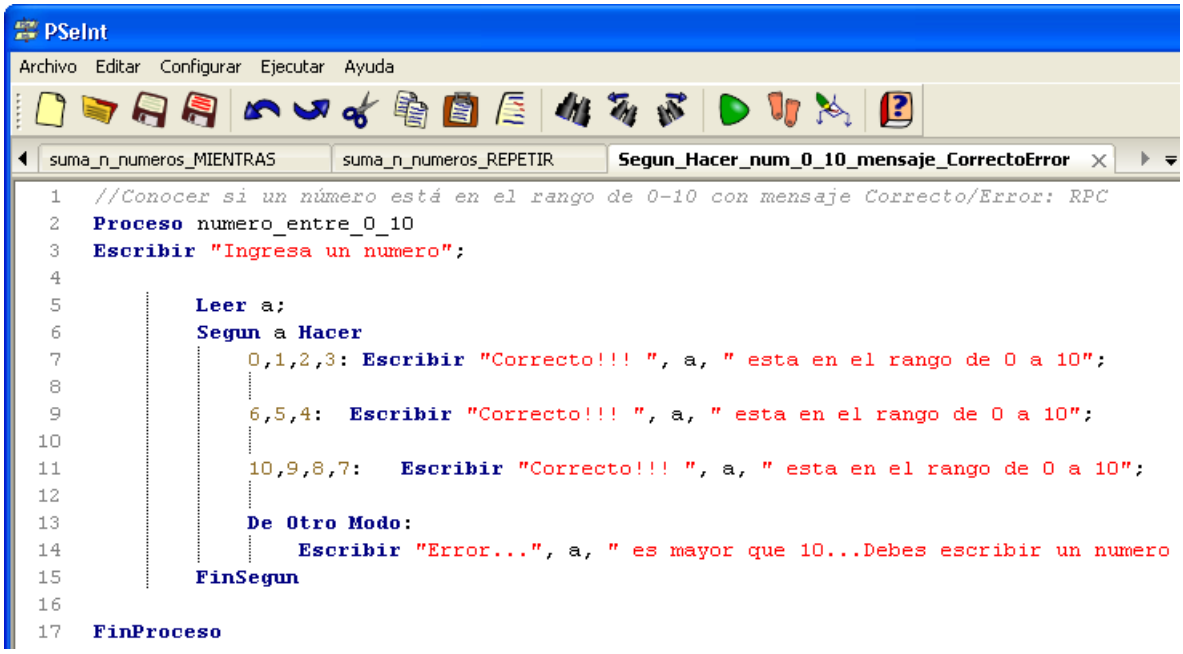


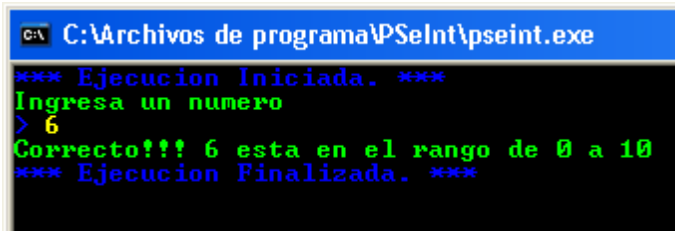
7. Conocer si un número 'n' está en el rango de 0 a 10 con mensaje de Correcto/Error utilizando SEGÚN HACER:

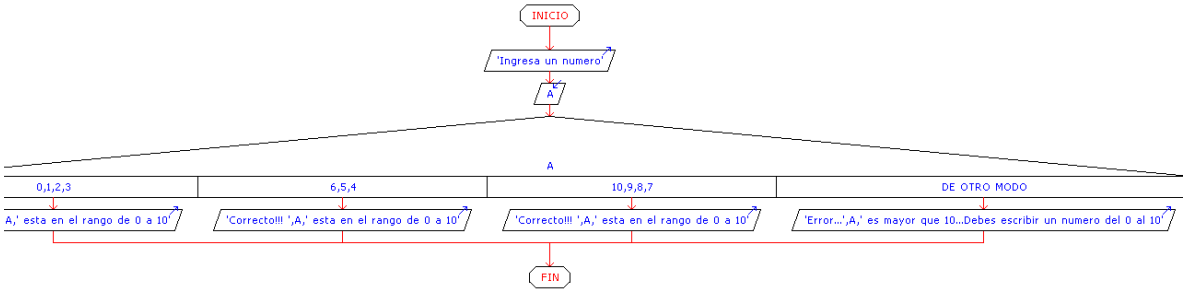
//Conocer si un número está en el rango de 0-10 con mensaje Correcto/Error: RPC

```
    Proceso numero_entre_0_10
    Escribir "Ingresa un numero";
```

```
Leer a;
Segun a Hacer
        0,1,2,3: Escribir "Correcto!!! ", a, " está en el rango de 0 a 10";
        6,5,4: Escribir "Correcto!!! ", a, " está en el rango de 0 a 10";
        10,9,8,7: Escribir "Correcto!!! ", a, " está en el rango de 0 a
10";
        De Otro Modo:
                Escribir "Error...", a, " es mayor que 10...Debes escribir
                un numero del 0 al 10";
        FinSegun
FinProceso
```







