

Instituto Tecnológico de Sonora

Examen de Algoritmos

para todos los aspirantes a ingresar al programa educativo de
Ingeniero de Software en agosto de 2017

Como parte del perfil de Ingreso del Programa Educativo de Ingeniería en Software Plan 2016 se ha determinado provechoso para la trayectoria académica futura de nuestros aspirantes el que demuestren capacidad de identificar y resolver problemas mediante algoritmos computacionales. Por lo anterior se deberá presentar un Examen de Conocimientos sobre Algoritmos.

Para prepararse para el examen se proporcionará una guía, además de material de apoyo, que se encuentran publicados en el microsítio del Programa Educativo de Ingeniería de Software (www.itson.mx/isw). El resultado del examen se informará la semana posterior a su aplicación.

El Examen de Conocimientos sobre Algoritmos tendrá una duración de tres horas y se aplicará a los aspirantes a ingresar al programa de Ingeniería en Software en las siguientes fechas:

Fechas y lugares del Examen de Algoritmos

	Obregón	Navojoa	Guaymas
Fecha 1	Viernes 31 de marzo 16:00 a 19:00 horas CISCO LV 1013	Sábado 22 de abril 9:00 a 13:00 horas CIT Aula 2	Sábado 22 de abril 9:00 horas Edificio CITEV Aula 923
Fecha 2	Viernes 28 de abril 16:00 a 19:00 horas CISCO LV 1013	Sábado 20 de mayo 9:00 a 13:00 horas CIT Aula 2	Sábado 20 de mayo 9:00 horas Edificio CITEV Aula 923
Fecha 3	Viernes 26 de mayo 16:00 a 19:00 horas CISCO LV 1013	Sábado 17 de junio 9:00 a 13:00 horas CIT Aula 2	Sábado 17 de junio 9:00 horas Edificio CITEV Aula 923
Fecha 4	Viernes 23 de junio 16:00 a 19:00 horas CISCO LV 1013	-	-

Los aspirantes que no acrediten el examen podrán inscribirse en el Curso de Fundamentos de Algoritmos Computacionales, el cual tiene una duración de 24 horas, mismo que los preparará de forma suficiente para tomar el curso de Programación I. Este curso no tiene costo de inversión y el material requerido es cuaderno para notas, lápiz y borrador. El Curso de Fundamentos de Algoritmos Computacionales para alumnos a ingresar en agosto de 2017 se impartirá en la semana previa al inicio de cursos.

Curso de Fundamentos de Algoritmos Computacionales

	Obregón	Navojoa	Guaymas
Curso	Curso propedéutico para quienes no aprueben el Examen de Conocimientos sobre Algoritmos 21 a 25 de agosto	-	-

Más información:

Mtra. Martha Eloisa Larrínaga Hernández
Responsable de Programa Obregón
mlarrinaga@itson.edu.mx

Mtro. Alonso Gómez Ávila
Responsable de Programa Guaymas
alonso.gomez@itson.edu.mx

Mtro. José de Jesús Soto Padilla
Responsable de Programa Navojoa
jose.soto@itson.edu.mx



ITSON

Educar para
Trascender

**Examen de Conocimientos sobre Algoritmos para el
Ingreso a la Licenciatura en Ingeniería en Software**

GUÍA PARA EL SUSTENTANTE

Marzo de 2016

Presentación

Como parte del perfil de Ingreso del Programa Educativo de Ingeniería en Software Plan 2016 el aspirante debe demostrar *capacidad de identificar y resolver problemas mediante algoritmos computacionales*. Por lo anterior, como requisito de admisión se deberá acreditar un **Examen de Conocimientos sobre Algoritmos**.

¿Qué evalúa el Examen de Conocimientos sobre Algoritmos?

El examen está organizado de forma que pueda evaluarse el nivel de conocimientos en tres dimensiones: Secuencial, Condicional, Repetición. Entendiendo cada una de éstas:

- **Secuencial:** Diseñar un algoritmo secuencial para resolver un problema planteado a través del modelo de entrada, proceso y salida (modelo de solución). Comprendiéndose en este punto que un algoritmo secuencial es aquel que únicamente se incluye instrucciones que se ejecutan una detrás de otra, una sola vez.
- **Condicional:** Diseñar un algoritmo condicional para resolver un problema planteado a través del modelo de entrada, proceso y salida (modelo de solución). Comprendiéndose en este punto que un algoritmo condicional es aquel que incluyen instrucciones que se ejecutan o no de acuerdo a una toma de decisión (camino alternativo), una sola vez.
- **Repetición:** Diseñar un algoritmo de repetición para resolver un problema planteado a través del modelo de entrada, proceso y salida (modelo de solución). Comprendiéndose en este punto que un algoritmo de repetición es aquel que incluye instrucciones que se ejecutan de forma repetida de acuerdo a una toma de decisión.

Las preguntas o reactivos del Examen de Conocimientos de Algoritmos están diseñadas para medir esas dimensiones en tres niveles:

1. **Nivel Básico:** Leer un pseudocódigo, identificar elementos.
2. **Nivel Aceptable:** Plantear un pseudocódigo a partir de un problema simple.
3. **Nivel Sobresaliente:** Resolver problemas con "supuestos" y combinación de estructuras".

¿Qué temas son incluidos en las preguntas del Examen?

Los temas identifican los conocimientos fundamentales en el dominio de Algoritmos:

1. Estructuras secuenciales.
2. Estructuras condicionales.
 - a. Simples
 - b. Dobles
3. Estructuras condicionales.
 - a. Simples
 - b. Anidadas
4. Estructuras de repetición.
 - a. Simples
 - b. Anidadas
5. Identificación de datos de entrada, proceso y salida (Modelo solución).
6. Identificación de variables y constantes.
7. Operaciones básicas.
 - a. Suma
 - b. Multiplicación
 - c. División
 - d. Módulo o residuo
8. Operadores.
 - a. Aritméticos
 - b. Relacionales
 - c. Lógicos
9. Precedencia de operaciones aritméticas.
10. Concepto de asignación de valores.
11. Expresiones Lógicas y Relacionales.
 - a. Aritméticas

- b. Lógicas
 - c. Relacionales
12. Conversiones de fórmulas algebraicas a expresiones aritméticas.
 13. Lectura de un algoritmo (Seguimiento del código para predecir un resultado).
 14. Uso de instrucciones de entrada y salida.
 15. Concepto y uso de constantes.
 16. Concepto y uso de variables.
 17. Distinción entre variables y etiquetas.
 18. Identificar tipo de estructura adecuada para la resolución del problema.
 19. Identificar cuando dos estructuras son equivalentes (generan el mismo resultado).
 20. Representación de rango en las estructuras condicionales y de repetición.
 21. Comparación entre un programa secuencial y uno de repetición (que sean equivalentes).
 22. Verificar elementos de una estructura de repetición (mientras).
 23. Comprobar si un ciclo es infinito o no (condiciones de salida).
 24. Concepto de contador y acumulador (inicialización de los ciclos).
 25. Ámbito del ciclo (determinar que va dentro o fuera del ciclo).
 26. Elegir entre estructuras condicionales o de repetición.
 27. Uso combinado de estructuras. Ciclos definidos e indefinidos (dependen de un parámetro para finalizar, generalmente no dependen de un contador y no es posible determinar cuántas iteraciones se realizarán previo a la ejecución del algoritmo).
 28. Elección de valores de prueba para comprobar ciclos.
 29. Ubicación de las expresiones aritméticas (sumatorias, promedios etc.).
 30. Acumuladores y contadores en estructuras anidadas.

Formato del Examen

El Examen de Conocimientos sobre Algoritmos es en línea y de opción múltiple, dividido en tres secciones: Estructura Secuencial, Estructura Condicional y Estructura de Repetición.

Aunque el examen completo tiene una duración máxima de tres horas, cada sección individual también tiene la misma duración, por lo que te recomendamos organizar el tiempo de cada sección de tal forma que puedas completarlas todas en las 3 horas indicadas. Una vez iniciada una sección, no podrás abandonarla hasta terminar la sección por completo.

En esta modalidad de examen, por cada sección podrás:

- Revisar las preguntas (reactivos) del examen en la pantalla de una computadora.
- Responder los reactivos seleccionando la opción correcta con el ratón (mouse) de la computadora.
- Regresar a reactivos ya resueltos o que no respondiste inicialmente para responderlos de nueva cuenta.

En caso de que requiera hacer algún cálculo, el aplicador le proporcionará hojas para dicho fin. Al finalizar la sesión de examen las deberás regresar al aplicador y no podrás sustraerlas del espacio asignado para la aplicación.

¿Qué tipo de preguntas se incluyen?

En el examen se utilizan reactivos o preguntas de opción múltiple, que contienen dos elementos:

- **Una pregunta, afirmación, enunciado o gráfico**, acompañado de una instrucción que plantea un problema explícitamente.
- **Opciones de respuesta**, que son enunciados, palabras, cifras o combinaciones de números y letras que guardan relación con la base del reactivo, donde sólo una opción es la correcta. El orden de las respuestas de cada reactivo es aleatorio, y varía por cada participante.

Durante el examen encontrarás diferentes formas de preguntar. En algunos casos se hace una pregunta directa, en otros se pide completar una información, algunos le solicitan

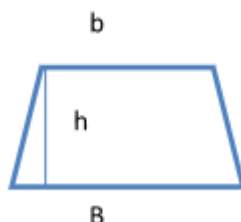
elegir un orden determinado, otros requieren la elección de elementos de una lista dada y otros más le piden relacionar columnas.

Comprender estos formatos te permitirá llegar mejor preparado al examen. Con el fin de apoyarte en ese objetivo, a continuación se presentan algunos ejemplos.

1. Considerando el Siguiete Problema:

El perímetro de un trapezio es igual a la suma de todos sus lados y su área es igual a la suma de la base mayor (B) más la base menor (b) dividido todo entre 2 para después multiplicar este resultado por la altura (h).

Considerando el trapezio de la siguiente figura y de acuerdo al Modelo de Solución, identifique los elementos de Entrada, Proceso y Salida Necesarios para resolver el problema del cálculo del área del Trapecio.



Opciones de Respuesta:

a) Entrada: B,b,h Proceso: $A=b+B/2*h$ Salida: Valor de A	Incorrecto: La expresión $b+B$ debería estar encerrada entre paréntesis, de otra forma se evalúa primero la operación $B/2$, luego se multiplica por h, y al final se le suma b.
b) Entrada: B,b,h Proceso: $A=b+(B/2)*h$ Salida: Valor de A	Incorrecto: La expresión $b+B$ debería estar encerrada entre paréntesis, de otra forma se evalúa primero la operación $B/2$, luego se multiplica por h, y al final se le suma b.
c) Entrada: B,b,h Proceso: $A=(b+B)/2*h$ Salida: Valor de A	Correcto: El orden de evaluación se presenta tal y como se establece en el enunciado que define la mecánica para el cálculo del área de un trapecio.
d) Entrada: B,b,h Proceso: $A=b+B/(2*h)$	Incorrecto: En este caso, primero se evalúa la expresión $2*h$, enseguida se divide el valor de B entre el resultado anterior, y por último se le suma el valor de b.

Salida: Valor de A	
<p>2. Trabajadores de cierta empresa reciben un bono de fin de año adicional a su salario de acuerdo a su antigüedad en la empresa. Si tiene menos de 1 año de trabajo, recibe 40 % de su salario como bono; si ha trabajado entre 1 y 10 años recibe el 70% de su salario, y si tiene más de 10 años, recibe un 90 % de su salario como bono.</p> <p>Una solución al problema es la siguiente :</p> <pre> Leer salario, meses si meses < 12 Entonces bono = salario * 0.40 sino si meses <= 120 Entonces bono = salario * 0.70 sino bono = salario * 0.90 Fin Si Fin Si Escribir "Bono a pagar al empleado ... ", bono </pre> <p>A partir del algoritmo mostrado, seleccione un arreglo de estructuras condicionales equivalente, que igual solucione el problema:</p>	
<p>a) Si meses < 12 Entonces Bono = salario * 0.40 Fin si Si meses >= 12 Entonces Bono = salario * 0.70 Fin si Si meses > 120 Entonces Bono = salario * 0.90 Fin si</p>	<p>Correcta: Aun cuando aparentemente no funciona, sí resuelve el problema, ya que en el caso de una antigüedad > 120, simplemente calcula dos veces el monto, una vez para la condición meses >= 12 y luego otra vez para la condición meses > 120</p>
<p>b) Si meses < 12 Entonces Bono = salario * 0.40 Fin si Si meses >= 12 y meses <= 120 Entonces Bono = salario * 0.70</p>	<p>Incorrecta: Cuando los meses son menores a 12, se calcula un bono a un porcentaje del 40 %, pero en la siguiente condición al evaluar a falso, se calcula el bono a un monto de 90%.</p>

<p>Sino Bono = salario * 0.90 Fin si</p>	
<p>3. Utilizando las siguientes instrucciones, construya un algoritmo que permita calcular la suma de los números pares desde 0 hasta antes de un número entero positivo proporcionado por el usuario:</p> <p>a) $i = i + 2$ b) Mientras($i < \text{numero}$) c) $i = 0$ d) Leer número e) Escribir i f) Fin mientras</p>	
<p>a) d, e, a, b, f, c</p>	<p>Incorrecta: d) Leer número e) Escribir i a) $i = i + 2$ b) Mientras($i < \text{numero}$) f) Fin mientras c) $i = 0$</p>
<p>b) d, c, b, e, a, f</p>	<p>Correcta: d) Leer número c) $i = 0$ b) Mientras($i < \text{numero}$) e) Escribir i a) $i = i + 2$ f) Fin mientras</p>
<p>c) d, c, a, b, e, f</p>	<p>Incorrecta: d) Leer número c) $i = 0$ a) $i = i + 2$ b) Mientras($i < \text{numero}$) e) Escribir i f) Fin mientras</p>

Consejos para sentar el Examen

- Procura visitar o ubicar con anticipación el lugar donde se llevará a cabo el examen, identificar las vías de acceso y los medios de transporte que garanticen la llegada a tiempo.
- Descansa bien la víspera del examen.
- Ingiere alimentos saludables y suficientes. No olvide sus medicamentos en caso de tener que tomarlos.
- Asegúrate de llevar:
 - Tú número de CURP (Cuenta Única de Registro de Población), ya que la utilizarás como usuario y contraseña para ingresar a la plataforma electrónica.
 - La credencial para votar expedida por el Instituto Nacional Electoral (INE), o el pasaporte expedido por la Secretaría de Relaciones Exteriores para el caso de los mexicanos, o en caso de menores de edad, una credencial reciente con fotografía, para identificarte.
 - Dos o tres lápices del número 2½, una goma de borrar y un sacapuntas.
 - Llega por lo menos 30 minutos antes de iniciar el examen, con lo cual evitará presiones y tensiones innecesarias.

