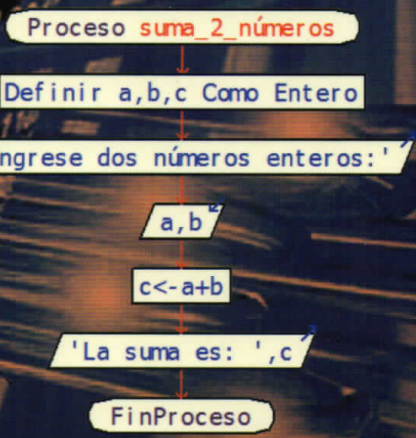
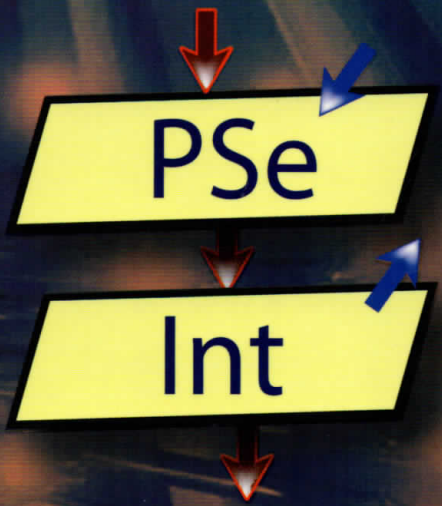
1era edición

UNIVERSIDAD POLITÉCNICA ESTATAL DEL CARCHI

UPEC

Fundamentos de Programación usando PSeInt





IVÁN GARCÍA S.

2014

Fundamentos de Programación Usando PSeInt

UNIVERSIDAD POLITÉCNICA ESTATAL DEL CARCHI

Dr. Hugo Ruíz Enríquez
RECTOR

AUTOR:

MSc. Iván García Santillán
Docente Titular Auxiliar UPEC (Carrera de Informática; Administración de Empresas)

LIBRO REVISADO POR:

MSc. Marco Pusdá (Universidad Técnica del Norte - Ecuador)
MSc. Patricio Iles (Universidad Técnica del Norte - Ecuador)

ISBN : 978-9942-914-18-7

Derecho de Autor Nº 045156
Primera Edición
Diciembre 2014
Tulcán - Carchi - Ecuador

Comisión de Publicaciones UPEC
CP.UPEC. 0.005-2014

DISEÑO Y DIAGRAMACIÓN:

IMAGO
Veracruz N34-90 y Av. América
3316378 - 0996017113
graficasimago@hotmail.com
Quito - Ecuador

PRÓLOGO

El libro de *Fundamentos de programación usando PSeInt* trata sobre el análisis y diseño de algoritmos, es decir, que se centra en la lógica de la programación más que en la codificación utilizando algún lenguaje de programación específico como C, C++, Java, C#, PHP, VB, pascal, python, javaScript, Matlab. Esto permite desarrollar en los estudiantes de cualquier disciplina, las capacidades mentales necesarias para poder programar computadoras. Para el diseño de algoritmos se hace uso de los diagramas de flujo y pseudocódigo utilizando el programa *PSeInt*, que es software libre. Además se utiliza los paradigmas de la programación estructurada y modular (descendente o top down) que son la base que sustentan la programación orientada a objetos. Esta última (POO) queda fuera del alcance de este libro.

El libro puede ser utilizado en la asignatura de fundamentos de programación de la carrera de Informática o afines, así como también en otras disciplinas como Electrónica, Mecatrónica, Industrial, Administración de Empresas, etc. Mantiene un lenguaje fácil de entender para los estudiantes y contiene una variedad de ejercicios resueltos y propuestos. Los enunciados de los ejercicios planteados en el texto resultan ser de la experiencia docente del autor, así como de la selección adecuada de otras fuentes de consulta como libros e Internet.

INTRODUCCIÓN

La presente obra denominada *Fundamentos de Programación usando PSeInt* se ha creado tanto para estudiantes y profesionales de informática o afines como para otras disciplinas con el objetivo de proporcionar una guía teórica y práctica que sea fácil y, a su vez, no pierda el rigor técnico de la programación de computadoras.

El texto está dividido en ocho capítulos, los cuales cubren los fundamentos de la programación. En el capítulo I se hace una introducción a la Programación, con tópicos como el ciclo de la programación, paradigmas de programación, tipos de aplicaciones y lenguajes de programación. El capítulo II hace una introducción al programa PSeInt y su entorno de trabajo. En el capítulo III se tratan los algoritmos básicos utilizando Diagramas de Flujo y el enfoque de la programación estructurada. En el capítulo IV se presentan algoritmos más complejos utilizando Pseudocódigo. En el capítulo V se analiza el diseño modular conocido también como análisis descendente o Top Down. Aquí también se utiliza el paso de parámetros por valor y por referencia. En el capítulo VI se trabaja con los arreglos unidimensionales y bidimensionales. En el capítulo VII se usa la recursividad como una técnica diferente para resolver problemas computacionales. En el capítulo VIII trata la conversión de los algoritmos a código fuente utilizando los lenguajes de programación disponibles en PSeInt (C, C++, java, C#, PHP, VB.net, pascal, python, javaScript, Matlab). Los ocho capítulos son indispensables para el análisis, diseño, codificación y depuración de programas de computadora.

El libro en total contiene 65 ejercicios resueltos y 68 ejercicios propuestos. En la resolución de los ejercicios no necesariamente se presenta el mejor algoritmo. Por lo tanto, queda para el estudiante el análisis detallado de los algoritmos resueltos para entender su lógica de programación y proponer alternativas de solución más eficientes. Para la resolución de ciertos ejercicios propuestos, es posible que el estudiante deba consultar ciertos conceptos o fórmulas en otras fuentes de consulta pertinente al problema respectivo. Esto le permitirá desarrollar la habilidad de autoaprendizaje, esencial en un profesional en formación.

Adicionalmente, existe el buzón de correo electrónico *ivan.garcia@upec.edu.ec* para interactuar con sus lectores y recibir cualquier sugerencia.

El Autor

AGRADECIMIENTO

A toda la comunidad que conforma la Universidad Politécnica Estatal del Carchi que ha hecho posible la culminación de la presente obra.

A mis alumnos de la asignatura de Fundamentos de Programación de la Carrera de Informática y Administración de Empresas, por permitirme poner de manifiesto todo lo aprendido en estos últimos años de docencia.

A Pablo Novara, creador del *PSeInt*, por acoger algunas sugerencias para el mejoramiento del programa.

A todos ellos mis sentimientos de aprecio y gratitud.

DEDICATORIA

A mi esposa Sary y mis hijos Andy y Yamileth, testigos y motivo de mi esfuerzo, constancia y dedicación.

Tabla de contenido

CAPÍTULO I

INTRODUCCIÓN A LA PROGRAMACIÓN

En este capítulo usted encontrará:

- *La programación*
- *El ciclo de la programación*
- *Los paradigmas de la programación*
  - *Estructurada*
  - *Modular (descendente o top-down)*
  - *Orientada a objetos*
- *Tipos de aplicaciones*
  - *Consola*
  - *Escritorio*
  - *Web*
  - *Móvil*
- *Lenguajes de programación*
- *Tareas propuestas*

1. Introducción a la Programación

1.1 La Programación

La programación de computadoras surge en los años 50 (López, 2011) con la finalidad de resolver problemas específicos con la ayuda del computador. Por ejemplo: calcular el impuesto a la renta de una persona natural, fondos de reserva de un empleado, décimo tercer sueldo, etc.

Con la evolución tecnológica (principalmente de la electrónica) se empieza a construir computadoras con mayor capacidad de procesamiento y almacenamiento, y con ello la proliferación de lenguajes de programación cada vez más fáciles y potentes para el desarrollo de programas. Algunos de los lenguajes de programación son: C, C++, Java, C#, VB, php, pascal, Python, javaScript, entre otros.

La Programación es el proceso de analizar, diseñar, codificar o traducir, depurar y mantener el código fuente de los programas computacionales, los cuales son escritos en algún lenguaje de programación (Joyanes, 2008).

1.2 El ciclo de la programación

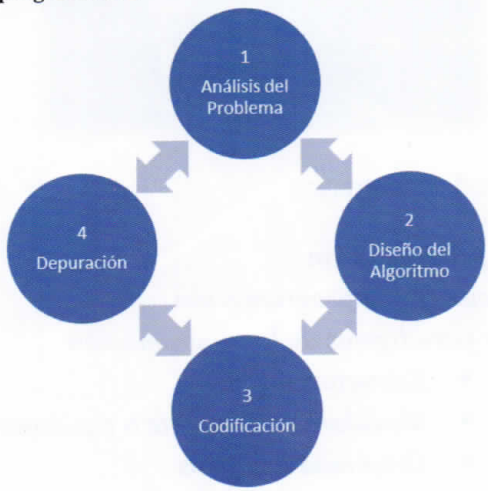


Figura 1.1 El ciclo de Programación
Fuente: López (2009)

En el **análisis del problema** se debe comprender con precisión en qué consiste el problema, es decir, cuáles son los datos disponibles (entrada) y resultados (salida), las restricciones (que está permitido o prohibido hacer y/o utilizar) y los procesos (pasos, fórmulas) necesarios para convertir dichos datos (insumos) en resultados.

Por ejemplo: Realizar el análisis del problema para obtener el área de un círculo.

Entrada: Radio del círculo
Salida: Área del círculo
Proceso: Calcular el área del círculo a través de la fórmula: $A = \pi r^2$

En el **Diseño** se elabora un algoritmo (diagrama de flujo o pseudocódigo) que resuelva el problema planteado. En este libro se utilizará el programa ***PSeInt*** para el diseño de los algoritmos (véase el capítulo 2).

En la **Codificación**, el algoritmo se traduce a un lenguaje de programación específico y se obtiene un programa. Puede utilizarse java, C++, C#, php, etc.

En la **depuración**, los resultados se prueban y validan, se afinan ciertos detalles y corrigen ciertos errores: *sintaxis, lógicos, en tiempo de ejecución*. La sintaxis son las reglas que rigen el uso de las palabras, símbolos y la puntuación de cada lenguaje de programación. Los *errores de sintaxis* son los más fáciles de detectar y corregir. Los *errores lógicos* producen como resultado una salida incorrecta y son más difíciles de detectar y corregir. Los *errores en tiempo de ejecución* se producen cuando el programa está en ejecución y se presenta alguna anomalía como por ejemplo: mal ingreso de los datos de entrada, no está disponible la red, dispositivos de almacenamiento, impresión, etc.

La **documentación** de un programa también es una parte fundamental y se la puede hacer en cada etapa o al final.

Otra manera más detallada del ciclo de desarrollo del programa es el propuesto por Farrell (2013):

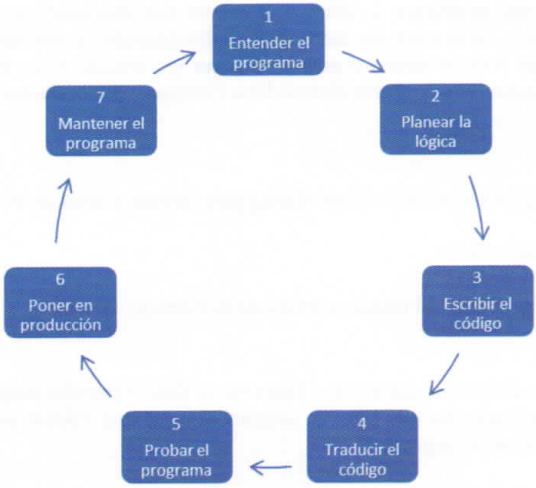


Figura 1.2 El ciclo del desarrollo del programa
Fuente: Farrell (2013)

1.3 Los paradigmas de la Programación

Existen algunas técnicas de programación cuya finalidad consiste en mejorar el proceso de creación de programas, así como su mantenimiento y documentación. Algunas de ellas son: programación estructurada, modular y orientada a objetos.

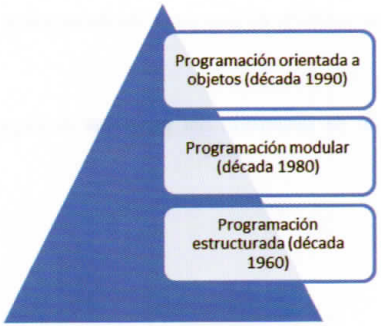


Figura 1.3 Paradigmas de la programación
Fuente: Adaptado de López (2011)

1.3.1 La programación estructurada

Según López (2011), la *programación estructurada* es la base en la que se sustenta la programación orientada a objetos, por lo que se debe dominar primero. Los elementos de la programación estructurada principalmente son: tipos de datos, estructuras de control (secuenciación, selección, repetición).

Los tipos de datos principalmente son: número (entero, real), carácter/cadena de caracteres, fecha, booleano (verdadero, falso).

La ***secuenciación*** es la capacidad de ejecutar las instrucciones secuenciales una tras otras. La ***selección*** es la capacidad de escoger, entre alternativas, si algo se ejecuta o no, y la ***repetición*** es la capacidad de ejecutar una o más instrucciones varias veces.

1.3.2 La programación modular (descendente o top-down)

Conocida también como diseño *descendente o Top-Down*, se puede considerar como una evolución de la programación estructurada. Divide un programa en varios módulos (subprogramas) más simples con la finalidad de hacerlo más comprensible y manejable, así como solucionar problemas más grandes y complejos de una mejor forma. Cada módulo realiza una tarea específica y algunos de ellos pueden necesitar de otros para funcionar. Un módulo además puede contener varios métodos: procedimientos o funciones, (véase la figura 1.4). En resumen, esta técnica permite diseñar programas bien estructurados, documentados, fáciles de entender y mantener.

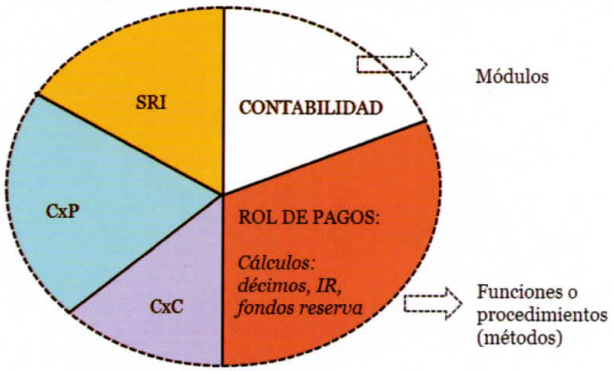


Figura 1.4 Programa de gestión contable

1.3.3 La programación orientada a objetos (POO)

La POO se puede concebir como una evolución de la programación modular, debido al requerimiento de programas más ambiciosos (aplicaciones gráficas, móviles, web). Aquí se caracteriza por los conceptos de objetos, clases, encapsulación, herencia y polimorfismo. Un programa consiste en un conjunto de objetos que se comunican entre sí a través de mensajes. Cada objeto se compone de datos y métodos como se muestra en la figura 1.5.

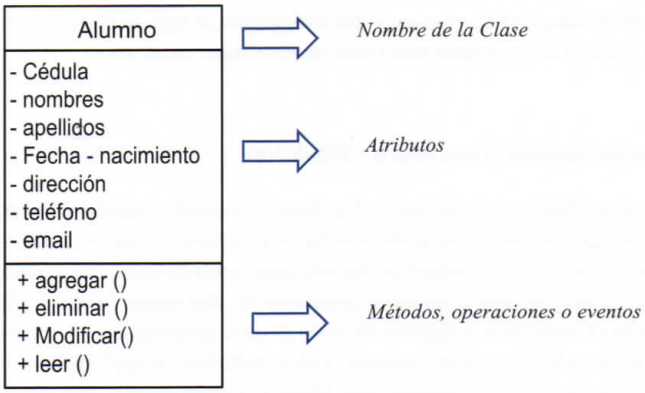


Figura 1.5 Clase Alumno

En este libro usaremos los paradigmas de la programación estructurada y modular para la resolución de los ejercicios. Queda fuera del alcance la programación orientada a objetos.

1.4 Tipos de Aplicaciones

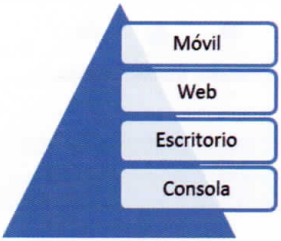


Figura 1.6 Tipos de Aplicaciones

1.4.1 Aplicación de Consola

Estas aplicaciones se ejecutan a través de una interfaz de línea de comandos (CLI), es decir, en una ventana de MS-DOS, Gnome, KDE, etc. No tiene interfaz gráfica y se ejecuta en modo texto. Tienen la ventaja de consumir menos recursos del equipo por lo que suelen ejecutarse rápidamente. Algunas aplicaciones de consola son: programa de contabilidad Tini, Tmax, etc.

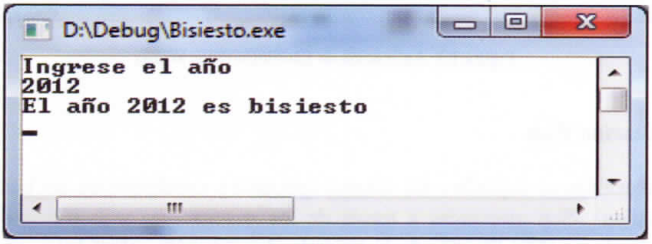


Figura 1.7 Aplicación de Consola (Verifica si el año ingresado es bisiesto)

1.4.2 Aplicación de Escritorio

Estas aplicaciones son ejecutadas y administradas directamente por el sistema operativo del equipo (Windows, Linux, Mac OS) y su rendimiento depende de las configuraciones del propio hardware (RAM, procesador, disco duro, video, etc.). Además estas aplicaciones están basadas en una interfaz gráfica (GUI) muy atractiva e intuitiva para el usuario. Algunas aplicaciones de escritorio son: Word, Excel, Access, Power Point, SPSS Statistics, AutoCAD, Photoshop, Illustrator, Winamp, Virtual DJ, etc.

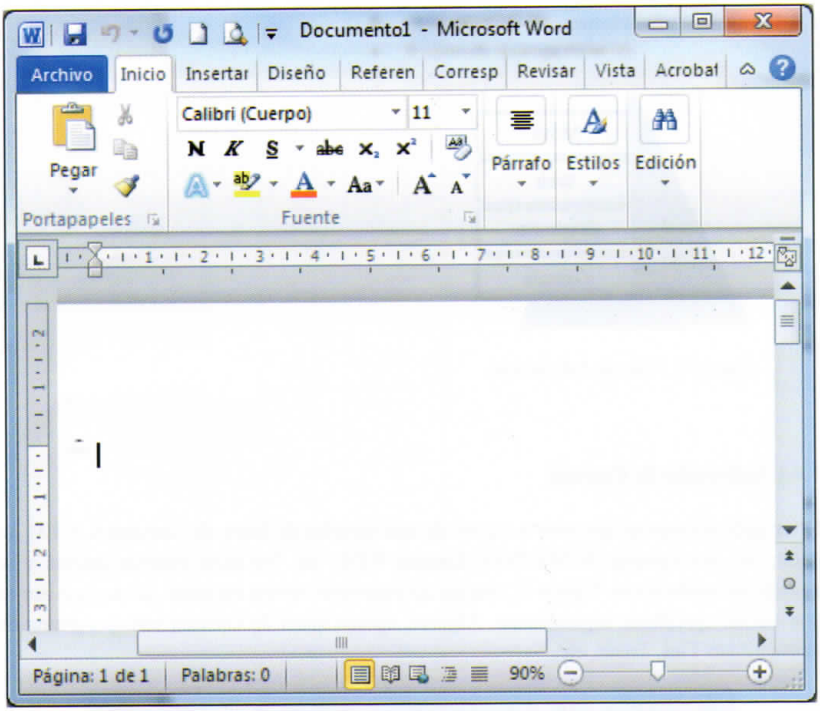


Figura 1.8 Aplicación de Escritorio (MS. Word)

1.4.3 Aplicación Web

Estas aplicaciones no dependen del sistema operativo y configuración del hardware del propio equipo. Son ejecutadas a través de los navegadores web (Google Chrome, Internet Explorer, Mozilla Firefox) tipiando la dirección electrónica (URL). La administración de estas aplicaciones se realiza por el proveedor o propietario de dicha aplicación. Algunas aplicaciones web son los servicios en línea del: SRI (servicio de rentas internas), IESS (instituto ecuatoriano de seguridad social), banca virtual, Facebook, Twitter, E-Bay, Amazon, MercadoLibre, etc.

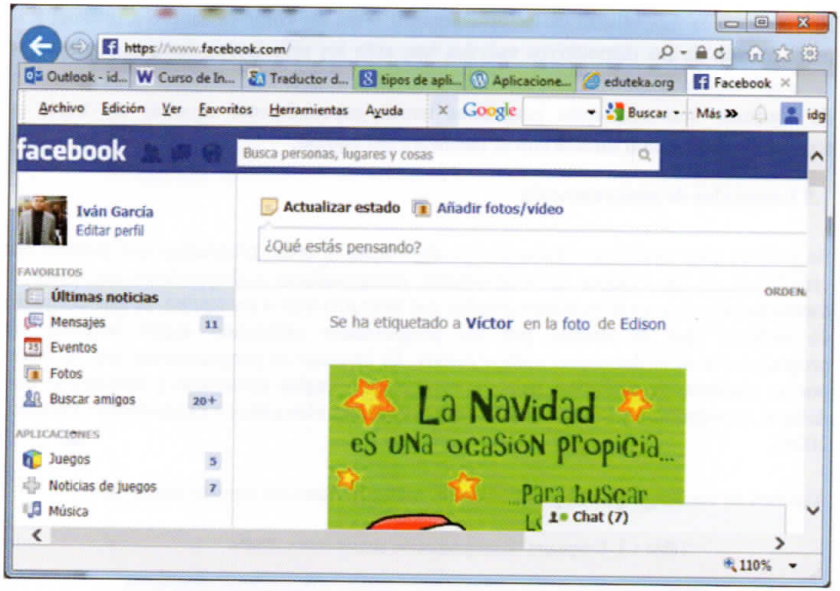


Figura 1.9 Aplicación Web (Facebook)

1.4.4 Aplicación Móvil

Estas aplicaciones están diseñadas para ser ejecutadas en teléfonos inteligentes, tabletas y otros dispositivos móviles. Se debe tener en cuenta las limitaciones de estos dispositivos como: procesador, pantalla, sistema operativo (Android, Symbian, Windows phone, BlackBerry, iOS, otros), etc. Algunas aplicaciones móviles son los servicios en línea de la banca virtual, juegos, utilidades, etc.

Figura 1.10 Aplicación Móvil

En resumen, la flexibilidad de las aplicaciones Web, así como el acceso a Internet y la proliferación de los dispositivos móviles han sido los principales motivos por los que estas aplicaciones son cada vez más utilizadas. En cambio, las aplicaciones de consola y de escritorio son más aptas para otras tareas, especialmente aquellas en donde se requiere la interacción directa con el hardware del equipo.

1.5 Lenguajes de programación

Se utilizan para codificar (traducir) los algoritmos y crear programas que puedan ser ejecutados en los equipos (principalmente computadoras o dispositivos móviles). A través de ellos los seres humanos puedan dar instrucciones a un equipo. A este conjunto de órdenes que es escrito por un programador utilizando algún lenguaje de programación se le denomina ***código fuente***. El lenguaje de programación está formado por un conjunto de símbolos, palabras reservadas y reglas sintácticas y semánticas que definen su estructura y el significado o sentido de sus elementos y expresiones (Joyanes, 2008).

Algunos de los lenguajes de programación más populares en nuestro medio son:

Tabla 1.1 Lenguajes de programación comúnmente usados

Lenguaje	Tipos de aplicación	Licenciamiento
VB.net	Consola, escritorio, web, móvil	Software privativo
C#.net	Consola, escritorio, web, móvil	Software privativo
C++	Consola, escritorio	Software libre
Java	Consola, escritorio, web, móvil	Software libre
Oracle	Web, móvil	Software privativo
PHP	Web, móvil	Software libre
Matlab	Consola, escritorio, web, móvil	Software privativo

Estos lenguajes de programación se llaman de ***alto nivel*** y se caracterizan por expresar los algoritmos de una manera adecuada (sencillo y comprensible) para el ser humano, en contraposición con los lenguajes de ***bajo nivel*** que son entendibles por las máquinas (computador).

Algunos lenguajes de programación son ***software libre***, es decir, el código fuente esté disponible para que a más de su uso, cualquiera pueda estudiarlo, modificarlo o redistribuirlo. En contraposición con otros que son ***software privativo*** en donde, por lo general, se permite su uso bajo ciertas condiciones y además se pague un canon al propietario o titular.

Según López (2011), no se debe enseñar al estudiante a programar directamente con un lenguaje de programación, debido al doble problema que resulta aprender la lógica de

programación y la sintaxis del lenguaje al mismo tiempo. Por esta razón, aquí se hará énfasis en los algoritmos, a través de diagramas de flujo y pseudocódigo utilizando el programa PSeInt.

1.6 Tareas propuestas

1.6.1 Realice un cuadro sinóptico con la siguiente información solicitada: Lenguaje de programación, propietario, licencia (privativo, software libre), tipo de aplicaciones que permite desarrollar, % de uso (a nivel de Ecuador, Latinoamérica, mundial), curva de aprendizaje (véase figura 1.11), sitio web oficial.

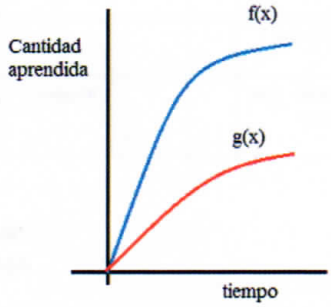


Figura 1.11 Curva de aprendizaje;
f(x) indica que en poco tiempo se aprende mucho;
g(x) indica que en mucho tiempo se aprende poco

1.6.2 Realice un cuadro sinóptico sobre software empresarial existente en su profesión, con la siguiente información: nombre del software, costo, sitio web oficial, tipo de aplicación, módulos que posee (funcionalidades), lenguaje de programación y base de datos que utiliza, país de origen.

1.6.3 Considerando la arquitectura de 3 capas de una aplicación web: cliente, servidor de aplicaciones y bases de datos (véase figura 1.12), complete la tabla con la información que mejor se ajusta, de acuerdo al tipo de licencia (privativo, software libre) de la tecnología indicada.

Tecnología	Servidor de base de datos	Servidor de Aplicaciones / Web	Cliente (navegador)	Sistema Operativo	Extensión de la página web	Costo estimado en licencias
Java	*MySQL* *Postgres*	*TomEE* *JBoss*	*Firefox* *Chrome*	*Linux*	**.jsp*	*0.00*
C#.net **VB.net**			*IE*	*Windows*		
Php					**.php*	
Oracle		*WebLogic*				
Matlab		*Matlab Web Server*		*Windows*		

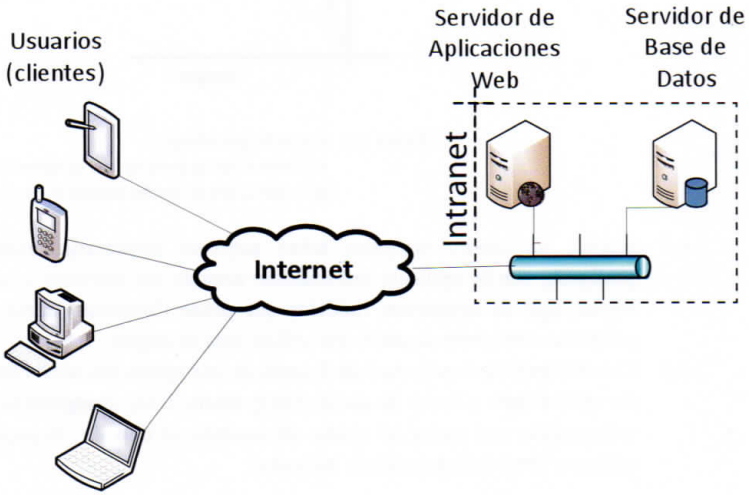


Figura 1.12 Arquitectura de 3 capas de una aplicación web

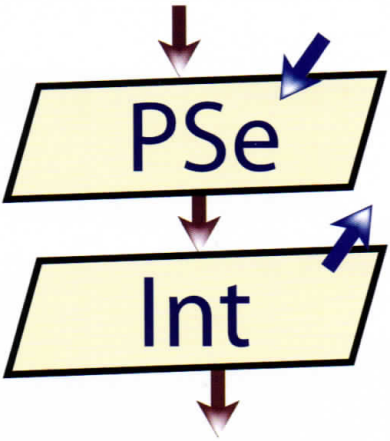
1.6.4 Realice un cuadro sinóptico sobre las ventajas y desventajas del software libre y privativo; tipos de licencia (GNU, GPL, CopyLeft, Creative Common, etc.) y decreto 1014 de software libre en Ecuador.

1.6.5 Realice un cuadro sinóptico sobre la propiedad intelectual del software y los organismos de control y/o apoyo en el Ecuador: IEPI (Instituto Ecuatoriano de Propiedad Intelectual), AESOFT (Asociación Ecuatoriana de Software), ASLE (Asociación de Software Libre del Ecuador).

1.6.6 Si utilizar software libre en una empresa, tiene varias ventajas importantes y además contribuye a reducir los costes, ¿por qué existen muchas empresas que se deciden por el software privativo?