8. Calculadora Suma, Resta: Multiplicación y División

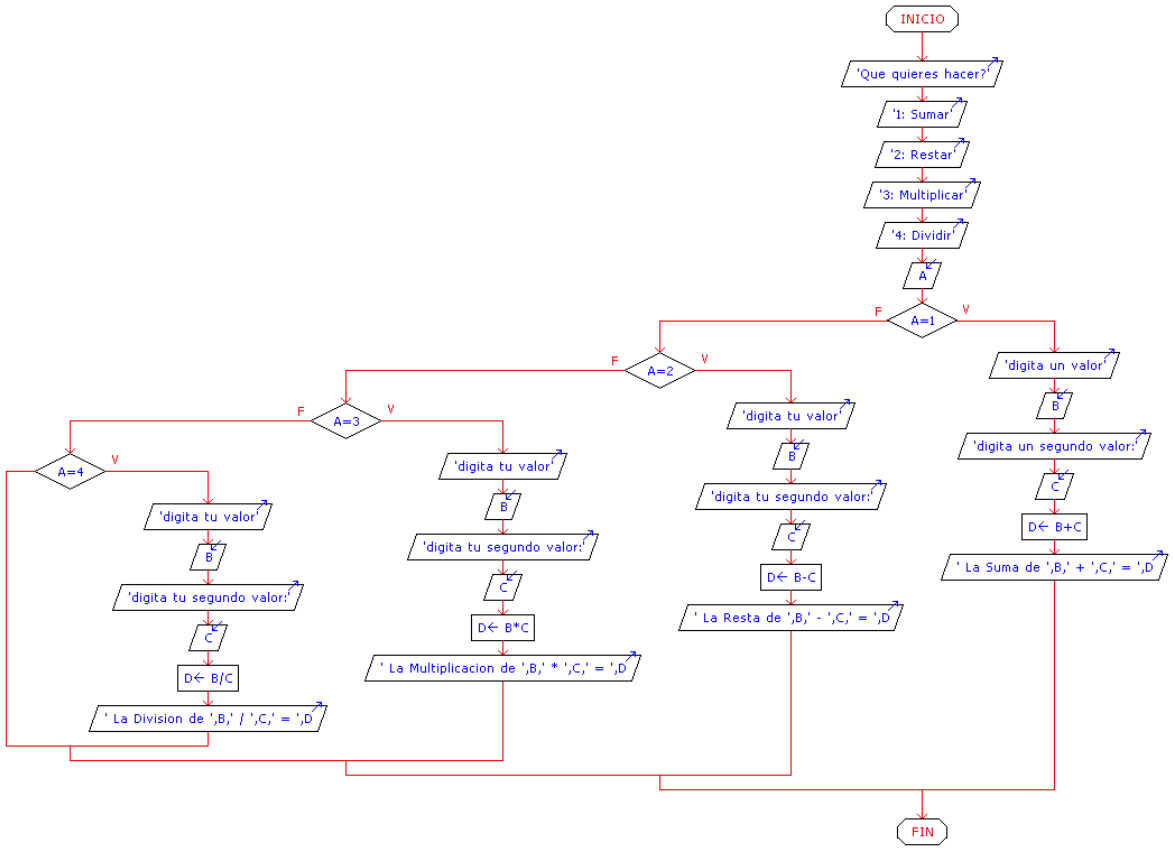
```
//Calculadora Suma, Resta, Multiplicación y División

        Proceso calculadora
                escribir "Que quieres hacer?";
                escribir "1: Sumar";
                escribir "2: Restar";
                escribir "3: Multiplicar";
                escribir "4: Dividir";
                leer a;
                Si a=1 Entonces
                        escribir "digita un valor";
                        leer b;
                        escribir "digita un segundo valor:";
                        leer c
                        d<-b+c;
                        escribir " La Suma de ", b, " + ", c, " = ", d
                        Sino
                        Si a=2 Entonces
                        escribir "digita tu valor";
                        leer b;
                        escribir "digita tu segundo valor:";
                        leer c
                        d<-b-c;
                        escribir " La Resta de " , b, " - ", c, " = ", d
                Sino
                        Si a=3 Entonces
                        escribir "digita tu valor";
                        leer b;
                        escribir "digita tu segundo valor:";
                        leer c
                        d<-b*c;
                        escribir " La Multiplicación de " , b, " * ", c," = " , d
                Sino
                        Si a=4 Entonces
                        escribir "digita tu valor";
                        leer b;
                        escribir "digita tu segundo valor:";
                        leer c
                        d<-b/c;
                        escribir " La División de " , b, " / ", c, " = ", d
                Sino
                FinSi
                FinSi
```

```
        FinSi
    FinSi
FinProceso
```

suma_n_numeros_REPETIR | Segun_Hacer_num_0_10_mensaje_CorrectoError | calculadora_Sum_Res_Mul

```
1  //Calculadora Suma, Resta, Multiplicación y División: RPC
2  Proceso calculadora
3      escribir "Que quieres hacer?";
4      escribir "1: Sumar";
5      escribir "2: Restar";
6      escribir "3: Multiplicar";
7      escribir "4: Dividir";
8      leer a;
9      Si a=1 Entonces
10         escribir "digita un valor";
11         leer b;
12         escribir "digita un segundo valor:";
13         leer c
14         d<-b+c;
15         escribir " La Suma de ", b, " + ", c, " = ", d
16     Sino
17         Si a=2 Entonces
18             escribir "digita tu valor";
19             leer b;
20             escribir "digita tu segundo valor:";
21             leer c
22             d<-b-c;
23             escribir " La Resta de " , b, " - ", c, " = ", d
24         Sino
25             Si a=3 Entonces
26                 escribir "digita tu valor";
27                 leer b;
28                 escribir "digita tu segundo valor:";
29                 leer c
30                 d<-b*c;
31                 escribir " La Multiplicacion de " , b, " * ", c," = " , d
32             Sino
33                 Si a=4 Entonces
34                     escribir "digita tu valor";
35                     leer b;
36                     escribir "digita tu segundo valor:";
37                     leer c
38                     d<-b/c;
39                     escribir " La Division de " , b, " / ", c, " = ", d
40                 Sino
41                 FinSi
42             FinSi
43         FinSi
44     FinSi
45 FinProceso
```

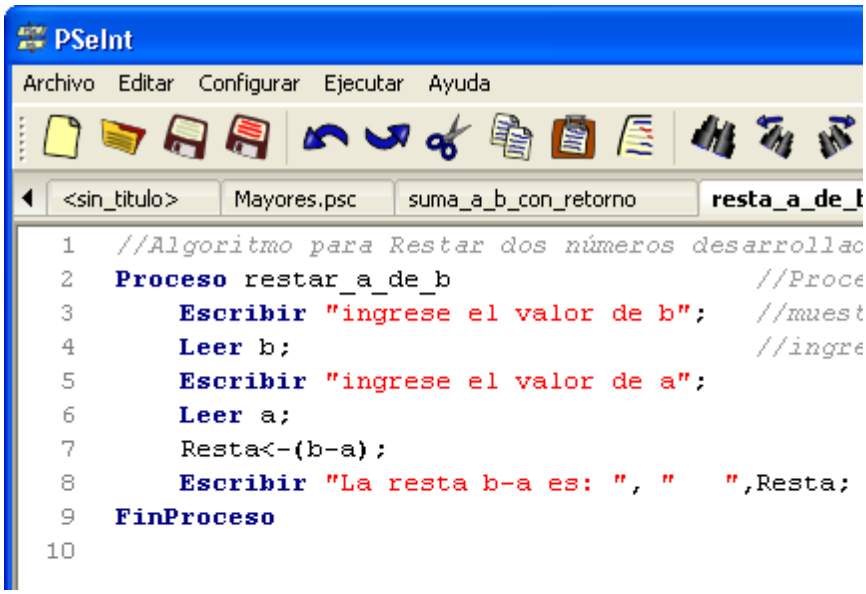
```
C:\Archivos de programa\PSeInt\pseint.exe
*** Ejecucion Iniciada. ***
Que quieres hacer?
1: Sumar
2: Restar
3: Multiplicar
4: Dividir
> 3
digita tu valor
> 25
digita tu segundo valor:
> 5
 La Multiplicacion de 25 * 5 = 125
*** Ejecucion Finalizada. ***
```