Reglas durante la administración del Examen

- No se permitirá el acceso a ningún sustentante 30 minutos después de iniciada la aplicación del Examen.
- Debe portar una identificación oficial (la credencial para votar expedida por el Instituto Nacional Electoral (INE), o el pasaporte expedido por la Secretaría de Relaciones Exteriores para el caso de los mexicanos), o para menores de edad, una credencial oficial reciente con fotografía; de no hacerlo, es causa suficiente para que no se permita la realización del examen.
- No está permitido fumar, comer o ingerir bebidas dentro del lugar de aplicación donde se está resolviendo el examen.
- No se permite utilizar teléfono o cualquier dispositivo móvil de comunicación, por lo que deberá apagarlo antes de ingresar.
- Las salidas momentáneas serán controladas por el aplicador. En ellas no está permitido sacar ningún documento del examen ni materiales que se estén empleando para su realización.
- Cualquier intento de copiar a otro sustentante o situación de intercambio de respuestas o cualquier otro mecanismo para llevarse el contenido del examen, causará su inmediata suspensión.

Capítulo 2

Algoritmos, Pseudocódigos y Programación Estructurada

En el Capítulo 1: Introducción a la Programación, se mencionó que la computadora sólo entiende una serie de instrucciones codificadas en el lenguaje máquina de la propia computadora. En este lenguaje, las instrucciones son una secuencia de ceros y unos (código binario) y escribir un programa directamente en este lenguaje sería una tarea laboriosa y tediosa. También es seguro que el proceso de detectar errores en el código y su corrección sería un proceso que nos tomaría aún más tiempo y nos produciría un buen dolor de cabeza.

Afortunadamente, los primeros programadores al enfrentarse a los problemas ya mencionados, buscaron la forma de reducir el esfuerzo que implicaba escribir programas. La solución que encontraron fue la de crear lenguajes, llamados de alto nivel, cuya sintaxis fuese más parecida a la del lenguaje humano permitiéndole al programador escribir programas en un lenguaje que se entiende directamente. Es la misma computadora la que se encargara de traducir los programas en el código de alto nivel al código máquina mediante unos programas llamados traductores.

Procedimiento Para Crear un Programa

El procedimiento para crear un programa que la computadora pueda ejecutar, es el siguiente:

1. **Comprender el problema** al que se le está buscando solución y especificar qué información se le proporcionará al programa, **datos de entrada** y qué tipo de información arrojará el programa como respuesta, **datos de salida**.
2. **Diseñar el algoritmo** del programa que convertirá los datos de entrada al programa en los datos de salida del programa.
3. **Codificar el algoritmo** en un lenguaje de alto nivel.
4. Usar un editor de textos para **editar el código**. Este código se conoce como **programa fuente**.

5. Convertir el programa fuente a **código máquina** usando un compilador. Este proceso se llama **compilar el programa**. Normalmente nuestro código fuente tendrá **errores de sintaxis** que el compilador detectará y nos lo indicará. Debemos regresar al editor para corregirlos y repetir el paso 5 hasta que nuestro código no tenga errores de sintaxis. El código máquina resultante, también llamado **código ejecutable** es un programa que la máquina puede ejecutar.
6. **Ejecutar** el programa alimentándolo con datos para los cuales conozcamos los resultados que nos arrojará el programa y **comparar los resultados obtenidos del programa con los resultados esperados**. Es frecuente que los resultados obtenidos no coincidan con los esperados. Esto se debe a que nuestro algoritmo contiene algún **error de lógica** que deberá de encontrarse y corregirse.

En este tema estudiaremos los primeros dos pasos del procedimiento para crear un programa. Los restantes se verán en los tutoriales del sistema de desarrollo empleado en el curso.

Comprensión del Problema

A fin de que podamos resolver cualquier problema, en cualquier disciplina no sólo en computación, es necesario comprender el problema a resolver. Es muy frecuente que comencemos a plantear soluciones antes de tener en claro el problema a resolver. Esto nos trae como consecuencia que los resultados obtenidos sean incorrectos y que tengamos que buscar una nueva solución. Además hay que considerar que en la práctica, los problemas a los que se les busca solución mediante la computadora son problemas complejos y que por lo general requieren del trabajo de varias personas, por períodos de tiempo largo. De aquí la importancia que tiene entender con claridad el problema antes de dedicarle recursos a resolverlo.

Una parte fundamental de la comprensión de un problema es la identificar correctamente qué respuestas son las que deseamos que nos proporcione el programa y cuál es la información que necesitamos proporcionarle al programa.

Algoritmo y Pseudocódigo

Una vez entendido el problema, el siguiente punto es establecer la secuencia de pasos necesarios para obtener las respuestas deseadas a partir de esos datos. Esta secuencia de pasos escrita en manera formal y sistemática se conoce como **algoritmo**.

Para describir este algoritmo se requiere de un lenguaje. Este lenguaje debe permitirnos, aparte de la descripción de los pasos para resolver el problema, modelar la representación de la solución. Esto último significa que a partir de la descripción de la solución, otra persona además de la que escribió el algoritmo sea capaz de llegar a la misma solución. Tal lenguaje se llama **pseudocódigo**.

Un algoritmo escrito en pseudocódigo, es un conjunto de **sentencias** escritas siguiendo cierta sintaxis o reglas de construcción.

Sentencias

Las sentencias, llamadas también **estructuras de control**, son construcciones para dirigir el flujo de acciones que la computadora efectuará sobre los datos para obtener los resultados.

Hay tres tipos de sentencias o estructuras de control: La sentencia secuencial o compuesta, la sentencia condicional o selección y la sentencia repetitiva o iteración.

Sentencia Compuesta

Una característica fundamental de un algoritmo, es que está formado por un conjunto de instrucciones o pasos que se ejecutan en una secuencia bien definida. A este conjunto de instrucciones que se ejecutan en secuencia se le conoce como **sentencia compuesta**. Cada una de las instrucciones de una sentencia compuesta puede ser una sentencia simple, una sentencia compuesta, una sentencia condicional o una sentencia repetitiva. Una **sentencia simple**, es una instrucción que no puede descomponerse en instrucciones más sencillas.

En pseudocódigo, una sentencia compuesta se escribe escribiendo cada una de sus sentencias que la componen en un renglón por separado:

```
sentencia1  
[sentencia2]...
```

Note que una sentencia compuesta puede estar formada de una o más sentencias.

Ejemplo sobre la Sentencia Compuesta

Considere el siguiente problema:

Escribir el pseudocódigo para un programa que convierta una velocidad dada en km/hr a m/s.

Podemos ver que la respuesta esperada de este programa es un número que representa la velocidad expresada en m/s. Además, el programa tiene como dato de entrada un número que representa la velocidad expresada en km/hr. El factor de conversión no es un dato de entrada puesto que su valor no cambia y puede considerarse como una constante en el programa.

La búsqueda de la solución de este problema se hará en varias aproximaciones. Empezaremos con una solución muy general y la iremos refinando en aproximaciones sucesivas. A esta técnica se le conoce con el nombre de **diseño descendente**. En la primera aproximación se establece lo **que** hay que hacer para resolver el problema sin preocuparnos en el **cómo**. En las siguientes aproximaciones se refinará la solución detallando el cómo. La primera aproximación de la solución podría ser el siguiente pseudocódigo:

```
lee velocidad
convierte velocidad de km/hr a m/s
escribe velocidad
```

La primera sentencia hace que la computadora lea un número suministrado por el usuario del programa y que representa la velocidad en km/hr. Ese número se almacena en una localidad de la memoria RAM llamada **variable**. Cada variable tiene un nombre. La variable utilizada en este programa se llama *velocidad*. La segunda sentencia toma el valor de la velocidad en km/hr y los convierte a m/s, dejando el resultado en la misma variable *velocidad* y por último la última instrucción toma el valor de la velocidad en m/s almacenado en la variable *velocidad* y lo escribe para que el usuario conozca el resultado.

Note que las tres sentencias del pseudocódigo deben de ejecutarse en el orden en que fueron escritas. La computadora no puede hacer la conversión de velocidad hasta no conocer la velocidad en km/hr. Tampoco puede escribir la velocidad en m/s antes de haber calculado esta velocidad. Estas tres sentencias constituyen una sentencia compuesta.

En las siguientes aproximaciones de la solución, se refinará cada uno de los pasos de la primera aproximación tratando de expresar el cómo se deben realizar cada uno de esos pasos. Al final esos pasos deberán quedar expresados en términos de las operaciones que es capaz de realizar la computadora, que son ciertas operaciones aritméticas y lógicas. También, ya se ha mencionado que la computadora posee ciertos dispositivos de entrada y salida por medio de los cuales le podemos alimentar o extraer información. Por ello, podemos pensar que la computadora puede “leer” y “escribir” un dato y que deben de existir instrucciones que le indiquen a la computadora que realice esas funciones. En pseudocódigo esas instrucciones se denotan por las **palabras clave**: **lee** y **escribe**. Podemos pensar que **lee** y **escribe** son sentencias simples. La sentencia **lee**, permite que el usuario suministre un dato a través de un dispositivo de entrada, normalmente el teclado, y lo almacena en una variable. La sentencia **escribe** despliega un resultado en un dispositivo de salida, normalmente el monitor.

Nos resta por analizar la segunda sentencia de nuestro pseudocódigo: Convierte la velocidad de km/hr a m/s. Aquí nos podemos preguntar: ¿Existe una instrucción para efectuar la conversión de una velocidad en km/hr a m/s? La respuesta es no. La tarea de convertir velocidades, así como cualquier otra conversión, puede expresarse a partir de

otras tareas más elementales. La tarea de convertir una velocidad en km/hr a m/s puede escribirse como:

```
toma el dato almacenado en la variable velocidad
multiplícalo por 1000
divide el resultado anterior entre 3600
almacena el resultado en la variable velocidad
```

Los cuatro pasos anteriores pueden representarse en forma abreviada mediante la siguiente expresión:

```
velocidad = velocidad * 1000/3600
```

Donde el símbolo = llamado operador de asignación, significa que el valor original de la variable velocidad va a ser reemplazado por el resultado de:

```
velocidad * 1000/3600
```

Esto es, el resultado de multiplicar (*) el valor que está almacenado en la variable velocidad por 1000 y dividirlo (/) entre 3600. Luego una segunda aproximación al programa deseado estaría dada por el siguiente pseudocódigo:

```
lee velocidad
velocidad = velocidad * 1000/3600
escribe velocidad
```

En este programa sólo fueron necesarias dos aproximaciones a fin de que todos los pasos del programa fuesen instrucciones que la computadora pueda ejecutar. En programas más complejos será necesario un mayor número de aproximaciones.

Ejercicio sobre la Sentencia Compuesta

Escribir el pseudocódigo para un programa que lea los tres lados de un triángulo y escriba su área.

Sentencia Condicional

Una segunda sentencia, la sentencia condicional, le permite a la computadora seleccionar entre dos cursos alternativos a seguir, dependiendo de una condición dada. La sintaxis de la sentencia condicional es:

```
si(expresión)
    sentencia1
[otro
    sentencia2]
```

si es una palabra clave con la que inicia la sentencia condicional. La palabra clave **otro** separa las dos sentencias compuestas: *sentencia1* y *sentencia2*. La expresión entre paréntesis al evaluarse sólo puede tomar los valores falso o verdadero. Si el valor de

expresión es verdadero, la computadora ejecuta la *sentencia1* en caso contrario, la computadora ejecutará la *sentencia2*.

Note que la construcción:

```
[otro
  sentencia2]
```

es opcional (está encerrada entre corchetes). Si se omite y el valor de *expresión* es falso, entonces la computadora no hará nada.

Ya mencionamos que una sentencia compuesta puede estar formada de una o más sentencias. Si las sentencias compuestas de la sentencia condicional sólo tienen una sentencia pueden escribirse también de la siguiente manera:

```
si(expresión) sentencia1
[otro sentencia2]
```

Si las sentencias compuestas tienen más de una sentencia, deberán de escribirse como:

```
si(expresión) {
  sentencia11
  sentencia12 ...
}
[otro {
  sentencia21
  sentencia22 ...
}]
```

Note que las sentencias que forman *sentencia1* se han delimitado por llaves, {}. Esto nos indica que todas las sentencias se ejecutarán si *expresión* es verdadera. Si omitimos las llaves y escribimos:

```
si(expresión)
  sentencia11
  sentencia12 ...
[otro {
  sentencia21
  sentencia22 ...
}]
```

la construcción es incorrecta, debido a que si *expresión* es verdadera sólo se ejecutaría *sentencia11*. *sentencia12 ...* se considerarían como sentencias independientes de la sentencia condicional y la construcción:

```
[otro
  sentencia21
  sentencia22 ...]
```

ya no formaría parte de la sentencia condicional, lo cual no es posible ya que esta construcción no puede existir por sí sola. También las sentencias que forman

sentencia2 se han delimitado por llaves, para indicar que todas las sentencias se ejecutarán si *expresión* es falso. Si omitimos las llaves y escribimos:

```
si(expresión) {  
    sentencia11  
    sentencia12 ...  
}  
[otro  
    sentencia21  
    sentencia22 ...]
```

la construcción aunque correcta, no producirá el efecto deseado ya que si *expresión* es falso sólo se ejecutaría *sentencia21. sentencia22 ...* se considerarían como sentencias independientes de la sentencia condicional.