CAPÍTULO II

INTRODUCCIÓN A PseInt



En este capítulo usted encontrará:

- *Qué es PSeInt*
- *El entorno de trabajo*
- *Perfiles (personalización)*
- *Editor de Diagramas de flujo*
- *Editor de Pseudocódigo*
- *Ejecución del algoritmo*
- *Guardar el algoritmo*
- *Exportación a código fuente*
- *Tareas propuestas*

2. Introducción a PSeInt

2.1 Qué es PSeInt

Según Novara (2014):

> PSeInt viene de las palabras '***PSeudo Intérprete***' y es una herramienta para asistir a un estudiante en sus primeros pasos en programación. Mediante un simple e intuitivo pseudolenguaje en español (complementado con un editor de diagramas de flujo), le permite centrar su atención en los conceptos fundamentales de la algoritmia computacional, minimizando las dificultades propias de un lenguaje y proporcionando un entorno de trabajo con numerosas ayudas y recursos didácticos.

El programa es una aplicación de escritorio y se encuentra disponible libremente para Windows, Linux y Mac OS desde el sitio web oficial http://pseint.sourceforge.net/. La última versión disponible al momento de la edición del libro es la *20140921* con la cual se trabaja aquí.

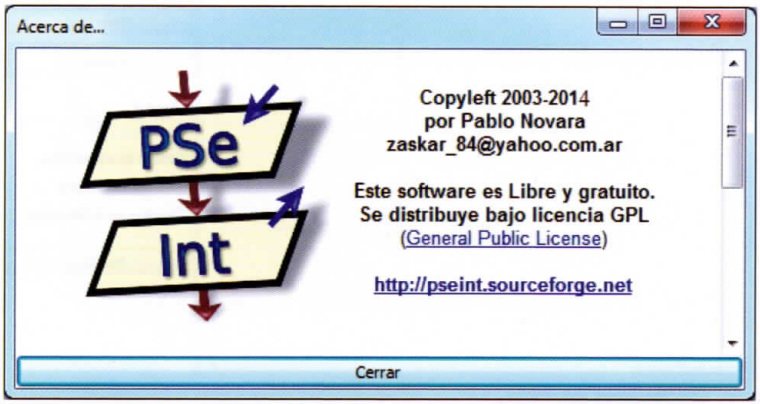


Figura 2.1 Acerca del Programa PSeInt

Algunas características y funcionalidades de PSeInt son:

- Presenta herramientas de edición para escribir algoritmos en pseudocódigo en español
- Permite generar y editar el diagrama de flujo del algoritmo
- El lenguaje pseudocódigo utilizado es configurable (perfiles)
- Puede interpretar (ejecutar) los algoritmos escritos, incluso paso a paso.
- Determina y marca los errores de sintaxis y en tiempo de ejecución.

- Permite exportar el algoritmo de pseudocódigo a código C, C++, C#, java, php, VB.net, pascal, python, javaScript, Matlab.
- Ofrece un sistema de ayuda integrado y con ejemplos
- Es multiplataforma (Windows, GNU/Linux y Mac OS X)
- Es totalmente libre y gratuito (licencia GPL)

2.2 El entorno de trabajo

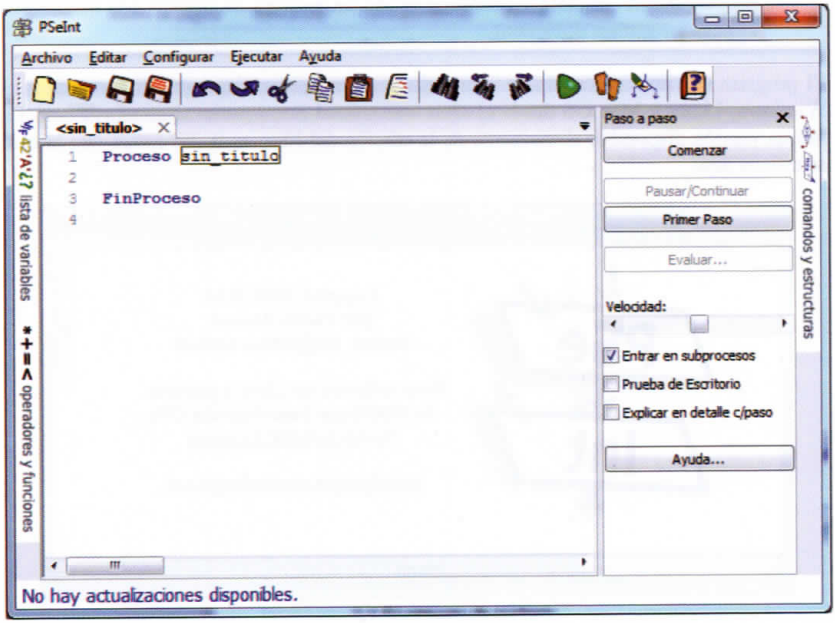


Figura 2.2 Entorno de trabajo de PSeInt

2.3 Perfiles

Los perfiles permiten configurar ciertas restricciones en el pseudocódigo. Existen perfiles predefinidos en el programa, como el *Flexible*, y otros creados bajo solicitud de ciertas instituciones como es el caso del perfil '*UPEC*' (Universidad Politécnica Estatal del Carchi) que utilizaremos en el resto del libro (véase figura 2.3). Este perfil obliga a definir o declarar las variables antes de ser usadas, lo cual resulta ser una buena práctica de programación y evitar posibles errores de programación.

Para seleccionar el perfil vaya al menú '*Configurar*', Opciones del lenguaje (*perfiles*).

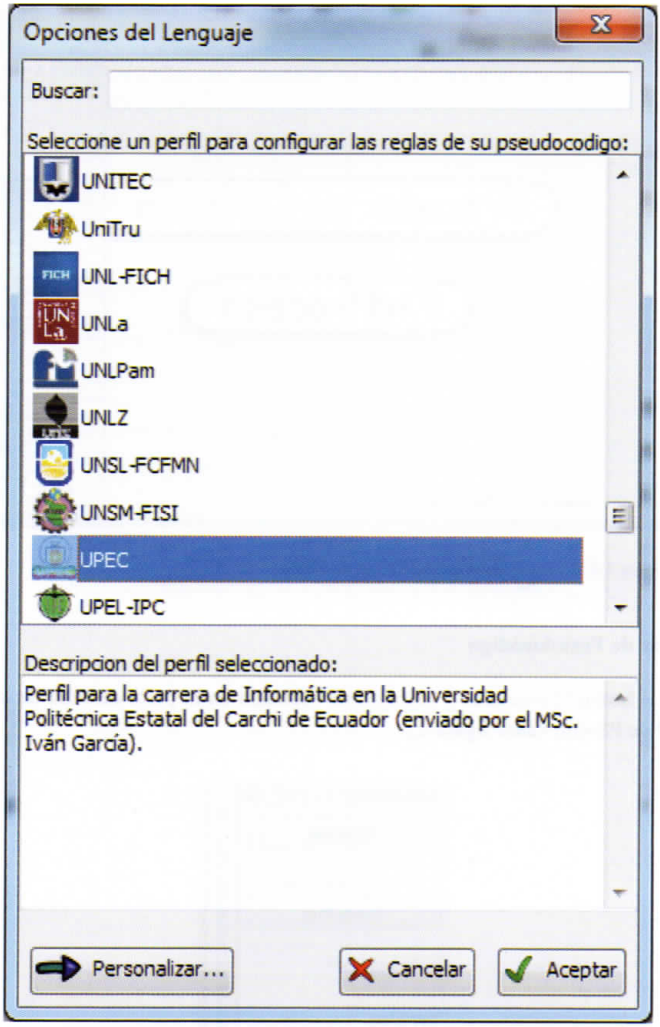


Figura 2.3 Perfiles de PSeInt

2.4 Editor de Diagramas de Flujo

Diríjase al menú '*Archivo*', opción '*editar diagrama de flujo*'.

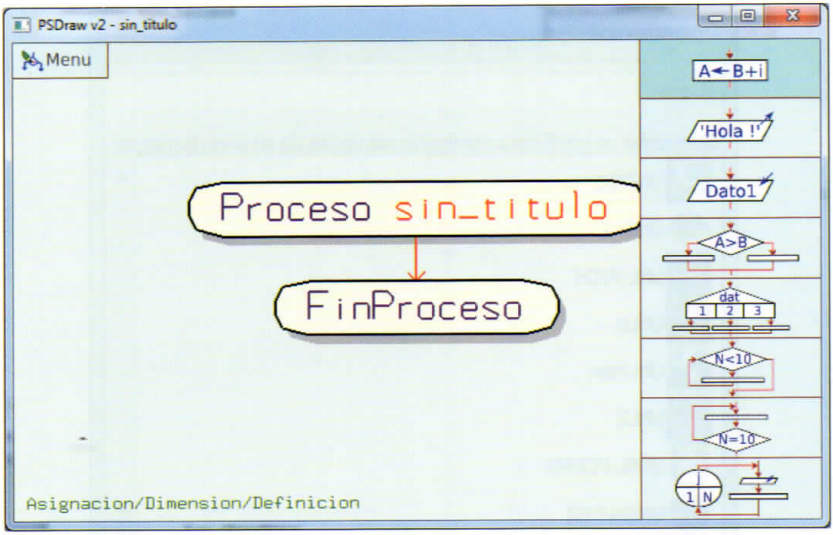


Figura 2.4 Editor de diagramas de Flujo de PSeInt

2.5 Editor de Pseudocódigo

Diríjase al botón '*Comandos y Estructuras*' ubicado en la parte derecha de la pantalla principal de PSeInt, véase figura 2.5

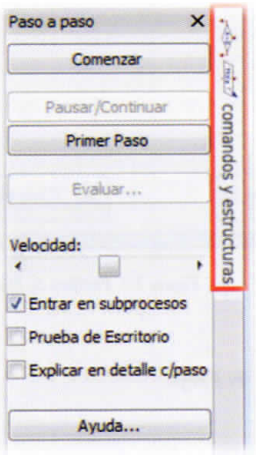


Figura 2.5 Comandos y estructuras de PSeInt

Luego aparece la ventana de *Comandos* como se indica en la figura 2.6.

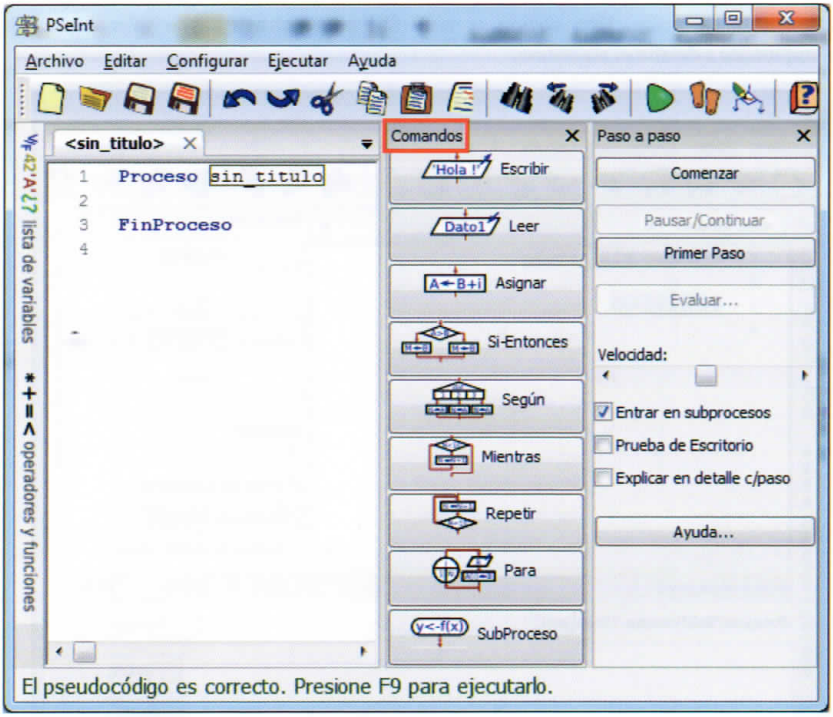


Figura 2.6 Editor de Pseudocódigo de PSeInt

2.6 Ejecución del algoritmo

Diríjase al menú '*Ejecutar*', ahí encontrará algunas opciones disponibles, entre ellas:

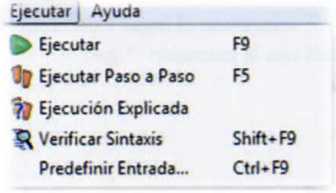


Figura 2.7 Opciones de ejecución del algoritmo

Además activando la opción de '*Prueba de Escritorio*' puede ejecutar por pasos el algoritmo según su ritmo e ir examinando los valores que toman las diferentes variables durante la ejecución. Esta opción es muy útil para corregir los *errores lógicos* y comprender a detalle el algoritmo.

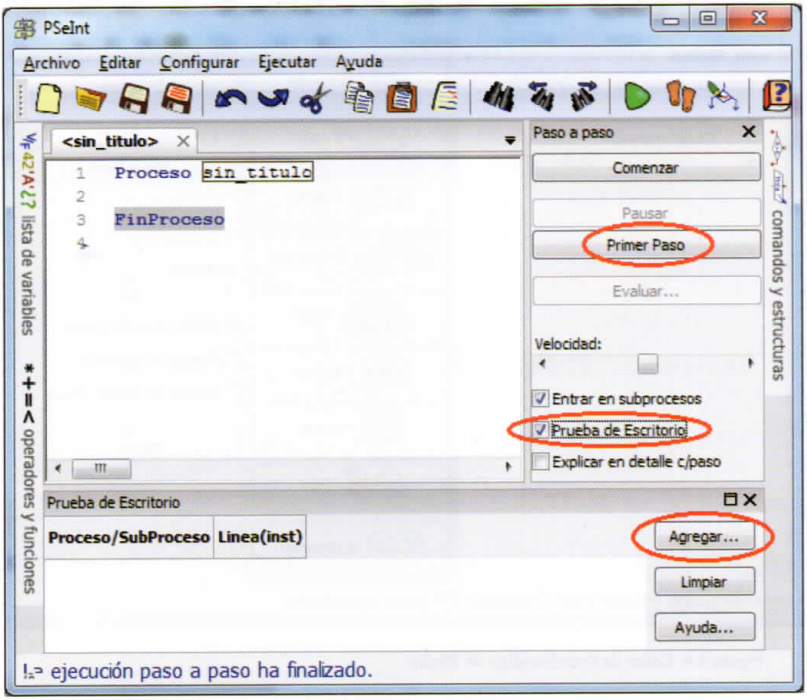


Figura 2.8 Prueba de Escritorio en PSeInt

2.7 Guardar el algoritmo

Diríjase al menú '*Archivo*', opción '*Guardar como…*', seleccione el lugar y un nombre adecuado para el algoritmo. Este archivo se almacenará con la extensión '**.psc*'

2.8 Exportación a código fuente

Diríjase al menú '*Archivo*', opción '*Exportar*', ahí encontrará varias opciones para crear el código fuente en algún lenguaje de programación disponible, imagen o html.

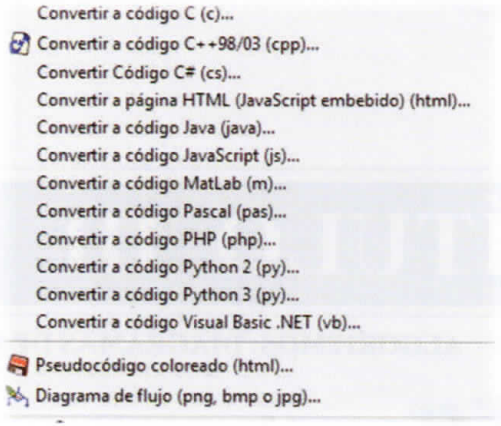


Figura 2.9 Opciones de exportación de PSeInt

2.9 Tareas propuestas

2.9.1 Descargue la última versión disponible del programa PSeInt desde el sitio web oficial e instale en su computador.

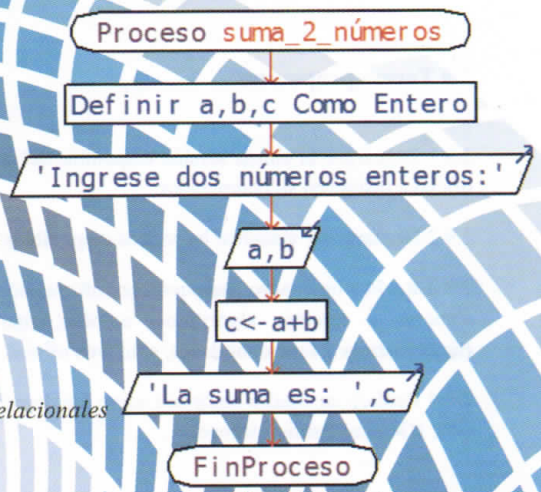
2.9.2 Descargue y visualice los videos disponibles sobre el funcionamiento de PSeInt

2.9.3 Realice un cuadro sinóptico con información de otros programas disponibles para elaborar y ejecutar algoritmos: nombre, tipo de licencia, tipo de aplicación, características y funcionalidades, sitio web oficial.

2.9.4 Realice un cuadro sinóptico sobre los derechos (libertades) que ofrece la licencia GPL (General Public License)

CAPÍTULO III

ALGORITMOS: DIAGRAMAS DE FLUJO



En este capítulo usted aprenderá:

- *Qué son los algoritmos*
- *Diagrama de flujo*
- *Forma general*
- *Operadores: algebraicos, lógicos, relacionales*
- *Funciones incorporadas*
  - *Matemáticas*
  - *Cadena de caracteres*
  - *Otras*
- *Variables y tipos de datos*
- *Estructuras básicas de control*
  - *La secuenciación: asignación, lectura, escritura*
  - *La selección: si-entonces, según*
  - *La repetición: mientras, repetir-hasta, para*
- *Estructuras de control anidadas*
- *Miscelánea de ejercicios*
- *Observaciones del capítulo*
- *Ejercicios propuestos*

3. Algoritmos: Diagramas de Flujo

3.1 Qué son los algoritmos

Para López (2011), un algoritmo es una secuencia ordenada y cronológica de pasos o instrucciones que han de seguirse para resolver un problema y debe tener las siguientes características:

- Debe ser simple, claro y preciso (determinista, es decir, que si se ejecuta varias veces con los mismos datos iniciales, siempre producirá la misma salida).
- Tener un orden lógico
- Tener un principio y un fin

Ejemplo 1: Realizar un algoritmo para '*enviar un correo electrónico*'

1 *Encender el computador y modem*
2 *Abrir el navegador de internet*
3 *Tipiar la dirección URL del servicio de correo electrónico*
4 *Ingresar las credenciales*
5 *Presionar el botón Nuevo/Redactar*
6 *Escribir el asunto del mensaje*
7 *Redactar el mensaje*
8 *Adjuntar archivos*
9 *Agregar los destinatarios*
10 *Presionar el botón Enviar*
11 *Cerrar la sesión*
12 *Cerrar el navegador*
13 *Apagar el computador y modem*

Ejemplo 2: Realizar un algoritmo para '*imprimir un documento en la impresora*'

1 *Encender el computador e impresora*
2 *Ejecutar el editor de texto (MS Word)*
3 *Abrir el documento de interés*
4 *Configurar las páginas (orientación, tamaño, márgenes, ..*
5 *Configurar las propiedades de la impresora*
6 *Poner hojas en blanco en la impresora*
7 *Presionar el botón Imprimir*
8 *Cerrar el programa MS Word*
9 *Apagar el computador e impresora*

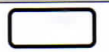
Computacionalmente un algoritmo puede expresarse de distintas maneras, principalmente en un *diagrama de flujo* o *pseudocódigo*.

Para López (2011) los diagramas de flujo resultan obsoletos porque no soportan todas las estructuras de control de la programación estructurada en forma natural. Además su tamaño crece considerablemente en la medida que el problema sea hace más complejo. Por el contrario, estos son fáciles de entender y utilizar para estudiantes de otras disciplinas ajenas a la informática. Por esta razón se va a dedicar este capítulo a los diagramas de flujo para la resolución de problemas sencillos.

3.2 Diagramas de Flujo

Es una representación gráfica de un algoritmo. Los elementos o símbolos básicos a usar son los mostrados en la tabla 3.1.

Tabla 3.1 Símbolos básicos del diagrama de flujo

Símbolo	Descripción
	Inicio y fin
	Proceso o instrucción
	Decisión
	Entrada de datos por teclado Salida de datos por pantalla
	Conector dentro de la misma página
	Conector fuera de la página
	Línea de conexión y dirección de flujo
	Llamada a un subproceso (método)

Fuente: Adaptado de López (2009)

3.3 Forma general de un algoritmo

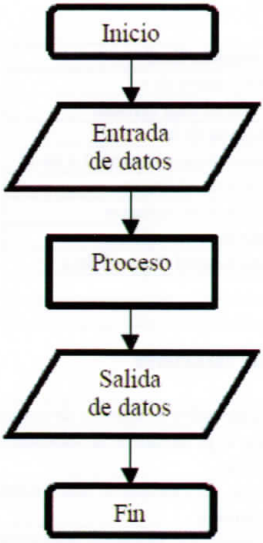


Figura 3.1 Forma general de un algoritmo

3.4 Operadores básicos en PSeInt

Algebraicos	+	Suma
	-	Resta
	*	Multiplicación
	/	División
	^	Potencia
	MOD	Módulo o resto
Lógicos	Y	Conjunción
	O	Disyunción
	NO	Negación
Relacionales	=	igual
	< >	diferente
	<	Menor
	>	Mayor
	< =	Menor o igual
	> =	Mayor o igual

3.5 Funciones incorporadas en PSeInt

3.5.1 Funciones Matemáticas

abs	abs(x) calcula el valor absoluto de x
trunc	trunc(x) devuelve la parte entera de x
redon	redon(x) redondea x al entero más cercano
rc	rc(x) calcula la raíz cuadrada de x
azar	azar (n) retorna un número aleatorio entre 0 y (n-1)
sen	sen(x) calcula en seno de x en radianes
cos	cos(x) calcula en coseno de x en radianes
tan	tan(x) calcula la tangente de x en radianes
ln	ln(x) calcula el logaritmo natural (base e) de x
exp	exp(x) calcula e^x
PI	Equivale a 3.1415926

3.5.2 Funciones de Cadena de caracteres

Longitud	Longitud(s) devuelve el número de caracteres
Subcadena	Subcadena(s, i, j) devuelve la subcadena desde la posición i hasta j
Concatenar	Concatenar(s1, s2) devuelve una cadena con los contenidos unidos
ConvetirANumero	ConvetirANumero(s) devuelve convertido de texto a número
ConvertirATexto	ConvertirATexto(n) devuelve convertido de número a texto
Mayúsculas	Mayúsculas(s) devuelve su contenido convertido a mayúsculas
Minúsculas	Minúsculas(s) devuelve su contenido convertido a minúsculas

3.5.3 Otras funciones

Borrar Pantalla	Limpia la Pantalla de resultados
Esperar Tecla	Detiene la ejecución del algoritmo hasta que el usuario presione cualquier tecla para continuar

3.6 Variables y tipos de datos

Las variables sirven para almacenar datos y está formada por un espacio en memoria del computador y un nombre simbólico asociado a dicho espacio. Los nombres de las variables deben empezar con una letra y no deben contener símbolos raros. También no deben coincidir con las *palabras reservadas* del propio programa PSeInt.

Los tipos de datos primitivos disponibles en PSeInt son: carácter/cadena de caracteres, entero, lógico y real. Para declarar o definir una variable se usa la palabra reservada '*definir*' y la siguiente sintaxis:

```
definir x Como Carácter
definir y Como Entero
definir z Como Lógico
definir w Como Real
```

Si desea declarar más variables del mismo tipo puede separarlas por comas. Para asignar el valor a una variable tipo carácter, se debe usar las comillas simples o dobles. El separador decimal en las variables numéricas es el punto. Antes de utilizar una variable en alguna operación, es altamente recomendado que esta debe estar previamente inicializada con algún valor explícito para evitar posibles errores.

Según Farrell (2013), existen tres convenciones para nombrar las variables:

- **Notación de camello.-** La variable empieza con una letra minúscula y cualquier palabra subsiguiente comienza con una mayúscula, ejemplo: lastName, firstName.
- **La caja de Pascal.-** La primera letra es mayúscula, ejemplo: LastName, FirstName.
- **Notación húngara.-** El tipo de dato es parte del nombre de la variable, ejemplo: stringLastName, stringFirstName.

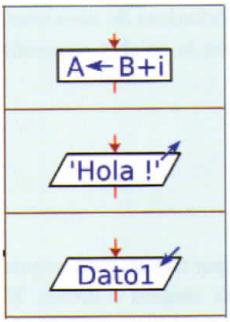
Aquí en el documento utilizaremos la notación de camello.

3.7 Estructuras básicas de control

Permiten modificar el flujo de ejecución de las instrucciones de un algoritmo o programa.

3.7.1 La secuenciación

Las instrucciones se ejecutan secuencialmente una después de la otra. Aquí podemos encontrar básicamente:



Asignación, definición o dimensión (arreglos)

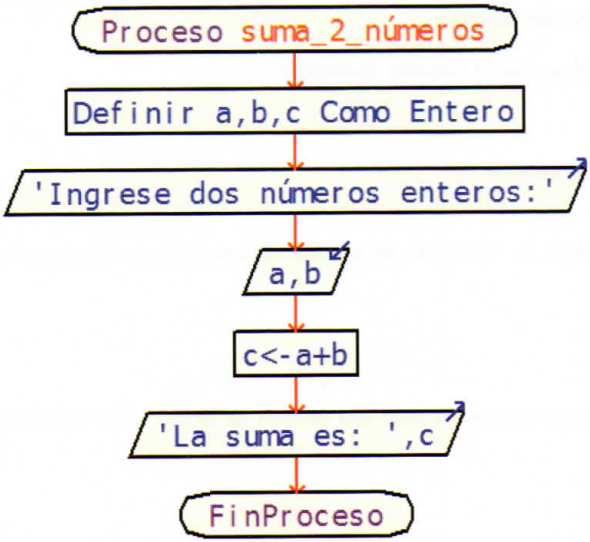
Escribir (salida de datos)

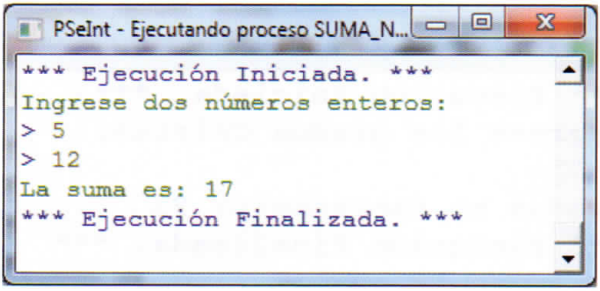
Leer (entrada de datos)

3.7.2 Ejercicios de Secuenciación

3.7.2.1 Realizar un algoritmo para calcular la suma de dos números enteros leídos por teclado

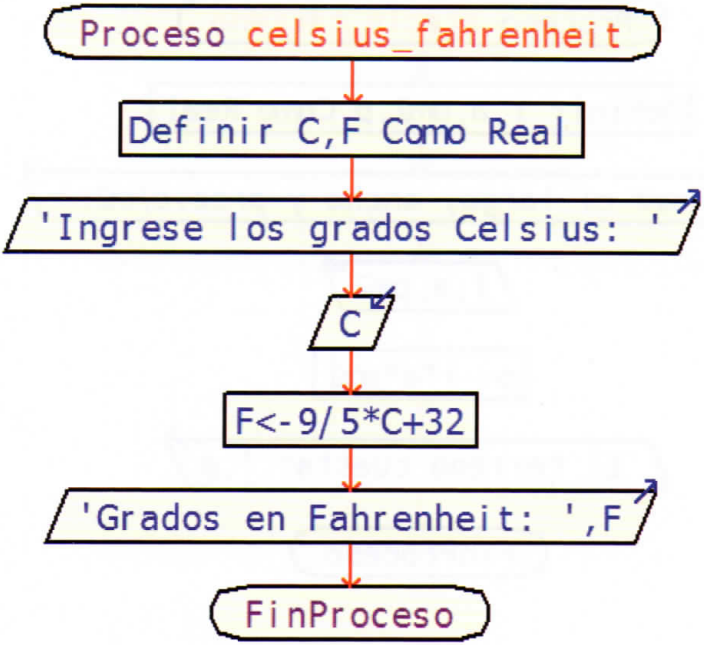
Entrada: dos números
Salida: un número (suma)
Proceso: sumar los dos números enteros usando el operador algebraico (+)

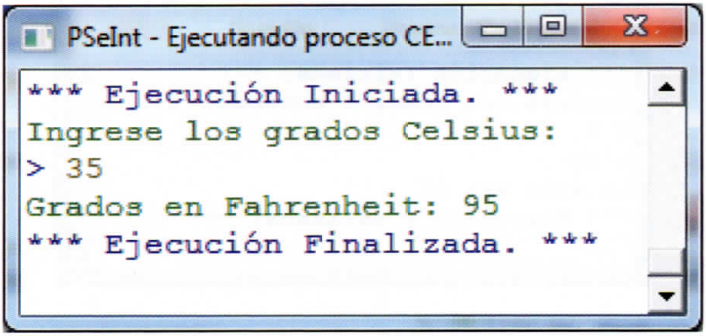




3.7.2.2 Realizar un algoritmo para convertir de grados Celsius (C) ingresados por teclado a Fahrenheit (F).

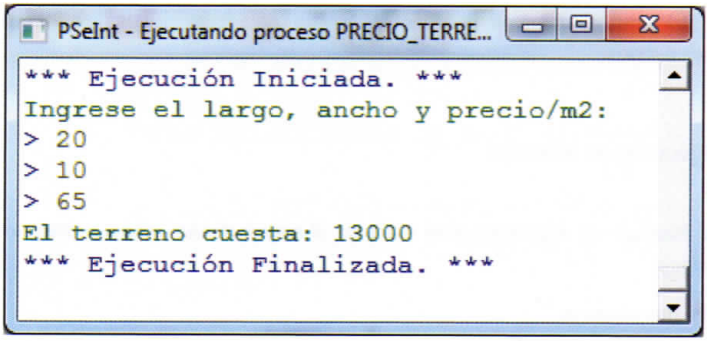
Entrada: un número real (Celsius)
Salida: un número real (Fahrenheit)
Proceso: convertir usando la fórmula: $F = \frac{9}{5}C + 32$





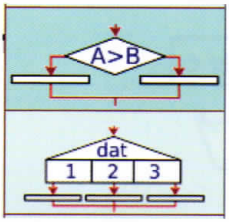
3.7.2.3 Realizar un algoritmo para calcular el precio de un terreno cuadrado o rectangular ingresando por teclado el largo, ancho y precio de cada m^2.