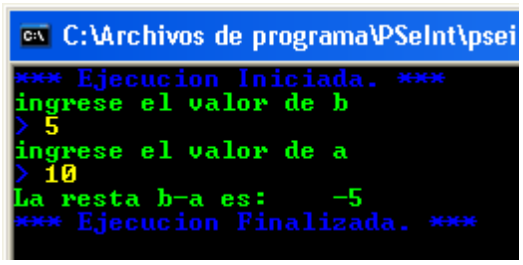


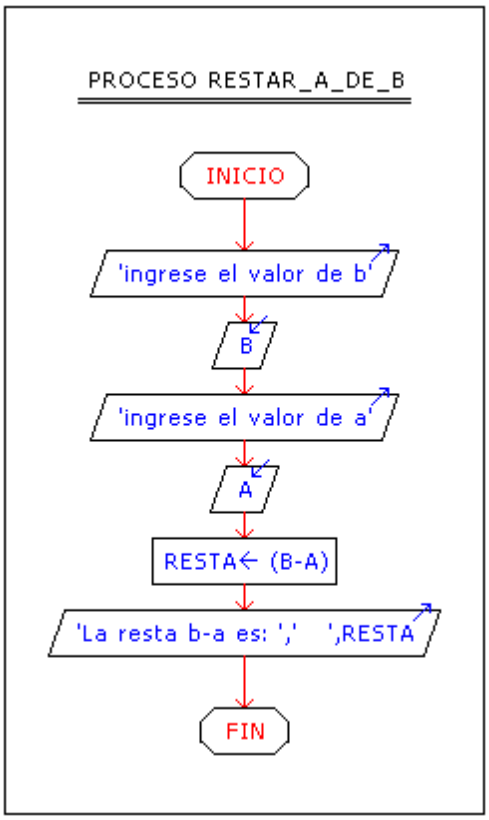
9. Restar a de b

//Algoritmo para Restar dos números desarrollado por RPC

Proceso restar_a_de_b //Proceso: Restar a de b; note que no hay espacios: restar_a_de_b
Escribir "ingrese el valor de b"; //muestra en pantalla la instrucción de ingresar el valor de 'b'
Leer b; //ingresa por teclado el valor de 'b'
Escribir "ingrese el valor de a";
Leer a;
Resta<-(b-a);
Escribir "La resta b-a es: ", " ",Resta; // note que existe un espacio: " ",Resta; la variable "Resta" es el valor de b-a
FinProceso // fin del proceso





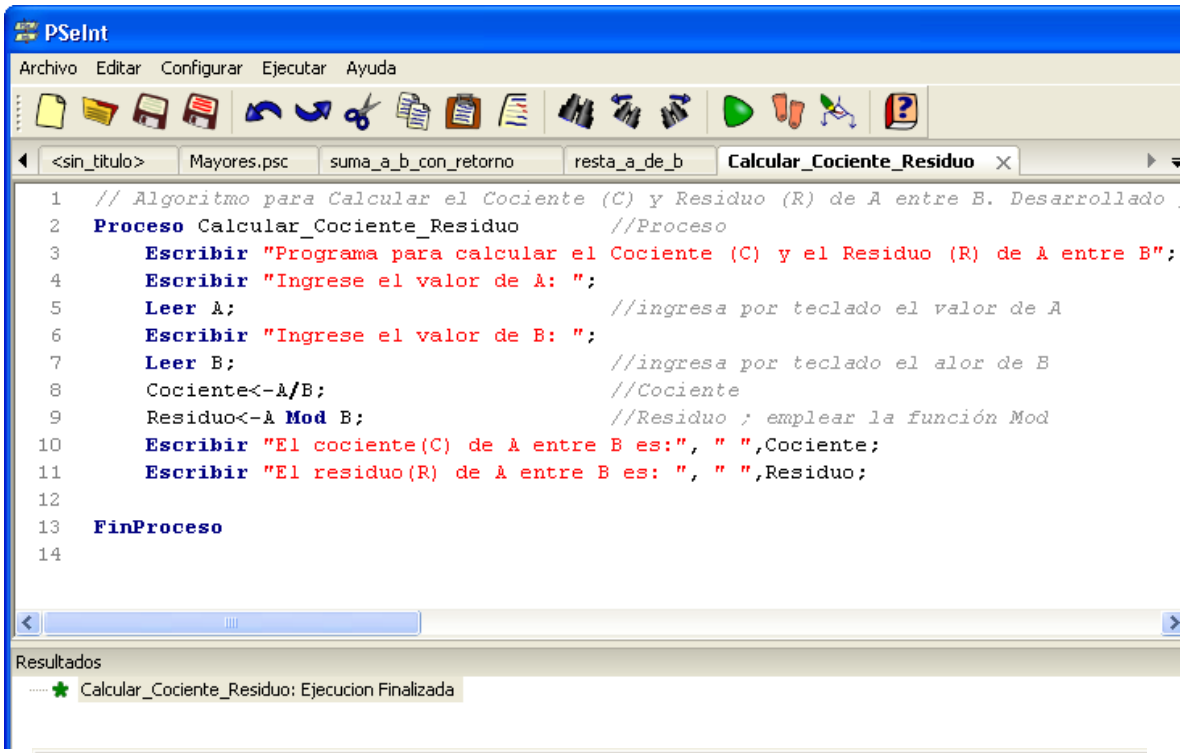


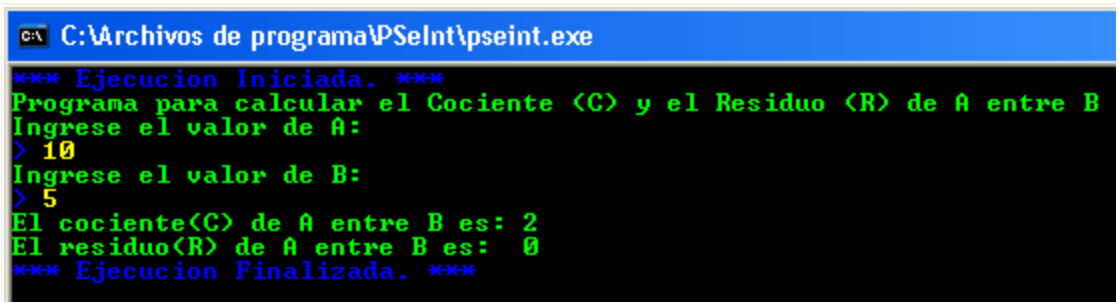
10. Calcular el cociente y residuo de la división de dos números A y B