Ejemplos sobre la Sentencia Condicional

1. Escribir el pseudocódigo para un programa que lea tres números enteros y escriba el mayor de ellos.

En este problema, los datos de entrada son tres números positivos, *a*, *b* y *c* y la respuesta esperada es un número, el mayor de los tres. Una primera aproximación a la solución del problema puede ser:

```
lee a, b, c  
encuentra el mayor de a, b y c  
escribe mayor
```

donde *a*, *b* y *c* son las variables que contienen los números de entrada y *mayor* es la variable que contiene el mayor de los tres números. En este pseudocódigo, el único paso que necesita refinarse es el segundo, ya que el primero y el tercero corresponden a sentencias simples. Para encontrar el mayor de *a*, *b* y *c*, la computadora debe comparar los tres números. Sin embargo, la computadora sólo puede comparar dos números a la vez. Por lo que el segundo paso debe separarse en dos partes:

```
encuentra el mayor de a y b y almacénalo en mayor  
encuentra el mayor de c y mayor y almacénalo en mayor
```

Para encontrar el mayor de *a* y *b*, la computadora debe preguntarse: ¿es *a* mayor que *b*?. Esto lo podemos expresar mediante la sentencia condicional siguiente.

```
si(a > b) mayor = a  
otro mayor = b
```

Aquí la computadora evalúa la expresión *a* > *b*. Si es verdadero, esto es, si *a* > *b*, le asignará a la variable *mayor* el valor de *a*, en caso contrario almacenará en *mayor* el valor de *b*. De cualquier forma, al terminar de ejecutar la sentencia, en *mayor* tendremos el mayor de *a* y *b*.

De la misma manera, para encontrar el mayor de `c` y `mayor`, podemos usar la sentencia condicional siguiente.

```
si(c > mayor) mayor = c
```

Si la expresión `c > mayor` es verdadero significa que el mayor de los tres números es `c` y por lo tanto lo almacenamos en la variable `mayor`. Note que en este caso no hay una sentencia 2 ya que si la expresión `c > mayor` es falso, la variable `mayor` ya contiene el mayor de los tres números y no es necesario cambiar el valor de esta variable.

El pseudocódigo para este programa queda entonces como:

```
lee a, b, c

si(a > b) mayor = a
otro mayor = b
si(c > mayor) mayor = c

escribe mayor
```

2. Escribe el pseudocódigo de un programa, que realice las cuatro operaciones fundamentales de la aritmética. El programa deberá leer el primer número, el carácter que representa la operación deseada y el segundo número. El programa desplegará el resultado.

En este programa la entrada estará formada de dos números que representan los operandos y un carácter que representa el operador. La salida consistirá de un número que representa el resultado o un mensaje de error si la operación es inválida.

La primera aproximación de la solución del problema puede ser:

```
lee resultado
lee operador
lee operando
calcula resultado
escribe resultado
```

El primer operando se almacena en una variable llamada `resultado` y es en ésta donde queda el resultado al final del programa. La variable `operador` contiene el operador (carácter empleado para indicar la operación) y la variable `operando` contiene el segundo de los operandos. En este pseudocódigo, el único paso que necesita refinarse es el cuarto. Los otros corresponden a sentencias simples de lectura y escritura. La forma en que se calcula el resultado depende de la operación a realizar y está determinada por el contenido de la variable `operador`. El cálculo del resultado se puede escribir mediante las siguientes sentencias condicionales en cascada:

```
si(operador == '+') resultado = resultado + operando
```

```
otro si(operador == '-') resultado = resultado - operando
otro si(operador == '*') resultado = resultado * operando
otro si(operador == '/')
    si(operando != 0.0) resultado = resultado / operando
    otro escribe "Error: División entre cero"
otro escribe "Error: Operador desconocido"
```

En la expresión de la sentencia condicional:

```
operador == '+'
```

el símbolo `==` significa "igual a", por lo que la computadora se pregunta si el carácter almacenado en la variable `operador` es igual al carácter `'+'`. Si es cierto se sumarán los contenidos de las variables `resultado` y `operando` y el resultado se almacenará en la variable `resultado`. En caso contrario se investiga si la operación deseada es la resta, etc. Note que si la operación deseada es la división, se compara el valor en la variable `operando` con cero. Si el valor en `operando` no es cero podemos efectuar la división, en caso contrario desplegaremos un mensaje de error ya que la división entre cero no es válida.

El pseudocódigo para este programa queda como:

```
lee resultado
lee operador
lee operando

si(operador == '+') resultado = resultado + operando
otro si(operador == '-') resultado = resultado - operando
otro si(operador == '*') resultado = resultado * operando
otro si(operador == '/')
    si(operando != 0.0) resultado = resultado / operando
    otro escribe "Error: División entre cero"
otro escribe "Error: Operador desconocido"

escribe resultado
```

Ejercicios sobre la Sentencia Condicional

1. El costo de un telegrama ordinario es de \$1000 si el número de palabras es hasta 10, por cada palabra adicional se cobra \$200. Si el telegrama es urgente los costos son de \$2000 y \$400 respectivamente. Escribir el pseudocódigo para un programa que lea el tipo del telegrama (una sola letra, 'O' para ordinario y 'U' para urgente) y el número de palabras del telegrama y escriba el costo de éste.
2. Escribir el pseudocódigo para un programa que lea tres números positivos suministrados en orden ascendentes los cuales representan las longitudes de los lados de un triángulo. El programa deberá determinar si los tres lados forman un triángulo y su tipo.

Sentencia Repetitiva

La sentencia repetitiva, le permite a la computadora ejecutar una serie de pasos o instrucciones en forma repetitiva mientras se cumpla una condición dada. La sintaxis de esta sentencia es la siguiente:

```
mientras(expresión)
    sentencia
```

La sentencia repetitiva inicia con la palabra clave **mientras**. La expresión entre paréntesis al evaluarse sólo puede tomar los valores falso o verdadero. Primero la computadora evalúa la *expresión*. Si es verdadero, la computadora ejecuta la sentencia compuesta, *sentencia*, enseguida vuelve a evaluar la *expresión* y continuará ejecutando la sentencia compuesta mientras *expresión* sea verdadero. Si valor de *expresión* es falso, entonces la sentencia repetitiva termina.

Al igual que con la sentencia condicional, si la sentencia compuesta de la sentencia repetitiva solo tiene una sentencia puede escribirse como

```
mientras(expresión) sentencia
```

Si la sentencias compuesta tiene más de una sentencia, deberá de escribirse como:

```
mientras(expresión) {
    sentencia1
    sentencia2 ...
}
```

Note que las sentencias que forman *sentencia* se han delimitado por llaves, {}. Esto nos indica que todas las sentencias se ejecutarán si *expresión* es verdadero. Si omitimos las llaves y escribimos:

```
mientras(expresión)
    sentencia1
    sentencia2 ...
```

la construcción aunque correcta, no producirá el efecto deseado ya que si *expresión* es verdadero sólo se ejecutaría *sentencia1*. *sentencia2* ... se considerarían como sentencias independiente de la sentencia repetitiva.

Ejemplos sobre la Sentencia Repetitiva

1. Escribir el pseudocódigo para un programa que genere una tabla que muestre el capital acumulado de una inversión, a una tasa de interés anual, con recapitalización mensual. El programa deberá leer el monto de capital que se desea invertir, la tasa de interés anual y el número de meses que se desea tabular.

Para este programa, los datos de entrada son tres números: el monto del capital, la tasa de interés anual, y el número de meses a tabular. La salida del programa será una tabla de la forma:

Mes	Capital
1	dddd.dd
2	dddd.dd
...	

Una primera aproximación a la solución a este problema puede ser el siguiente pseudocódigo:

```
lee capital, tasa y meses
genera tabla
```

La primera instrucción es una sentencia simple por lo que sólo necesitamos refinar es la segunda instrucción: `genera tabla`. Para generar esta tabla requerimos que la computadora repita el siguiente paso, tantas veces como meses se deseen en la tabla:

```
genera renglón
```

Para lograr que la sentencia compuesta se repita se utiliza la siguiente sentencia repetitiva:

```
mes = 1
mientras(mes <= meses) {
    genera renglón
    mes = mes + 1
}
```

La variable `mes` se utiliza como contador de las veces que se generan renglones de la tabla y debe inicializarse a uno ya que la primera vez que se ejecuta la sentencia corresponde al mes uno. El último paso de la sentencia compuesta anterior es necesario para que la computadora lleve la cuenta de cuantas veces ha ejecutado dicha sentencia. En este caso la variable `mes` se utiliza como un contador que se incrementa en uno cada vez que se ejecuta la sentencia.

Por lo tanto, una segunda aproximación a la solución estará dada por el siguiente pseudocódigo:

```
lee capital, tasa y meses

mes = 1
mientras(mes <= meses) {
    genera renglón
    mes = mes + 1
}
```

En el pseudocódigo anterior, sólo hay que refinar el paso genera renglón. Para generar cada renglón de la tabla se requieren los dos pasos siguientes:

```
calcula capital
escribe mes y capital
```

La variable `capital` en la que inicialmente se guardó el valor del capital a invertir, se usa para almacenar el valor del capital conforme se va acumulando mensualmente. Luego la tercera aproximación a la solución estará dada por el siguiente pseudocódigo:

```
lee capital, tasa y meses

mes = 1
mientras(mes <= meses) {
    calcula capital
    escribe mes y capital
    mes = mes + 1
}
```

En el pseudocódigo anterior, resta por refinar el paso de calcular el capital acumulado mensualmente. El capital acumulado en un mes se puede calcular a partir del capital acumulado en el mes anterior de la siguiente forma:

```
capital = capital + interés
```

esto es el capital de este mes es igual al capital del mes anterior más el interés ganado. El interés es el producto del capital por la tasa de interés mensual. Como el dato disponible es la tasa de interés anual, la dividimos entre 12 para obtener la tasa de interés mensual:

```
capital = capital + capital * tasa/12
```

o agrupando

```
capital = capital * (1 + tasa/12)
```

Luego, nuestro pseudocódigo final es:

```
lee capital, tasa y meses

mes = 1
mientras(mes <= meses) {
    capital = capital * (1 + tasa/12)
    escribe mes y capital
    mes = mes + 1
}
```

2. Crea un programa que calcule el área bajo la curva $y = x^2$ y que se encuentra entre las rectas $x = x_i$ y $x = x_f$. Aproxime el área bajo la curva como la suma de las áreas de n rectángulos inscritos bajo la curva, tal como se muestra en la figura 2-

1. El programa deberá pedir los valores de x_i y x_f , así como el número de rectángulos a usarse.

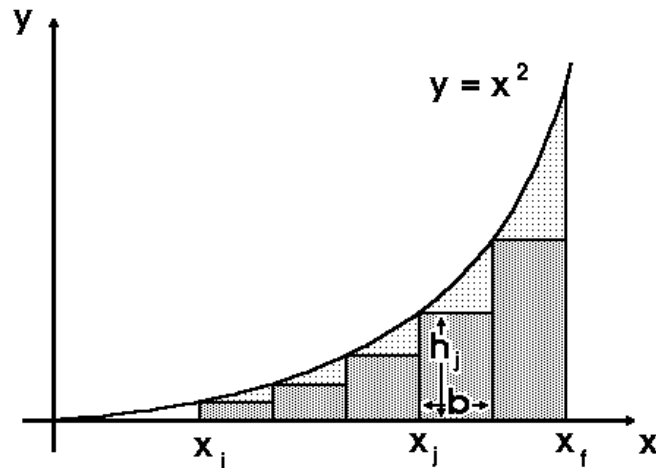


Figura 2-1

En este programa, los datos de entrada son tres números: los valores de las abscisas entre las que se calculará el área y el número de rectángulos que se usarán en éste cálculo. La salida del programa será un número, el área bajo la curva.

Una primera aproximación para el pseudocódigo de este programa es:

```
lee xi, xf, n
calcula área
escribe área
```

En el pseudocódigo anterior, el paso que se requiere desglosar más es el en que se calcula el área. Dado que el área bajo la curva se va a aproximar mediante el área de los n rectángulos inscritos bajo la curva, tenemos que:

$$\text{área} = \text{área}_1 + \text{área}_2 + \dots + \text{área}_j + \dots + \text{área}_n$$

Donde $\text{área}_1, \text{área}_2, \dots, \text{área}_j, \dots, \text{área}_n$ son las áreas de los n rectángulos. Como el área de un rectángulo está dada por:

$$\text{área}_j = b \times h_j$$

Donde b y h_j son la base y la altura del rectángulo, respectivamente. El área de los n rectángulos es:

$$\text{área} = b \times h_1 + b \times h_2 + \dots + b \times h_j + \dots + b \times h_n$$

$$\text{área} = b \times (h_1 + h_2 + \dots + h_j + \dots + h_n)$$

Para obtener el área de los n rectángulos multiplicamos la base por la suma de las alturas de los n rectángulos. Luego el paso del cálculo del área bajo la curva puede desglosarse como:

```
calcula base
calcula sumaH
área = sumaH * base
```

La base de cada rectángulo puede calcularse mediante la expresión:

$$\text{base} = (x_f - x_i)/n$$

Para calcular la suma de las alturas requiere que se calcule cada una de las alturas de los rectángulos. La altura de cada rectángulo es el valor de la función para el valor de la abscisa izquierda de cada rectángulo: $y = x^2$. Por lo que la suma de las alturas puede calcularse mediante de la siguiente manera:

```
sumaH = 0
x = xi
mientras(x < xf) {
    y = x2
    sumaH = sumaH + y
    x = x + base
}
```

El pseudocódigo final para este programa es:

```
lee xi, xf, n

base = (xf - xi)/n
sumaH = 0
x = xi
mientras(x < xf) {
    y = x2
    sumaH = sumaH + y
    x = x + base
}
área = sumaH * base

escribe área
```

Ejercicios sobre la Sentencia Repetitiva

1. Escribir el pseudocódigo para un programa que lea las edades de un grupo de alumnos y encuentre la edad promedio.
2. Escribir el pseudocódigo para un programa que determine el número de meses necesario para que una inversión, colocada a una tasa de interés anual dada y con recapitalización mensual, se duplique.