Entrada: tres números reales
Salida: un número real (precio del terreno)
Proceso: calcular usando la fórmula: $precio = largo * ancho * precio\ del\ m^2$



3.7.3 La selección

Se ejecuta una(s) instrucción(es) u otra, dependiendo de la evaluación de una condición (verdadero o falso). Aquí podemos encontrar:



Si–entonces (condicional simple)

Según (selección múltiple)

Una condición puede estar formada por varias sub-condiciones unidas a través de operadores lógicos Y, O, NO (expresiones lógicas compuestas). La tabla de verdad de cada operador es:

A	B	**Y**
V	V	V
V	F	F
F	V	F
F	F	F

A	B	**O**
V	V	V
V	F	V
F	V	V
F	F	F

A	**NO**
V	F
F	V

Cada parte (sub-condición) de una expresión (condición) que usa un operador sólo se evalúa hasta donde sea necesario para determinar si la expresión entera es verdadera o falsa. Algunas partes es posible que no se lleguen a probar. Esta característica se conoce como ***evaluación de cortocircuito***.

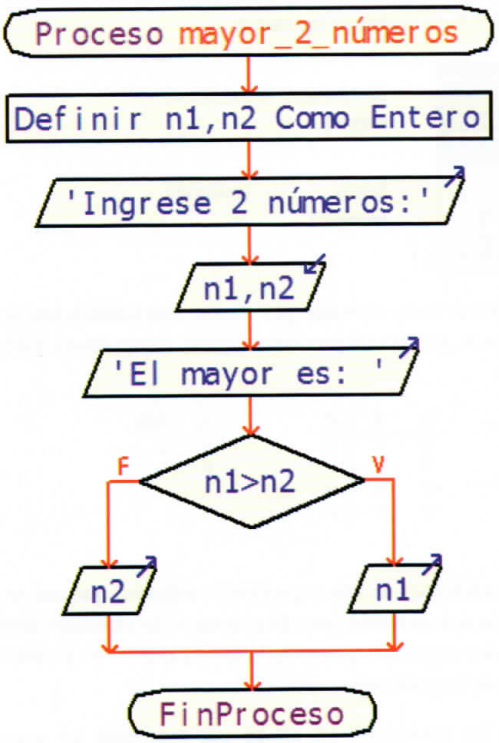
Cuando combina los operadores (*Y, O*) en una expresión, los operadores '*Y*' tienen *precedencia*, es decir, sus valores booleanos se evalúan primero. Se puede usar los paréntesis para evitar ciertas confusiones o anular el orden predeterminado de las operaciones.

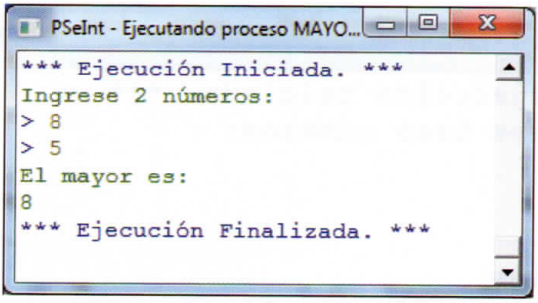
Nota: En PSeInt, actualmente la estructura *'según'* no admite rangos de números en las opciones o casos.

3.7.4 Ejercicios de Selección

3.7.4.1 Realizar un algoritmo para calcular el mayor de dos números enteros leídos por teclado

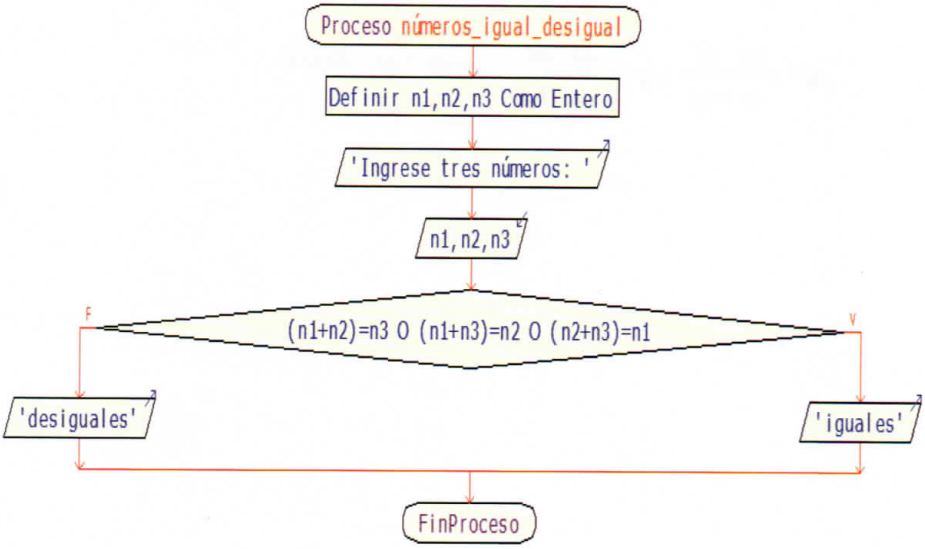
Entrada: dos números
Salida: un número (mayor)
Proceso: calcular el mayor de los números usando una selección y el operador relacional

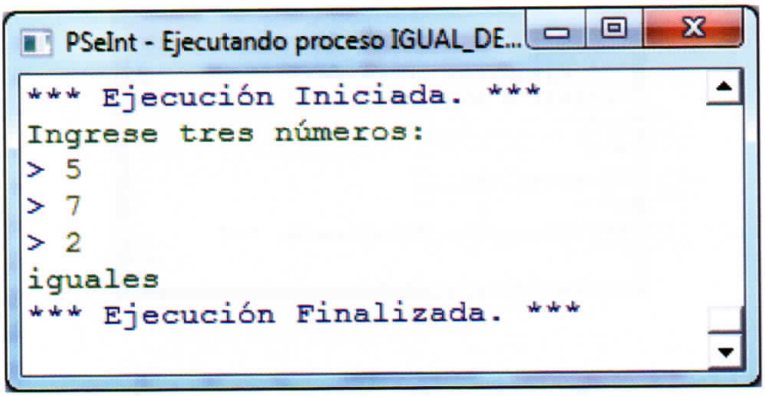




3.7.4.2 Realizar un algoritmo que lea por teclado tres números e imprima 'iguales' si la suma de dos cualquiera de ellos es igual al otro número, de lo contrario, imprima 'desiguales.

Entrada: tres números enteros
Salida: iguales/desiguales
Proceso: Determinar en base a la suma de dos números y comparar con el otro utilizando una selección y operadores lógicos.



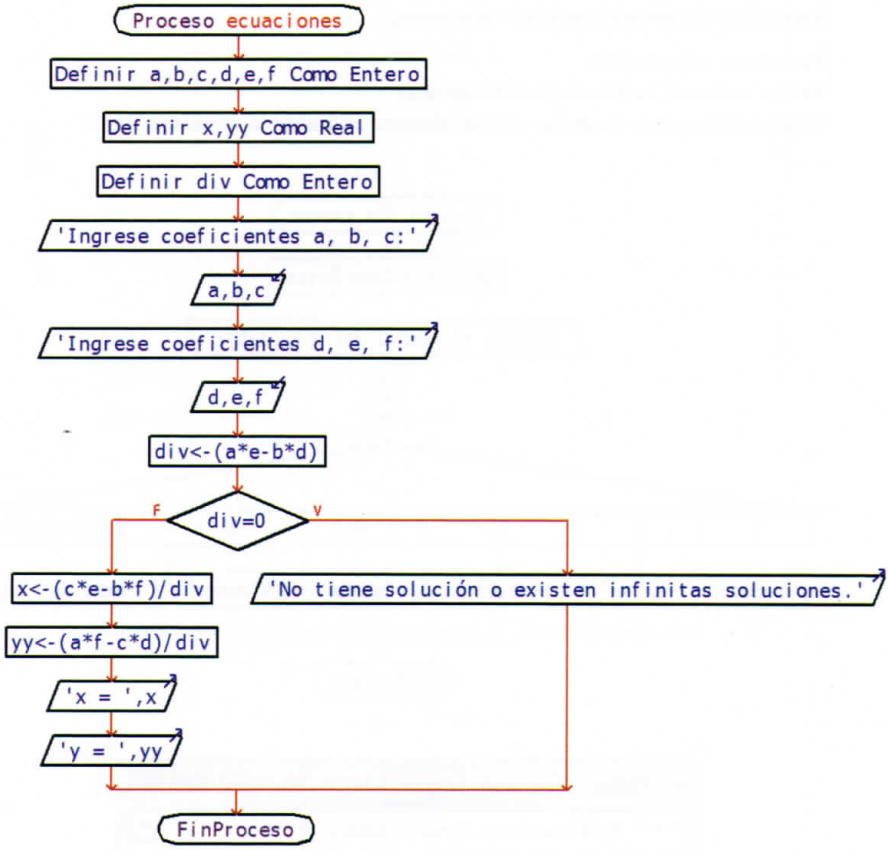


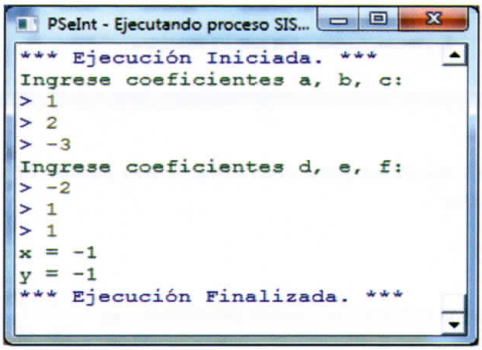
3.7.4.3 Realizar un algoritmo que resuelva un sistema de ecuaciones lineales de dos incógnitas ingresando sus coeficientes por teclado:

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$$

Entrada: seis números (a-f)
Salida: dos números (x, y)
Proceso: resolver el sistema de ecuaciones en base a las siguientes ecuaciones:

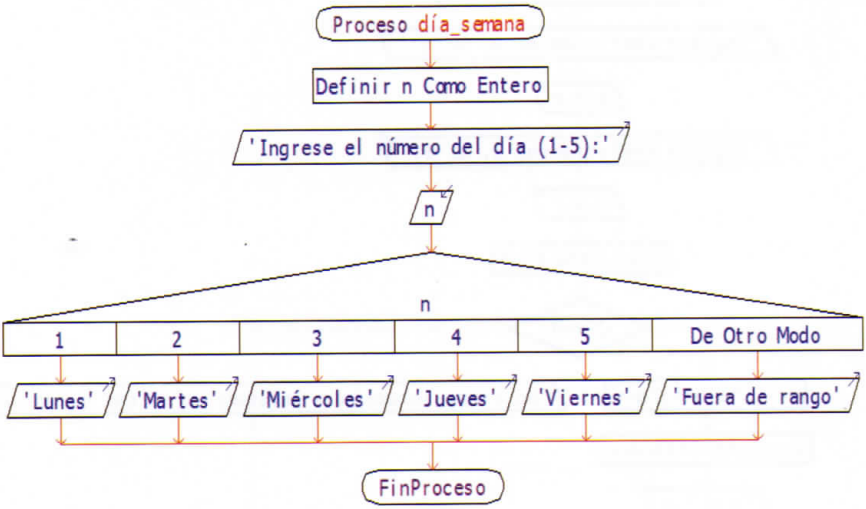
$$x = \frac{ce - bf}{ae - bd}\ ;\ y = \frac{af - cd}{ae - bd}\ ;\ donde\ (ae - db) \neq 0$$

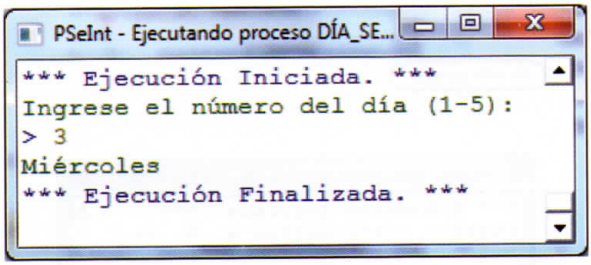

Proceso ecuaciones
Definir a,b,c,d,e,f Como Entero
Definir x,yy Como Real
Definir div Como Entero
'Ingrese coeficientes a, b, c:'
a,b,c
'Ingrese coeficientes d, e, f:'
d,e,f
div<-(a*e-b*d)
F
div=0
V
x<-(c*e-b*f)/div
'No tiene solución o existen infinitas soluciones.'
yy<-(a*f-c*d)/div
'x = ',x
'y = ',yy
FinProceso


PSeInt - Ejecutando proceso SIS...
*** Ejecución Iniciada. ***
Ingrese coeficientes a, b, c:
> 1
> 2
> -3
Ingrese coeficientes d, e, f:
> -2
> 1
> 1
x = -1
y = -1
*** Ejecución Finalizada. ***

3.7.4.4 Realizar un algoritmo que permita ingresar por teclado el número del día laborable de la semana e imprimir su nombre.

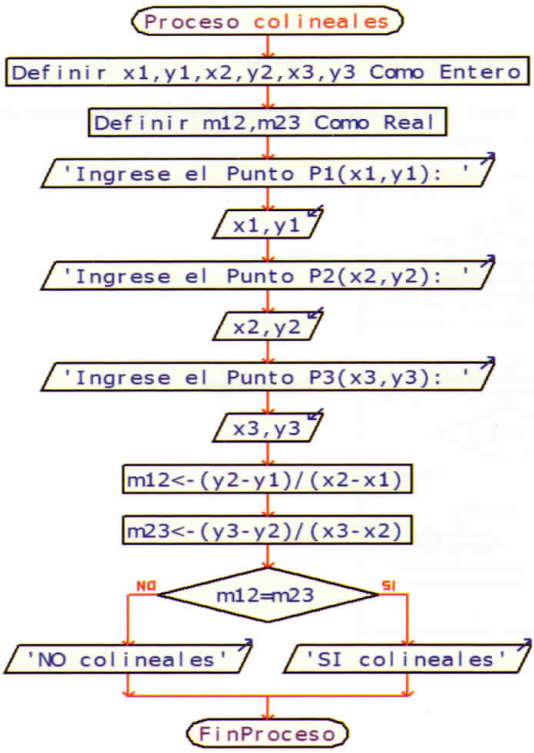
Entrada: un número (1-5)
Salida: cadena de caracteres (nombre del día)
Proceso: Determinar el nombre del día laborable utilizando la estructura '*según*'

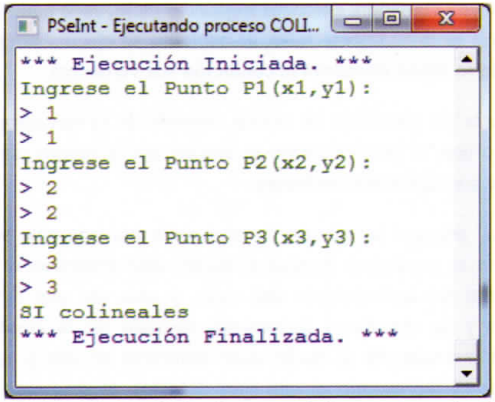




3.7.4.5 Realizar un algoritmo para detérminar si tres puntos P(x,y) ingresando por teclado son colineales

Entrada: P1, P2, P3
Salida: Si / No
Proceso: Determinar la colinealidad usando pendientes $m = \frac{y_2 - y_1}{x_2 - x_1}$

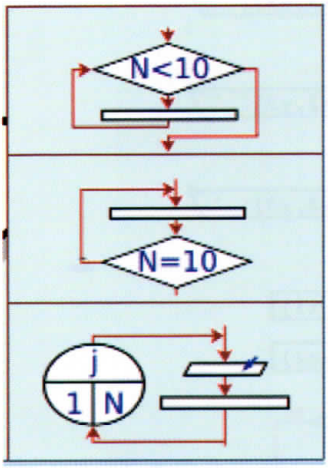




¿Qué sucede con el algoritmo cuando la recta es vertical?

3.7.5 La Repetición (bucle, ciclo o iteración)

Se inicia o repite una(s) instrucción(es), dependiendo de la evaluación de una condición. Aquí podemos encontrar:



Mientras

Repetir-hasta

Para

Funcionamiento:

En el '*Mientras*' y el '*Para*' la condición se evalúa antes de ingresar al bucle (*ciclo preprueba*) y si esta es verdadera entonces ingresa al ciclo, caso contrario se sale. Por lo que, si la condición es inicialmente falsa, el bucle no se ejecuta ninguna vez. Para los dos casos, el bucle se repite **mientras** la condición sea verdadera.

En el '*Repetir-hasta*' la condición se evalúa después de la ejecución del bucle (*ciclo posprueba*). Por lo que, el bucle siempre se ejecuta por lo menos una vez. El bucle se repite **hasta** que la condición sea verdadera.

En el bucle '*Para*' primero se ejecuta la *asignación* por una sola vez, seguida de la *condición* y si esta es verdadera ingresa al bucle, caso contrario se sale. En caso de ingresar se ejecutan las instrucciones del ciclo, y una vez que llega al final de la estructura regresa y se ejecuta el *incremento* seguido de la *condición*. Si esta es verdadera ingresa nuevamente al bucle, caso contrario se sale y así sucesivamente. Nótese que cuando el *incremento* es uno (por defecto), el programa no lo pone en el gráfico pero si lo considera en la ejecución.

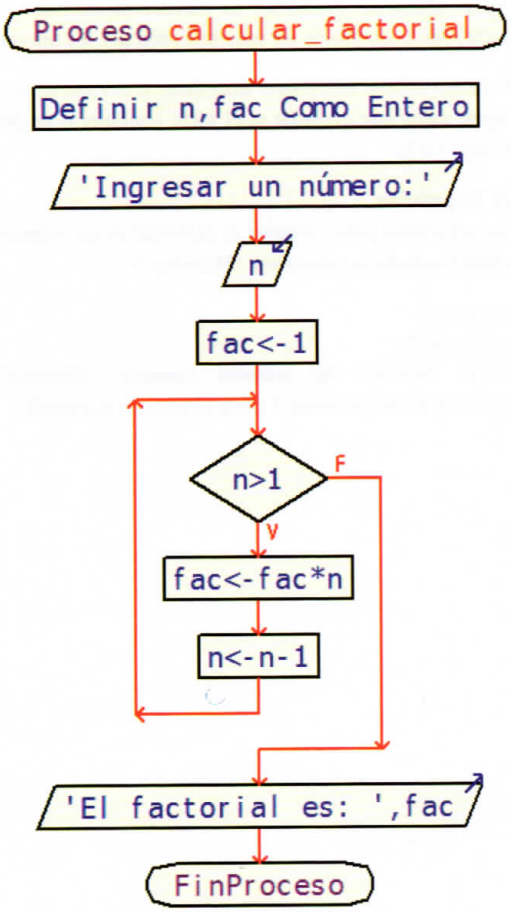
El bucle *'Para'* y el *'Mientras'* funcionan de manera similar, sólo que el *'Para'* es mucho más compacto, por lo que suele ser muy utilizado por los programadores.

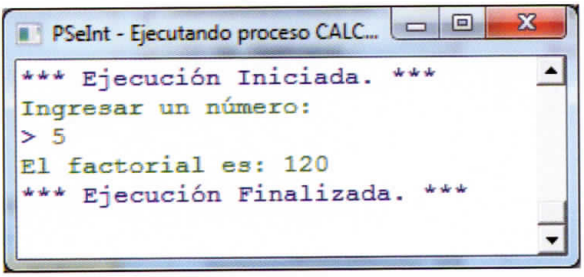
En cualquier ciclo deben ocurrir tres pasos: inicializar una variable de control del ciclo, compararla con algún valor que controle si el ciclo continúa o se detiene y alterar la variable que controla el ciclo.

3.7.6 Ejercicios de Repetición

3.7.6.1 Realizar un algoritmo para calcular el factorial de un número entero positivo ingresado por teclado (usando la estructura 'Mientras')

Entrada: un número entero
Salida: un número (factorial)
Proceso: calcular el factorial del número (usando *'Mientras'*) a través de multiplicaciones sucesiva desde 1 hasta el número ingresado.

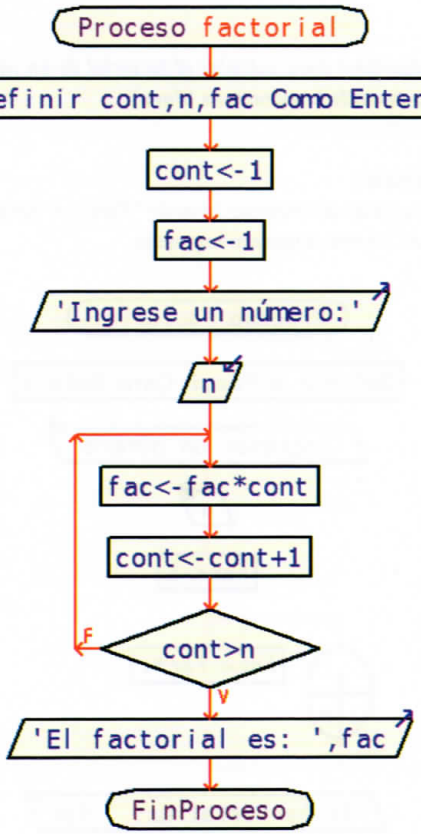

Proceso calcular_factorial
Definir n,fac Como Entero
'Ingresar un número:'
n
fac<-1
n>1
F
V
fac<-fac*n
n<-n-1
'El factorial es: ',fac
FinProceso

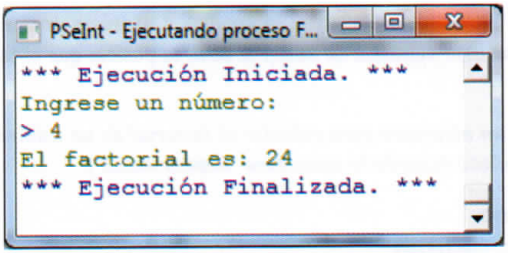

PSeInt - Ejecutando proceso CALC...
*** Ejecución Iniciada. ***
Ingresar un número:
> 5
El factorial es: 120
*** Ejecución Finalizada. ***

Observe que el bucle '*Mientras*' tiene la condición al inicio, y si es evaluada como verdadera ingresa, caso contrario se sale. Por esto, es posible que el bucle no se ejecute ninguna vez.

3.7.6.2 Realizar un algoritmo para calcular el factorial de un número entero positivo ingresado por teclado (usando la estructura 'Repetir-hasta')

Entrada: un número
Salida: un número (factorial)
Proceso: calcular el factorial del número (usando '*repetir-hasta*') a través de multiplicaciones sucesiva desde 1 hasta el número ingresado.

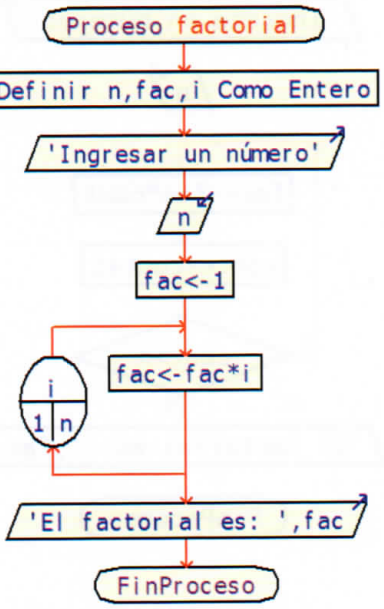


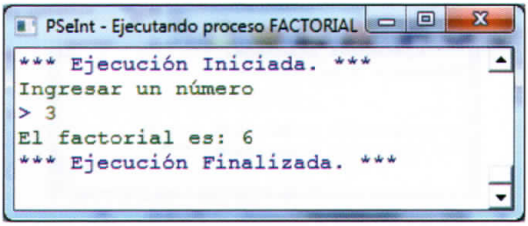


Observe que el bucle '*Repetir-hasta*' tiene la condición al final, y si es evaluada como falsa ingresa nuevamente, caso contrario se sale. Por esto, el bucle se ejecuta al menos una vez.

3.7.6.3 Realizar un algoritmo para calcular el factorial de un número entero positivo ingresado por teclado (usando la estructura 'Para')

Entrada: un número
Salida: un número (factorial)
Proceso: calcular el factorial del número (usando '*Para*') a través de multiplicaciones sucesiva desde 1 hasta el número ingresado.



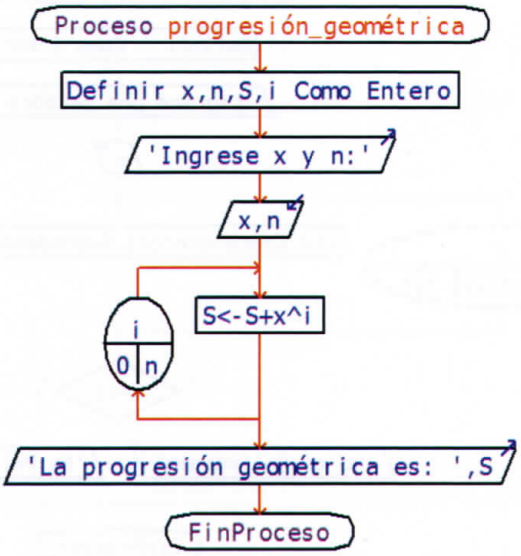


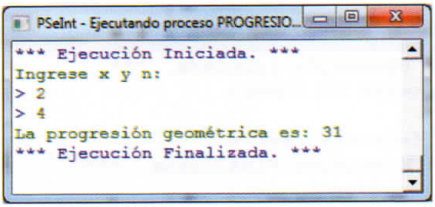
Nótese que si el incremento en el bucle es uno (por defecto), el programa no lo hace constar en el diagrama pero si lo considera internamente en la ejecución.

3.7.6.4 - Realizar un algoritmo para calcular la suma de la progresión geométrica ingresando por teclado x y n.

$$s = 1 + x^2 + x^3 + x + \cdots + x^n$$

Entrada: x, n
Salida: un número (S)
Proceso: calcular a través de las potencias y las sumas sucesivas en un bucle.

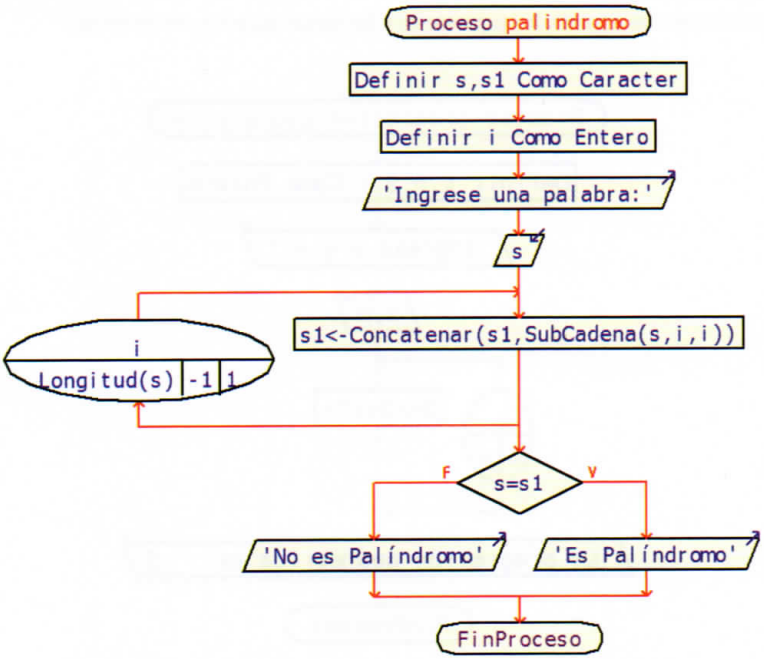


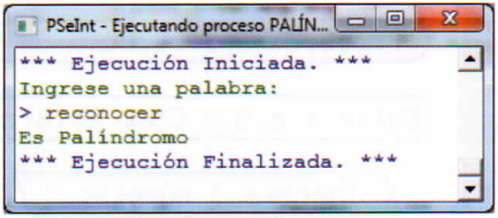


Nótese que la variable (*i*) del bucle empieza en cero y su incremento es de uno.

3.7.6.5 Realizar un algoritmo para determinar si una palabra ingresada por teclado es Palíndromo.

Entrada: palabra
Salida: Si/No
Proceso: Determinar si es palíndromo, es decir, si se lee igual de izquierda a derecha y viceversa, ejemplo: radar, oso, reconocer, somos, sometemos. Se invierte la palabra y se compara con la original.