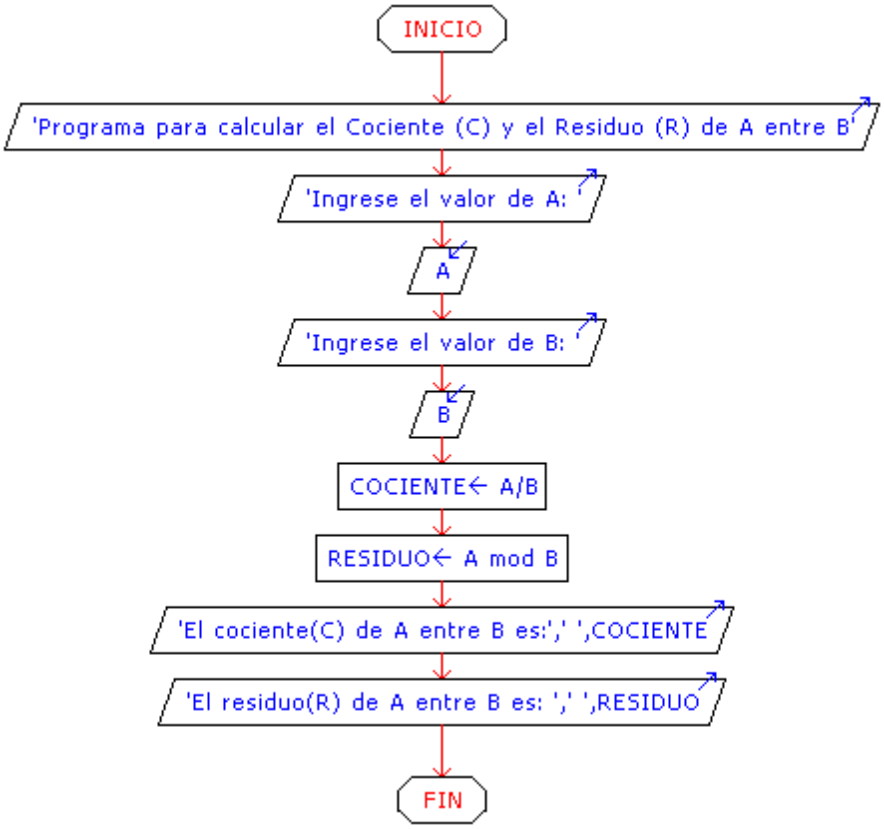
// Algoritmo para Calcular el Cociente (C) y Residuo (R) de A entre B. Desarrollado por RPC

Proceso Calcular_Cociente_Residuo //Proceso
Escribir "Programa para calcular el Cociente (C) y el Residuo (R) de A entre B";

```
        Escribir "Ingrese el valor de A: ";
        Leer A; //ingresa por teclado el valor de A
        Escribir "Ingrese el valor de B: ";
        Leer B; //ingresa por teclado el valor de B
        Cociente<-A/B; //Cociente
        Residuo<-A Mod B; //Residuo ; emplear la función Mod
        Escribir "El cociente(C) de A entre B es:", " ",Cociente;
        Escribir "El residuo(R) de A entre B es: ", " ",Residuo;
FinProceso
```







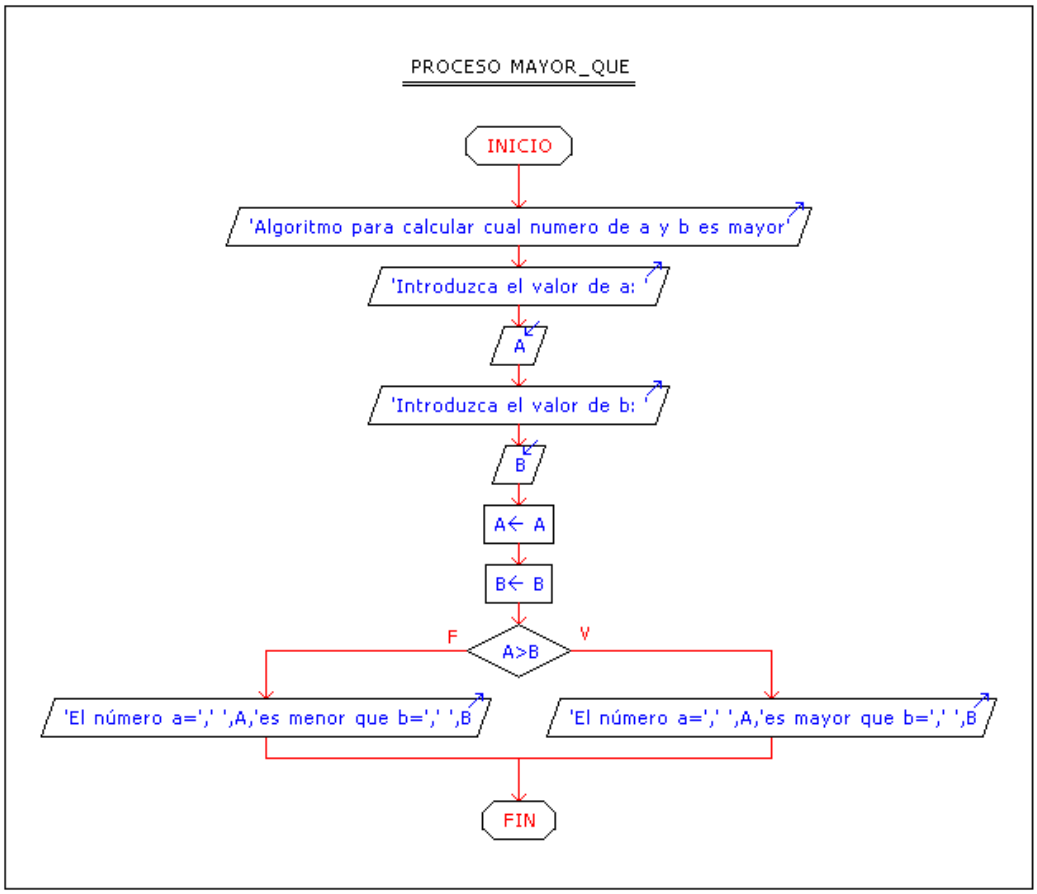
11. Determinar el mayor de dos números ‘a’ y ‘b’