Programación Estructurada

Las tres estructuras de control vistas, la **sentencia compuesta**:

```
sentencia1  
[sentencia2]...
```

la **sentencia condicional**:

```
si(expresión) sentencia1  
[otro sentencia2]
```

y la **sentencia repetitiva**:

```
mientras(expresión)  
sentencia
```

todas tienen un sólo punto de entrada o inicio y un sólo punto de salida o final. Su punto de entrada es la parte superior y su punto de salida está en la parte inferior. Lo anterior nos permitirá crear programas más complejos combinando las estructuras de control de diferentes maneras, uniendo las salidas de unas con las entradas de otras. Para este fin existe una **regla fundamental de composición** que establece lo siguiente:

Las estructuras de control que se pueden formar combinando de manera válida las sentencias secuenciales, condicionales y repetitivas también serán válidas.

Lo anterior da origen al concepto de **programación estructurada**. Un programa estará bien construido si está formado por estructuras de control válidas, de acuerdo con la regla precedente.

Por ejemplo, podemos construir una sentencia secuencial a partir de una sentencia secuencial y una sentencia condicional:

```
sentencia1  
sentencia2  
  
si(expresión) {  
    sentencia3  
    sentencia4  
}  
otro {  
    sentencia5  
    sentencia6  
}
```

Note que esta nueva sentencia sigue teniendo una sola entrada y una sola salida.

Un segundo ejemplo es una sentencia condicional donde su primera sentencia compuesta está formada de otra sentencia condicional seguida de una sentencia iterativa y su segunda sentencia también contiene una sentencia condicional:

```

si(expresión1) {
    si(expresión2) {
        sentencia1
        sentencia2
    }
    otro {
        sentencia3
        sentencia4
    }

    mientras(expresión3) {
        sentencia5
        sentencia6
    }
}
otro
    si(expresión4) {
        sentencia7
        sentencia8
    }
    otro {
        sentencia9
        sentencia10
    }
}

```

Ejemplo Sobre Programación Estructurada

Modifique el pseudocódigo del ejemplo 2 sobre la sentencia condicional para que la calculadora sea capaz de realizar operaciones consecutivas, mostrando los resultados parciales. Por ejemplo si tecleamos:

```

4
+
3

```

la computadora nos mostrará el resultado de la suma:

```

4
+
3
7

```

y estará esperando que tecleemos el operador y operando para la siguiente operación. Por ejemplo supongamos que al resultado anterior le deseamos restar 5. Entonces teclearemos:

```

4
+
3
7
-
5

```

La computadora desplegará el resultado de la resta:

```
4
+
3
7
-
5
2
```

y volverá a esperar que tecleemos el operador y operando para la siguiente operación. Si deseamos que el programa termine presionaremos la tecla '='.

En este programa la entrada estará formada por números que representan los operandos y caracteres que representan los operadores. La salida consistirá de números que representan los resultados parciales y mensajes de error si la operación es inválida.

Una primera aproximación al programa está dada por el siguiente pseudocódigo:

```
lee resultado
lee operador
mientras(operador != '=') {
    lee operando
    calcula resultado
    escribe resultado
    lee operador
}
```

En el primer paso del programa, estamos leyendo el primer número y lo estamos almacenando en una variable llamada resultado. En el segundo paso leemos el carácter que representa la operación a realizar. En la expresión de la sentencia repetitiva:

```
operador != '='
```

el símbolo `!=` significa "diferente de", por lo que el ciclo se repetirá mientras el operador sea diferente del carácter '='.

En el primer paso de la sentencia compuesta de la sentencia repetitiva leemos el otro número con el que se va hacer la operación y lo almacenamos en la variable operando.

El paso en que se calcula el resultado es similar al del ejemplo 2 sobre la sentencia condicional, por lo que el pseudocódigo del programa queda como:

```
lee resultado
lee operador

mientras(operador != '=') {
    lee operando

    si(operador == '+') resultado = resultado + operando
    otro si(operador == '-') resultado = resultado - operando
    otro si(operador == '*') resultado = resultado * operando
}
```

```

    otro si(operador == '/')
        si(operando != 0.0) resultado = resultado / operando
        otro escribe "Error: División entre cero"
    otro escribe "Error: Operador desconocido"

    escribe resultado

    lee operador
}

```

Problemas

1. Escribir el pseudocódigo para un programa que lea un número que represente una distancia en metros y escriba su equivalente en pies y en pulgadas.
2. Escribir el pseudocódigo para un programa que lea un número que represente el radio de una esfera y escriba su área y volumen.
3. Escriba el pseudocódigo para un programa que indique a una cajera de banco el número y denominación de los billetes que necesita darle a un cliente al hacer un retiro. La cajera deberá darle al cliente billetes de la más alta denominación posible, esto es, el menor número de billetes. Suponga que los retiros deben de ser en cantidades múltiples de 10 pesos y que hay billetes de \$10, \$20, \$50 y \$100 pesos.
4. Un año bisiesto es aquel año que es divisible entre 4 pero no es divisible entre 100 a menos que sea divisible entre 400 en cuyo caso si es bisiesto. Crea el pseudocódigo de un programa que determine si un año dado es un año bisiesto o no.
5. La capacitancia de un capacitor de placas paralelas con vacío por dieléctrico está dada por:

$$C = \epsilon_0 \frac{A}{d}$$

Donde $\epsilon_0 = 8.85 \times 10^{-12}$ F/m es la permitividad del vacío, A es el área de una de las placas del capacitor y d es la distancia entre las placas.

Crea el pseudocódigo de un programa que calcule la capacitancia de un capacitor de placas paralelas. Las placas paralelas pueden tener forma rectangular o circular. El programa pedirá el tipo de placas del capacitor, 'R' o 'C' y dependiendo del tipo las dimensiones a y b de los lados del rectángulo o el radio r. Adicionalmente el programa pedirá la distancia d, entre placas.

6. Escribir el pseudocódigo para un programa que genere una tabla de los cuadrados y los cubos de los números enteros del 1 al 10. La tabla resultante debe tener la siguiente forma:

Número	Cuadrado	Cubo
1	1	1
2	4	8
etc.		

7. Escribir el pseudocódigo para un programa que tabule la ecuación:

$$y = x^3 - 2x + 3$$

El programa deberá pedir los límites inferior, superior y el valor del incremento de x .

8. Crea el pseudocódigo para un programa que calcule el promedio de cada alumno de un grupo, para lo cual se leerán su matrícula y cuatro calificaciones. Indicar fin de datos con matrícula = 0. El programa deberá imprimir la matrícula, las cuatro calificaciones y el promedio de cada alumno. Al final deberá imprimir la calificación promedio global del grupo.



INSTITUTO TECNOLÓGICO DE SONORA
Educar para Trascender

Pseudocódigo y PSEINT

Programa Educativo
Ingeniero en Software

Marzo 2016

INDICE

INTRODUCCIÓN	4
¿QUÉ ES PSEINT?	5
La interfaz y el área de trabajo.....	5
EL PSEUDOCÓDIGO	7
FORMA GENERAL DE UN ALGORITMO EN PSEUDOCÓDIGO	7
TIPOS DE DATOS	7
Tipos de Datos Simples	8
Estructuras de Datos: Arreglos	8
Dimensionamiento (Arreglos-Arrays)	8
EXPRESIONES.....	9
Operadores.....	9
Funciones matemática.....	10
PRIMITIVAS SECUENCIALES (COMANDOS DE ENTRADA, PROCESO Y SALIDA).....	11
Lectura o entrada.....	11
Asignación o proceso.....	11
Escritura o salida	11
ESTRUCTURAS DE CONTROL (PROCESO)	12
Condicionales	12
Si-Entonces (If-Then).....	12
Selección Múltiple (Select If).....	12
Repetitivas.....	13
Mientras Hacer (while)	13
Repetir Hasta Que (do-while)	14
Para (for).....	14
EJECUCIÓN PASO A PASO.....	145
EJEMPLOS DE ALGORITMOS.....	167
EJERCICIOS RESUELTOS UTILIZANDO PSEINT	212

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, así:

- **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 la secuencia que debe ejecutar el algoritmo

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

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

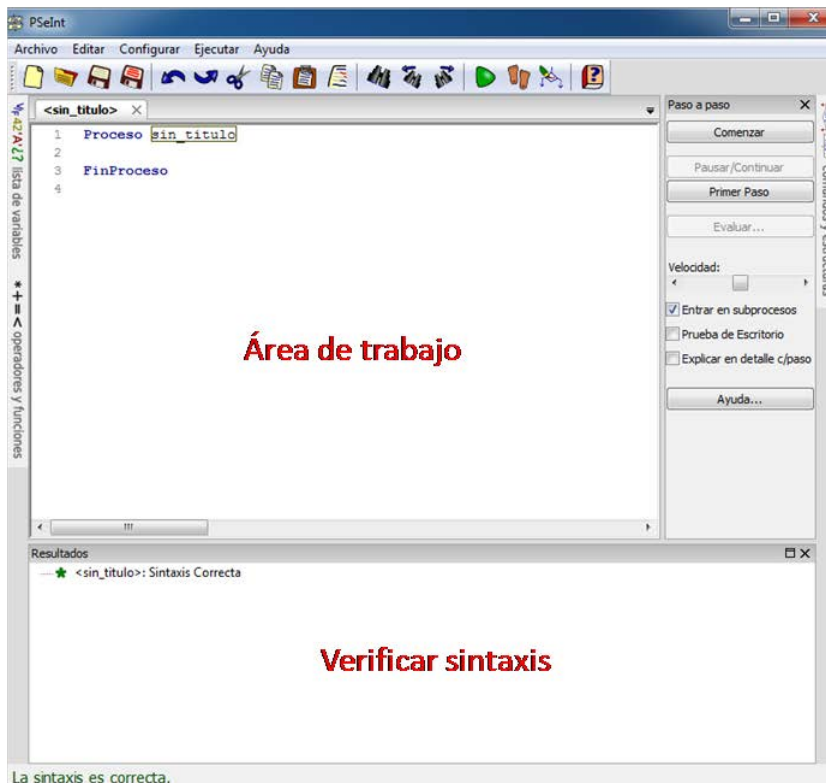
¿Qué es PSEINT?

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>

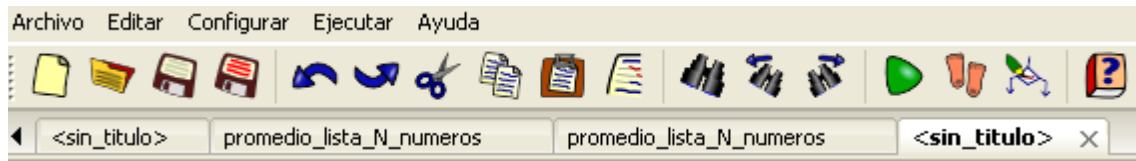
La interfaz y el área de trabajo






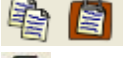








← Barra de Menú

← 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, Carácter.
- 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> (<max1>, ..., <maxN>);

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

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
*	Multipliación	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:

<i>Función</i>	<i>Significado</i>
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.

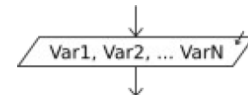
Primitivas Secuenciales (Comandos de Entrada, Proceso y Salida)

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

Lectura o entrada

La instrucción Leer permite ingresar información desde el ambiente.

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

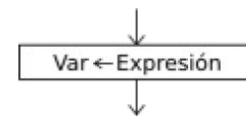


Esta instrucción lee N valores desde el ambiente (en este caso el teclado) y los asigna a las N variables mencionadas. Pueden incluirse una o más variables, por lo tanto el comando leerá uno o más valores.

Asignación o proceso

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

<variable> <- <expresión> ;

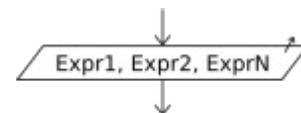


Al ejecutarse la asignación, primero se evalúa la expresión de la derecha y luego se asigna el resultado a la variable de la izquierda. El tipo de la variable y el de la expresión deben coincidir.

Escritura o salida

La instrucción Escribir permite mostrar valores al ambiente.

Escribir <expr1> , <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.

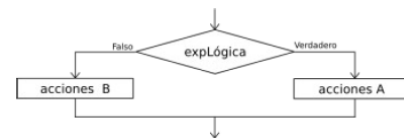
Estructuras de Control (Proceso)

- 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.



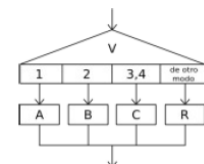
Si <condición> Entonces
 <instrucciones>
 Sino
 <instrucciones>
 FinSi

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

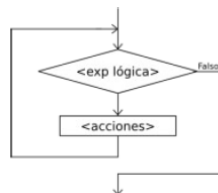
Esta instrucción permite ejecutar opcionalmente varias acciones posibles, dependiendo del valor almacenado en una variable de tipo numérico. Al ejecutarse, se evalúa el contenido de la variable y se ejecuta la secuencia de instrucciones asociada con dicho valor.