Nótese que el bucle '*Para*' utiliza un contador (variable *i*) en forma decreciente (-1). El bucle ('*Para*') y la decisión ('*si-entonces*') están en forma secuencial una después de la otra, pero no anidadas.

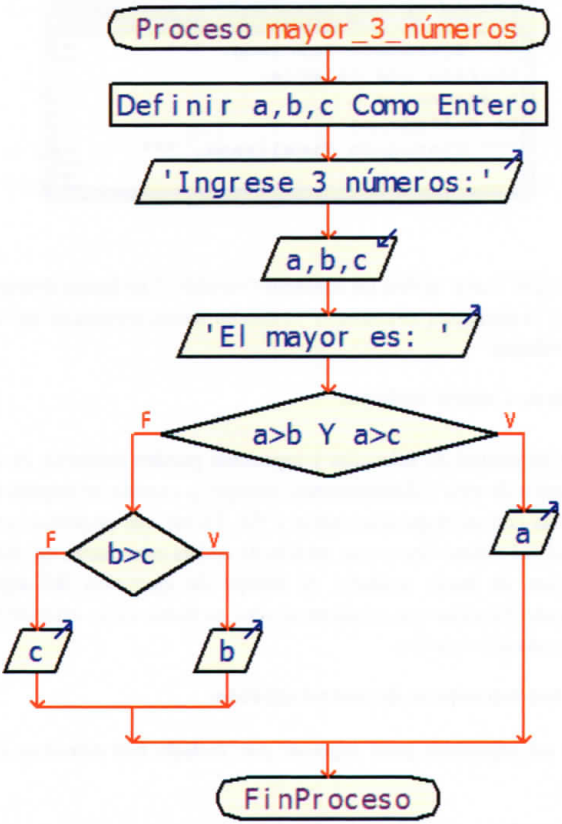
3.8 Estructuras de Control anidadas

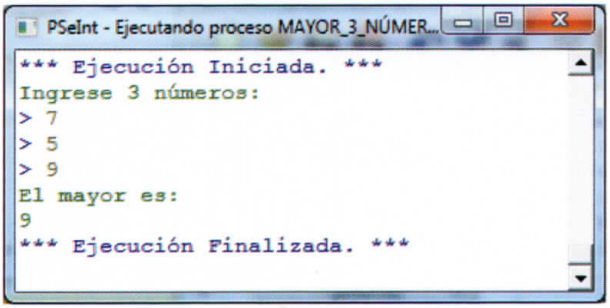
Las estructuras de control de selección y repetición pueden *anidarse*, es decir, pueden ponerse una dentro de otra indistintamente, siempre y cuando se respeta la sintaxis de cada una de ellas con su respectivo inicio y fin. Es así que podemos tener anidación entre estructuras del mismo tipo o una mezcla de ambas estructuras. Se debe considerar que para ejecutar un bucle anidado, el tiempo de ejecución del algoritmo crece considerablemente. El ciclo que contiene al otro se llama *ciclo exterior* y el que está contenido se llama *ciclo interior*.

3.9 Ejercicios con estructuras de control anidadas

3.9.1 Realizar un algoritmo para ingresar por teclado tres números e imprimir el mayor de ellos.

Entrada: tres números
Salida: un número (mayor)
Proceso: Calcular el mayor de los números usando una condición compuesta y simple anidada.





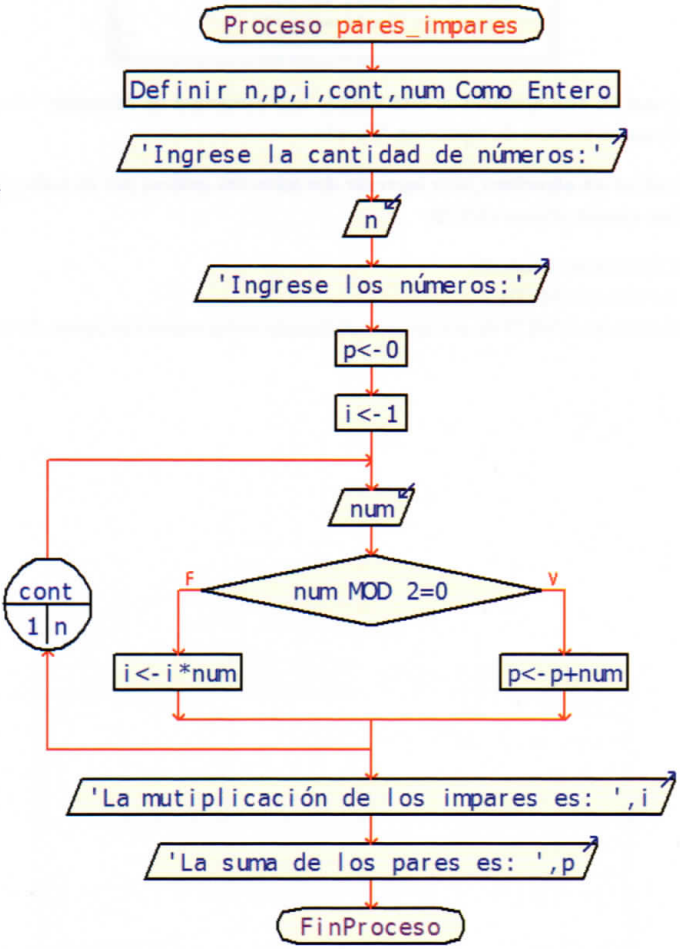
Observe que en la primera condición compuesta se está usando el operador lógico de conjunción (Y). Así tenemos una estructura '*si-entonces*' dentro de otra estructura '*si-entonces*'.

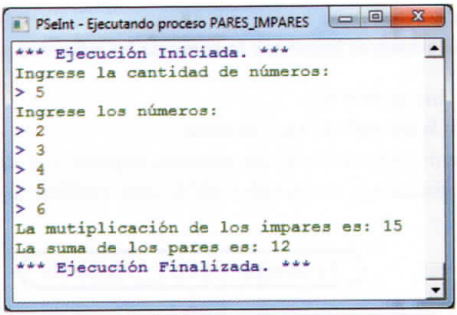
3.9.2 Realizar un algoritmo para ingresar n números leídos por teclado e imprima la multiplicación de los números impares y la suma de los números pares.

Entrada: cantidad y lista de números

Salida: dos números: la multiplicación y la suma

Proceso: Calcular la multiplicación de los números impares y la suma de los pares. Se utiliza una decisión y el operador MOD para verificar si el número es par o impar.

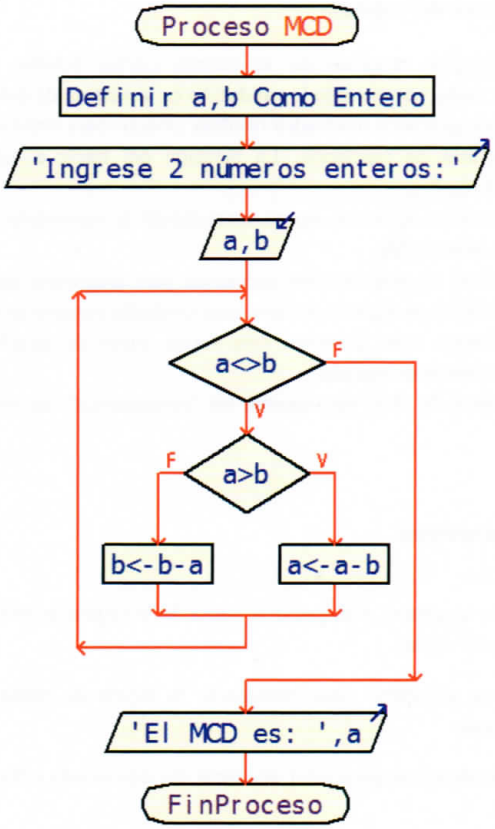




Observe que en este ejercicio se está usando una estructura de selección '*si-entonces*' dentro de una estructura de repetición '*Para*'.

3.9.3 Realizar un algoritmo para ingresar dos números enteros por teclado e imprimir el máximo común divisor (MCD).

Entrada: dos números enteros
Salida: un número (MCD)
Proceso: calcular el MCD de dos enteros utilizando restas sucesivas dentro de un bucle.




```
PSeInt - Ejecutando proceso MCD
*** Ejecución Iniciada. ***
Ingrese 2 números enteros:
> 6
> 4
El MCD es: 2
*** Ejecución Finalizada. ***
```

Observe que en este ejercicio se está usando una estructura de selección '*si-entonces*' dentro de una estructura de repetición '*Mientras*'.

3.10 Observaciones del capítulo

- En los ejemplos resueltos no se cubren ciertos detalles o requerimientos implícitos, como por ejemplo la validación en la entrada de datos por teclado.
- No todos los ejercicios mostrados resultan idóneos para realizarlos utilizando las tres estructuras de repetición. La elección del bucle a usar dependerá del problema a resolver.
- En los ejercicios resueltos, no necesariamente se representa el algoritmo más eficiente o más legible.
- Se debe tomar las precauciones necesarias para asegurarse que la ejecución de un bucle finalice en algún momento, caso contrario entraría en un bucle infinito.
- Cada algoritmo debe probarse con varios casos de prueba, es decir, con diferentes valores de entrada.
- Los diagramas de flujo no soportan los '*comentarios*' (se verá en el siguiente capítulo).

3.11 Ejercicios propuestos

3.11.1 Elaborar un algoritmo para pagar la cuota de la tarjeta de crédito utilizando los servicios de la banca virtual.

3.11.2 Elaborar un algoritmo para reemplazar la tarjeta de memoria RAM de un computador personal.

3.11.3 Elaborar un algoritmo para subir una tarea enviada al aula virtual de la asignatura de programación.

Utilizando diagramas de flujo, realice algoritmos para resolver los siguientes ejercicios propuestos:

3.11.4 Convertir una medida de almacenamiento en TB (terabyte) ingresada por teclado a GB (gigabyte), MB (megabyte), y KB (kilobyte).

3.11.5 Convertir una medida de longitud en metros ingresada por teclado a yardas, pulgadas, pies y centímetros.

3.11.6 Calcular la cantidad de pintura necesaria para pintar las paredes y techo (rectangulares) de una habitación ingresando por teclado sus dimensiones (alto, ancho, profundidad) en metros y el rendimiento de la pintura (m^2/lt). No considere las aberturas como ventanas o puertas.

$$litros\ por\ mano\ (lt) = \frac{superficie\ a\ pintar\ (m^2)}{rendimiento\ de\ la\ pintura\ \left(\frac{m^2}{lt}\right)}$$

3.11.7 Resolver la ecuación de segundo grado (raíces), ingresando por teclado los coeficientes a, b y c. Si las raíces son imaginarias deberá mostrar un mensaje.

3.11.8 Determinar si tres puntos P(x,y) ingresados por teclado son colineales. Utilice la fórmula de la distancia:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

3.11.9 Determinar si un año ingresado por teclado es bisiesto.

3.11.10 Calcular e imprimir el sueldo neto a percibir de un empleado de un almacén considerando sus ingresos (sueldo fijo y comisiones por ventas) y egresos (seguro social). La comisión es un porcentaje del sueldo fijo y se maneja en base a los artículos vendidos. El seguro social es el 9.35% del sueldo fijo. Ingrese por teclado el sueldo fijo y el número de artículos vendidos.

# artículos vendidos	% comisión de ventas
1-25	2%
26-50	4%
51-75	6%
76 o más	8%

3.11.11 Calcular la potenciación de dos números ingresados por teclado (base y exponente). No utilice el operador algebraico potencia (^) incorporado en PSeInt.

3.11.12 Determinar si un número ingresado por teclado es primo.

3.11.13 Determinar si un número ingresado por teclado es perfecto.

3.11.14 Calcular la calificación obtenida de un empleado público que atiende a los usuarios por ventanilla. Cada usuario califica la atención recibida ingresando un número entre 1 (menor puntuación) y 5 (mayor). El programa se ejecuta en forma continua hasta que el gerente introduce un número negativo para salir. Al final, el programa imprime la frecuencia de cada puntuación y el promedio de las calificaciones de la jornada laboral.

CAPÍTULO IV

ALGORITMOS: PSEUDOCÓDIGO

```
1  Proceso suma_2_números
2      Definir a,b,c Como Entero
3      Escribir 'Ingrese dos números enteros:'
4      Leer a,b
5      c<-a+b
6      Escribir 'La suma es: ',c
7  FinProceso
```

En este capítulo usted encontrará:

- *Definición y forma general*
- *Estructuras básicas*
  - *La secuenciación: asignación, lectura, escritura*
  - *La selección: si-entonces, según*
  - *La repetición: mientras, repetir-hasta, para*
- *Estructuras de control anidadas*
- *Miscelánea de ejercicios*
- *Observaciones del capítulo*
- *Ejercicios propuestos*

4. Algoritmos: Pseudocódigo

4.1 Definición y forma general

Según Farral (2013), el *pseudocódigo* es una representación en inglés o español de los pasos lógicos que se requieren para resolver un problema. *Pseudo* es un prefijo que significa *falso* y *codificar* significa ponerlo en un lenguaje de programación; por consiguiente, *pseudocódigo* significa *código falso* o declaraciones que en apariencia se han escrito en un lenguaje de programación pero no necesariamente siguen todas las reglas de sintaxis de alguno en específico, por lo que resultan adecuados para aprender la lógica de programación.

La forma general de un algoritmo en pseudocódigo es:

```
1  Proceso sin_titulo
2      // instrucciones
3  FinProceso
```

Nótese que algunas instrucciones empieza con una doble barra inclinada (//), esto significa que son ***comentarios***, es decir, no se toman en cuenta al momento de ejecutar el algoritmo y sirven para dar alguna explicación al lector o programador, así como también para la documentación del programa.

Las palabras de color azul (*Proceso, FinProceso*) se llaman ***palabras reservadas*** de PSeInt. Los números en gris indican la secuencia de las instrucciones.

En este capítulo se muestran en pseudocódigo algunos de los ejercicios realizados en el capítulo anterior utilizando diagramas de flujo.

4.2 Estructuras básicas de control

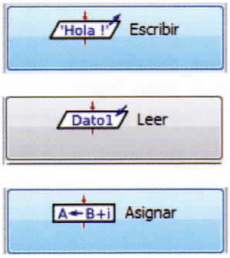
4.2.1 La Secuenciación

Aquí podemos encontrar básicamente los siguientes comandos:

```
Escribir lista_de_expresiones

Leer lista_de_variables

variable<-expresion
```

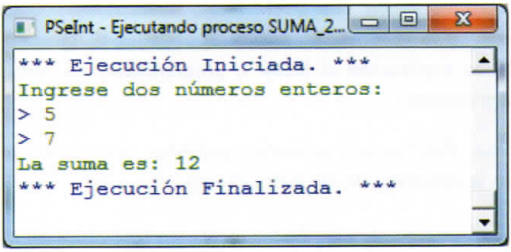


4.2.2 Ejercicios de Secuenciación

4.2.2.1 Realizar un algoritmo para calcular la suma de dos números enteros leídos por teclado

Entrada: dos números
Salida: un número (suma)
Proceso: sumar los dos números enteros usando el operador algebraico (+)

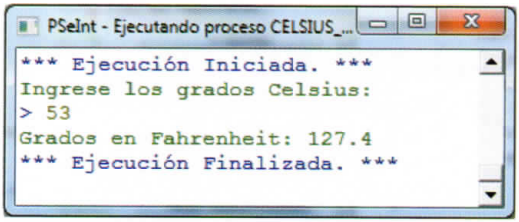
```
1  Proceso suma_2_números
2      Definir a,b,c Como Entero
3      Escribir 'Ingrese dos números enteros:'
4      Leer a,b
5      c<-a+b
6      Escribir 'La suma es: ',c
7  FinProceso
```



4.2.2.2 Realizar un algoritmo para convertir de grados Celsius (C) ingresados por teclado a Fahrenheit (F).

Entrada: un número real (Celsius)
Salida: un número real (Fahrenheit)
Proceso: convertir usando la fórmula: $F = \frac{9}{5}C + 32$

```
1  Proceso celsius_fahrenheit
2      Definir C,F Como Real
3      Escribir 'Ingrese los grados Celsius: '
4      Leer C
5      F<-(9/5)*C+32
6      Escribir 'Grados en Fahrenheit: ',F
7  FinProceso
```



4.2.3 La Selección

```
Si expresion_logica Entonces
    acciones_por_verdadero
Sino
    acciones_por_falso
Fin Si
```

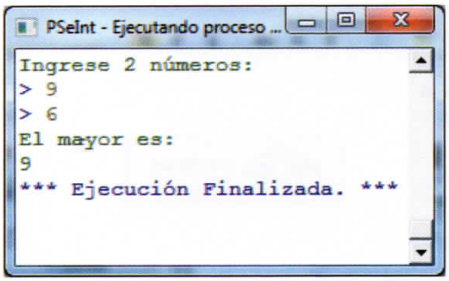

```
Segun variable_numerica Hacer
    opcion_1:
        secuencia_de_acciones_1
    opcion_2:
        secuencia_de_acciones_2
    opcion_3:
        secuencia_de_acciones_3
    De Otro Modo:
        secuencia_de_acciones_dom
Fin Segun
```

4.2.4 Ejercicios de Selección

4.2.4.1 Realizar un algoritmo para calcular el mayor de dos números enteros leídos por teclado

Entrada: dos números
Salida: un número (mayor)
Proceso: calcular el mayor de los números usando una selección y el operador relacional.

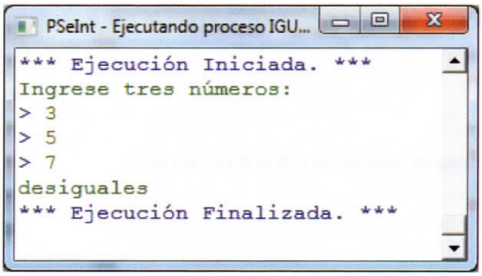
```
Proceso mayor_2_números
    Definir n1,n2 Como Entero
    Escribir 'Ingrese 2 números:'
    Leer n1,n2
    Escribir 'El mayor es: '
    Si n1>n2 Entonces
        Escribir n1
    Sino
        Escribir n2
    FinSi
FinProceso
```



4.2.4.2 Realizar un algoritmo que lea por teclado tres números e imprima 'iguales' si la suma de dos cualquiera de ellos es igual al otro número, de lo contrario, imprima 'desiguales.

Entrada: tres números
Salida: iguales/desiguales
Proceso: Determinar en base a la suma de dos números y comparar con el otro utilizando una selección compuesta y operadores lógicos.

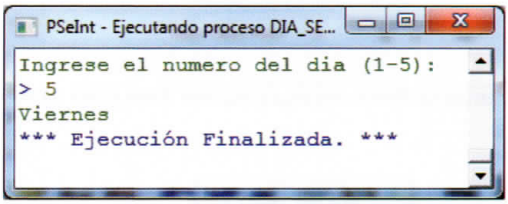
```
Proceso igual_desigual
    Definir n1,n2,n3 Como Entero
    Escribir 'Ingrese tres números: '
    Leer n1,n2,n3
    Si (n1+n2)=n3 O (n1+n3)=n2 O (n2+n3)=n1 Entonces
        Escribir 'iguales'
    Sino
        Escribir 'desiguales'
    FinSi
FinProceso
```



4.2.4.3 Realizar un algoritmo para ingresar por teclado el número del día laborable de la semana e imprimir su nombre

Entrada: un número (1-5)
Salida: cadena de caracteres (nombre del día)
Proceso: Determinar el nombre del día laborable utilizando la estructura '*según*'

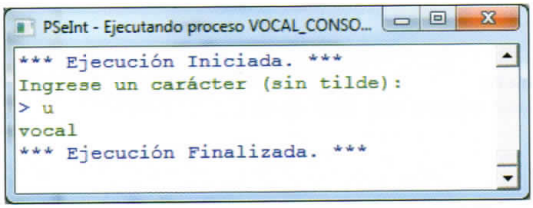
```
1   Proceso día_semana
2       Definir n Como Entero
3       Escribir 'Ingrese el número del día (1-5):'
4       Leer n
5       Segun n  Hacer
6           1:
7               Escribir 'Lunes'
8           2:
9               Escribir 'Martes'
10          3:
11              Escribir 'Miércoles'
12          4:
13              Escribir 'Jueves'
14          5:
15              Escribir 'Viernes'
16          De Otro Modo:
17              Escribir 'Fuera de rango'
18      FinSegun
19  FinProceso
```



4.2.4.4 Realizar un algoritmo para determinar si un carácter ingresado por teclado es una vocal o consonante

Entrada: un carácter
Salida: vocal/consonante
Proceso: Determinar si es vocal o consonante usando la estructura '*según*'

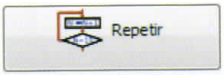
```
1   Proceso vocal_consonante
2       definir c Como Caracter
3       Escribir "Ingrese un carácter (sin tilde):"
4       Leer c
5       c<-Minusculas(c)
6       Segun c Hacer
7           'a', 'e', 'i', 'o', 'u':
8               Escribir "vocal"
9           De Otro Modo:
10              Escribir "consonante"
11          Fin Segun
12  FinProceso
```



4.2.5 La Repetición

```
Mientras expresion_logica Hacer
    secuencia_de_acciones
Fin Mientras
```


```
Repetir
    secuencia_de_acciones
Hasta Que expresion_logica
```



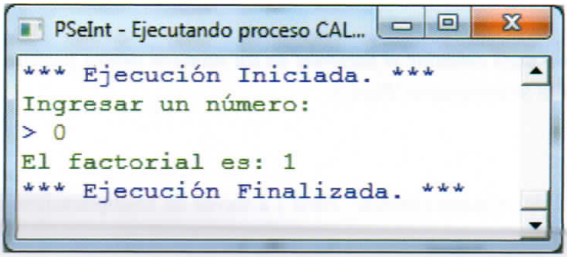

```
Para variable_numerica<-valor_inicial Hasta valor_final Con Paso paso Hacer
    secuencia_de_acciones
Fin Para
```

4.2.6 Ejercicios de Repetición

4.2.6.1 Realizar un algoritmo para calcular el factorial de un número entero positivo ingresado por teclado (usando la estructura 'Mientras')

Entrada: un número
Salida: un número (factorial)
Proceso: calcular el factorial del número (usando '*Mientras*') a través de multiplicaciones sucesivas del 1 hasta el número.

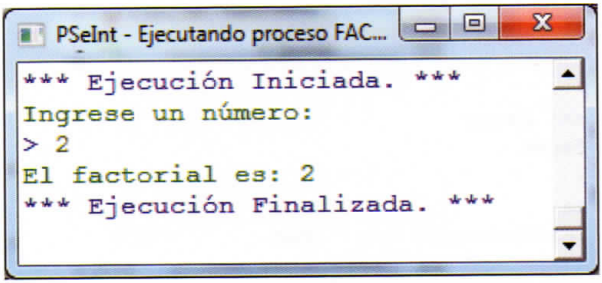
```
1   Proceso calcular_factorial
2       Definir n,fac Como Entero
3       Escribir 'Ingresar un número:'
4       Leer n
5       fac<-1
6       Mientras n>1 Hacer
7           fac<-fac*n
8           n<-n-1
9       FinMientras
10      Escribir 'El factorial es: ',fac
11  FinProceso
```



4.2.6.2 Realizar un algoritmo para calcular el factorial de un número entero positivo ingresado por teclado (usando la estructura 'Repetir-hasta')

Entrada: un número
Salida: un número (factorial)
Proceso: calcular el factorial del número (usando '*repetir-hasta*') a través de multiplicaciones sucesivas del 1 hasta el número.

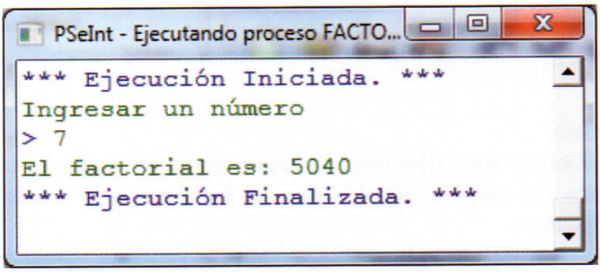
```
1   Proceso factorial
2       Definir cont,n,fac Como Entero
3       cont<-1
4       fac<-1
5       Escribir 'Ingrese un número:'
6       Leer n
7       Repetir
8           fac<-fac*cont
9           cont<-cont+1
10      Hasta Que cont>n
11      Escribir 'El factorial es: ',fac
12  FinProceso
```



4.2.6.3 Realizar un algoritmo para calcular el factorial de un número entero positivo ingresado por teclado (usando la estructura 'Para')

Entrada: un número
Salida: un número (factorial)
Proceso: calcular el factorial del número (usando '*Para*') a través de multiplicaciones sucesivas del 1 hasta el número.

```
1   Proceso factorial
2       Definir n,fac,i Como Entero
3       Escribir 'Ingresar un número'
4       Leer n
5       fac<-1
6       Para i<-1 Hasta n Hacer
7           fac<-fac*i
8       FinPara
9       Escribir 'El factorial es: ',fac
10  FinProceso
```

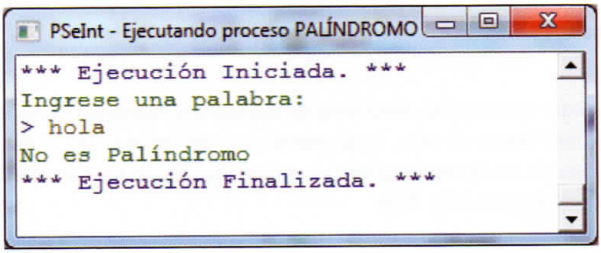


Nótese que el incremento en el bucle es de uno (por defecto).

4.2.6.4 Realizar un algoritmo para determinar si una palabra ingresada por teclado es Palíndromo.

Entrada: palabra
Salida: Si/No
Proceso: Determinar si es palíndromo, es decir, si se lee igual de izquierda a derecha y viceversa, ejemplo: radar, oso, reconocer, somos, sometemos. Se debe invertir la palabra usando un bucle y comparar con la original.

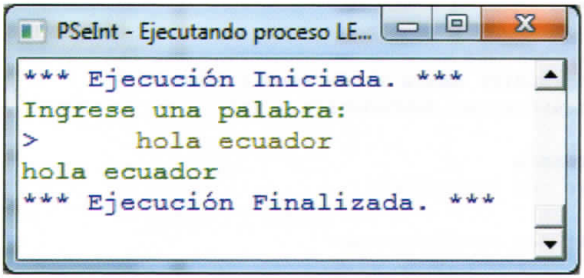
```
1   Proceso palíndromo
2       Definir s,s1 Como Caracter
3       Definir i Como Entero
4       Escribir 'Ingrese una palabra:'
5       Leer s
6       // bucle decreciente
7       Para i<-Longitud(s) Hasta 1 Con Paso -1 Hacer
8           s1<-Concatenar(s1,SubCadena(s,i,i))
9       FinPara
10      Si s=s1 Entonces
11          Escribir 'Es Palíndromo'
12      Sino
13          Escribir 'No es Palíndromo'
14      FinSi
15  FinProceso
```



4.2.6.5 Realizar un algoritmo para eliminar los espacios en blanco ubicados a la izquierda (inicio) de una palabra ingresada por teclado.

Entrada: palabra
Salida: palabra (sin espacios)
Proceso: Contar los espacios en blanco del inicio utilizando un bucle. Recuperar la subcadena a partir del contador obtenido hasta el final usando la función incorporada de PSeInt.

```
1   Proceso left_trim
2       definir s Como Caracter
3       definir i Como Entero
4       Escribir "Ingrese una palabra:"
5       Leer s
6       i<-1
7       Mientras SubCadena(s,i,i) = " " Hacer    // un espacio
8           i<-i+1
9       Fin Mientras
10      s <- SubCadena(s,i,Longitud(s))
11      Escribir s
12  FinProceso
```



4.3 Estructuras de control anidadas

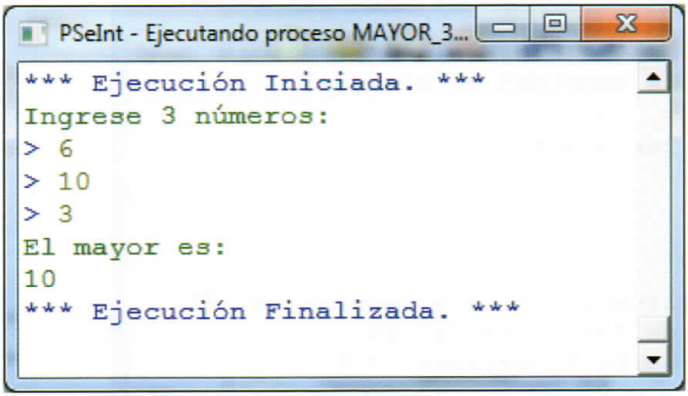
Como se indicó en el capítulo anterior, las estructuras de selección y repetición pueden anidarse, es decir, poner una dentro de otra. Estas pueden ser del mismo tipo o una mezcla de ellas. A continuación se muestran en pseudocódigo algunos de los ejercicios realizados anteriormente con diagramas de flujo.

4.4 Ejercicios de estructuras de control anidadas

4.4.1 Realizar un algoritmo para ingresar por teclado tres números e imprimir el mayor de ellos

Entrada: tres números
Salida: un número (mayor)
Proceso: Calcular el mayor de los números utilizando la estructura compuesta y simple anidada.