//Algoritmo que determina el mayor de dos números 'a' y 'b'. Desarrollado por RPC

```
Proceso mayor_que //proceso mayor_que
        Escribir "Algoritmo para calcular cual numero de a y b es mayor";
        Escribir "Introduzca el valor de a: " //muestra en pantalla la
instrucción
        Leer a; //ingresa por teclado el valor de 'a'
        Escribir "Introduzca el valor de b: "
        Leer b;
        a<-a; // a=a; si escribieramos a=0, la comparación sería entre ceros
(error)
        b<-b; // idem al anterior
        Si a>b Entonces //Condicional Si (If) a>b Entonces que?
                Escribir "El número a=", " ", a, "es mayor que b=", " ", b;
        Sino
                Escribir "El número a=", " ", a, "es menor que b=", " ", b;
        FinSi //Fin de la condicional
FinProceso //Fin del proceso
```

```
1  //Algoritmo que determina el mayor de dos números 'a' y 'b'. Desarrollado por RPC
2  Proceso mayor_que            //proceso mayor_que
3      Escribir "Algoritmo para calcular cual numero de a y b es mayor";
4      Escribir "Introduzca el valor de a: "       //muestra en pantalla la instrucción
5      Leer a;                  //ingresa por teclado el valor de 'a'
6      Escribir "Introduzca el valor de b: "
7      Leer b;
8      a<-a;                    // a=a; si escribieramos a=0, la comparación sería entre
9      b<-b;                    // idem al anterior
10     Si a>b Entonces          //Condicional Si (If) a>b Entonces que?
11         Escribir "El número a=", " ", a, "es mayor que b=", " ", b;
12     Sino
13         Escribir "El número a=", " ", a, "es menor que b=", " ", b;
14     FinSi                    //Fin de la condicional
15
16 FinProceso                   //Fin del proceso
```

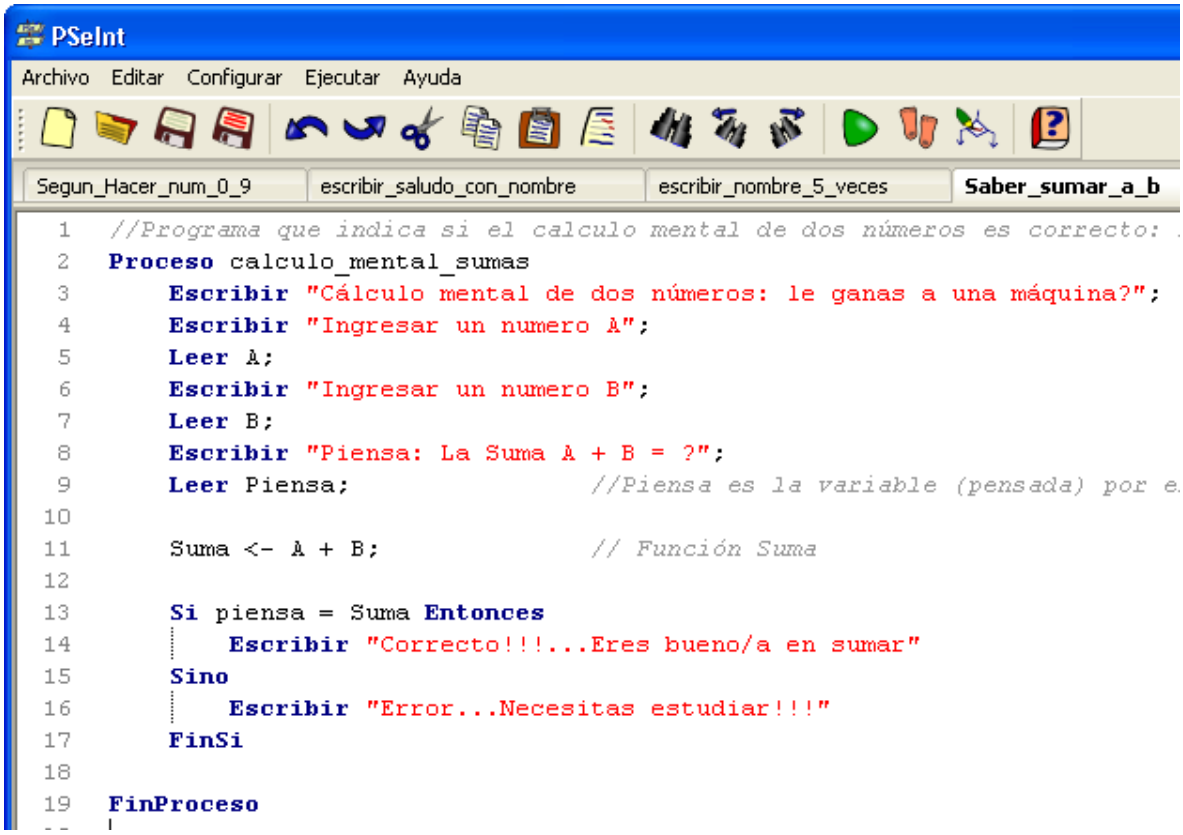
```
C:\Archivos de programa\PSeInt\pseint.exe
*** Ejecucion Iniciada. ***
Programa para calcular el Cociente (C) y el Residuo (R) de A entre B
Ingrese el valor de A:
> 10
Ingrese el valor de B:
> 5
El cociente(C) de A entre B es: 2
El residuo(R) de A entre B es:  0
*** Ejecucion Finalizada. ***
```