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>
```



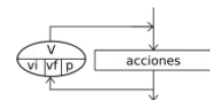
Al ejecutarse esta instrucción, la secuencia de instrucciones que forma el cuerpo del ciclo se ejecuta una vez y luego se evalúa la condición. Si la condición es falsa, el cuerpo del ciclo se ejecuta nuevamente y se vuelve a evaluar la condición. Esto se repite hasta que la condición sea verdadera.

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 las 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

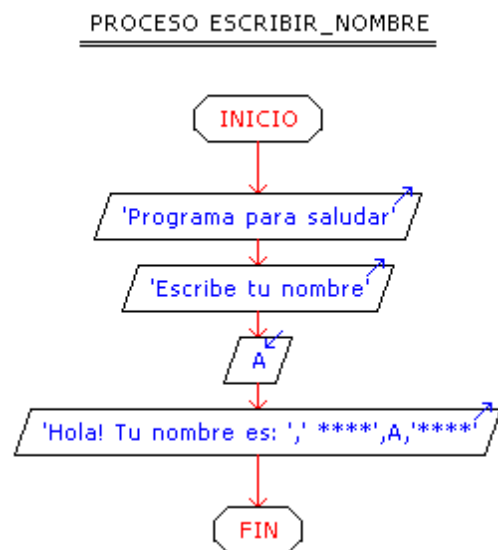
FinProceso //fin del proceso

```

1  //Programa para Escribir un saludo con el nombre: RPC
2  Proceso Escribir_nombre
3      Escribir "Programa para saludar";           //muestra en pantall
4      Escribir "Escribe tu nombre";             //instrucción
5      Leer a;                                   //ingresa por teclac
6      Escribir "Hola! Tu nombre es: ", " ****", a, "****"; //m
7  FinProceso                                     //fin del proceso
  
```

```

C:\Archivos de programa\PSeint\pseint.exe
*** Ejecucion Iniciada. ***
Programa para saludar
Escribe tu nombre
> Ronald PC
Hola! Tu nombre es:  ****Ronald PC****
*** Ejecucion Finalizada. ***
  
```



2. Sumar dos números 'a' y 'b'

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

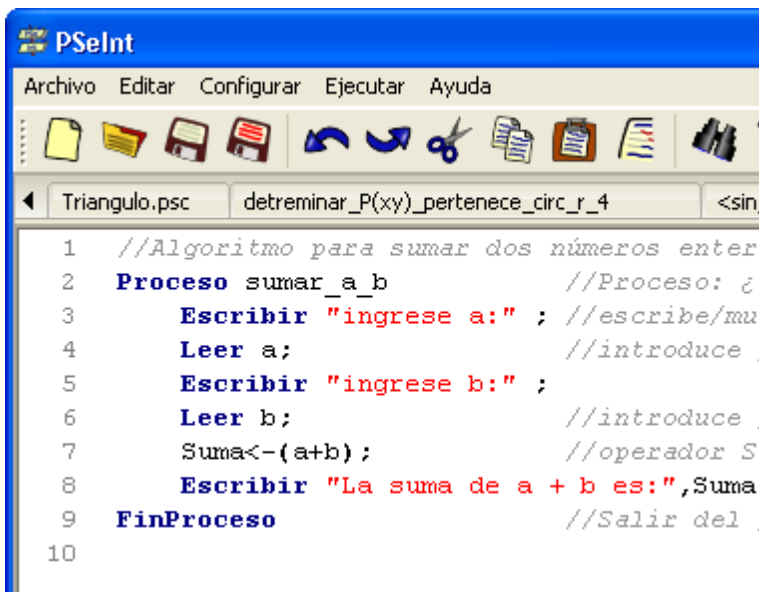
Proceso sumar_a_b //Proceso: ¿qué desea hacer el 'programa?': sumar a y b

```

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

```

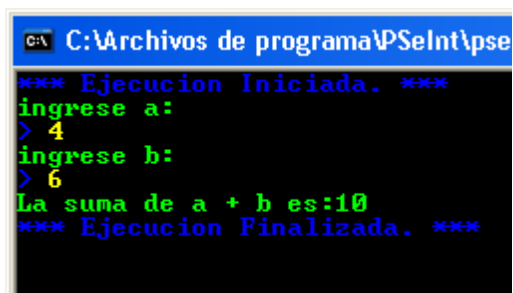
FinProceso



```

1 //Algoritmo para sumar dos números enteros
2 Proceso sumar_a_b //Proceso: ¿qué desea hacer el 'programa?': sumar a y b
3     Escribir "ingrese a:" ; //escribe/muestra en pantalla
4     Leer a; //introduce por teclado el valor de 'a'
5     Escribir "ingrese b:" ;
6     Leer b; //introduce por teclado el valor de 'b'
7     Suma<-(a+b); //operador Suma=a+b
8     Escribir "La suma de a + b es:",Suma ; //escribe/muestra en pantalla
9     + el valor Suma
10 FinProceso

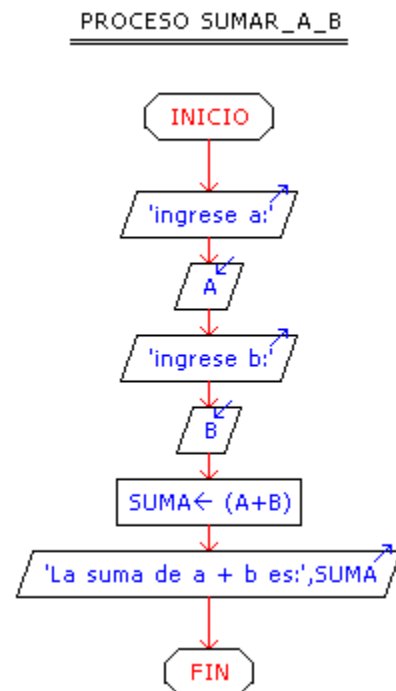
```



```

C:\Archivos de programa\PSeInt\pse
*** Ejecucion Iniciada. ***
ingrese a:
> 4
ingrese b:
> 6
La suma de a + b es:10
*** Ejecucion Finalizada. ***

```



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 tu 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"

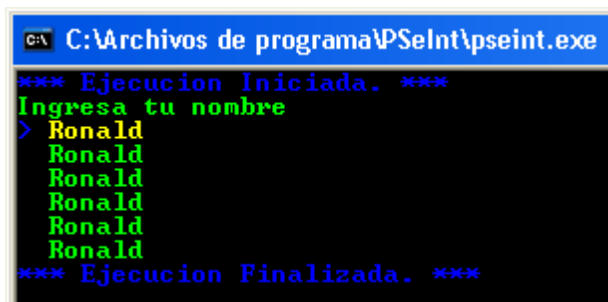
FinProceso //fin del proceso



```

1  //Programa para Escribir un nombre y re
2  Proceso repetir_nombre
3      Escribir "Ingresa tu nombre";
4      Leer nombre;
5      Para i<-1 Hasta 5 Con Paso 1 Hacer
6          Escribir " ", nombre;
7      FinPara
8
9  FinProceso
10

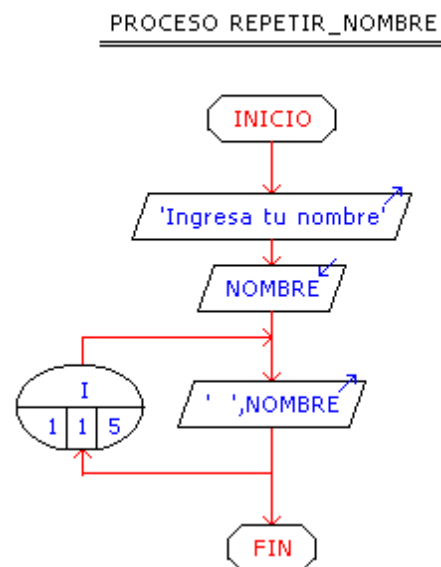
```



```

C:\Archivos de programa\PSeInt\pseint.exe
*** Ejecucion Iniciada. ***
Ingresa tu nombre
> Ronald
Ronald
Ronald
Ronald
Ronald
Ronald
*** Ejecucion Finalizada. ***

```



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

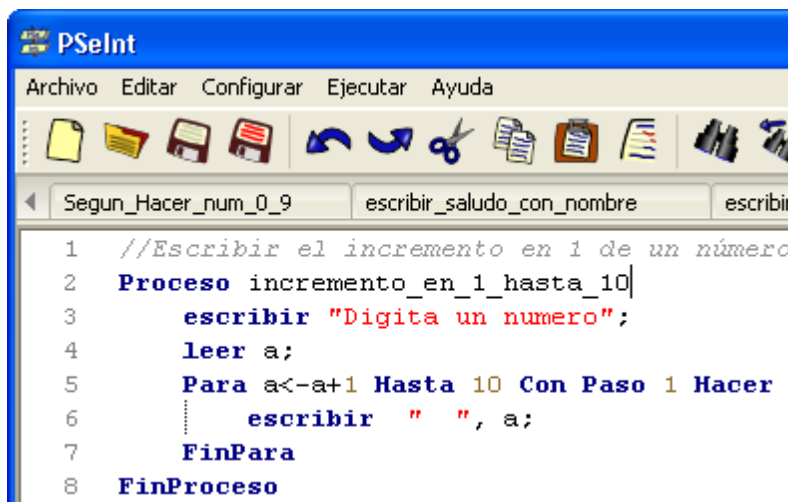
leer a; //ingresa la variable 'a' (número menor a 10)

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

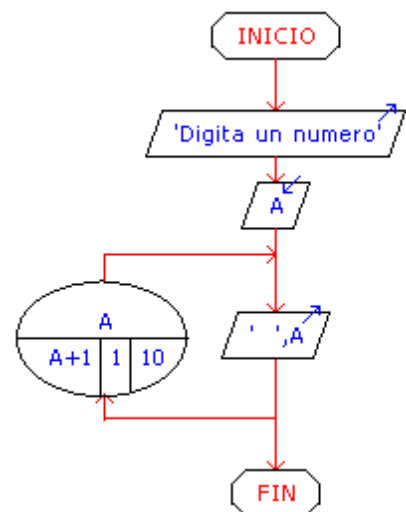
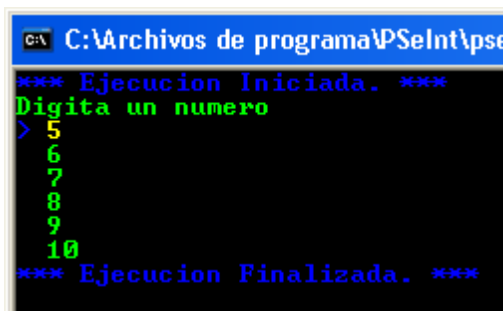
FinPara //Fin del comando Para

FinProceso //Fin del proceso



```

1 //Escribir el incremento en 1 de un número
2 Proceso incremento_en_1_hasta_10
3     escribir "Digita un numero";
4     leer a;
5     Para a<-a+1 Hasta 10 Con Paso 1 Hacer
6         escribir " ", a;
7     FinPara
8 FinProceso
  
```

```

C:\Archivos de programa\PSeInt\pse
*** Ejecucion Iniciada. ***
Digita un numero
> 5
6
7
8
9
10
*** Ejecucion Finalizada. ***
  
```

5. Sumar n números utilizando MIENTRAS

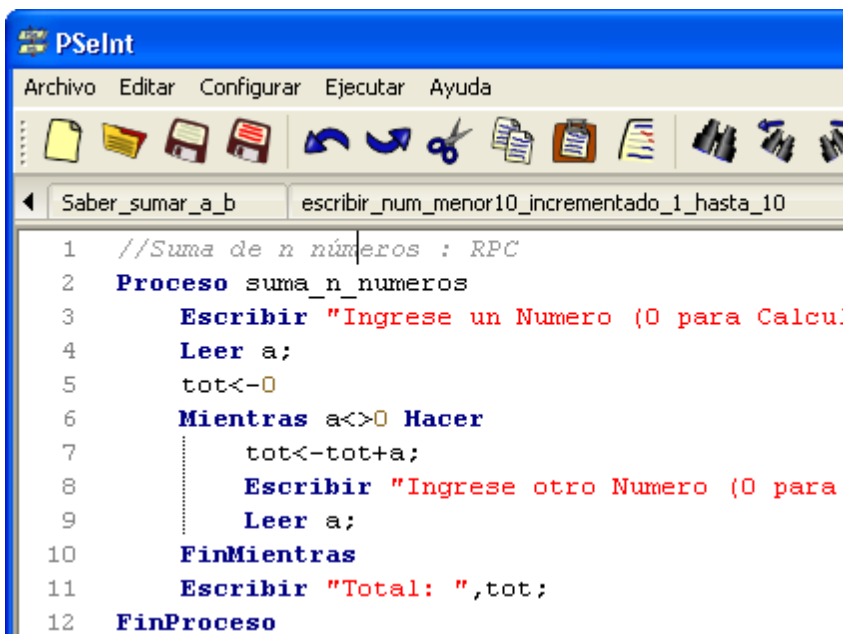
//Suma de n números : RPC

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

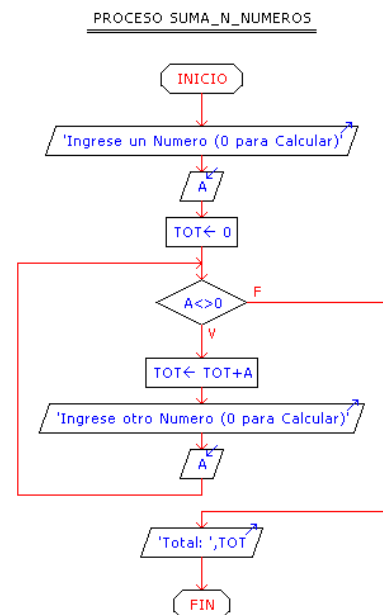
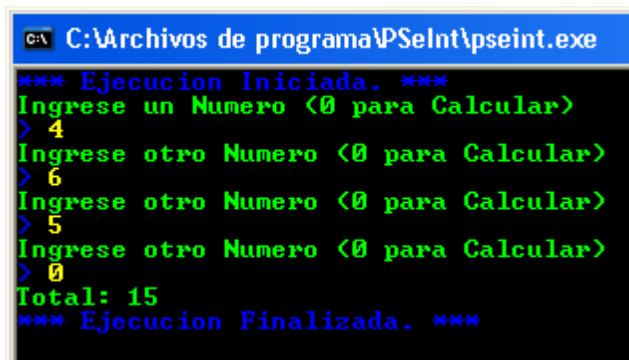
```



```

1 //Suma de n números : RPC
2 Proceso suma_n_numeros
3     Escribir "Ingrese un Numero (0 para Calcular)";
4     Leer a;
5     tot<-0
6     Mientras a<>0 Hacer
7         tot<-tot+a;
8         Escribir "Ingrese otro Numero (0 para Calcular)";
9         Leer a;
10    FinMientras
11    Escribir "Total: ",tot;
12 FinProceso