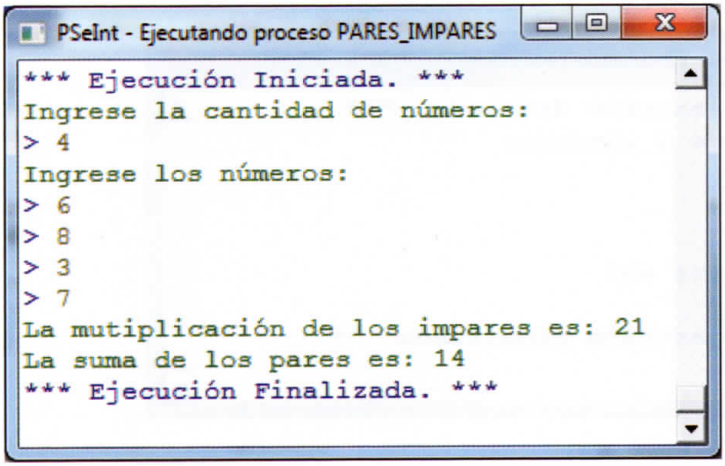
```
1   Proceso mayor_3_números
2       Definir a,b,c Como Entero
3       Escribir 'Ingrese 3 números:'
4       Leer a,b,c
5       Escribir 'El mayor es: '
6       Si a>b Y a>c Entonces    // condición compuesta
7           Escribir a
8       Sino
9           Si b>c Entonces      // condición simple
10              Escribir b
11          Sino
12              Escribir c
13          FinSi
14      FinSi
15  FinProceso
```



4.4.2 Realizar un algoritmo para ingresar n números leídos por teclado e imprima la multiplicación de los números impares y la suma de los números pares.

Entrada: cantidad y lista de números
Salida: dos números (la multiplicación y la suma)
Proceso: Calcular la multiplicación de los números impares y la suma de los pares utilizando un bucle y el operador MOD para discriminar entre par e impar.

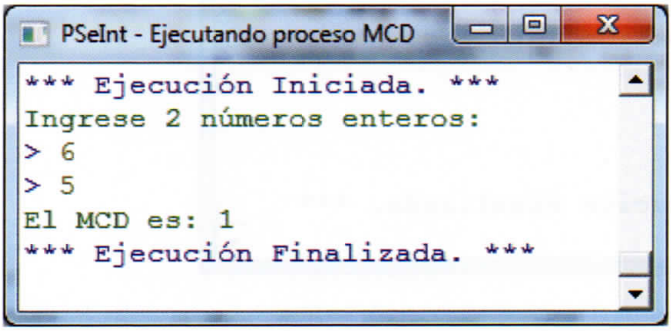
```
1   Proceso pares_impares
2       Definir n,p,i,cont,num Como Entero
3       Escribir 'Ingrese la cantidad de números:'
4       Leer n
5       Escribir 'Ingrese los números:'
6       p<-0
7       i<-1
8       Para cont<-1 Hasta n Hacer
9           Leer num
10          Si num MOD 2=0 Entonces // ¿el número es par?
11              p<-p+num
12          Sino
13              i<-i*num
14          FinSi
15      FinPara
16      Escribir 'La mutiplicación de los impares es: ',i
17      Escribir 'La suma de los pares es: ',p
18  FinProceso
```



4.4.3 Realizar un algoritmo para ingresar dos números enteros por teclado e imprimir el máximo común divisor (MCD).

Entrada: dos números enteros
Salida: un número (MCD)
Proceso: calcular el MCD de dos enteros realizando restas sucesivas en un bucle.

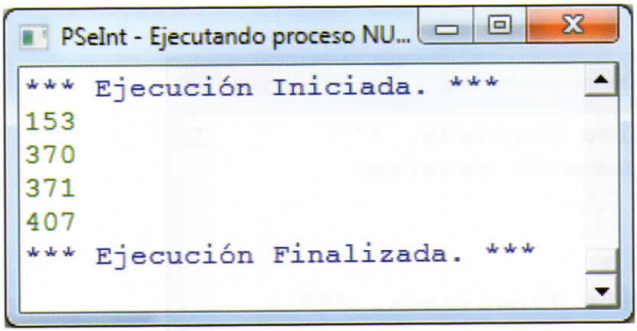
```
1   Proceso MCD
2       Definir a,b Como Entero
3       Escribir 'Ingrese 2 números enteros:'
4       Leer a,b
5       Mientras a<>b Hacer
6           Si a>b Entonces
7               a<-a-b
8           Sino
9               b<-b-a
10          FinSi
11      FinMientras
12      Escribir 'El MCD es: ',a
13  FinProceso
```



4.4.4 Realizar un algoritmo para calcular e imprimir los números narcisistas de tres cifras.

Entrada: ninguna
Salida: lista de números narcisistas
Proceso: calcular los números narcisistas de tres cifras (del 100 al 999). *Un número narcisista es aquel que es igual a la suma de cada uno de sus dígitos elevados a la 'n' potencia (donde 'n' es el número de cifras del número).* Separar cada dígito del número, elevar al cubo y sumar. Comparar la suma con el número.

```
1   Proceso números_narcisistas
2       definir n, num, d1, d2, d3 Como Entero
3       Para num<-153 Hasta 999 Con Paso 1 Hacer
4           d1<-num/100      // toma el cociente (parte entera)
5           n<-num mod 100   // toma el residuo (módulo)
6           d2<-n/10
7           d3<-n mod 10
8           n<- d1^3 + d2^3 + d3^3
9           Si n=num Entonces
10              Escribir num
11          Fin Si
12      Fin Para
13  FinProceso
```



Otra forma de resolver el mismo ejercicio usando las funciones incorporadas de cadena de caracteres de PSeInt es la siguiente:

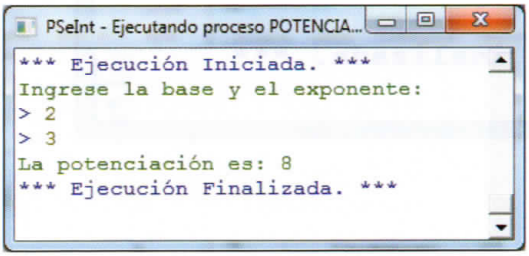
```
Proceso números_narcisistas
    definir n, num, d1, d2, d3 Como Entero
    definir s Como Caracter
    Para num<-100 Hasta 999 Con Paso 1 Hacer
        s<-ConvertirATexto(num)
        d1<-ConvertirANumero(SubCadena(s,1,1))
        d2<-ConvertirANumero(SubCadena(s,2,2))
        d3<-ConvertirANumero(SubCadena(s,3,3))
        n<- d1^3 + d2^3 + d3^3
        Si n=num Entonces
            Escribir num
        Fin Si
    Fin Para
FinProceso
```

4.5 Miscelánea de ejercicios

4.5.1 Realizar un algoritmo para calcular la potencia dado dos números ingresados por teclado (base y exponente). No utilice el operador algebraico potencia (^) de PSeInt.

Entrada: dos números enteros (base y exponente)
Salida: un número (la potencia)
Proceso: calcular la potencia utilizando multiplicaciones sucesivas en un bucle.

```
Proceso potenciación
    definir b,e,r Como Entero
    Escribir "Ingrese la base y el exponente:"
    Leer b,e
    r<-1
    Mientras e>0 Hacer
        r<-r*b
        e<-e-1
    FinMientras
    Escribir "La potenciación es: ",r
FinProceso
```



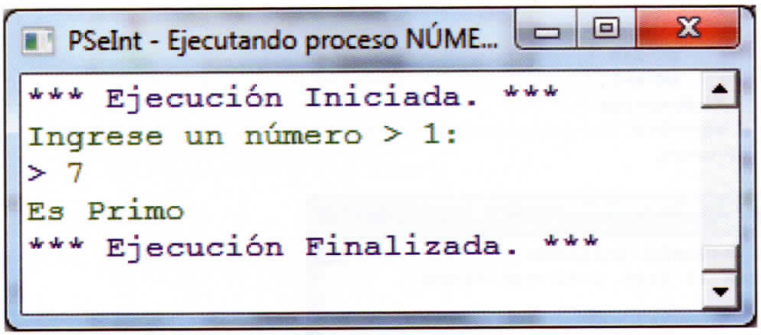
4.5.2 Realizar un algoritmo para determinar si un número ingresado por teclado es primo.

Entrada: un número entero >1

Salida: Si/No

Proceso: determinar si es primo o no. Un número primo es un número natural mayor que uno y que tiene únicamente dos divisores distintos, él mismo número y la unidad. Se utiliza en operador matemático MOD para comprobar la divisibilidad de un número. Se utiliza un bucle que va desde 2 hasta la raíz cuadrada del número.

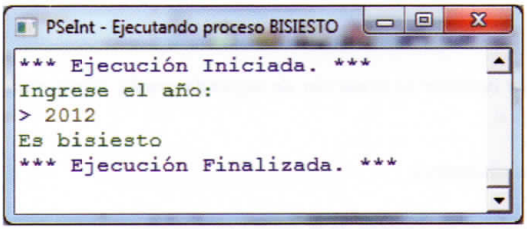
```
1   Proceso número_primo
2       definir n,i como entero
3       definir num como logico
4       Escribir "Ingrese un número > 1:"
5       Leer n
6       num<-Verdadero
7       i<-2
8       // es suficiente comprobar hasta la raíz cuadrada del número
9       Mientras i<=rc(n) Y num Hacer   // num es un valor centinela
10          Si n mod i = 0 Entonces // ¿es divisible?
11              num<-Falso
12          Fin Si
13          i<-i+1
14      Fin Mientras
15      Si num Entonces
16          Escribir "Es Primo"
17      Sino
18          Escribir "NO es Primo"
19      Fin Si
20  FinProceso
```



4.5.3 Realizar un algoritmo para determinar si un año ingresado por teclado es bisiesto.

Entrada: un número (año)
Salida: Si/No
Proceso: determinar si es bisiesto o no. El año bisiesto tiene 366 días, siendo el 29 de febrero el día extra. Un año es bisiesto si es divisible entre cuatro, exceptuando los últimos años de cada siglo -divisibles entre 100-, siempre y cuando no sean divisibles entre 400.

```
1   Proceso bisiesto
2       definir año Como Entero
3       Escribir "Ingrese el año:"
4       Leer año
5   -   Si año mod 4 =0 Y (año mod 100<>0 O año mod 400=0) Entonces
6       |   Escribir "Es bisiesto"
7       Sino
8       |   Escribir "NO es bisiesto"
9       Fin Si
10  FinProceso
```

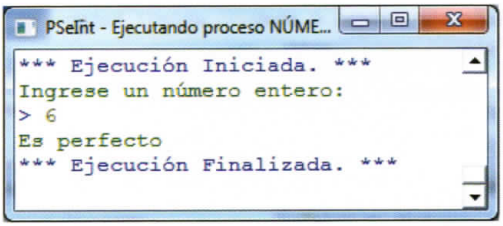


Observe en la línea #5, se utiliza condiciones compuestas y los paréntesis para indicar su precedencia.

4.5.4 Realizar un algoritmo para determinar si un número ingresado por teclado es perfecto.

Entrada: un número entero
Salida: Si/No
Proceso: determinar si es perfecto o no. Un número es perfecto cuando la suma de sus divisores positivos, sin incluirse, da el mismo número. El 6 es perfecto porque 1 + 2 + 3 = 6. Otro perfecto es el 28.

```
1   Proceso número_perfecto
2       definir N, i, sum como entero
3       Escribir "Ingrese un número entero:"
4       Leer N
5       sum <-0
6       Para i<-1 Hasta N-1 Con Paso 1 Hacer
7           Si N mod i = 0 Entonces // ¿es divisible?
8               sum = sum + i
9           FinSi
10      Fin Para
11      Si sum = N Entonces
12          Imprimir "Es perfecto"
13      Sino
14          imprimir "No es perfecto"
15      Fin Si
16  FinProceso
```



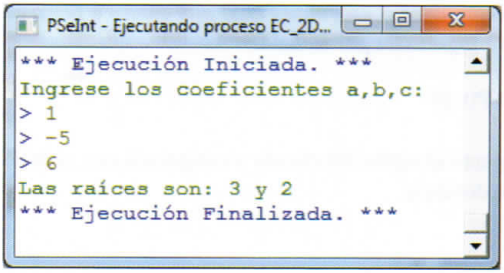
4.5.5 Realizar un algoritmo para resolver la ecuación de segundo grado, ingresando por teclado los coeficientes a, b y c.

Entrada: tres números enteros (coeficientes)
Salida: dos números (x1 y x2)
Proceso: calcular las raíces de la ecuación a través de la siguiente ecuación:

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

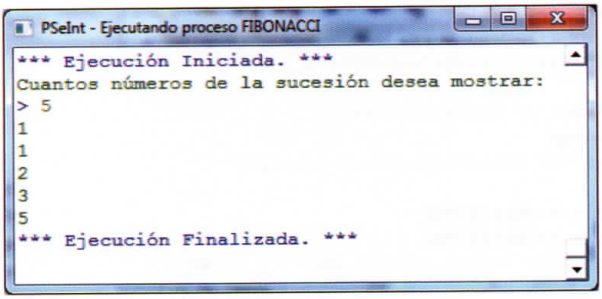
```
1   Proceso ec_2do_grado
2       definir a,b,c Como Entero
3       definir d, x1, x2 Como Real
4       Escribir "Ingrese los coeficientes a,b,c: "
5       Leer a,b,c
6       d<-(b^2)-4*a*c  // discriminante
7       Si d<0 Entonces
8           Escribir "No tiene soluciones reales"
9       Sino
10          x1<-(-b + rc(d))/(2*a)
11          x2<-(-b - rc(d))/(2*a)
12          Escribir "Las raíces son: ", x1, " y ", x2
13      Fin Si
14  FinProceso
```



4.5.6 Realizar un algoritmo para calcular e imprimir los n términos de la sucesión Fibonacci

Entrada: un número entero
Salida: n términos de la sucesión
Proceso: Calcular la sucesión Fibonacci. En esta sucesión los dos primeros números son uno y para el resto, cada elemento es la suma de los dos anteriores. Los primeros números de la serie son: 1, 1, 2, 3, 5, 8, 13, 21, ...

```
1   Proceso fibonacci
2       definir a,b,c,n,i Como Entero
3       Escribir "Cuantos números de la sucesión desea mostrar:"
4       Leer n
5       a<-1
6       b<-0
7       c<-1
8       Para i<-1 Hasta n Con Paso 1 Hacer
9           c<-a+b
10          a<-b
11          b<-c
12          Escribir c
13      Fin Para
14  FinProceso
```



4.5.7 Realizar un algoritmo para determinar si un punto P(x,y) está dentro o fuera de un círculo de centro C(x,y) y radio r ingresados por teclado.

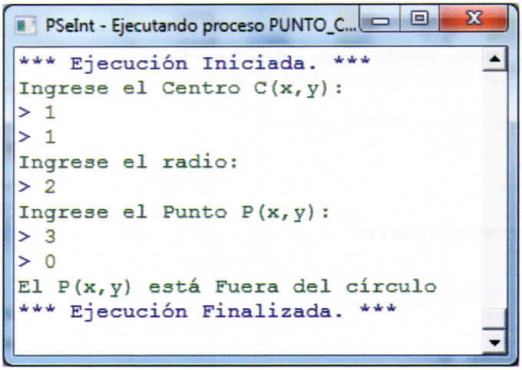
Entrada: centro C(x,y), radio, punto P(x,y)

Salida: Dentro/Fuera

Proceso: Calcular la distancia del punto al centro del círculo y compararla con el radio utilizando la fórmula de la distancia.

$$d=\sqrt{(x_2 - x_1)^2+ (y_2 - y_1)^2}$$

```
1   Proceso punto_circulo
2       definir cx, cy, r, px, py, d Como Real
3       Escribir "Ingrese el Centro C(x,y): "
4       Leer cx, cy
5       Escribir "Ingrese el radio: "
6       Leer r
7       Escribir "Ingrese el Punto P(x,y): "
8       Leer px, py
9       d=rc((px-cx)^2+(py-cy)^2)    // fórmula de la distancia
10      Si d<=r Entonces
11          Escribir "El P(x,y) está Dentro del círculo"
12      Sino
13          Escribir "El P(x,y) está Fuera del círculo"
14      Fin Si
15  FinProceso
```

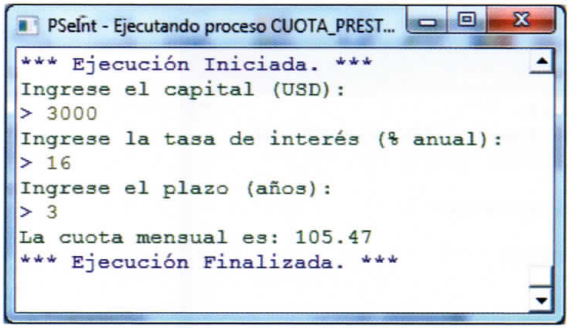


4.5.8 Realizar un algoritmo para calcular el salario neto semanal de un trabajador en función del número de horas trabajadas y la tasa de impuestos según las condiciones dadas. La tarifa y el impuesto aplican únicamente a las horas extras, es decir, sobre las 40 horas/semana.

# horas	Tarifa	Tasa de impuesto
<= 40	1 (veces)	0%
41 – 50	1.5	5%
> 50	2	8%

Entrada: # de horas/semana trabajadas y el costo/hora normal
Salida: salario neto, impuesto
Proceso: calcular el salario neto según la tabla indicada usando la estructura *si-entonces*

```
1   Proceso cuota_préstamo
2       Definir cuota,c,i Como Real
3       Definir p Como Entero
4       Escribir 'Ingrese el capital (USD):'
5       Leer c
6       Escribir 'Ingrese la tasa de interés (% anual):'
7       Leer i
8       Escribir 'Ingrese el plazo (años): '
9       Leer p
10      i<-i/100/12 // interés mensual
11      p<-p*12     // # de meses
12      cuota<-c*(i/(1-(1+i)^(-p))) // cuota fija
13      // redondea a dos decimales
14      cuota<- trunc(cuota*100)/100
15      Escribir 'La cuota mensual es: ',cuota
16  FinProceso
```



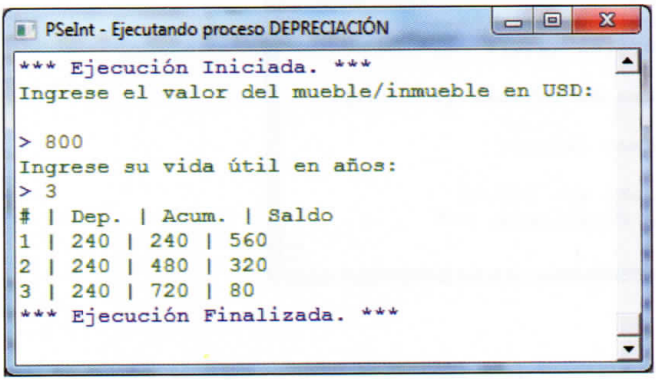
4.5.10 Realizar un algoritmo para calcular la tabla de depreciación de un bien mueble/inmueble ingresado por teclado.

Entrada: valor actual y # de años (vida útil)
Salida: Tabla de depreciación: año, depreciación anual, depreciación acumulada, saldo
Proceso: Calcular la depreciación según las siguientes fórmulas:

$$valor\ residual = 10\%\ del\ Valor\ actual$$

$$depreciadión\ anual = \frac{valor\ actual - valor\ residual}{\#\ de\ años}$$

```
1   Proceso depreciación
2       definir va,años,dep,dacum,saldo Como Real
3       definir i Como Entero
4       Escribir "Ingrese el valor del mueble/inmueble en USD: "
5       Leer va
6       Escribir "Ingrese su vida útil en años: "
7       Leer años
8       dep<-(va-0.1*va)/años    // fórmula de la depreciación anual
9       dacum<-0
10      Escribir "# | Dep. | Acum. | Saldo"
11      Para i<-1 Hasta años Con Paso 1 Hacer
12          dacum<-dacum+dep
13          saldo<- va-dacum
14          Escribir i," | ", dep, " | ", dacum, " | ", saldo
15      Fin Para
16  FinProceso
```



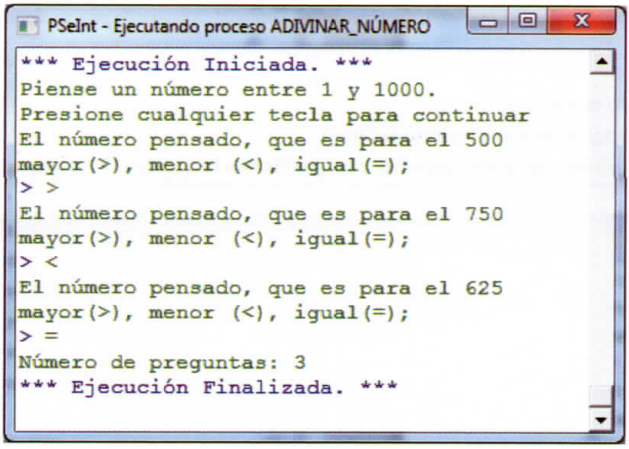
Nótese que PSeInt, al momento, no dispone de los caracteres de escape (*tabulación*) para poder formatear correctamente la salida de la tabla de depreciación.

4.5.11 Realizar un algoritmo que adivine el número pensado por usted entre 1 y 1000. *El algoritmo debe ir mostrando números y usted responderá con los símbolos '<', '>' o '=', según sea el número pensado menor, mayor o igual al mostrado en la pantalla. Cuando adivine deberá imprimir el número de preguntas que ha necesitado para adivinarlo. No debe realizar más de 10 preguntas.*

Entrada: '<', '>' , '='
Salida: Número (preguntas realizadas)
Proceso: Dividir el rango de números en 2 y mostrar al usuario y según su respuesta, descartar la parte superior o inferior. Repetir el proceso (bucle) hasta adivinar el número.

```
1   Proceso adivinar_número
2       Definir i,f,p,N Como Entero
3       Definir A Como Caracter
4       Escribir "Piense un número entre 1 y 1000."
5       Escribir "Presione cualquier tecla para continuar"
6       Esperar Tecla    // hace una pausa
7       i<-0
8       f<-1000
9       p<-0
10      Repetir
11          N<-(i+f)/2
12          Escribir 'El número pensado, que es para el ',N
13          Escribir 'mayor(>), menor (<), igual(=);'
14          Leer A
15          Si A='<' Entonces
16              f<-N
17          Sino
18              i<-N
19          FinSi
20          p<-p+1
21      Hasta Que A='=' // compara con el carácter igual
22      Escribir 'Número de preguntas: ', p
23  FinProceso
```

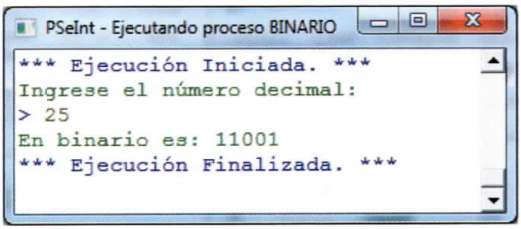
En este ejemplo se ha pensado el número 625



4.5.12 Realizar un algoritmo para convertir un número decimal ingresado por teclado a binario.

Entrada: número entero en decimal
Salida: cadena de caracteres (número en binario)
Proceso: Realizar divisiones sucesivas para 2 e ir guardando el residuo. Un número binario se representa utilizando solamente las cifras cero y uno.

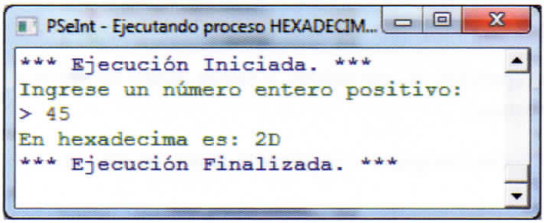
```
1   Proceso binario
2       definir s Como Caracter
3       definir n Como Entero
4       Escribir "Ingrese el número decimal: "
5       Leer n
6       Mientras n>=2 Hacer
7           s <- Concatenar(ConvertirATexto(n MOD 2),s)
8           n<-n/2
9       Fin Mientras
10      s <- Concatenar(ConvertirATexto(n),s)
11      Escribir "En binario es: ",s
12  FinProceso
```



4.5.13 Realizar un algoritmo para convertir un número decimal ingresado por teclado a hexadecimal.

Entrada: número entero en decimal
Salida: cadena de caracteres (número en hexadecimal)
Proceso: Realizar divisiones sucesivas para 16 e ir guardando el residuo. Un número hexadecimal se representa utilizando los números del 0-9 y las letras de la A-F.

```
1   Proceso hexadecimal
2       definir n, i Como Entero
3       definir hex, dig_hex Como Caracter
4       dig_hex = "0123456789ABCDEF"
5       Escribir "Ingrese un número entero positivo:"
6       Leer n
7       hex<-"" // cadena vacía
8       Repetir
9           i<- n mod 16 + 1
10          hex <- Concatenar(SubCadena(dig_hex,i,i), hex)
11          n <- trunc(n/16)
12      Hasta Que n=0
13      Escribir "En hexadecimal es: ", hex
14  FinProceso
```



4.6 Observaciones del capítulo

- Los algoritmos se vuelven más legibles cuando el pseudocódigo está formateado, es decir, se utiliza indentado (menú '*Editar*', opción '*corregir indentado*').
- El pseudocódigo es mucho más potente y compacto que los diagramas de flujo, así como más parecido a los lenguajes de programación disponibles. Razón por lo cual es utilizado ampliamente, principalmente para resolver problemas de complejidad considerable.
- En los ejercicios resueltos, no necesariamente se representa el algoritmo más eficiente o más legible.
- El pseudocódigo permite incorporar '*comentarios*' para dar una explicación adicional, a diferencia de los diagramas de flujo.

4.7 Ejercicios propuestos

Realizar algoritmos utilizando pseudocódigo para:

4.7.1 Leer por teclado los tres vértices de un triángulo cualesquiera e imprima su área y el tipo de triángulo (equilátero, isósceles, escaleno).

$$p = \frac{a + b + c}{2}$$

$$A = \sqrt{p(p - a)\ (p - b)\ (p - c)}$$ *Formula de Herón*

4.7.2 Leer por teclado el día y mes de nacimiento de una persona e imprima su signo zodiacal.

4.7.3 Eliminar los espacios ubicados a la derecha de una palabra ingresada por teclado.

4.7.4 Leer por teclado una frase de texto e imprimir la cantidad de vocales que existen por cada una y en total. Considere las tildes, mayúsculas y minúsculas.

4.7.5 Ingresar por teclado una frase e imprima el número de palabras que existe en el texto. Considere que puede haber varios espacios al inicio, entre o fin de las palabras.

4.7.6 Calcular la frecuencia de aparición de una palabra o patrón dentro de una frase ingresados por teclado (frase y patrón).

4.7.7 Ingresar por teclado una fecha cualquiera e imprima la fecha del día siguiente. Considérese los años bisiestos.

4.7.8 Ingresar por teclado una frase y ponga en mayúsculas la letra inicial de cada palabra.

4.7.9 Sustituir todas las apariciones de una palabra por otra en una frase. Ingrese por teclado la frase y las dos palabras (la que busca y la que reemplaza).

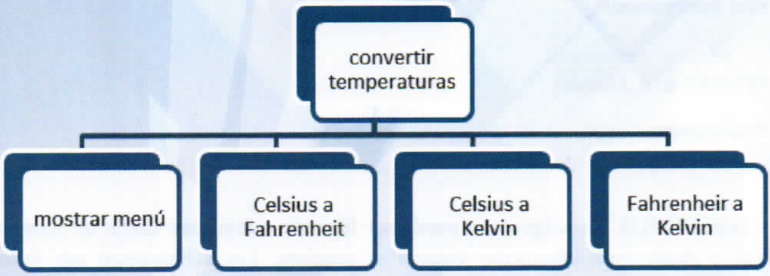
4.7.10 Convertir un número binario a decimal

4.7.11 Convertir un número hexadecimal a decimal

4.7.12 Determinar si un punto P(x,y) está dentro o fuera de un triángulo. Ingrese por teclado el punto y los tres vértices.

CAPÍTULO V

DISEÑO MODULAR (DESCENDENTE O TOP DOWN)



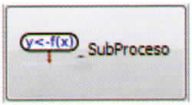
En este capítulo usted encontrará:

- *Procesos/Subprocesos*
- *Ejercicios de modularidad*
- *Variables locales y globales*
- *Paso de Parámetros: por valor y por referencia*
- *Ejercicios de paso de parámetros*
- *Ejercicios propuestos*

5. Diseño modular (descendente o top-down)

5.1 Subprocesos (métodos: funciones y procedimientos)

Como se mencionó en el capítulo I (sobre la programación modular), un problema se puede dividir en subproblemas más sencillos que abordan una tarea específica a la vez. Para esto se puede crear varios subprocesos o métodos (funciones/procedimientos) en el algoritmo utilizando el comando '*subproceso*', cuya sintaxis es la siguiente:




```
1  SubProceso variable_de_retorno <- Nombre ( Argumentos )
2
3  Fin SubProceso
4
5  // método principal
6  Proceso sin_titulo
7
8  FinProceso
```

Según López (2011), un subproceso puede ser llamado o invocado desde un proceso principal o desde otro subproceso cuando se requiera. Los subprocesos son útiles cuando:

- Existe una tarea específica que debe ejecutarse en más de una ocasión (reutilización).
- Cuando un problema es complejo o extenso.

Si el subproceso no retorna ningún valor, se puede omitir la *variable_de_retorno* y el operador de *asignación* (←). En este caso se le conoce como ***procedimiento***. Si el subproceso retorna un valor se conoce como ***función***. Si el subproceso no recibe ningún valor (***parámetro***) se puede colocar los paréntesis vacíos u omitirse.