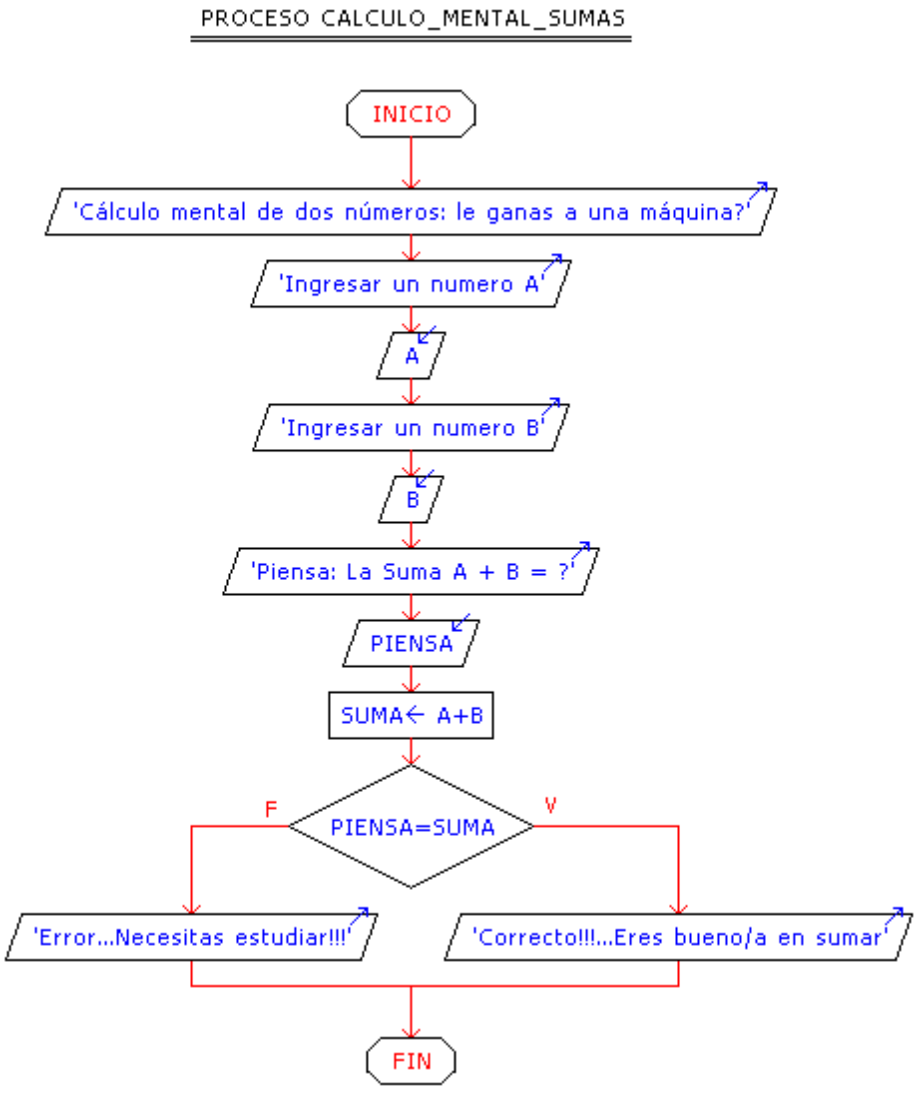
12. Cálculo mental de dos números: le ganas a una máquina?"

//Programa que indica si el cálculo mental de dos números es correcto: RPC

```
Proceso cálculo_mental_sumas
        Escribir "Cálculo mental de dos números: le ganas a una máquina?";
        Escribir "Ingresar un numero A";
        Leer A;
        Escribir "Ingresar un numero B";
        Leer B;
        Escribir "Piensa: La Suma A + B = ?";
        Leer Piensa; //Piensa es la variable (pensada) por el usuario
        Suma <- A + B; // Función Suma
        Si piensa = Suma Entonces
                Escribir "Correcto!!!...Eres bueno/a en sumar"
        Sino
                Escribir "Error...Necesitas estudiar!!!"
        FinSi
FinProceso
```



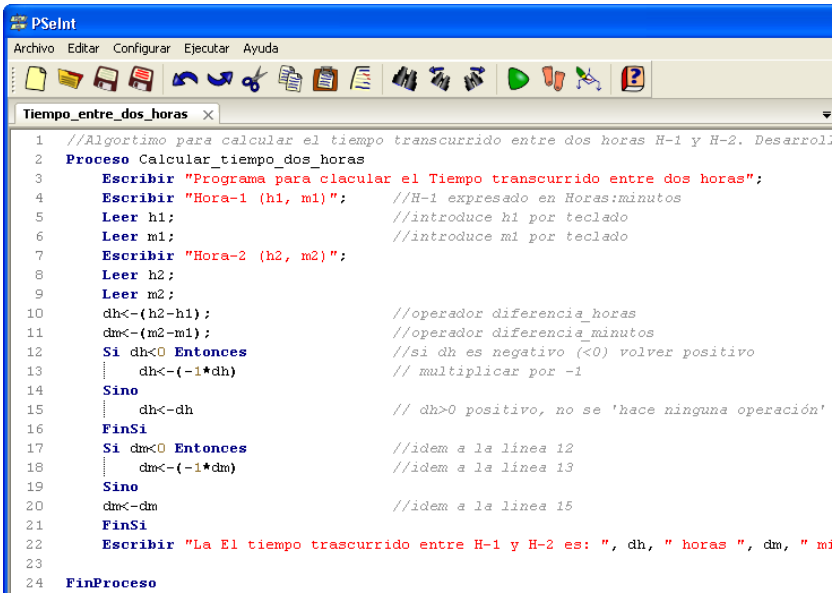
```
C:\Archivos de programa\PSeInt\pseint.exe
*** Ejecucion Iniciada. ***
CΒlculo mental de dos n·meros: le ganas a una mΒquina?
Ingresar un numero A
> 4
Ingresar un numero B
> 56
Piensa: La Suma A + B = ?
> 60
Correcto!!!...Eres bueno/a en sumar
*** Ejecucion Finalizada. ***
```