```

```

C:\Archivos de programa\PSeInt\pseint.exe
*** Ejecucion Iniciada. ***
Ingrese un Numero (0 para Calcular)
> 4
Ingrese otro Numero (0 para Calcular)
> 6
Ingrese otro Numero (0 para Calcular)
> 5
Ingrese otro Numero (0 para Calcular)
> 0
Total: 15
*** Ejecucion Finalizada. ***

```

6. Sumar n números utilizando REPETIR

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

```

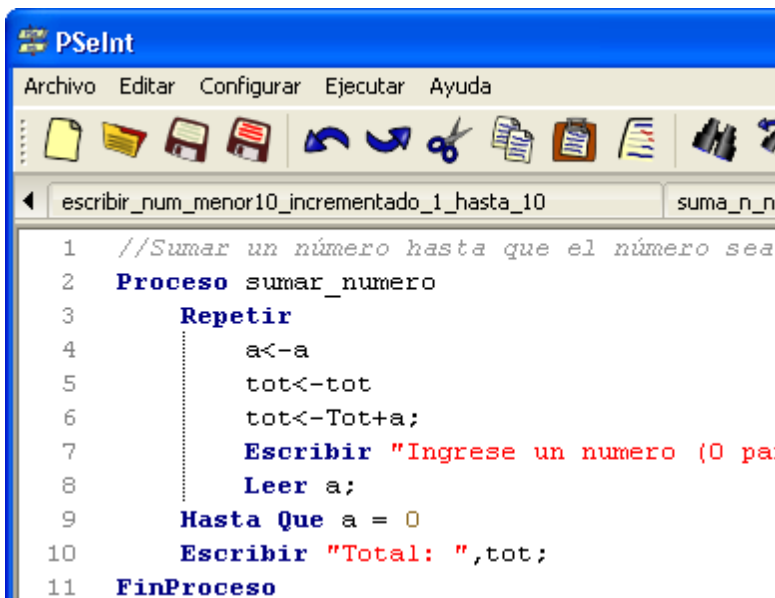
Proceso sumar_numero
Repetir

```

```

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

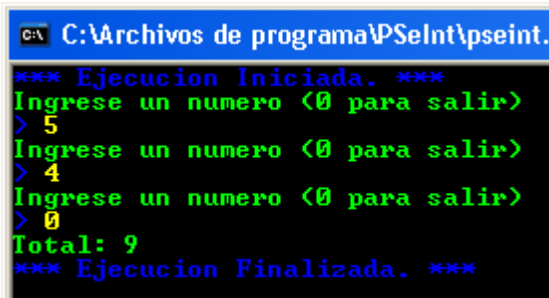
```



```

1 //Sumar un número hasta que el número sea
2 Proceso sumar_numero
3     Repetir
4         a<-a
5         tot<-tot
6         tot<-Tot+a;
7         Escribir "Ingrese un numero (0 para salir)";
8         Leer a;
9     Hasta Que a = 0
10    Escribir "Total: ",tot;
11 FinProceso

```

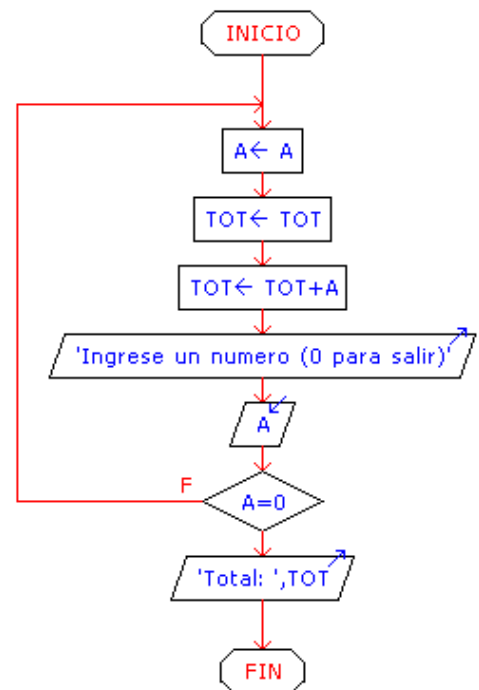


```

C:\Archivos de programa\PSeInt\pseint.
*** Ejecucion Iniciada. ***
Ingrese un numero <0 para salir>
> 5
Ingrese un numero <0 para salir>
> 4
Ingrese un numero <0 para salir>
> 0
Total: 9
*** Ejecucion Finalizada. ***

```

PROCESO SUMAR_NUMERO



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

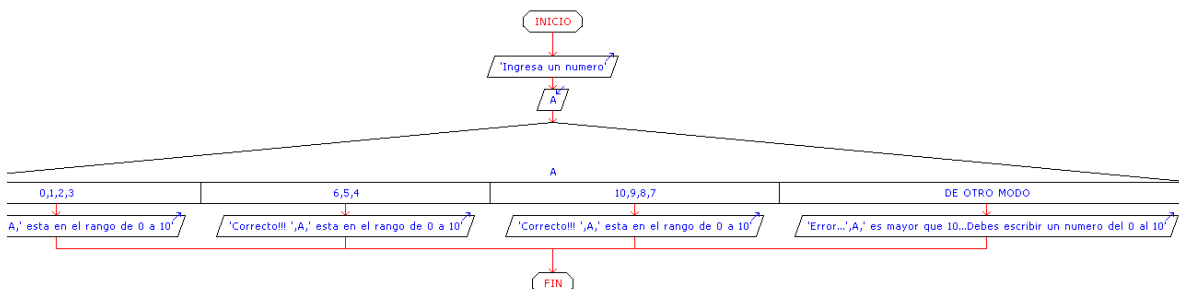
FinProceso

```

1 //Conocer si un número está en el rango de 0-10 con mensaje Correcto/Error: RPC
2 Proceso numero_entre_0_10
3   Escribir "Ingresa un numero";
4
5   Leer a;
6   Segun a Hacer
7     0,1,2,3: Escribir "Correcto!!! ", a, " esta en el rango de 0 a 10";
8
9     6,5,4: Escribir "Correcto!!! ", a, " esta en el rango de 0 a 10";
10
11    10,9,8,7: Escribir "Correcto!!! ", a, " esta en el rango de 0 a 10";
12
13    De Otro Modo:
14      Escribir "Error...", a, " es mayor que 10...Debes escribir un numero
15  FinSegun
16
17 FinProceso
  
```

```

C:\Archivos de programa\PSeInt\pseint.exe
*** Ejecucion Iniciada. ***
Ingresa un numero
> 6
Correcto!!! 6 esta en el rango de 0 a 10
*** Ejecucion Finalizada. ***
  
```



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
         $d \leftarrow b + c$ ;
        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
                 $d \leftarrow b - c$ ;
                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
                     $d \leftarrow b * c$ ;
                    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
                         $d \leftarrow b / c$ ;
                        escribir " La División de ", b, " / ", c, " = ", d
                    Sino
                        FinSi
                    FinSi
                FinSi
            FinSi
        FinSi
    FinSi

```

FinSi
 FinSi
 FinProceso

```

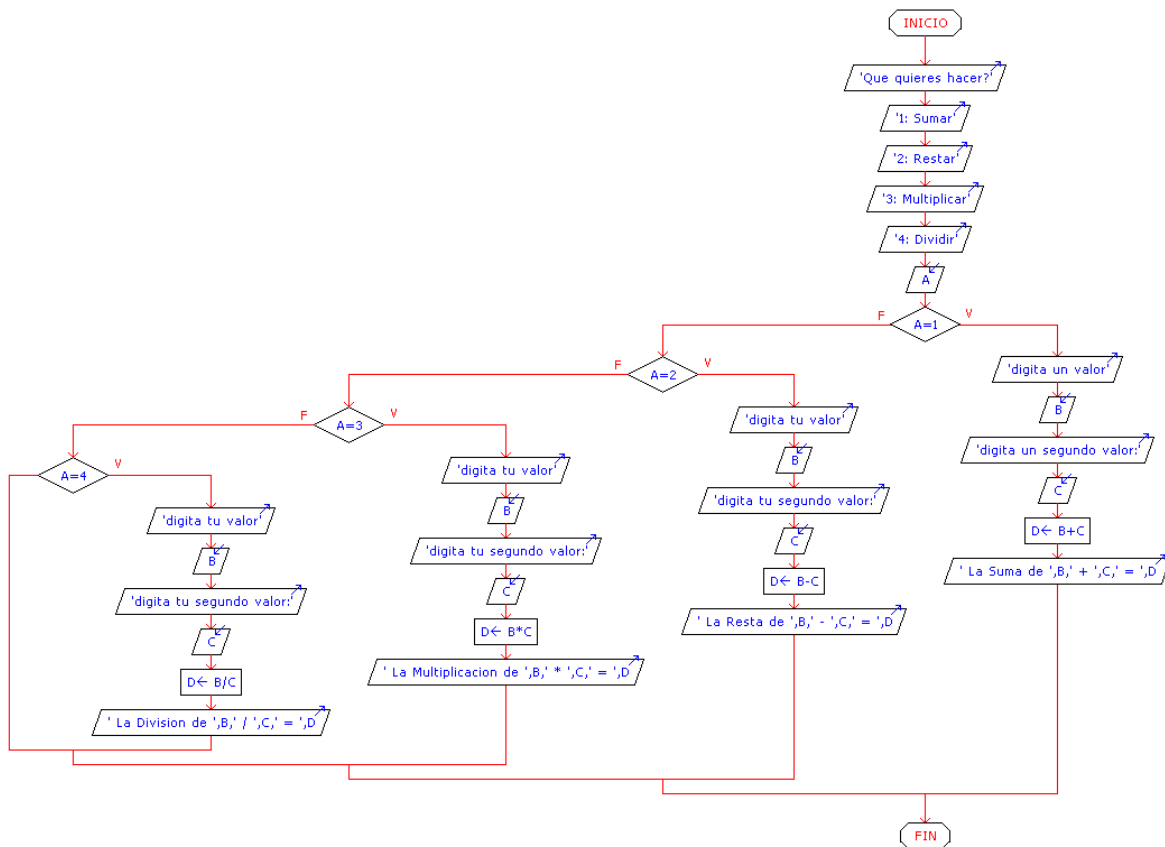
1 //Calculadora Suma, Resta, Multiplicación y División: RPC
2 Proceso calculadora
3   escribir "Que quieres hacer?";
4   escribir "1: Sumar";
5   escribir "2: Restar";
6   escribir "3: Multiplicar";
7   escribir "4: Dividir";
8   leer a;
9   Si a=1 Entonces
10    escribir "digita un valor";
11    leer 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\PSelnt\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

```

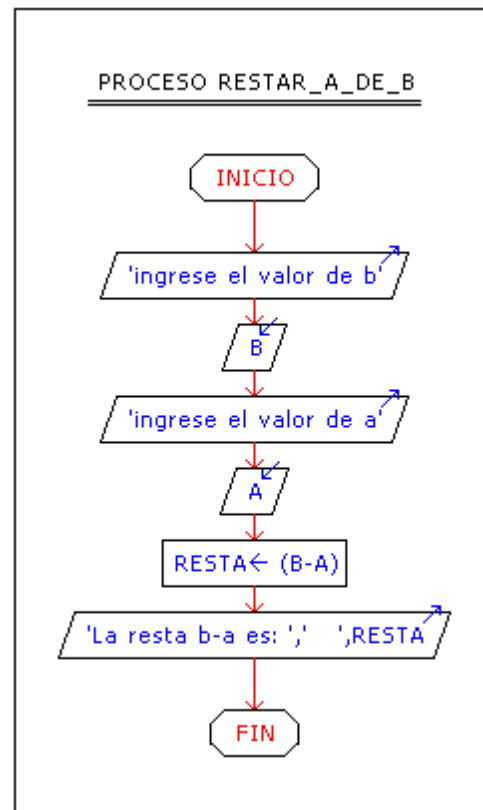
1 //Algoritmo para Restar dos números desarrollac
2 Proceso restar_a_de_b //Proce
3   Escribir "ingrese el valor de b"; //muest
4   Leer b; //ingre
5   Escribir "ingrese el valor de a";
6   Leer a;
7   Resta<-(b-a);
8   Escribir "La resta b-a es: ", " ",Resta;
9 FinProceso
10

```

```

C:\Archivos de programa\PSeInt\psei
*** Ejecucion Iniciada. ***
ingrese el valor de b
> 5
ingrese el valor de a
> 10
La resta b-a es: -5
*** Ejecucion Finalizada. ***