Para invocar al subproceso se debe utilizar su nombre y entre paréntesis los ***argumentos***. Hay que tener mucho cuidado que los argumentos coincidan con los parámetros en cantidad, tipo y orden. De igual manera, entre la *variable_de_ retorno* con la variable que recoge o almacena dicho valor.

Según Sznajdleder (2012), existe una diferencia sutil entre argumentos y parámetros:

- A un subproceso le pasamos ***argumentos***
- El subproceso recibe ***parámetros***.

En López (2011), se le llama ***parámetro formal*** al definido en el subproceso y ***parámetro actual*** al utilizado al hacer la llamada al subproceso.

Llamar a un subproceso quiere decir que se usa su nombre para atraerlo, causando que se ejecute. Cuando las tareas del subproceso están completas, el control regresa al punto desde el que se llamó en el programa principal u otro subproceso.

Los argumentos de un subproceso serán definidos en el proceso que realice la llamada, y por lo tanto, el subproceso los recibirá con el mismo tipo de dato ya definido. Por esto, no se debe volver a definir otra vez el tipo de dato de la variable que es argumento dentro del subproceso.

De igual forma, no es necesario que los nombres de las variables (*argumentos*) cuando invoca o llama al subproceso coincidan con los nombres de las variables (*parámetros*) definidos en el subproceso.

Según Farrell (2013), algunas ventajas de la modularización son:

- **Abstracción.-** Es el proceso de poner atención en las propiedades importantes mientras se ignoran los detalles no esenciales. Permite ver todo el panorama.
- **Reutilización**.- Los subprocesos pueden usarse más de una vez en el mismo programa o en otros.
- **Trabajo en paralelo**.- Al dividir una tarea o problema grande en módulos, se puede con más facilidad asignar las subtareas entre varias personas.

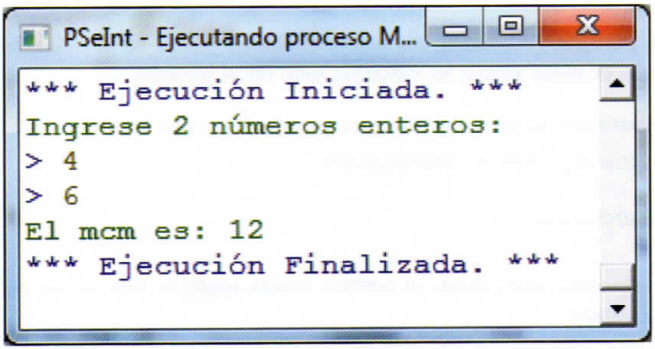
A continuación presentamos algunos algoritmos en pseudocódigo que hace uso de los subprocesos como consecuencia de la modularización.

5.2 Ejercicios de modularidad

5.2.1 Realizar un algoritmo que calcule el mínimo común múltiplo (m.c.m) de dos números leídos por teclado

Entrada: dos números enteros
Salida: Número entero (mcm)
Proceso: calcular el mcm utilizando la fórmula: $mcm(a,b)=\frac{a*b}{MCD\ (a,b)}$. El MCD es el máximo común divisor realizado anteriormente.

```
1   // devuelve el MCD de los parámetros a y b
2   SubProceso ret <- MCD (a,b) // aquí a y b son parámetros
3       definir ret como entero
4       Mientras a<>b Hacer
5           Si a>b Entonces
6               a<-a-b
7           Sino
8               b<-b-a
9           FinSi
10      FinMientras
11      ret<-a
12  Fin SubProceso
13
14  // método principal
15  Proceso mcm
16      definir a,b,c Como Entero
17      Escribir "Ingrese 2 números enteros: "
18      Leer a,b
19      c<-MCD(a,b) // invoca al subproceso. Aquí a y b son argumentos
20      Escribir "El mcm es: ",(a*b)/c
21  FinProceso
```

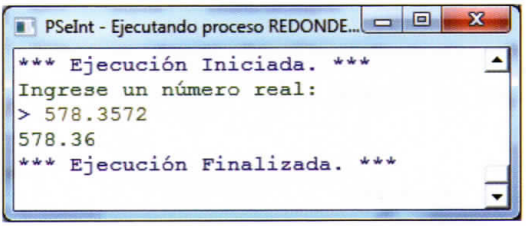


El subproceso MCD es una función porque retorna un valor.

5.2.2 Realizar un algoritmo para redondear un número real ingresado por teclado a dos decimales.

Entrada: un número real
Salida: Número real (redondeado a 2 decimales)
Proceso: utilizar la función REDON y multiplicar por cien y dividir por 100.

```
1   // devuelve el número redondeado a 2 decimales
2   SubProceso ret <- round ( n )
3       definir ret Como Real
4       ret <- REDON(n*100)/100
5   Fin SubProceso
6
7   // método principal
8   Proceso redondear
9       definir r, n Como Real
10      Escribir "Ingrese un número real:"
11      Leer n
12      r <- round(n)    // invoca al subproceso
13      Escribir r;
14  FinProceso
```



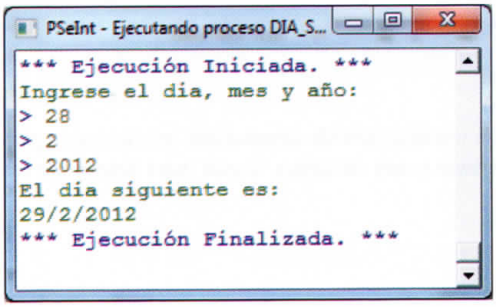
Al momento PSeInt no cuenta con una función incorporada que redondee a n-decimales.

5.2.3 Hacer un algoritmo que dados el día, mes y año, calcule cual es el siguiente día. Se debe considerar los años bisiestos.

Entrada: tres enteros (día, mes y año)
Salida: día, mes y año (siguiente día)
Proceso: Calcular el día siguiente. Se utiliza el método desarrollado anteriormente para determinar si un año es bisiesto y una estructura *'según'* para determinar los días de cada mes.

```
1  // verifica si el parámetro año es bisiesto
2  SubProceso ret <- esBisiesto ( año )
3      definir ret como logico
4      Si año mod 4 =0 Y (año mod 100<>0 O año mod 400=0) Entonces
5          ret <- verdadero
6      Sino
7          ret <- Falso
8      Fin Si
9  Fin SubProceso

10
11 // método principal
12 Proceso dia_siguiente
13     Definir d,m,a,dias Como Entero
14     Escribir 'Ingrese el dia, mes y año: '
15     Leer d, m, a
16     segun m hacer
17         1,3,5,7,8,10,12: dias<-31
18         4,6,9,11: dias<-30
19         2:
20             dias<-28
21             si esBisiesto(a) entonces    // invoca al subproceso
22                 dias<-dias+1
23             FinSi
24     FinSegun
25     d<-d+1
26     si d> dias entonces
27         d<-1    // primer día del mes
28         m<-m+1
29     FinSi
30     si m> 12 entonces
31         m<-1    // primer mes (enero)
32         a<-a+1
33     FinSi
34     Escribir 'El dia siguiente es: '
35     Escribir d,'/',m,'/',a
36 FinProceso
```

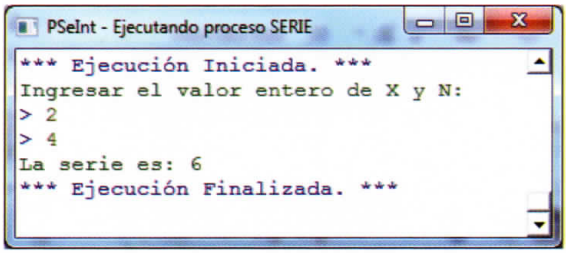


5.2.4 Realizar un algoritmo para calcular el valor de S, ingresando por teclado los valores de x y n.

$$S = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

Entrada: dos enteros (x y n)
Salida: número real (S)
Proceso: Calcular el valor de S. Se utiliza el método desarrollado anteriormente para calcular el factorial de un número. Se hace uso de un bucle desde 1 hasta n para las sumas.

```
1   //devuelve el factorial del parámetro n
2   SubProceso ret <- factorial (n)
3       Definir ret,fac,i Como Entero
4       fac<-1
5       Para i<-1 Hasta n Hacer
6           fac<-fac*i
7       FinPara
8       ret<-fac
9   Fin SubProceso
10
11  // método principal
12  Proceso serie
13      Definir x, n, i Como Entero
14      definir S Como Real
15      Escribir 'Ingresar el valor entero de X y N: '
16      Leer x, n
17      S<-0
18      Para i<-1 Hasta n Hacer
19          S <- S + x^i/factorial(i)    // invoca al subproceso
20      FinPara
21      Escribir 'La serie es: ',S
22  FinProceso
```



5.2.5 Calcular e imprimir la tabla de amortización para un préstamo bancario utilizando el sistema de amortización francés. El capital (USD), tasa de interés (anual) y plazo (meses) será ingresado por teclado.

Entrada: tres números (capital, tasa y plazo)
Salida: tabla de amortización

# mes	cuota fija	capital	interés	saldo
1				
2				
3				
...				

Proceso: calcular la cuota fija mensual, capital pagado, interés pagado, y saldo del préstamo utilizando el sistema de amortización francés, es decir, la cuota fija, el capital incrementa y el interés disminuye en el tiempo.

$$cuota\ mensual = c * \frac{i}{1 - (1+i)^{-p}}$$

Siendo c el capital prestado, i la tasa de interés mensual y p el número de cuotas (# meses).

```
1  // devuelve el parámetro n redondeado a 2 decimales
2  SubProceso ret <- round ( n )
3      definir ret Como Real
4      ret <- REDON(n*100)/100
5  Fin SubProceso
```

```
6
7   // método principal
8   Proceso tabla_amortizacion_frances
9       definir tiempo, i Como Entero
10      definir cuota, tasa, deuda, int, cap como real
11      ESCRIBIR "TABLA DE AMORTIZACION CON EL MÉTODO FRANCÉS"
12      Escribir "Ingrese el valor del préstamo (USD):"
13      leer deuda
14      escribir "Ingrese el número de cuotas (meses):"
15      leer tiempo
16      escribir "Ingrese la tasa de interes anual (en %):"
17      leer tasa
18      tasa <- tasa/100/12 // interés mensual
19      cuota <- deuda*(tasa/(1-(1+tasa)^(-tiempo)))     // cuota fija
20      escribir "N° ","CUOTA    ","CAPITAL  ","INTERÉS    ","SALDO"
21      Para i<-1 Hasta tiempo Con Paso 1 Hacer
22          int<-tasa*deuda      // interés pagado
23          cap<-cuota-int       // capital pagado
24          deuda<-deuda-cap     // saldo
25          // invoca al subproceso
26          escribir i,"  ", round(cuota),"    ", round(cap) Sin Saltar
27          escribir "    ", round(int),"    ",round(deuda)
28      Fin Para
29  FinProceso
```

```
PSeInt - Ejecutando proceso TABLA_AMORTIZACION_FR...
*** Ejecución Iniciada. ***
TABLA DE AMORTIZACION CON EL MÉTODO FRANCÉS
Ingrese el valor del préstamo (USD):
> 3000
Ingrese el número de cuotas (meses):
> 6
Ingrese la tasa de interes anual (en %):
> 16
N° CUOTA   CAPITAL  INTERÉS   SALDO
1  523.59   483.59   40   2516.41
2  523.59   490.04   33.55   2026.37
3  523.59   496.57   27.02   1529.8
4  523.59   503.19   20.4   1026.6
5  523.59   509.9   13.69   516.7
6  523.59   516.7   6.89   0
*** Ejecución Finalizada. ***
```

Nótese que PSeInt, al momento, no dispone de los caracteres de escape (*tabulación*) para poder formatear correctamente la salida de la tabla de amortización.

5.2.6 Realizar un algoritmo para calcular el área de la circunferencia circunscrita a un triángulo rectángulo cuyos vértices son ingresados por teclado.

Entrada: tres vértices

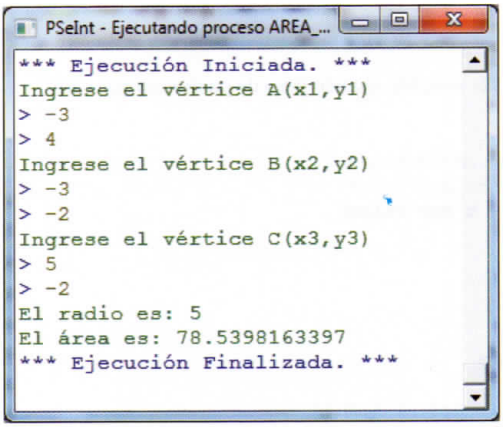
Salida: dos números (radio y área)

Proceso: Calcular el circuncentro, el radio y el área. El Circuncentro de un triángulo ***rectángulo*** es el punto medio de la hipotenusa. El radio estaría dado por la distancia del circuncentro a cualquier vértice. El área está dada por la fórmula $A = \pi r^2$. La fórmula de la distancia es:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
1   // devuelve el mayor de los tres números
2   SubProceso ret <- mayorTresNumeros ( a,b,c )
3     definir ret Como Real
4     Si a>b Y a>c Entonces    // condición compuesta
5       ret<- a
6     Sino
7       Si b>c Entonces       // condición simple
8         ret<- b
9       Sino
10        ret<- c
11      FinSi
12    FinSi
13  Fin SubProceso

15  // método principal
16  Proceso area_circunferencia
17      Definir x1, y1, x2, y2, x3, y3 Como Entero
18      definir AB,BC,AC,h,r como real
19      escribir "Ingrese el vértice A(x1,y1)"
20      leer x1,y1
21      escribir "Ingrese el vértice B(x2,y2)"
22      leer x2,y2
23      escribir "Ingrese el vértice C(x3,y3)"
24      leer x3,y3
25      AB<-rc((x1-x2)^2+(y1-y2)^2) // fórmula de la distancia
26      BC<-rc((x2-x3)^2+(y2-y3)^2)
27      AC<-rc((x1-x3)^2+(y1-y3)^2)
28      // Invoca al subproceso para determinar la hipotenusa
29      h<- mayorTresNumeros(AB, BC, AC)
30      r<-h/2  // radio
31      Escribir "El radio es: ",r
32      Escribir "El área es: ", (PI*r^2)
33  FinProceso
```



Aquí se hace uso de la constante PI incorporada en PSeInt.

5.3 Variables locales y globales

Las variables son ***locales*** a cada método donde han sido definidas o declaradas, es decir, se pueden usar solamente en ese método. Si se intenta hacer uso de dicha variable en otro proceso/subproceso donde no ha sido declarado, el programa PSeInt dará un mensaje de error. Las **variables globales** se definen fuera del cuerpo de un método y puede ser vista y manipulada por todos los métodos del algoritmo. Al momento en PSeInt no existe el concepto de variable *Global.*

5.4 Paso de parámetros por valor y referencia

En la definición de los *parámetros* de un proceso o subproceso se puede agregar las palabras claves '***Por Valor***' o '***Por Referencia***' para indicar como se pasa cada parámetro. Si se omiten, por defecto el paso será *Por valor* a excepción de los arreglos (veremos en el siguiente capítulo) que se pasan por referencia.

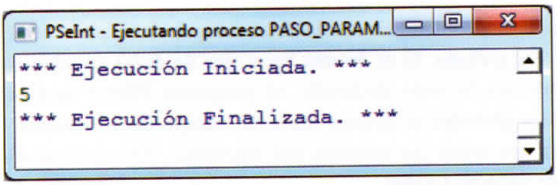
El *paso por referencia* implica que si en el subproceso se modifica el contenido del argumento, este modificará la variable que se utilizó en la invocación o llamada, es decir, se opera sobre la variable original (trabajan en el mismo espacio de memoria). *El paso por Valor* implica que si en el subproceso se modifica el contenido del argumento, este no se verá reflejado en la variable que se utilizó en la invocación, es decir, se opera sobre una copia de la variable (diferente espacio de memoria).

5.4.1 Ejercicios de paso de parámetros

5.4.1.1 Por valor (por defecto)

Entrada: un número entero
Salida: Número entero
Proceso: incrementar en una unidad la variable usando paso de parámetro por valor

```
1   // Este procedimiento no devueve nada.
2   // El paso de parámetro es por valor
3   SubProceso incrementar ( n Por Valor)
4       n<-n+1
5   Fin SubProceso
6
7   // método principal
8   Proceso paso_parámetros
9       definir a Como Entero
10      a<-5
11      incrementar(a)  // invoca al subproceso
12      Escribir a
13  FinProceso
```

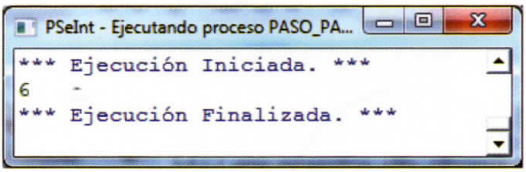


Observe que el cambio realizado en la variable '*n*' en el subproceso (procedimiento) no afecta a la variable original '*a*' del proceso.

5.4.1.2 Por Referencia

Entrada: un número entero
Salida: Número entero
Proceso: incrementar en una unidad la variable usando paso de parámetro por referencia

```
1   // Este procedimiento no devueve nada.
2   // El paso de parámetro es por referencia
3   SubProceso incrementar ( n Por Referencia)
4       n<-n+1
5   Fin SubProceso
6
7   // método principal
8   Proceso paso_parámetros
9       definir a Como Entero
10      a<-5
11      incrementar(a)  // invoca al subproceso
12      Escribir a
13  FinProceso
```



Observe que el cambio realizado en la variable '*n*' en el subproceso (procedimiento) si afecta a la variable original '*a*' del proceso.

En los ejercicios resueltos en los próximos capítulos se hará uso del paso de parámetros por valor y referencia de acuerdo al problema a resolver.

5.5 Ejercicios propuestos

Utilizando subprocesos, realizar algoritmos en pseudocódigo para:

5.5.1 Eliminar los espacios ubicados a la izquierda (inicio) y a la derecha (fin) de una palabra ingresada por teclado.

5.5.2 Insertar los separadores de miles y decimales a un número real (valor monetario en USD) ingresado por teclado, ejemplo: 12689.7→ 12,689.70 | 893→ 893.00

5.5.3 Descomponer un número ingresado por teclado en sus factores primos.

5.5.4 Calcular el número de días que faltan del año, a partir de una fecha ingresada por teclado.

5.5.5 Calcular el valor de T, ingresando por teclado los valores de x y n.

$$T = 1 + \frac{2x^2}{3!} + \frac{3x^4}{5!} + \frac{4x^6}{7!} + \cdots + \frac{nx^m}{p!}$$

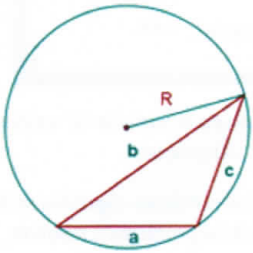
5.5.6 Calcular el valor de T, ingresando por teclado los valores de x y n.

$$T = 1 + \frac{2x^2}{3!} + \frac{3x^4}{5!} + \frac{4x^6}{7!} + \cdots + \frac{nx^m}{p!}$$

5.5.7 Convertir la temperatura de grados Fahrenheit ingresada por teclado a Kelvin y viceversa.

5.5.8 Intercambiar los valores de dos variables numéricas ingresadas por teclado, utilizando un subproceso y el paso de parámetros por referencia.

5.5.9 Calcular el área de la circunferencia circunscrita a un triángulo cualquiera (rectángulo, acutángulo u obtusángulo) cuyos vértices A, B, C son ingresados por teclado.



Sugerencia: puede usar la fórmula:

$$R = \frac{abc}{4(A_T)}$$

Dónde: A_T es el área del triángulo; *a, b, c* son los lados del triángulo; y *R* el radio de la circunferencia circunscrita al triángulo. El área del triángulo puede calcularse usando la fórmula de *Herón*:

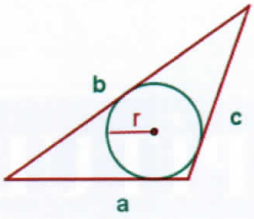
$p = \frac{a+b+c}{2}$ (Semiperímetro)

$$A_T = \sqrt{p(p-a)\,(p-b)\,(p-c)}$$

Los lados del triángulo (*a, b, c*) se pueden calcular con la fórmula de la distancia:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

5.5.10 Calcular el área de la circunferencia inscrita a un triángulo cualquiera (rectángulo, acutángulo u obtusángulo) cuyos vértices A, B, C son ingresados por teclado.



Sugerencia: puede usar la fórmula:

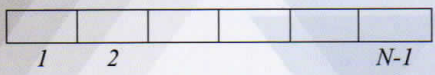
$$A_2 = rp$$

Dónde: A_T es el área del triángulo; r el radio de la circunferencia inscrita al triángulo y p el semiperímetro del triángulo.

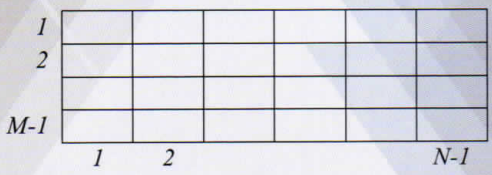
CAPÍTULO VI

ARREGLOS UNIDIMENSIONALES Y BIDIMENSIONALES

Arreglo Unidimensional



Arreglo bidimensional



En este capítulo usted encontrará:

- *Arreglos unidimensionales (Vectores)*
  - *Ejercicios de vectores*
  - *Ejercicios propuestos de vectores*
- *Arreglos bidimensionales (matriz)*
  - *Ejercicios de matrices*
  - *Ejercicios propuestos de matrices*

6 Arreglos

6.1 Arreglos unidimensionales (vectores)

Es un conjunto finito y homogéneo de datos, es decir, del mismo tipo de dato, y se almacenan bajo un mismo nombre, y se diferencian por la posición que tiene cada elemento dentro del arreglo (López, 2011). El primer elemento tendrá el índice uno, el segundo el dos y así sucesivamente. En algunos lenguajes de programación el primer índice es el cero (c#, java).

A continuación se muestra gráficamente en vector de cinco elementos:

Para declarar un vector utilizamos las palabras reservadas '*definir*' y '*dimensión*'. Por ejemplo, para definir un vector '*A*' de enteros y almacenar cinco elementos utilizamos la siguiente sintaxis:

```
definir A como entero
dimension A(5)
A(1)<- 10
A(2)<- 15
A(3)<- 20
A(4)<- 25
A(5)<- 30
```

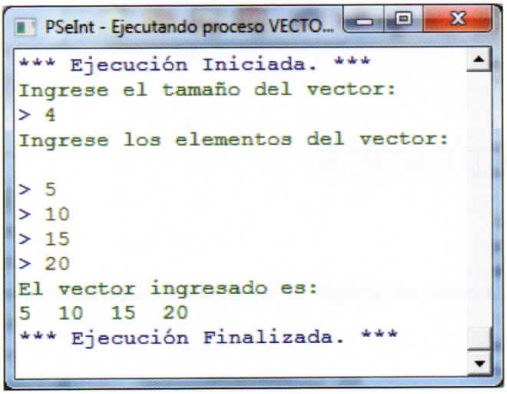
1	2	3	4	5
10	15	20	25	30

6.1.1 Ejercicios con Vectores

6.1.1.1 Realizar un algoritmo para crear un arreglo de n elementos ingresados por teclado.

Entrada: tamaño y elementos del vector
Salida: vector
Proceso: crear el vector y almacenar los elementos

```
1   // imprime los elementos del arreglo en una sola línea
2   SubProceso mostrarArreglo ( A, n )
3       definir i Como Entero
4       Para i<-1 Hasta n Con Paso 1 Hacer
5           Escribir A(i), "  " Sin Saltar
6       Fin Para
7       Escribir "" // salto de línea
8   Fin SubProceso
9
10  // método principal
11  Proceso vectores
12      definir A como entero
13      definir n, i Como Entero
14      Escribir "Ingrese el tamaño del vector: "
15      Leer n
16      dimension A(n)
17      Escribir "Ingrese los elementos del vector: "
18      Para i<-1 Hasta n Con Paso 1 Hacer
19          Leer A(i)
20      Fin Para
21      Escribir "El vector ingresado es: "
22      mostrarArreglo(A, n)     // invoca al subproceso
23  FinProceso
```

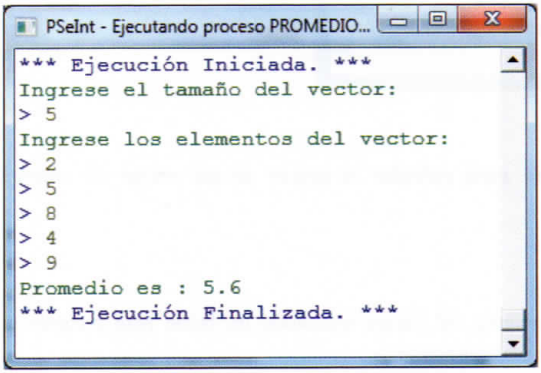


Observe que al invocar al subproceso '*mostrarArreglo*' el vector '*A*' se pasa por referencia y la variable '*n*' por valor.

6.1.1.2 Realizar un algoritmo para calcular el promedio de un vector de enteros ingresados por teclado.

Entrada: arreglo de enteros
Salida: número (promedio)
Proceso: calcula el promedio: suma de los números dividido para el tamaño utilizando un bucle.

```
1   Proceso promedio_vector
2       definir A como entero
3       definir n, sum, i Como Entero
4       Escribir "Ingrese el tamaño del vector: "
5       Leer n
6       sum<-0
7       dimension A(n)
8       Escribir "Ingrese los elementos del vector:"
9       Para i<-1 Hasta n Con Paso 1 Hacer
10          Leer A(i)
11          sum <- sum + A(i)
12      Fin Para
13      Escribir "Promedio es : " sum/n
14  FinProceso
```

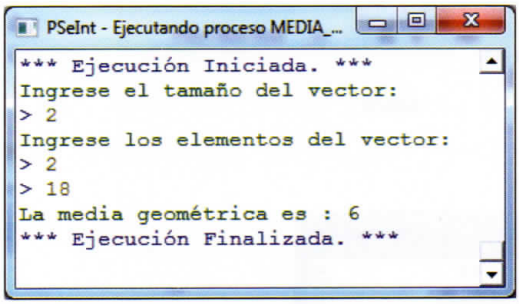


6.1.1.3 Realizar un algoritmo para calcular la media geométrica de un vector de enteros ingresados por teclado.

Entrada: arreglo de enteros
Salida: número real (media geométrica)
Proceso: calcula media geométrica según la fórmula:

$$x = \sqrt[n]{x_1 * x_2 * \cdots * x_n}$$

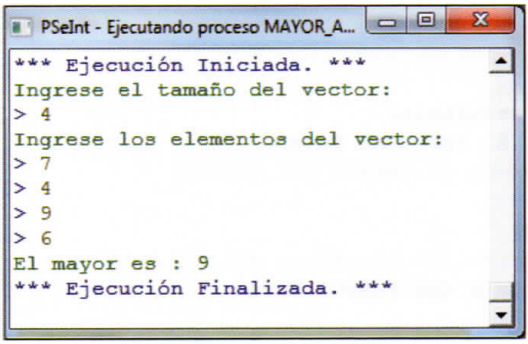
```
1   Proceso media_geométrica
2       definir A como entero
3       definir n, m, i Como Entero
4       Escribir "Ingrese el tamaño del vector: "
5       Leer n
6       m<-1
7       dimension A(n)
8       Escribir "Ingrese los elementos del vector:"
9       Para i<-1 Hasta n Con Paso 1 Hacer
10          Leer A(i)
11          m <- m * A(i)
12      Fin Para
13      Escribir "La media geométrica es : ", m^(1/n)    // raíz n-ésima
14  FinProceso
```



6.1.1.4 Realizar un algoritmo para calcular el mayor de un vector de enteros ingresados por teclado.

Entrada: arreglo de enteros
Salida: número (mayor)
Proceso: calcula el mayor elemento del vector utilizando un bucle para recorrer el vector y una selección para comparar.

```
1   Proceso mayor_arreglo
2       definir A como entero
3       definir n, m, i Como Entero
4       Escribir "Ingrese el tamaño del vector: "
5       Leer n
6       dimension A(n)
7       Escribir "Ingrese los elementos del vector:"
8       Para i<-1 Hasta n Con Paso 1 Hacer
9           Leer A(i)
10      Fin Para
11      m<-A(1) // primer elemento
12      Para i<-1 Hasta n Con Paso 1 Hacer
13          Si A(i) > m Entonces
14              m<-A(i)
15          Fin Si
16      Fin Para
17      Escribir "El mayor es : ", m
18  FinProceso
```

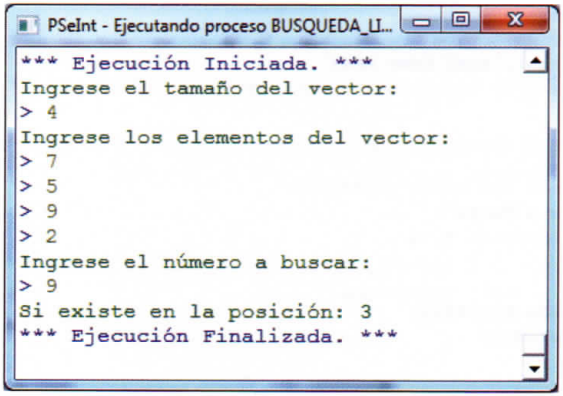


Observe que los dos bucles son secuenciales, no anidados. ¿Se podría resolver el ejercicio usando un solo bucle? ¿Funcionaría para números negativos únicamente?

6.1.1.5 Realizar un algoritmo para la búsqueda secuencial de un elemento en un arreglo de enteros ingresados por teclado.

Entrada: arreglo de enteros y número a buscar
Salida: número (posición)
Proceso: Verificar si un número existe en el arreglo utilizando la búsqueda secuencial. Si encuentra el número retorna la posición que ocupa en el arreglo (≥1), caso contrario retorna cero.