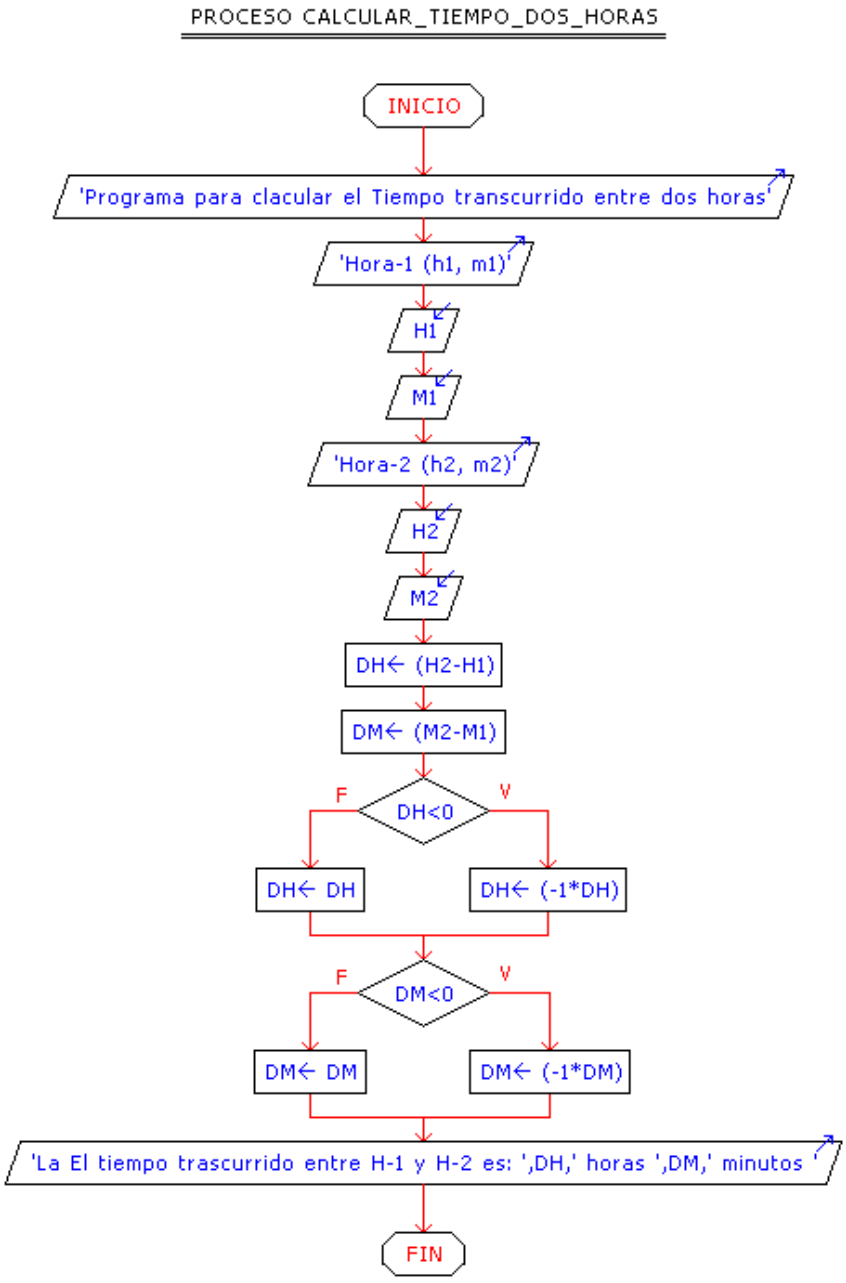
13. Determinar el tiempo transcurrido entre dos horas del día.

//Algortimo para calcular el tiempo transcurrido entre dos horas H-1 y H-2.
Desarrollado por RPC

```
Proceso Calcular_tiempo_dos_horas
    Escribir "Programa para clacular el Tiempo transcurrido entre dos
horas";
    Escribir "Hora-1 (h1, m1)"; //H-1 expresado en Horas:minutos
    Leer h1; //introduce h1 por teclado
    Leer m1; //introduce m1 por teclado
    Escribir "Hora-2 (h2, m2)";
    Leer h2;
    Leer m2;
    dh<-(h2-h1); //operador diferencia_horas
    dm<-(m2-m1); //operador diferencia_minutos
    Si dh<0 Entonces //si dh es negativo (<0) volver positivo
        dh<-(-1*dh) // multiplicar por -1
    Sino
        dh<-dh // dh>0 positivo, no se 'hace ninguna operación'
    FinSi
    Si dm<0 Entonces //idem a la línea 12
        dm<-(-1*dm) //idem a la línea 13
    Sino
        dm<-dm //idem a la línea 15
    FinSi
    Escribir "La El tiempo trascurrido entre H-1 y H-2 es: ", dh, " horas ",
    dm, " minutos ";
FinProceso
```



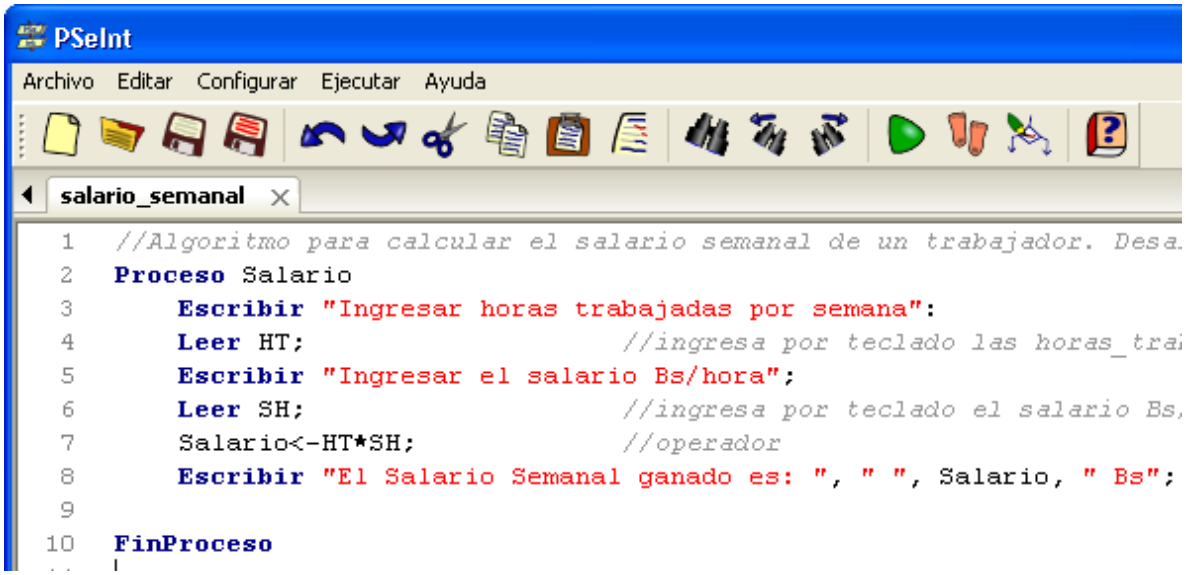
```
C:\Archivos de programa\PSeInt\pseint.exe
*** Ejecucion Iniciada. ***
Programa para clacular el Tiempo transcurrido entre dos horas
Hora-1 (h1, m1)
> 24
> 15
Hora-2 (h2, m2)
> 17
> 22
La El tiempo trascurrido entre H-1 y H-2 es: 7 horas 7 minutos
*** Ejecucion Finalizada. ***
```