```



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

```

```

1 // Algoritmo para Calcular el Cociente (C) y Residuo (R) de A entre B. Desarrollado.
2 Proceso Calcular_Cociente_Residuo //Proceso
3 Escribir "Programa para calcular el Cociente (C) y el Residuo (R) de A entre B";
4 Escribir "Ingrese el valor de A: ";
5 Leer A; //ingresa por teclado el valor de A
6 Escribir "Ingrese el valor de B: ";
7 Leer B; //ingresa por teclado el valor de B
8 Cociente<-A/B; //Cociente
9 Residuo<-A Mod B; //Residuo ; emplear la función Mod
10 Escribir "El cociente(C) de A entre B es:", " ",Cociente;
11 Escribir "El residuo(R) de A entre B es: ", " ",Residuo;
12
13 FinProceso
14

```

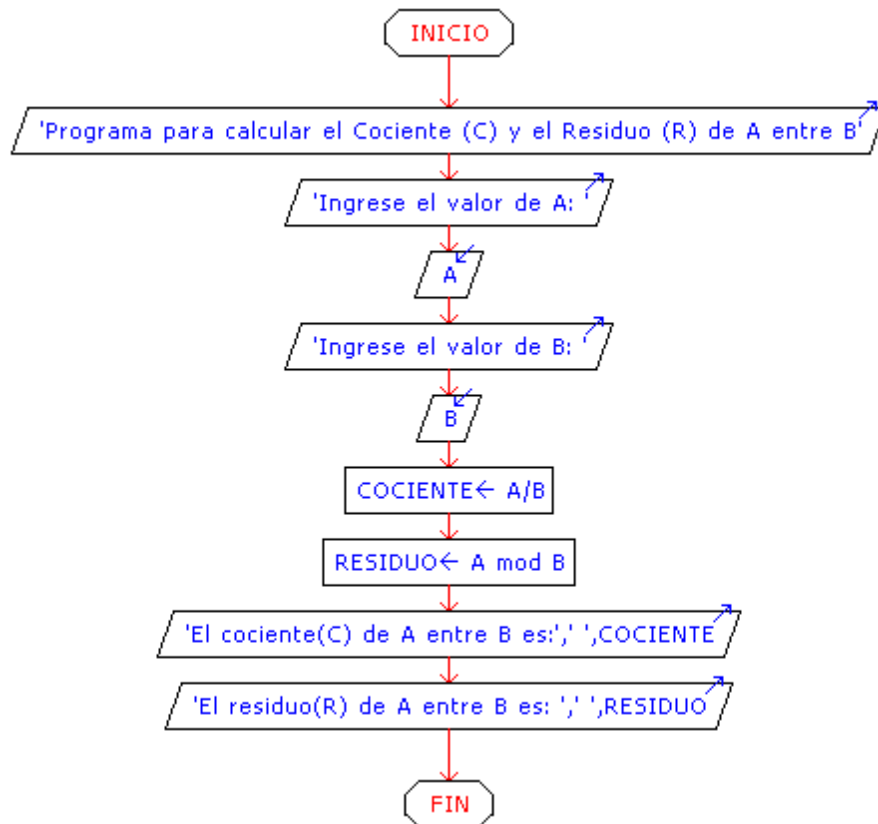
Resultados

★ Calculador_Cociente_Residuo: Ejecucion Finalizada

```

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. ***

```



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
  
```

```

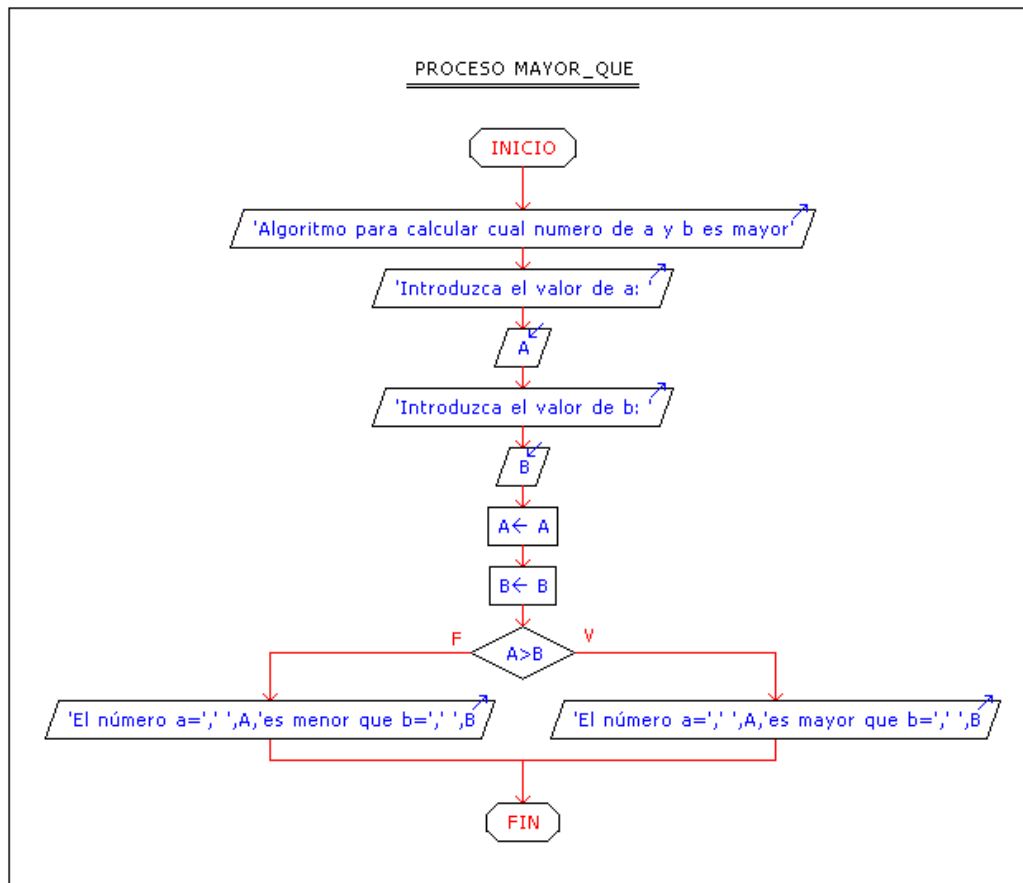
1 //Algoritmo que determina el mayor de dos números 'a' y 'b'. Desarrollado por RPC
2 Proceso mayor_que //proceso mayor_que
3   Escribir "Algoritmo para calcular cual numero de a y b es mayor";
4   Escribir "Introduzca el valor de a: " //muestra en pantalla la instrucción
5   Leer a; //ingresa por teclado 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\PSelnt\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

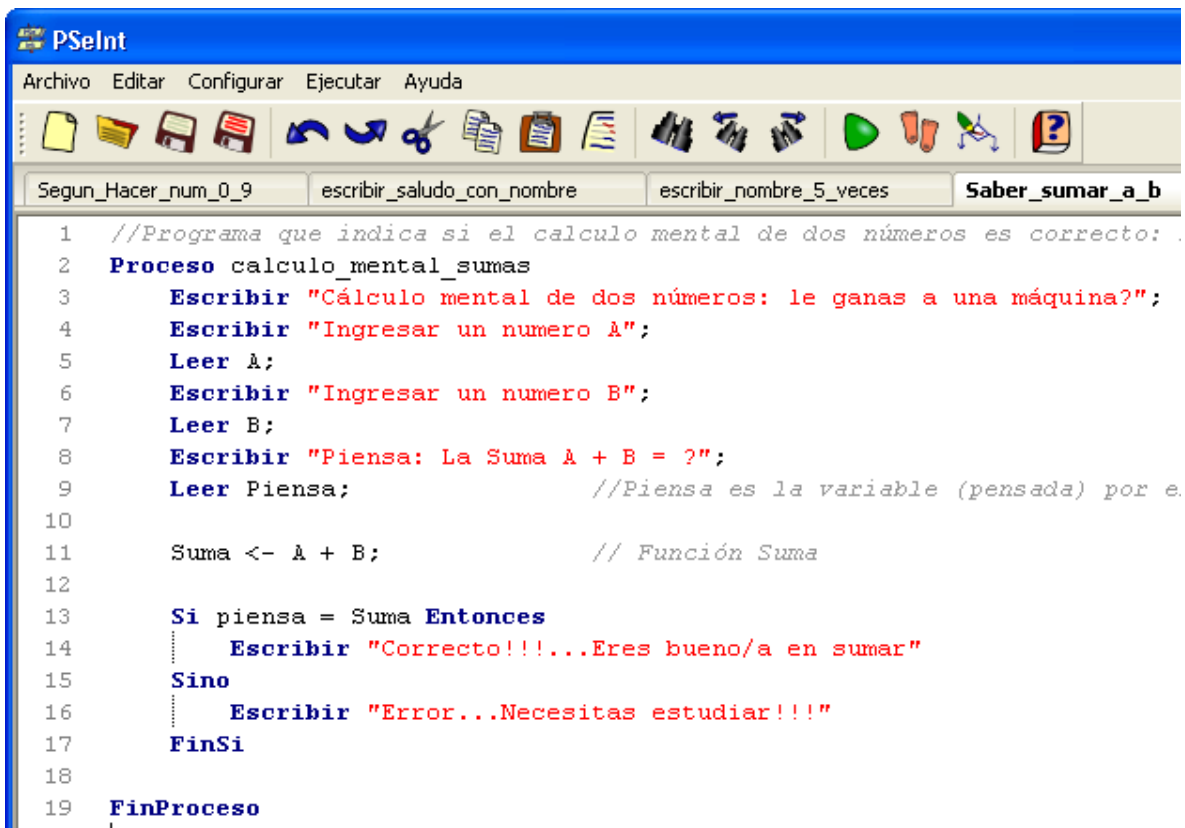
 Escribir "Correcto!!!...Eres bueno/a en sumar"

 Sino

 Escribir "Error...Necesitas estudiar!!!"

 FinSi

FinProceso



```

1  //Programa que indica si el calculo mental de dos números es correcto:
2  Proceso calculo_mental_sumas
3      Escribir "Cálculo mental de dos números: le ganas a una máquina?";
4      Escribir "Ingresar un numero A";
5      Leer A;
6      Escribir "Ingresar un numero B";
7      Leer B;
8      Escribir "Piensa: La Suma A + B = ?";
9      Leer Piensa;          //Piensa es la variable (pensada) por e.
10
11     Suma <- A + B;          // Función Suma
12
13     Si piensa = Suma Entonces
14         Escribir "Correcto!!!...Eres bueno/a en sumar"
15     Sino
16         Escribir "Error...Necesitas estudiar!!!"
17     FinSi
18
19 FinProceso

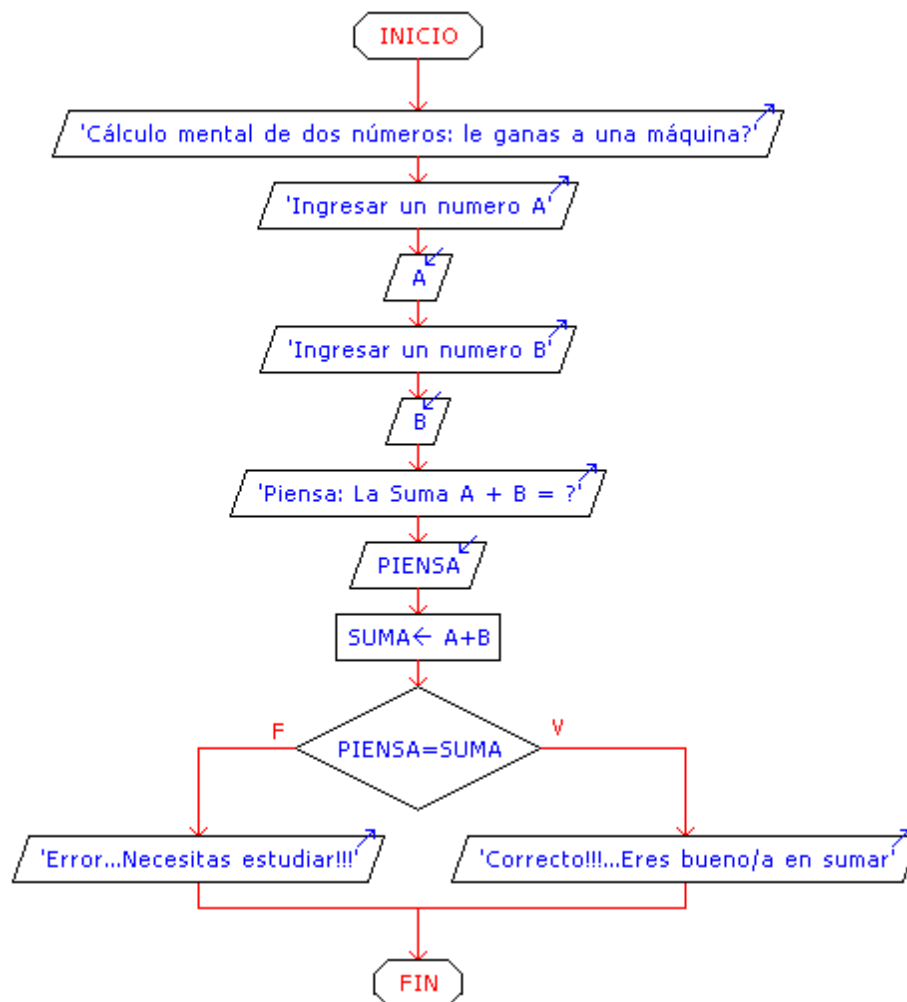
```

```

C:\Archivos de programa\PSelnt\pseint.exe
*** Ejecucion Iniciada. ***
Cálculo mental de dos números: le ganas a una máquina?
Ingresar un numero A
> 4
Ingresar un numero B
> 56
Piensa: La Suma A + B = ?
> 60
Correcto!!!...Eres bueno/a en sumar
*** Ejecucion Finalizada. ***

```

PROCESO CALCULO_MENTAL_SUMAS



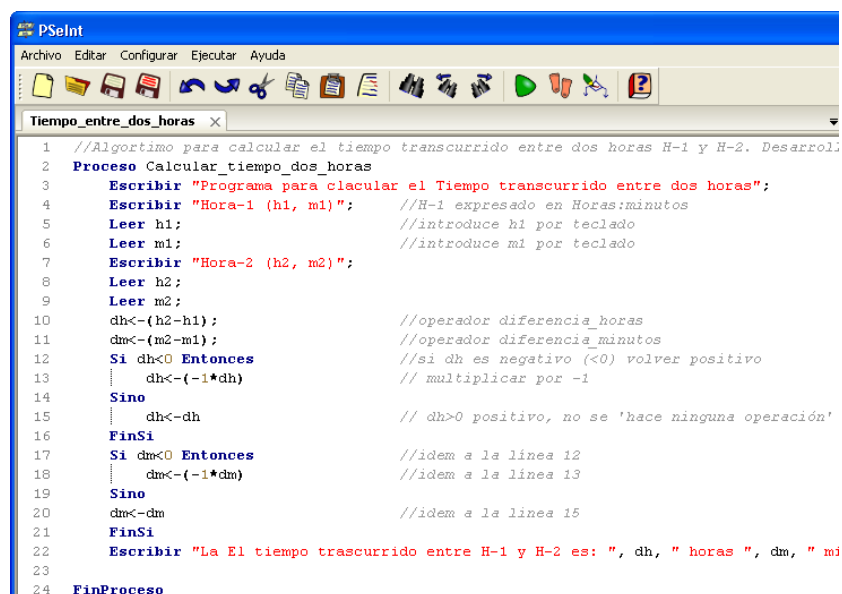
13. Determinar el tiempo transcurrido entre dos horas del día.

//Algoritmo 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