```
1  // si existe el número en el vector devuelve el índice
2  // caso contrario devuelve cero
3  SubProceso ret <- buscar ( A, n, num )
4      definir i, ret Como Entero
5      definir break Como Logico
6      break<- Verdadero    // es un valor centinela
7      ret<- 0
8      i<- 1
9      Mientras i <= n Y break Hacer
10         Si num = A(i) Entonces
11             ret<- i
12             break<- Falso
13         Sino
14             i<-i+1
15         Fin Si
16     Fin Mientras
17 Fin SubProceso
18
19 // método principal
20 Proceso busqueda_secuencial
21     definir n, i, A, num Como Entero
22     Escribir "Ingrese el tamaño del vector: "
23     Leer n
24     dimension A(n)
25     Escribir "Ingrese los elementos del vector:"
26     Para i<-1 Hasta n Con Paso 1 Hacer
27         Leer A(i)
28     Fin Para
29     Escribir "Ingrese el número a buscar: "
30     Leer num
31     i<- buscar(A, n, num)    // invoca al subproceso
32     Si i = 0 Entonces
33         Escribir "No existe el número"
34     Sino
35         Escribir "Si existe en la posición: ", i
36     Fin Si
37 FinProceso
```



Nótese que al invocar al subproceso '*buscar*' (línea 31), el vector *A* se pasa por referencia y las otras variables (*n* y *num*) por valor.

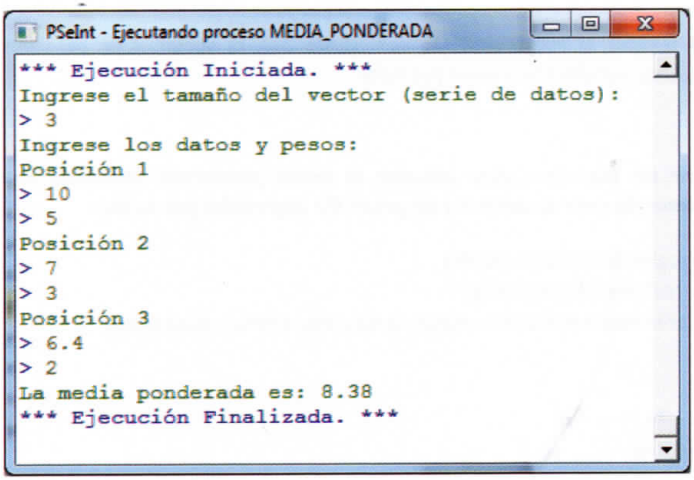
6.1.1.6 Realizar un algoritmo para calcular la media ponderada utilizando dos arreglos de enteros (la serie de datos X y los pesos W) ingresados por teclado.

Entrada: dos arreglos de números (X, W)
Salida: número real (media ponderada)
Proceso: calcular la media ponderada usando la siguiente fórmula matemática:

$$x = \{x_1, x_2, x_3, \ldots, x_n\}$$
$$w = \{w_1, w_2, w_3, \ldots, w_n\}$$

$$\bar{x} = \frac{x_1 w_1 + x_2 w_2 + x_3 w_3 + \cdots + x_n w_n}{w_1 + w_2 + w_3 + \cdots + w_n}$$

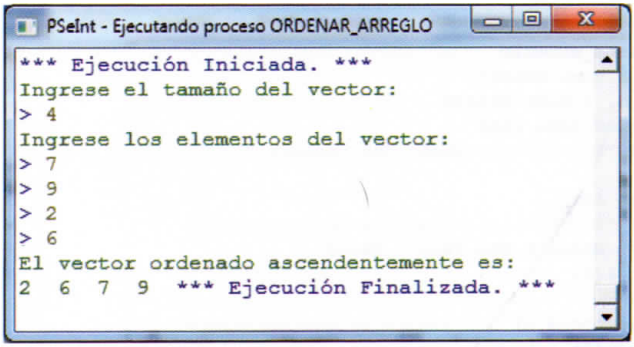
```
1  Proceso media_ponderada
2      Definir i, n  Como Entero
3      definir X, W, sum1, sum2 Como Real
4      Escribir 'Ingrese el tamaño del vector (serie de datos):'
5      Leer n
6      Dimension X(n)
7      Dimension W(n)
8      Escribir "Ingrese los datos y pesos:"
9      Para i<-1 Hasta n Hacer
10         Escribir "Posición ", i
11         Leer X(i)
12         Leer W(i)
13         sum2 <- sum2+ X(i)*W(i)
14         sum1 <- sum1+W(i)
15     FinPara
16     Escribir "La media ponderada es: ", sum2/sum1
17 FinProceso
```



6.1.1.7 Realizar un algoritmo para ordenar ascendentemente e imprimir un vector de enteros ingresados por teclado.

Entrada: arreglo de enteros
Salida: vector ordenado ascendentemente
Proceso: ordenar el vector con el método de intercambio usando dos bucles anidados.

```
1   Proceso ordenar_arreglo
2       definir A como entero
3       definir n, aux, i,j Como Entero
4       Escribir "Ingrese el tamaño del vector: "
5       Leer n
6       dimension A(n)
7       Escribir "Ingrese los elementos del vector:"
8       Para i<-1 Hasta n Con Paso 1 Hacer
9       |   Leer A(i)
10      Fin Para
11      // ordena usando el método de intercambio
12      Para i<-1 Hasta n-1 Con Paso 1 Hacer
13      |   Para j<-i+1 Hasta n Con Paso 1 Hacer
14      |   |   Si A(i) > A(j) Entonces
15      |   |   |   aux<-A(i)
16      |   |   |   A(i)<- A(j)
17      |   |   |   A(j)<-aux
18      |   |   Fin Si
19      |   Fin Para
20      Fin Para
21      Escribir "El vector ordenado ascendentemente es:"
22      Para i<-1 Hasta n Con Paso 1 Hacer
23      |   Escribir A(i), "  " Sin Saltar
24      Fin Para
25  FinProceso
```



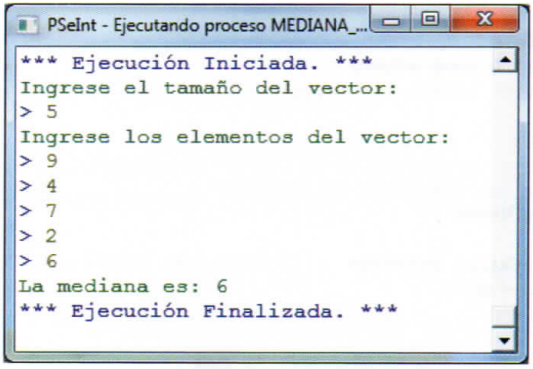
Se deja como ejercicio para el lector dividir el algoritmo en subprocesos: *ingresarVector*, *ordenarVector*, *mostrarVector* y el método principal. Utilice el paso de parámetros por valor y referencia donde sea necesario.

6.1.1.8 Realizar un algoritmo para calcular la mediana de un vector de enteros ingresados por teclado.

Entrada: arreglo de enteros
Salida: número real (mediana)
Proceso: calcula la mediana. Es el elemento central del arreglo previamente ordenado. Si el tamaño es par, la mediana es el promedio de los dos elementos centrales.

```
1  // ordena el vector A ascendentemente por intercambio
2  SubProceso ordenarVector(A, n)
3      definir i,j, aux Como Entero
4      Para i<-1 Hasta n-1 Con Paso 1 Hacer
5          Para j<-i+1 Hasta n Con Paso 1 Hacer
6              Si A(i) > A(j) Entonces
7                  aux<-A(i)
8                  A(i)<- A(j)
9                  A(j)<-aux
10             Fin Si
11         Fin Para
12     Fin Para
13 FinSubProceso
```

```
15 // método principal
16 Proceso mediana_arreglo
17     definir A como entero
18     definir n, i Como Entero
19     definir med como real
20     Escribir "Ingrese el tamaño del vector: "
21     Leer n
22     dimension A(n)
23     Escribir "Ingrese los elementos del vector:"
24     Para i<-1 Hasta n Con Paso 1 Hacer
25         Leer A(i)
26     Fin Para
27     ordenarVector(A,n)  // invoca al procedimiento
28     i<-trunc(n/2)
29     Si n mod 2=0 Entonces   // ¿el tamaño del vector es par?
30         med <-(A(i)+A(i+1))/2   // dos elementos centrales
31     Sino
32         med <-A(i+1)    // un elemento central
33     Fin Si
34     Escribir "La mediana es: ",med
35 FinProceso
```



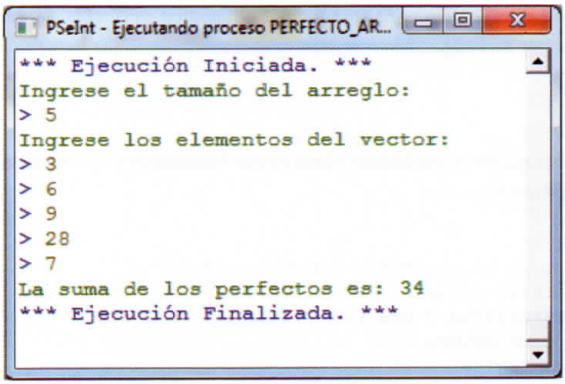
Observe que en la línea #27 se invoca al procedimiento '*ordenarVector*'. *El paso de parámetros* del arreglo '*A*' es por referencia y la variable '*n*' es por valor.

6.1.1.9 Realizar un algoritmo para calcular la suma de los números perfectos de un vector de enteros ingresados por teclado.

Entrada: arreglo de enteros
Salida: número (suma)
Proceso: calcula la suma de los números perfectos (véase como determinar si un número es perfecto en el capítulo 4)

```
1   // retorno verdadero si el parámetro num es perfecto
2   // caso contario retorna falso
3   SubProceso ret <- esPerfecto ( num )
4       definir i, sum como entero
5       definir ret como logico
6       sum <- 0
7       ret<- falso
8       Para i<-1 Hasta num-1 Con Paso 1 Hacer
9           Si num mod i = 0 Entonces
10              sum = sum + i
11          FinSi
12      Fin Para
13      Si sum = num Entonces
14          ret <- verdadero
15      Fin Si
16  Fin SubProceso
17
```

```
18  // método principal
19  Proceso perfecto_arreglo
20      definir n, i, sum, A como entero
21      Escribir "Ingrese el tamaño del arreglo: "
22      Leer n
23      Dimension A(n)
24      sum<-0
25      Escribir "Ingrese los elementos del vector:"
26      Para i<-1 Hasta n Hacer
27          Leer A(i)
28          Si esPerfecto(A(i)) Entonces     // invoca al subproceso
29              sum<- sum + A(i)
30          Fin Si
31      FinPara
32      Escribir "La suma de los perfectos es: ", sum
33  FinProceso
```

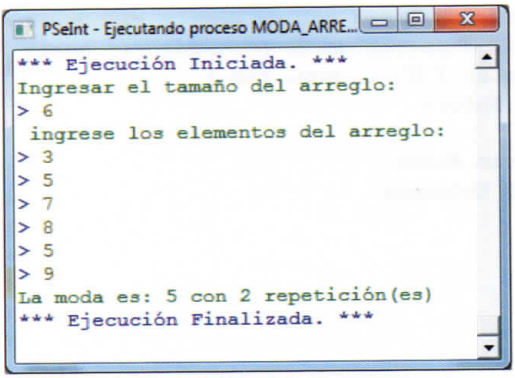


6.1.1.10 Realizar un algoritmo para calcular la moda de un arreglo de enteros ingresados por teclado.

Entrada: arreglo de enteros
Salida: dos números (moda y frecuencia)
Proceso: calcular la moda, es decir, el elemento que más veces se repite. Se utiliza un subproceso para contar un número en el arreglo.

```
1  // devuelve el número de veces que se repite el número 'num'
2  // en el arreglo 'A' a partir de la posición 'i' hasta 'tam'
3  SubProceso f <- contar ( A, i, num, tam )
4      definir f Como Entero
5      f<-0
6      Mientras i <= tam Hacer
7          Si num=A(i) Entonces
8              f<-f+1
9          Fin Si
10         i<-i+1
11     Fin Mientras
12 Fin SubProceso
```

```
14 // método principal
15 Proceso moda_arreglo
16     definir n, A, i, f, frec, num como entero
17     Escribir "Ingresar el tamaño del arreglo: "
18     Leer n
19     dimension A(n)
20     Escribir " ingrese los elementos del arreglo: "
21     Para i<-1 Hasta n Con Paso 1 Hacer
22         Leer A(i)
23     Fin Para
24     frec <-0
25     i<-1
26     Mientras i<= n Hacer
27         f<-contar(A,i,A(i),n)    // invoca al subproceso
28         Si f > frec Entonces
29             frec = f
30             num = A(i)
31         Fin Si
32         i = i+1
33     Fin Mientras
34     Escribir "La moda es: ", num, " con ", frec, " repetición(es)"
35 FinProceso
```



Nótese que al invocar al subproceso '*contar*', el arreglo '*A*' se pasa por referencia y las tres variables restantes por valor.

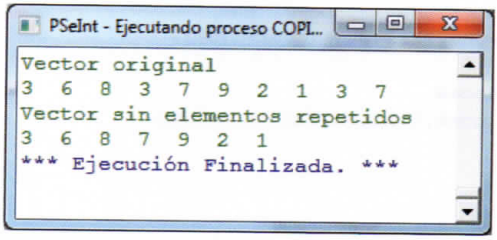
6.1.1.11 Realizar un algoritmo para copiar un arreglo de enteros ingresados por teclado a otro arreglo sin repetirse los elementos.

Entrada: arreglo de enteros
Salida: arreglo de enteros sin repetidos
Proceso: verifica la existencia de cada elemento del arreglo A en el B. Si no existe lo copia.

```
1   // imprime los elementos del arreglo en una sola linea
2   SubProceso mostrarArreglo ( A, n )
3       definir i Como Entero
4       Para i<-1 Hasta n Con Paso 1 Hacer
5           Escribir A(i), "  " Sin Saltar
6       Fin Para
7       Escribir "" // salto de linea
8   Fin SubProceso

10  // busca el número 'num' en el arreglo 'B' de tamaño 'j'
11  // si ya existe devuelve verdadero, caso contario falso
12  SubProceso ret <- existeNumArreglo ( num, B, j )
13      Definir ret Como Logico
14      Definir i Como Entero
15      ret<-Falso
16      Para i<-1 Hasta j Con Paso 1 Hacer
17          Si num=B(i) Entonces
18              ret<-Verdadero
19          Fin Si
20      Fin Para
21  Fin SubProceso
```

```
23  // método principal
24  Proceso copiar_arreglo
25      definir n, i, j, A, B Como Entero
26      Escribir "Ingrese el tamaño del vector:"
27      Leer n
28      Dimension  A(n), B(n)
29      j<-0
30      Escribir "Ingrese los elementos del vector:"
31      Para i<-1 Hasta n Con Paso 1 Hacer
32          Leer A(i)
33          // A continuación invoca al subproceso.
34          Si NO existeNumArreglo(A(i), B, j) Entonces
35              j<-j+1
36              B(j)<-A(i)
37          Fin Si
38      Fin Para
39      Limpiar Pantalla
40      Escribir "Vector original"
41      mostrarArreglo(A,n)      // invoca al subproceso
42      Escribir "Vector sin repetidos"
43      mostrarArreglo(B,j)      // invoca al subproceso
44  FinProceso
```

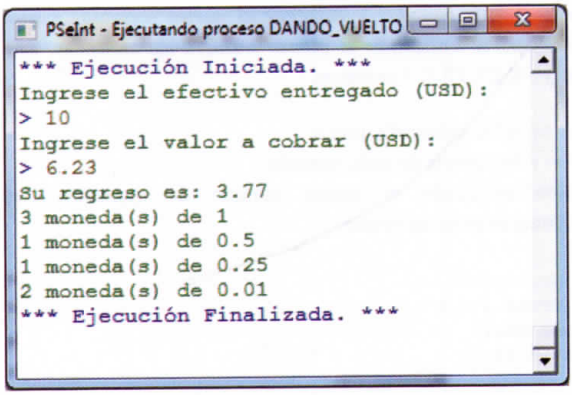


6.1.1.12 Realizar un algoritmo para calcular y entregar el vuelto (cambio) a una persona por concepto de transporte utilizando el menor número de monedas. Las denominaciones son 1 dólar y 50, 25, 10,5, 1 centavos.

Entrada: dos números reales: cantidad entregada, precio
Salida: cambio, denominación y frecuencia de cada moneda
Proceso: calcula el cambio utilizando el menor número de monedas. Las denominaciones se almacenan en un vector.

```
1  // redondea a dos decimales
2  SubProceso ret <- round ( n )
3      definir ret Como Real
4      ret <- REDON(n*100)/100
5  Fin SubProceso
```

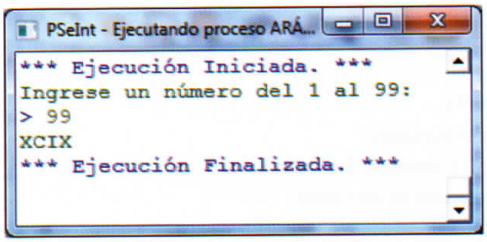
```
7   // método principal
8   Proceso dando_vuelto
9       definir valores,pasaje, vuelto, T como real
10      definir i, cont como entero
11      Dimension T(6)  // seis denominaciones de monedas
12      T(1)<-(1)
13      T(2)<-(0.50)
14      T(3)<-(0.25)
15      T(4)<-(0.10)
16      T(5)<-(0.05)
17      T(6)<-(0.01)
18      Escribir 'Ingrese el efectivo entregado (USD):'
19      Leer valores
20      Escribir 'Ingrese el valor a cobrar (USD):'
21      Leer pasaje
22      vuelto<-valores-pasaje
23      Escribir 'Su regreso es: ',vuelto
24      i<-1
25      Mientras vuelto> 0 Hacer
26          cont<-0
27          Si vuelto>=T(i) entonces
28              cont<-vuelto/T(i)
29              // A continuación invoca al subproceso
30              vuelto<-round(vuelto - (cont*T(i)))
31          FinSi
32          Si cont>0 Entonces
33              Escribir cont," moneda(s) de ',T(i)
34          FinSi
35          i<-i+1
36      FinMientras
37  FinProceso
```



6.1.1.13 Realizar un algoritmo que lea por teclado un número del 1 al 99 y convierta a romano.

Entrada: un número entero (entre 1 y 99)
Salida: cadena de caracteres (número en romano)
Proceso: convertir el número arábigo a romano. Los símbolos a usar son: I=1, V=5, X=10, L=50, C=100. Se almacenan en un vector.

```
1   Proceso arábigos_romanos
2       definir U, D Como Caracter
3       definir uni, dec, N como entero
4       Dimension U(10)
5       Dimension D(10)
6       // unidades
7       U(1)<-""     // cadena vacía
8       U(2)<-"I"
9       U(3)<-"II"
10      U(4)<-"III"
11      U(5)<-"IV"
12      U(6)<-"V"
13      U(7)<-"VI"
14      U(8)<-"VII"
15      U(9)<-"VIII"
16      U(10)<-"IX"
17      // decenas
18      D(1)<-""     // cadena vacía
19      D(2)<-"X"
20      D(3)<-"XX"
21      D(4)<-"XXX"
22      D(5)<-"XL"
23      D(6)<-"L"
24      D(7)<-"LX"
25      D(8)<-"LXX"
26      D(9)<-"LXXX"
27      D(10)<-"XC"
28      Escribir "Ingrese un número del 1 al 99: "
29      Leer N
30      uni<-(N mod 10)
31      dec<-(trunc(N/10) mod 10)
32      si N>=10 entonces
33      |   Escribir Concatenar(D(dec+1), U(uni+1))
34      sino
35      |   Escribir U(N+1)
36      FinSi
37  FinProceso
```



6.1.2 Ejercicios propuestos con Vectores

6.1.2.1 Realizar un algoritmo para invertir los elementos de un vector de enteros ingresados por teclado. Utilice el mismo vector.

6.1.2.2 Realizar un algoritmo para calcular la suma de los números primos de un vector de enteros ingresados por teclado.

6.1.2.3 Realizar un algoritmo para calcular la desviación estándar (σ) de un conjunto de datos (vector) ingresado por teclado.

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N} x_i - \bar{x})^2}$$

6.1.2.4 Realizar un algoritmo para ordenar un arreglo de enteros utilizando los métodos de selección e Inserción. ¿Cuál es mejor?

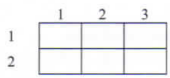
6.1.2.5 Realizar un algoritmo para la búsqueda binaria de un elemento en un arreglo de enteros ingresados por teclado.

6.1.2.6 Realizar un algoritmo que lea por teclado un número del 1 al 999 y convierta a romano.

6.2 Arreglos Bidimensionales (matrices)

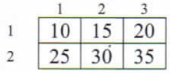
Es un conjunto finito y homogéneo de datos, es decir, del mismo tipo de dato, y se almacenan bajo un mismo nombre, y se diferencian por la posición que tiene cada elemento dentro del arreglo en forma rectangular o cuadrada (López, 2011). La matriz de orden MxN significa que tienen M filas y N columnas.

A continuación se muestra gráficamente una matriz de 2x3:



Para declarar una matriz se utiliza las palabras reservadas '*definir*' y '*dimensión*'. Por ejemplo para definir una matriz M_{2x3} de enteros y almacenar seis elementos utilizamos la siguiente sintaxis:

```
definir M Como Entero
dimension M(2,3)    // # de filas y columnas
M(1,1)<-10
M(1,2)<-15
M(1,3)<-20
M(2,1)<-25
M(2,2)<-30
M(2,3)<-35
```



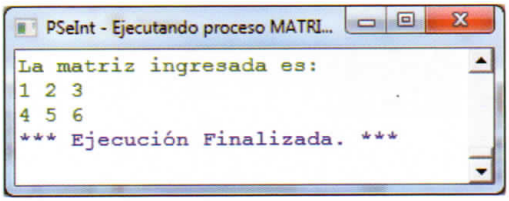
6.2.1 Ejercicios con Matrices

6.2.1.1 Realizar un algoritmo para crear una matriz de n elementos ingresados por teclado e imprimir.

Entrada: tamaño y elementos de la matriz
Salida: matriz
Proceso: crear la matriz y almacenar los elementos ingresados utilizando dos bucles anidados.

```
1   // imprime la matriz 'M' en forma rectangular o cuadrada
2   SubProceso mostrarMatriz(M, f, c)
3       Definir i,j como entero
4       Para i<-1 hasta f hacer
5           Para j<-1 hasta c hacer
6               Escribir M(i,j) , " " Sin Saltar
7           FinPara
8           Escribir "" // salto de línea
9       FinPara
10  FinSubProceso
```

```
12  // método principal
13  Proceso matrices
14      definir M como entero
15      definir f, c, i, j Como Entero
16      Escribir "Ingrese el tamaño (filas y columnas) de la matriz: "
17      Leer f,c
18      dimension M(f,c)
19      Para i<-1 Hasta f Con Paso 1 Hacer
20          Para j<-1 Hasta c Con Paso 1 Hacer
21              Escribir "Ingrese el elemento de fila ", i, " columna ",j
22              Leer M(i,j)
23          Fin Para
24      Fin Para
25      Limpiar Pantalla
26      Escribir "La matriz ingresada es:"
27      mostrarMatriz(M,f,c)     // invoca al subproceso
28  FinProceso
```



Al invocar al subproceso '*mostrarMatriz*' la matriz '*M*' se pasa por referencia y las dos variables restantes (*f* y *c*) por valor.

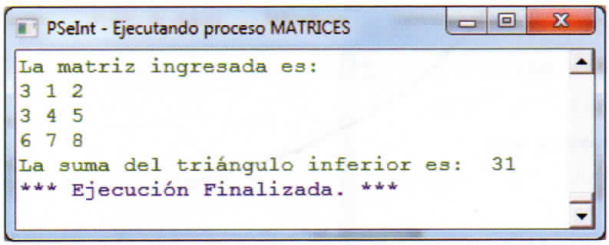
6.2.1.2 Realizar un algoritmo para sumar los elementos formados por el triángulo inferior de una matriz cuadrada de enteros ingresados por teclado (incluido la diagonal principal).

Entrada: tamaño y elementos de la matriz cuadrada
Salida: número (suma)
Proceso: Sumar los elementos del triángulo inferior. La matriz cuadrada tiene igual número de filas y columnas. El elemento está en el triángulo inferior cuando la fila es mayor o igual a la columna.

$$\mathbf{M}=\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}$$

```
...
...      // Aquí va el subproceso mostrarMatriz()
...
// ingresa los elementos de la matriz 'M' por teclado
// la matriz 'M' es pasada por referencia
SubProceso ingresarMatriz (M, f, c)
    definir i, j Como Entero
    Para i<-1 Hasta f Con Paso 1 Hacer
        Para j<-1 Hasta c Con Paso 1 Hacer
            Escribir "Ingrese elemento de fila ", i, " columna ",j
            Leer M(i,j)
        Fin Para
    Fin Para
Fin SubProceso

// método principal
Proceso matrices
    definir M como entero
    definir f, i, j, sum Como Entero
    Escribir "Ingrese el # de filas de la matriz cuadrada: "
    Leer f
    dimension M(f,f)     // matriz cuadrada
    ingresarMatriz(M, f, f) // invoca al subproceso
    sum <-0
    Para i<-1 Hasta f Con Paso 1 Hacer
        Para j<-1 Hasta f Con Paso 1 Hacer
            Si i >= j Entonces  // ¿está en el triángulo inferior?
                sum <- sum + M(i,j)
            Fin Si
        Fin Para
    Fin Para
    Limpiar Pantalla
    Escribir "La matriz ingresada es:"
    mostrarMatriz(M,f,f)     // invoca al subproceso
    Escribir "La suma del triángulo inferior es:  ", sum
FinProceso
```



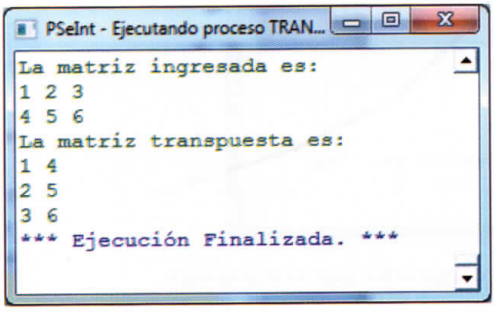
El subproceso '*mostrarMatriz*' es el mismo de los ejercicios anteriores.

6.2.1.3 Realizar un algoritmo para calcular la transpuesta de una matriz de enteros ingresados por teclado.

Entrada: tamaño y elementos de la matriz
Salida: matriz transpuesta
Proceso: Calcular la matriz transpuesta intercambiando las posiciones de los elementos:

$$(M^t)\,ji = Mij,\ 1 \leq i \leq m,\ 1 \leq j \leq n$$

```
...
...    // Aquí van los subprocesos mostrarMatriz() e ingresarMatriz()
...
// método principal
Proceso transpuesta
    definir M, Mt como entero
    definir f, c, i, j Como Entero
    Escribir "Ingrese el tamaño (filas y columnas) de la matriz: "
    Leer f, c
    dimension M(f,c)
    dimension Mt(c,f)
    ingresarMatriz(M, f, c) // invoca al subproceso
    Para i<-1 Hasta f Con Paso 1 Hacer
        Para j<-1 Hasta c Con Paso 1 Hacer
            Mt(j,i) <- M(i,j)    // transpone los elementos
        Fin Para
    Fin Para
    Limpiar Pantalla
    Escribir "La matriz ingresada es:"
    mostrarMatriz(M, f, c)       // invoca al subproceso
    Escribir "La matriz transpuesta es:"
    mostrarMatriz(Mt, c, f)      // invoca al subproceso
FinProceso
```

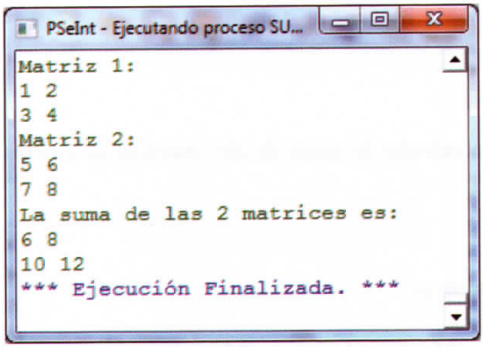


Los subprocesos "*mostrarMatriz*" e '*ingresarMatriz*' son los mismos utilizados en los ejercicios anteriores.

6.2.1.4 Realizar un algoritmo para calcular la suma de dos matrices de enteros ingresados por teclado.

Entrada: tamaño y elementos de las matrices
Salida: matriz (suma)
Proceso: Calcular la suma de matrices $(M)\ ij = (M_1)\ ij + (M_2)ij\ 1 \leq i \leq m,\ 1 \leq j \leq n$