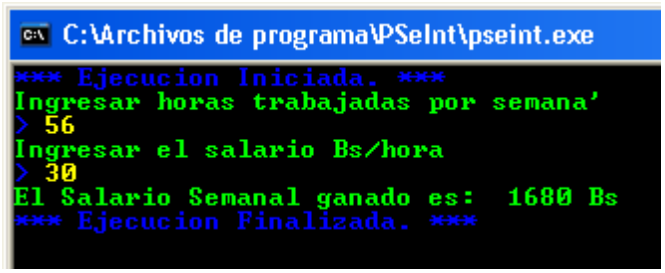


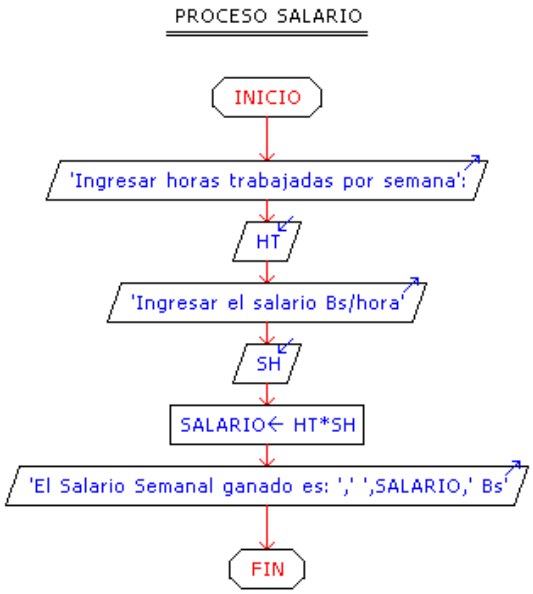
14. Calcular el salario semanal de un empleado

//Algoritmo para calcular el salario semanal de un trabajador. Desarrollado por RPC

```
Proceso Salario
    Escribir "Ingresar horas trabajadas por semana":
    Leer HT; //ingresa por teclado las horas_trabajadas_semana
    Escribir "Ingresar el salario Bs/hora";
    Leer SH; //ingresa por teclado el salario Bs/hora
    Salario<-HT*SH; //operador
    Escribir "El Salario Semanal ganado es: ", " ", Salario, " Bs";
FinProceso
```

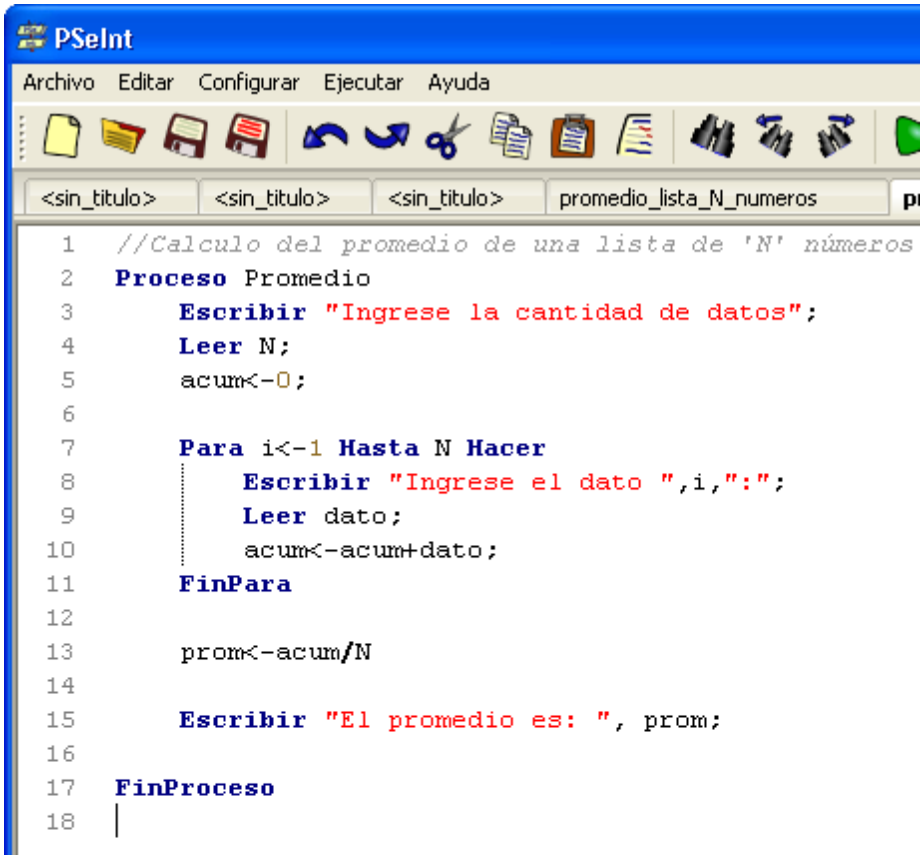






15. Cálculo del promedio de N números

```
//Calculo del promedio de una lista de 'N' números

    Proceso Promedio
        Escribir "Ingrese la cantidad de datos";
        Leer N;
        acum<-0;

        Para i<-1 Hasta N Hacer
            Escribir "Ingrese el dato ",i,":";
            Leer dato;
            acum<-acum+dato;
        FinPara

        prom<-acum/N

        Escribir "El promedio es: ", prom;

    FinProceso
```



```
C:\Archivos de programa\PSeInt\pse
*** Ejecucion Iniciada. ***
Ingrese la cantidad de datos
> 5
Ingrese el dato 1:
> 1
Ingrese el dato 2:
> 5
Ingrese el dato 3:
> 4
Ingrese el dato 4:
> 7
Ingrese el dato 5:
> 1
El promedio es: 3.6
*** Ejecucion Finalizada. ***
```