```



```

PSeInt
Archivo  Editar  Configurar  Ejecutar  Ayuda
Tiempo_entre_dos_horas x
1 //Algoritmo para calcular el tiempo transcurrido entre dos horas H-1 y H-2. Desarroll
2 Proceso Calcular_tiempo_dos_horas
3     Escribir "Programa para clacular el Tiempo transcurrido entre dos horas";
4     Escribir "Hora-1 (h1, m1)"; //H-1 expresado en Horas:minutos
5     Leer h1; //introduce h1 por teclado
6     Leer m1; //introduce m1 por teclado
7     Escribir "Hora-2 (h2, m2)";
8     Leer h2;
9     Leer m2;
10    dh<-(h2-h1); //operador diferencia_horas
11    dm<-(m2-m1); //operador diferencia_minutos
12    Si dh<0 Entonces //si dh es negativo (<0) volver positivo
13        dh<-(-1*dh) // multiplicar por -1
14    Sino
15        dh<-dh // dh>0 positivo, no se 'hace ninguna operación'
16    FinSi
17    Si dm<0 Entonces //idem a la línea 12
18        dm<-(-1*dm) //idem a la línea 13
19    Sino
20        dm<-dm //idem a la línea 15
21    FinSi
22    Escribir "La El tiempo trascurrido entre H-1 y H-2 es: ", dh, " horas ", dm, " m
23
24 FinProceso

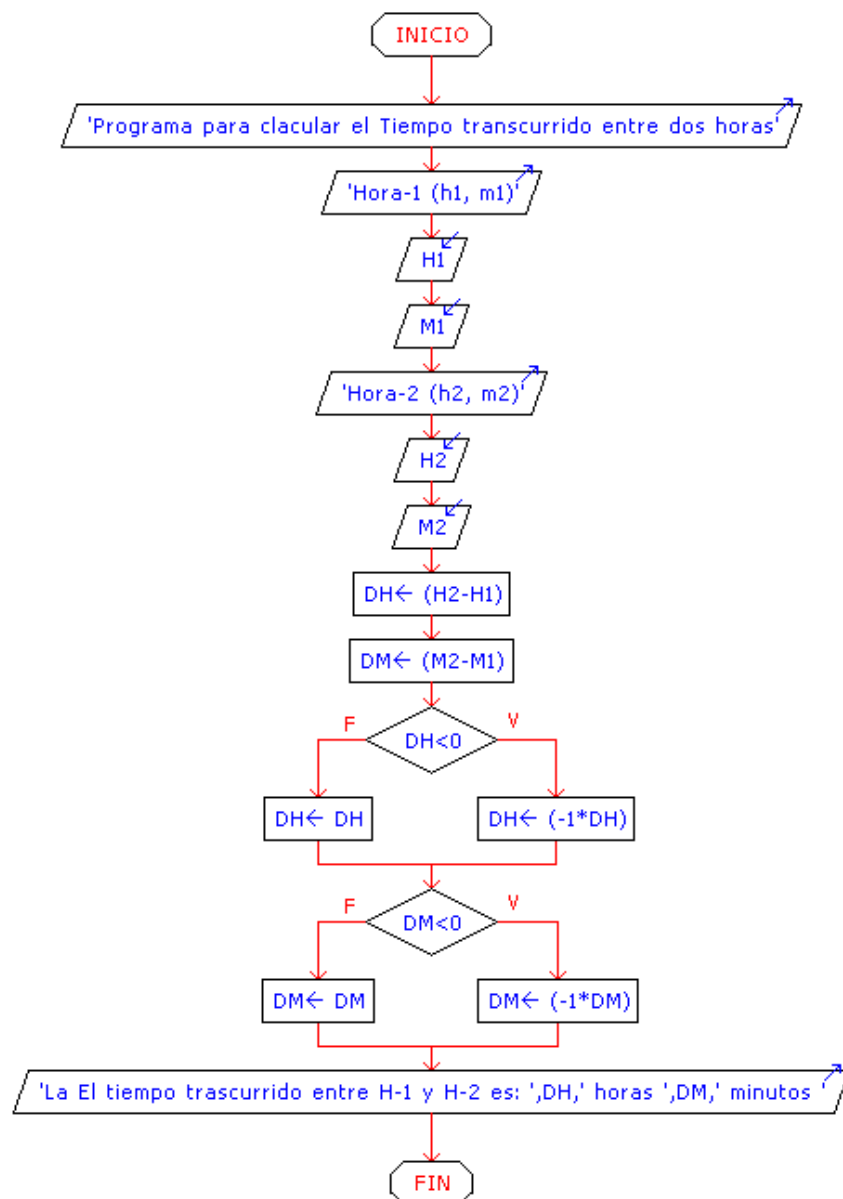
```

```

C:\Archivos de programa\PSelnt\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. ***

```

PROCESO CALCULAR_TIEMPO_DOS_HORAS



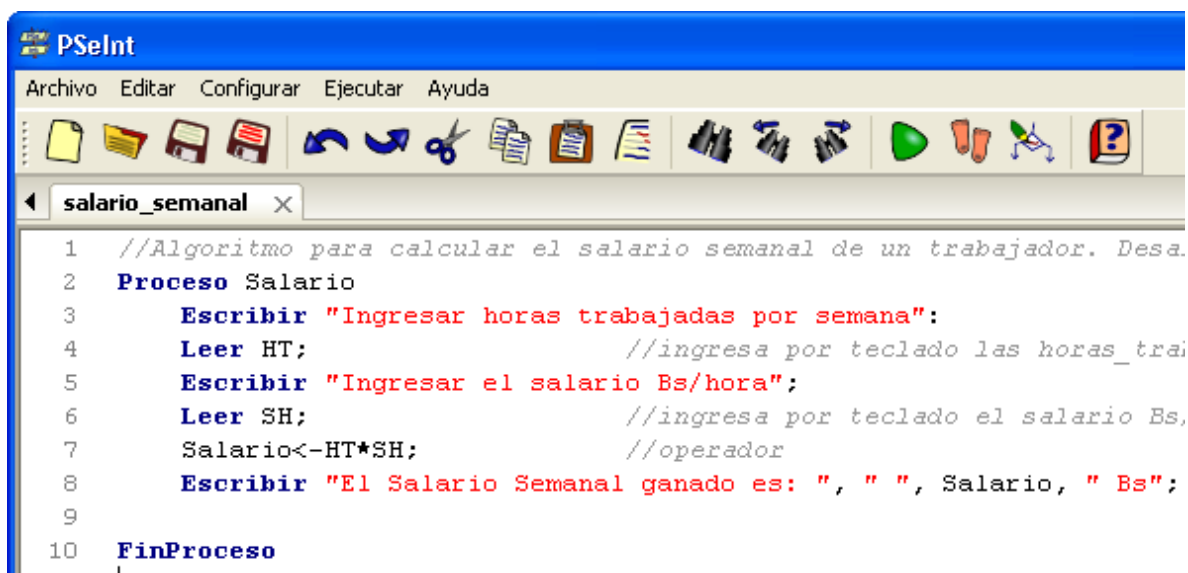
14. Calcular el salario semanal de un empleado

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

Proceso Salario

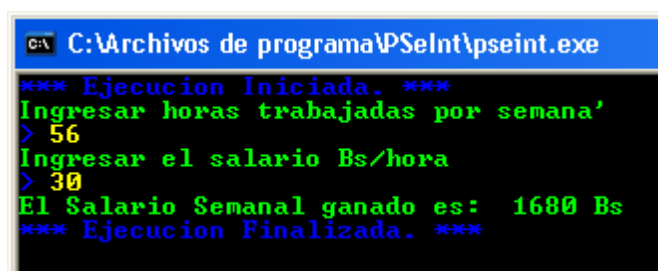
Escribir "Ingresar horas trabajadas por semana":
 Leer HT; //ingresa por teclado las horas_trabajadas_semana
 Escribir "Ingresar el salario Bs/hora";
 Leer SH; //ingresa por teclado el salario Bs/hora
 $\text{Salario} \leftarrow \text{HT} * \text{SH}$; //operador
 Escribir "El Salario Semanal ganado es: ", " ", Salario, " Bs";

FinProceso



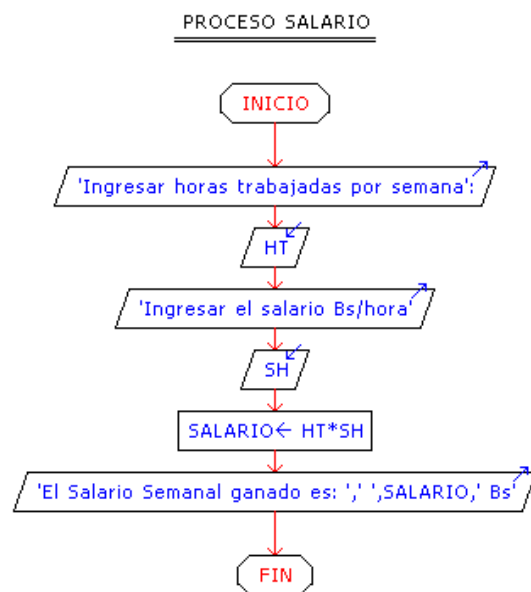
```

1 //Algoritmo para calcular el salario semanal de un trabajador. Desa.
2 Proceso Salario
3   Escribir "Ingresar horas trabajadas por semana":
4   Leer HT; //ingresa por teclado las horas_trabajadas_semana
5   Escribir "Ingresar el salario Bs/hora";
6   Leer SH; //ingresa por teclado el salario Bs/hora
7   Salario<-HT*SH; //operador
8   Escribir "El Salario Semanal ganado es: ", " ", Salario, " Bs";
9
10 FinProceso
  
```



```

C:\Archivos de programa\PSeInt\pseint.exe
*** Ejecucion Iniciada. ***
Ingresar horas trabajadas por semana'
> 56
Ingresar el salario Bs/hora
> 30
El Salario Semanal ganado es: 1680 Bs
*** Ejecucion Finalizada. ***
  
```



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;

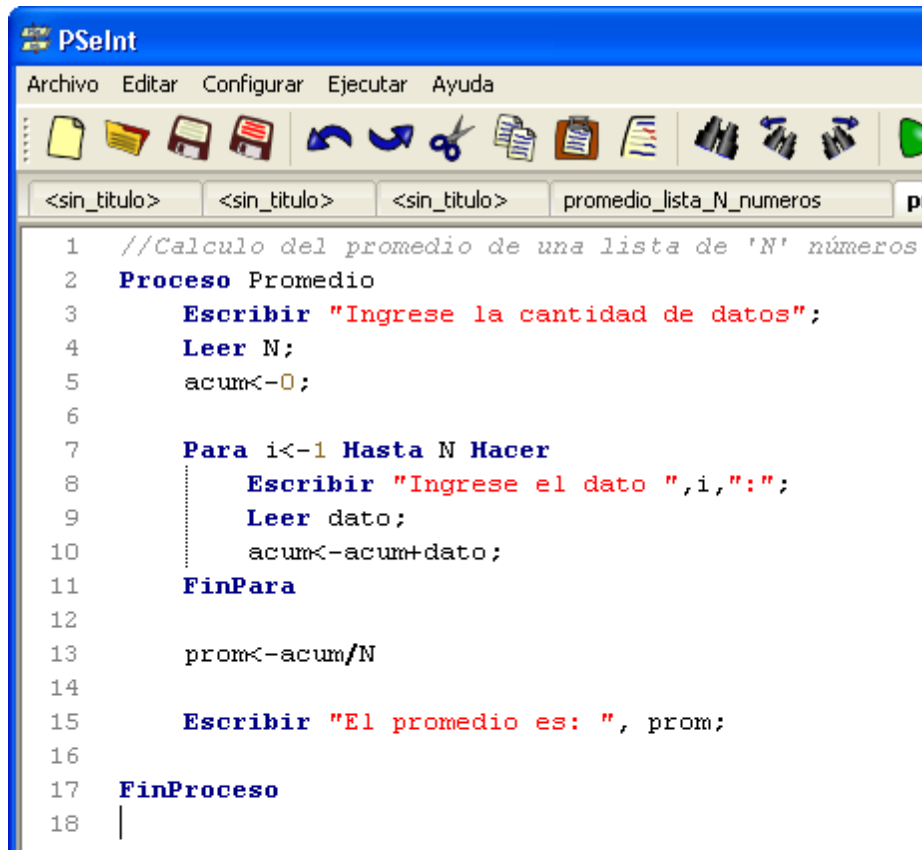
 acum<-acum+dato;

 FinPara

 prom<-acum/N

 Escribir "El promedio es: ", prom;

FinProceso



```

1  //Calculo del promedio de una lista de 'N' números
2  Proceso Promedio
3      Escribir "Ingrese la cantidad de datos";
4      Leer N;
5      acum<-0;
6
7      Para i<-1 Hasta N Hacer
8          Escribir "Ingrese el dato ",i,":";
9          Leer dato;
10         acum<-acum+dato;
11     FinPara
12
13     prom<-acum/N
14
15     Escribir "El promedio es: ", prom;
16
17 FinProceso
18 |
  
```

```

C:\Archivos de programa\PSelInt\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. ***

```