```
...
...      // Aquí van los subprocesos mostrarMatriz() e ingresarMatriz()
...
// método principal
Proceso suma_matrices
    definir M, M1, M2 como entero
    definir f, c, i, j Como Entero
    Escribir "Ingrese el tamaño (filas y columnas) de la matriz: "
    Leer f, c
    dimension M(f,c)
    dimension M1(f,c)
    dimension M2(f,c)
    Escribir "Ingrese la primera matriz:"
    ingresarMatriz(M1, f, c)     // invoca al subproceso
    Escribir "Ingrese la segunda matriz:"
    ingresarMatriz(M2, f, c)     // invoca al subproceso
    Para i<-1 Hasta f Con Paso 1 Hacer
        Para j<-1 Hasta c Con Paso 1 Hacer
            M(i,j) <- M1(i,j) + M2(i,j)
        Fin Para
    Fin Para
    Borrar Pantalla
    Escribir "Matriz 1:"
    mostrarMatriz(M1,f,c)    // invoca al subproceso
    Escribir "Matriz 2:"
    mostrarMatriz(M2,f,c)    // invoca al subproceso
    Escribir "La suma de las 2 matrices es:"
    mostrarMatriz(M,f,c)     // invoca al subproceso
FinProceso
```



Los subprocesos "*mostrarMatriz*" e '*ingresarMatriz*' son los mismos utilizados en los ejercicios anteriores.

6.2.1.5 Realizar un algoritmo para calcular la multiplicación de dos matrices de enteros ingresados por teclado.

Entrada: tamaño y elementos de las matrices
Salida: matriz (multiplicación)
Proceso: Calcular la multiplicación de matrices. Dadas dos matrices A y B.

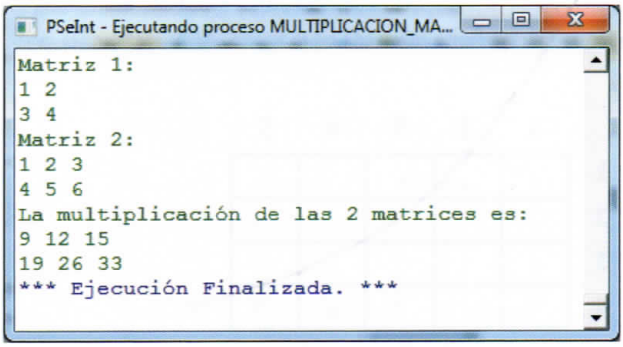
$$A = (A_{ij})_{mxn};\ B = (B_{ij})_{nxp}$$

$$C = AB = (c_{ij})_{mxp}$$

$$c_{ij} = \sum_{r=1}^{n} a_{ir}\, b_{rj}$$

```
...
...     // Aquí van los subprocesos mostrarMatriz() e ingresarMatriz()
...
```

```
22  // método principal
23  Proceso multiplicacion_matrices
24      definir M, M1, M2 como entero
25      definir f1, c1, f2, c2, i, j, k Como Entero
26      Escribir "Ingrese el tamaño (filas y columnas) de la matriz 1: "
27      Leer f1, c1
28      Escribir "Ingrese el tamaño (filas y columnas) de la matriz 2: "
29      Leer f2, c2
30      Si c1 <> f2 Entonces
31          Escribir "No se puede multiplicar. Corrija los tamaños."
32      Sino
33          dimension M(f1,c2)
34          dimension M1(f1,c1)
35          dimension M2(f2,c2)
36          Escribir "Ingrese la primera matriz:"
37          ingresarMatriz(M1, f1, c1)  // invoca al subproceso
38          Escribir "Ingrese la segunda matriz:"
39          ingresarMatriz(M2, f2, c2)  // invoca al subproceso
40          Para i<-1 Hasta f1 Con Paso 1 Hacer
41              Para j<-1 Hasta c2 Con Paso 1 Hacer
42                  M(i,j) <- 0
43                  Para k<-1 hasta c1 hacer
44                      M(i,j) <- M(i,j) + (M1(i,k) * M2(k,j))
45                  Fin Para
46              Fin Para
47          Fin Para
48          Borrar Pantalla
49          Escribir "Matriz 1:"
50          mostrarMatriz(M1,f1,c1) // invoca al subproceso
51          Escribir "Matriz 2:"
52          mostrarMatriz(M2,f2,c2) // invoca al subproceso
53          Escribir "La multiplicación de las 2 matrices es:"
54          mostrarMatriz(M,f1,c2)  // invoca al subproceso
55      FinSi
56  FinProceso
```



Los subprocesos '*ingresarMatriz*' y '*mostrarMatriz*' son los mismos de antes. Nótese que para realizar la multiplicación de matrices se requiere de tres bucles anidados.

6.2.2 Ejercicios propuestos con Matrices

6.2.2.1 Realizar un algoritmo para determinar si una matriz ingresada por teclado cumple la propiedad de simetría. Una matriz es simétrica cuando es una matriz cuadrada, y es igual a su transpuesta.

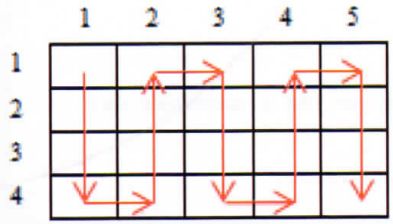
6.2.2.2 Realizar un algoritmo para determinar si una matriz ingresada por teclado cumple la propiedad de antisimetría. Una matriz antisimétrica es una matriz cuadrada cuya transpuesta es igual a su negativa, es decir, $M^T = -M$

6.2.2.3 Realizar un algoritmo para determinar si la matriz X es mayor que la matriz W. Las dos matrices son de orden *mxn* e ingresadas por teclado. Para que X sea mayor a W, cada elemento X_{ij} debe ser mayor o igual a W_{ij} y además debe existir, al menos, un elemento en X que sea mayor al correspondiente en W.

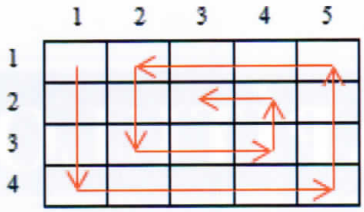
6.2.2.4 Realizar un algoritmo para calcular la suma de cada diagonal de una matriz cuadrada ingresada por teclado (véase la figura) y almacenar en un vector.

$$\mathbf{M} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}$$

6.2.2.5 Realizar un algoritmo para ingresar el tamaño (*M x N*) y los elementos de una matriz por teclado siguiendo la trayectoria mostrada en la figura. Al final imprima la matriz.



6.2.2.6 Realizar un algoritmo para ingresar el tamaño (*M x N*) y los elementos de una matriz por teclado siguiendo la trayectoria mostrada en la figura. Al final imprima la matriz.



6.2.2.7 Realizar un algoritmo para imprimir el *cuadrado mágico* de orden impar m *x m* ingresado por teclado, donde $3 \leq m \leq 11$.

CAPÍTULO VII

RECURSIVIDAD

$$n! = \begin{cases} si \quad n = 0 \rightarrow 1 \\ si \quad n \geq 1 \rightarrow n(n-1)! \end{cases}$$

En este capítulo usted encontrará:

- *Definición de recursividad*
- *Ejercicios de recursividad*
- *Ejercicios propuestos*

7. Recursividad

7.1 Definición

Según Joyanes (2008), un método es recursivo cuando se invoca o llama a sí mismo. La recursión se puede utilizar como una alternativa a la iteración. Una solución recursiva es normalmente menos eficiente en términos de tiempo y memoria que una solución iterativa debido a las operaciones auxiliares que conllevan las llamadas suplementarias a las funciones.

Aunque la recursividad es un concepto algo "complejo", algunos problemas de computación se pueden resolver de una manera natural y sencilla usando recursividad en lugar de las estructuras de repetición (bucles). Algunos ejemplos son: el factorial, el Fibonacci, torres de Hanoi, etc.

Cuando un método se llama a sí mismo, se asigna espacio en la *pila* (stack) para las nuevas variables que se crean por cada llamada. A pesar de tener los nombres iguales se tratan de variables diferentes.

Una pila es una estructura de datos lineal en donde el modo de acceso a sus elementos para almacenar y recuperar los datos es de tipo LIFO (*Last In First Out*, es decir, último en entrar, primero en salir). Gráficamente se puede ver como sigue:

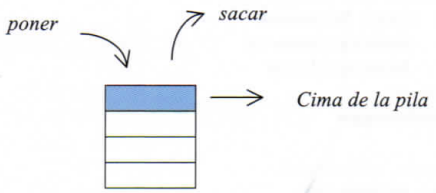


Figura 7.1 Estructura de la pila (stack)

La pila cuenta con dos operaciones básicas: *poner*, que coloca un objeto en la pila, y *sacar*, que retira el último elemento apilado. En cada momento sólo se tiene acceso a la parte superior de la pila (cima).

Al regresar de una llamada recursiva, se recuperan de la pila las variables locales y los parámetros respectivos y la ejecución se reanuda en el punto donde se efectuó la llamada a dicho método.

Los métodos recursivos deben cumplir dos condiciones:

a) Debe haber un caso base o inicial (salida no recursiva).
b) Cada llamada al mismo método debe ser cada vez más simple, hasta que se llegue al caso base.

Existen dos tipos de recursividad:

- **Directa**.- Cuando un subproceso se invoca a sí mismo.
- **Indirecta**.- Cuando un subproceso invoca a otro que a su vez invoca al primero (mutuamente).

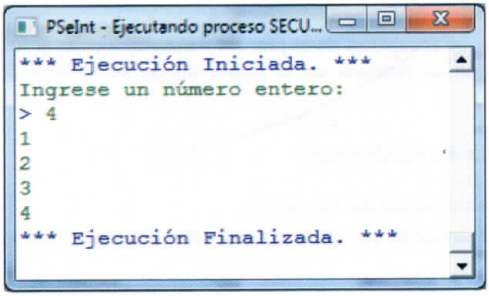
Una desventaja de la recursividad es que puede ocupar demasiado espacio en memoria (pila) y la corrida de escritorio (depuración) puede resultar complicada y tediosa.

7.2 Ejercicios de recursividad

7.2.1 Realizar un algoritmo recursivo para imprimir en secuencia desde el 1 hasta un número ingresado por teclado.

Entrada: un número entero
Salida: lista de números en secuencia
Proceso: Calcular la secuencia del 1 hasta n utilizando recursividad

```
1   // subproceso recursivo
2   SubProceso mostrar ( num )
3       Si num>0 Entonces
4       |   mostrar(num-1)  // se llama a sí mismo
5       |   Escribir num
6       Fin Si
7   Fin SubProceso
8
9   // método principal
10  Proceso secuencia
11      definir n Como Entero
12      Escribir "Ingrese un número entero:"
13      Leer n
14      mostrar(n)  // llamada al subproceso
15  FinProceso
```



Nótese que el *caso base* del subproceso recursivo '*mostrar*' se da cuando el parámetro '*num*' es igual a cero y no imprime nada.

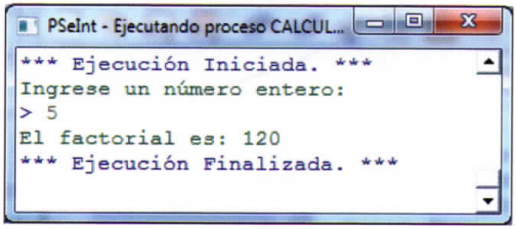
Si se realiza la corrida de escritorio se puede observar que la pila almacena los valores de la siguiente manera:

num	0
num	1
num	2
num	3
num	4

7.2.2 Realizar un algoritmo recursivo para calcular el factorial de un número ingresado por teclado.

Entrada: un número entero
Salida: número (factorial)
Proceso: Calcular el factorial utilizando la función: $n! \begin{cases} si\ n = 0 \rightarrow 1 \\ si\ n \geq 1 \rightarrow n(n-1)! \end{cases}$

```
1   // subproceso recursivo
2   SubProceso ret <- factorial ( num )
3       definir ret Como Entero
4       Si num>0 Entonces
5       |   ret <- num * factorial(num-1)
6       Sino
7       |   ret<-1  // caso base
8       Fin Si
9   Fin SubProceso
10
11  // método principal
12  Proceso calcular_factorial
13      definir n Como Entero
14      Escribir "Ingrese un número entero:"
15      Leer n
16      // A continuación se invoca al subproceso
17      Escribir "El factorial es: ", factorial(n)
18  FinProceso
```



Si se realiza la prueba de escritorio se puede observar que la pila almacena los valores de la siguiente manera:

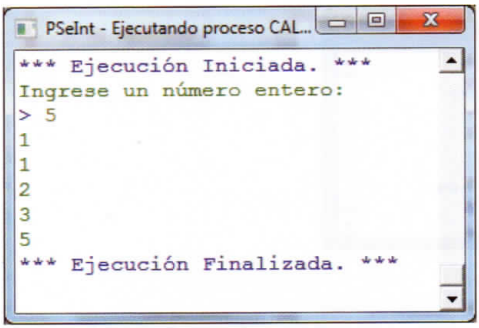
num	0	ret	1					
num	1	ret		1				
num	2	ret			2			
num	3	ret				6		
num	4	ret					24	
num	5	ret						120

7.2.3 Realizar un algoritmo recursivo para mostrar los n números de la serie Fibonacci ingresado por teclado.

Entrada: un número entero
Salida: lista de números (serie fibonacci)
Proceso: Calcular la serie fibonacci utilizando recursividad:

$$F(n) = F(n - 1) + F(n-2)$$

```
1   // subproceso recursivo
2   SubProceso ret <- fibonacci ( num )
3       definir ret Como Entero
4       Si num<2 Entonces
5           ret <- num  // caso base
6       Sino
7           ret<- fibonacci(num-1)+fibonacci(num-2)
8       Fin Si
9   Fin SubProceso
10
11  // método principal
12  Proceso calcular_fibonacci
13      definir n, i Como Entero
14      Escribir "Ingrese un número entero:"
15      Leer n
16      Para i<-1 Hasta n Con Paso 1 Hacer
17          Escribir fibonacci(i)   // invoca al subproceso
18      Fin Para
19  FinProceso
```

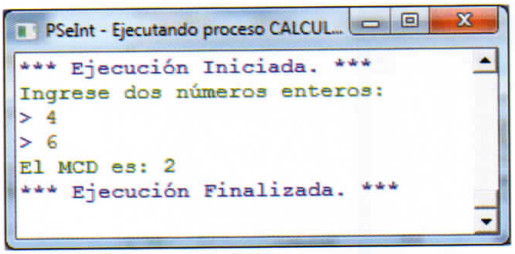


Nótese que realizar la prueba de escritorio se vuelve complejo.

7.2.4 Realizar un algoritmo recursivo para calcular el máximo común divisor de dos números ingresados por teclado.

Entrada: dos números enteros
Salida: número (MCD)
Proceso: Calcular el MCD utilizando recursividad y el operador MOD en cada llamada al método recursivo.

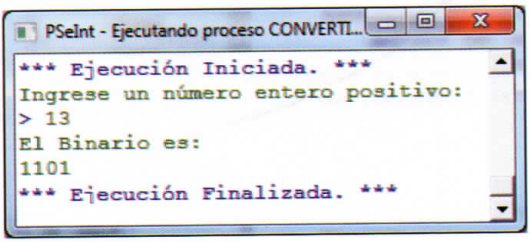
```
1   // subproceso recursivo
2   SubProceso ret <- MCD(a, b)
3       definir ret Como Entero
4       Si b=0 Entonces
5           ret <- a     // caso base
6       Sino
7           ret<-MCD(b, a mod b)
8       Fin Si
9   Fin SubProceso
10
11  // método principal
12  Proceso calcular_MCD
13      definir n1, n2 Como Entero
14      Escribir "Ingrese dos números enteros:"
15      Leer n1, n2
16      // A continuación se invoca al subproceso
17      Escribir "El MCD es: ", MCD(n1, n2)
18  FinProceso
```



7.2.5 Realizar un algoritmo recursivo para convertir un número decimal ingresado por teclado a binario.

Entrada: un número entero > 0
Salida: número binario (cadena de caracteres)
Proceso: Convertir de decimal a binario con divisiones sucesivas para 2 y usando recursividad.

```
1   // subproceso recursivo
2   SubProceso binario(num)
3       definir ret como caracter
4       Si num>0 Entonces
5           binario(trunc(num/2))
6           Escribir (num mod 2) Sin Saltar
7       Fin Si
8   Fin SubProceso
9
10  // método principal
11  Proceso convertir_binario
12      definir n Como Entero
13      Escribir "Ingrese un número entero positivo:"
14      Leer n
15      Escribir "El Binario es: "
16      binario(n)   // invoca al subproceso
17  FinProceso
```



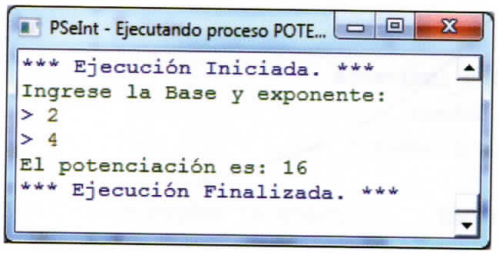
Nótese que el *caso base* del subproceso recursivo '*binario*' se da cuando el parámetro '*num*' es igual a cero y no retorna nada (cadena vacía).

7.2.6 Realizar un algoritmo recursivo para calcular la potencia de dos números ingresados por teclado.

Entrada: dos números (base y exponente)
Salida: número (potencia)
Proceso: calcular la potenciación utilizando recursividad. Se disminuye el exponente en uno en cada llamada recursiva.

$$\begin{cases} x^0 = 1 \\ si\ y > 0,\ x^y \equiv x.x^{y-1} \end{cases}$$

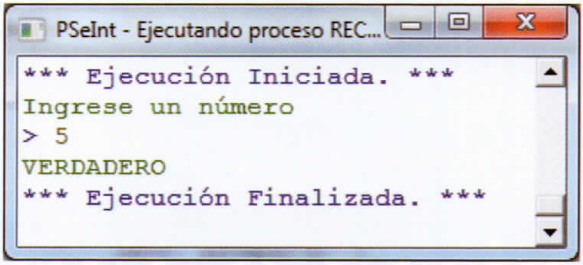
```
1  // subproceso recursivo
2  SubProceso ret <- potencia (base, expon)
3      definir ret Como Entero
4      Si expon = 0 Entonces
5          ret <- 1     // caso base
6      sino
7          ret <- base * potencia(base,expon-1)
8      FinSi
9  FinSubProceso
10
11 // método principal
12 Proceso potenciación
13     Definir base, exponente como Entero
14     Escribir "Ingrese la Base y exponente:"
15     Leer base, exponente
16     // A continuación se invoca al subproceso
17     Escribir "El potenciación es: ",Potencia(base,exponente)
18 FinProceso
```



7.2.7 Realizar un algoritmo recursivo indirecto (o mutuo) para determinar si un número ingresado por teclado es impar.

Entrada: un número
Salida: verdadero/falso
Proceso: determinar usando la recursividad indirecta.

```
1  // subproceso recursivo indirecto
2  SubProceso ret <- esPar ( n )
3      definir ret Como Logico
4      Si n=0 Entonces
5          ret<-Verdadero       // caso base
6      Sino
7          ret<- esImpar(n-1)  // invocación mutua (indirecto)
8      Fin Si
9  Fin SubProceso
10
11 // subproceso recursivo indirecto
12 SubProceso ret <- esImpar ( n )
13     definir ret Como Logico
14     Si n=0 Entonces
15         ret<- Falso      // caso base
16     Sino
17         ret<-esPar(n-1) // invocación mutua (indirecto)
18     Fin Si
19 Fin SubProceso
20
21 // método principal
22 Proceso impar_recursión_indirecta
23     Definir num Como Entero
24     Escribir "Ingrese un número"
25     Leer num
26     Escribir esImpar(num)    // invoca al subproceso
27 FinProceso
```



7.3 Ejercicios propuestos

7.3.1 Realizar un algoritmo recursivo para calcular la longitud (tamaño) de una cadena de caracteres.

7.3.2 Realizar un algoritmo recursivo para mostrar verdadero/falso dependiendo si una palabra ingresada por teclado es palíndromo.

7.3.3 Realizar un algoritmo recursivo para ordenar un arreglo (vector) de enteros.

7.3.4 Realizar un algoritmo recursivo para convertir un número arábigo ingresado por teclado (del 1 al 99) a romano.

7.3.5 Realizar un algoritmo recursivo para calcular el número de movimientos necesarios para resolver las Torres de Hanoi según el número de discos ingresado por teclado.

CAPÍTULO VIII

CONVERSIÓN DEL ALGORITMO A CÓDIGO FUENTE

```
#include<stdio.h>
using namespace std;

int main() {
    int a;
    int b;
    int c;
    printf("Ingrese dos numeros enteros:\n");
    scanf("%i",&a);
    scanf("%i",&b);
    c=a+b;
    printf("La suma es:%i\n",c);
    return 0;
}
```

En este capítulo usted encontrará:

- *Exportación a Java*
- *Exportación a C*
- *Exportación a C++*
- *Exportación a PHP*
- *Exportación a VB.net*
- *Exportación a Pascal*
- *Exportación a Python*
- *Exportación a JavaScript*
- *Exportación a Matlab*
- *Exportación a C#*
- *Ejercicios propuestos*

8 Conversión del algoritmo al código fuente

Para convertir el algoritmo en código fuente (programa) utilizamos la opción de PSeInt: '*Archivo*'→ '*exportar*' y elegimos el lenguaje de programación.

A continuación mostramos el algoritmo para sumar dos números leídos por teclado y su correspondiente código fuente en cada lenguaje de programación disponible actualmente en PSeInt:

Algoritmo:

```
1  Proceso números
2      Definir a,b,c Como Entero
3      Escribir 'Ingrese dos números enteros:'
4      Leer a,b
5      c<-a+b
6      Escribir 'La suma es: ',c
7  FinProceso
```

8.1 Exportación a Java

```
import java.io.*;
public class números {
    public static void main(String args[]) throws IOException {
        BufferedReader bufEntrada=new BufferedReader
                    (new InputStreamReader(System.in));
        int a;
        int b;
        int c;
        System.out.println("Ingrese dos números enteros:");
        a=Integer.parseInt(bufEntrada.readLine());
        b=Integer.parseInt(bufEntrada.readLine());
        c=a+b;
        System.out.println("La suma es:"+c);
    }
}
```

8.2 Exportación a C

```c
#include<stdio.h>
using namespace std;
int main() {
    int a;
    int b;
    int c;
    printf("Ingrese dos números enteros:\n");
    scanf("%i",&a);
    scanf("%i",&b);
    c=a+b;
    printf("La suma es:%i\n",c);
    return 0;
}
```

8.3 Exportación a C++

```cpp
#include<iostream>
using namespace std;
int main() {
    int a;
    int b;
    int c;
    cout<<"Ingrese dos números enteros:"<<endl;
    cin>>a>>b;
    c=a+b;
    cout<<"La suma es:"<<c<<endl;
    return 0;
}
```

8.4 Exportación a PHP

```php
<?php
    $stdin = fopen('php://stdin','r');
    settype($a,'integer');
    settype($b,'integer');
    settype($c,'integer');
    echo 'Ingrese dos números enteros:',PHP_EOL;
    fscanf($stdin,"%d",$a);
    fscanf($stdin,"%d",$b);
    $c = $a+$b;
    echo 'La suma es:',$c,PHP_EOL;
?>
```

8.5 Exportación a VB.net

```
Module NÚMEROS
    Sub Main()
        Dim a As Integer
        Dim b As Integer
        Dim c As Integer
        Console.WriteLine("Ingrese dos números enteros:")
        a = Integer.Parse(Console.ReadLine())
        b = Integer.Parse(Console.ReadLine())
        c = a+b
        Console.WriteLine("La suma es:",c)
    End Sub
End Module
```

8.6 Exportación a Pascal

```
Program números;
Var
    a: Integer;
    b: Integer;
    c: Integer;
Begin
    WriteLn('Ingrese dos números enteros:');
    ReadLn(a,b);
    c := a+b;
    WriteLn('La suma es:',c);
End.
```

8.7 Exportación a Python

```
if __name__ == '__main__':
    a = int()
    b = int()
    c = int()
    print("Ingrese dos números enteros:")
    a = int(input())
    b = int(input())
    c = a+b
    print("La suma es:",c)
```

8.8 Exportación a JavaScript

```
function números() {
    var a=new Number();
    var b=new Number();
    var c=new Number();
    document.write("Ingrese dos números enteros:",'<BR/>');
    a=Number(prompt());
    b=Number(prompt());
    c=a+b;
    document.write("La suma es:",c,'<BR/>');
}
```

8.9 Exportación a Matlab

```
function suma_2_números()
    a=0;
    b=0;
    c=0;
    disp('Ingrese dos números enteros:');
    a=input('');
    b=input('');
    c=a+b;
    disp(['La suma es: ',num2str(c)]);
end
```

8.10 Exportación a C#

```
using System;
namespace PSeInt {
    class suma_2_números {

        static void Main(string[] args) {
            int a;
            int b;
            int c;
            Console.WriteLine("Ingrese dos números enteros:");
            a = int.Parse(Console.ReadLine());
            b = int.Parse(Console.ReadLine());
            c = a+b;
            Console.WriteLine("La suma es: "+c);
        }

    }

}
```

Glosario

Acumulador.- Es una variable cuyo valor va incrementando en cantidades variables dentro de un ciclo de repetición, ej. $x=x+n$

Algoritmo.- Es una secuencia de pasos necesarios para resolver algún problema.

Arreglo.- Es una serie de valores del mismo tipo de dato. Los elementos tienen el mismo nombre pero se distinguen por los índices.

Búsqueda binaria.- Es una búsqueda que empieza en el medio de una lista o serie ordenada y luego determina si debería continuar hacia arriba o hacia abajo para encontrar el valor buscado. Es más eficiente que la búsqueda secuencial.

Búsqueda secuencial.- Es una búsqueda a lo largo de una serie o lista de un extremo a otro para encontrar el valor buscado.

Cliente.- Es una aplicación o un equipo que consume un servicio remoto en otro equipo, conocido como servidor, a través de una red de telecomunicaciones.

Código fuente.- Son las declaraciones que un programador escribe en un lenguaje de programación.

Contador.- Es una variable cuyo valor incrementa de uno en uno dentro de un ciclo de programación, ej. $x=x+1$

Diagrama de flujo.- Es una representación gráfica de un algoritmo y utiliza principalmente entradas, procesos y salidas.

Encapsulamiento.- Es la acción de contener las instrucciones de una tarea en un módulo.

Entrada.- Introducción de datos en el algoritmo por medio del teclado.

Estructura.- Es una unidad básica de lógica de programación: secuencia, selección, repetición.

Estructura apilada.- Es el resultado de la unión de estructuras (secuencia, selección, repetición) por sus extremos.

Estructura anidada.- Una estructura (secuencia, selección, repetición) dentro de otra.

IDE (*Integrated Development Environment*).- Es una herramienta que integra todos los recursos que necesita un programador para codificar, compilar, depurar, ejecutar y documentar programas.

Lenguajes de Programación.- Se usan para escribir los programas. Algunos son C, C++, C#, java, PHP, VB.net, pascal, python, javaScript.

Lógica.- Instrucciones que se dan a la computadora en un secuencia específica, sin omitir ninguna ni agregar superfluas.

Módulo.- Unidad de un algoritmo o programa y tiene una tarea específica.

Notación de camello.- Convención para nombrar las variables. Empieza con una letra minúscula y cualquier palabra subsiguiente comienza con una mayúscula, ejemplo: lastName.

Palabras reservadas.- Es un conjunto limitado de palabras que se reservan en un lenguaje, por ejemplo: *mientras, según, repetir, para*, etc.

Pila.- Es una ubicación de memoria en la que la computadora sigue el rastro a la dirección de memoria correcta a la que deberá regresar después de ejecutar un método.

Precedencia.- Es la cualidad de una operación que determina el orden en el cual se evalúa.

Programa.- Conjunto de instrucciones para una computadora.

Programación.- Es el arte de desarrollar y escribir programas.

Programación estructurada.- Es una técnica para el desarrollo de programas que sean claros, haciendo uso de tres estructuras de control: la secuencia, selección y repetición.

Programación modular.- Esta técnica divide un programa en varios módulos (subprogramas) más simples con la finalidad de hacerlo más comprensible, manejable y reutilizable, así como solucionar problemas más grandes y complejos de una mejor forma.

Programación orientada a objetos (POO).- Es un modelo de programación que se enfoca en los objetos o cosas y describe sus características (atributos) y comportamientos (métodos).

Prueba de escritorio.- Es el proceso de ejecutar paso a paso el algoritmo para evaluar y analizar la evolución de los datos y el funcionamiento del mismo.

Pseudocódigo.- Es una representación en inglés o español de los pasos lógicos que se requieren para resolver un problema.

Salida.- Muestra los resultados del algoritmo en un monitor o impresora.

Semántica.- Se refiere al significado o sentido que se da a la combinación de símbolos (palabras, expresiones) del lenguaje de programación.

Servidor.- Es un equipo que forma parte de una red de computadoras y provee servicios a otros equipos denominados clientes.

Servidor de Aplicaciones.- Ejecuta las aplicaciones empresariales de la organización y gestiona las funciones de la lógica de negocio y acceso a los datos de la aplicación.

Servidor de Base de datos.- Provee los servicios de base de datos (consulta, inserción, eliminación y actualización) a otros programas u otras computadoras.

Sintaxis.- Son las reglas gramaticales que gobierna los elementos (palabras, números y puntuación) del lenguaje de programación y sus combinaciones.

Tipo de datos.- Describe que valor se puede asignar a una variable: carácter, entero, real, booleano.

Valor centinela.- Valor que representa un punto de entrada o salida.

Variable.- Es una ubicación de memoria nombrada, cuyo valor puede variar.

Bibliografía

Farrell, Y. (2013). *Introducción a la programación lógica y diseño*. Séptima edición. México: Cengage Learning.

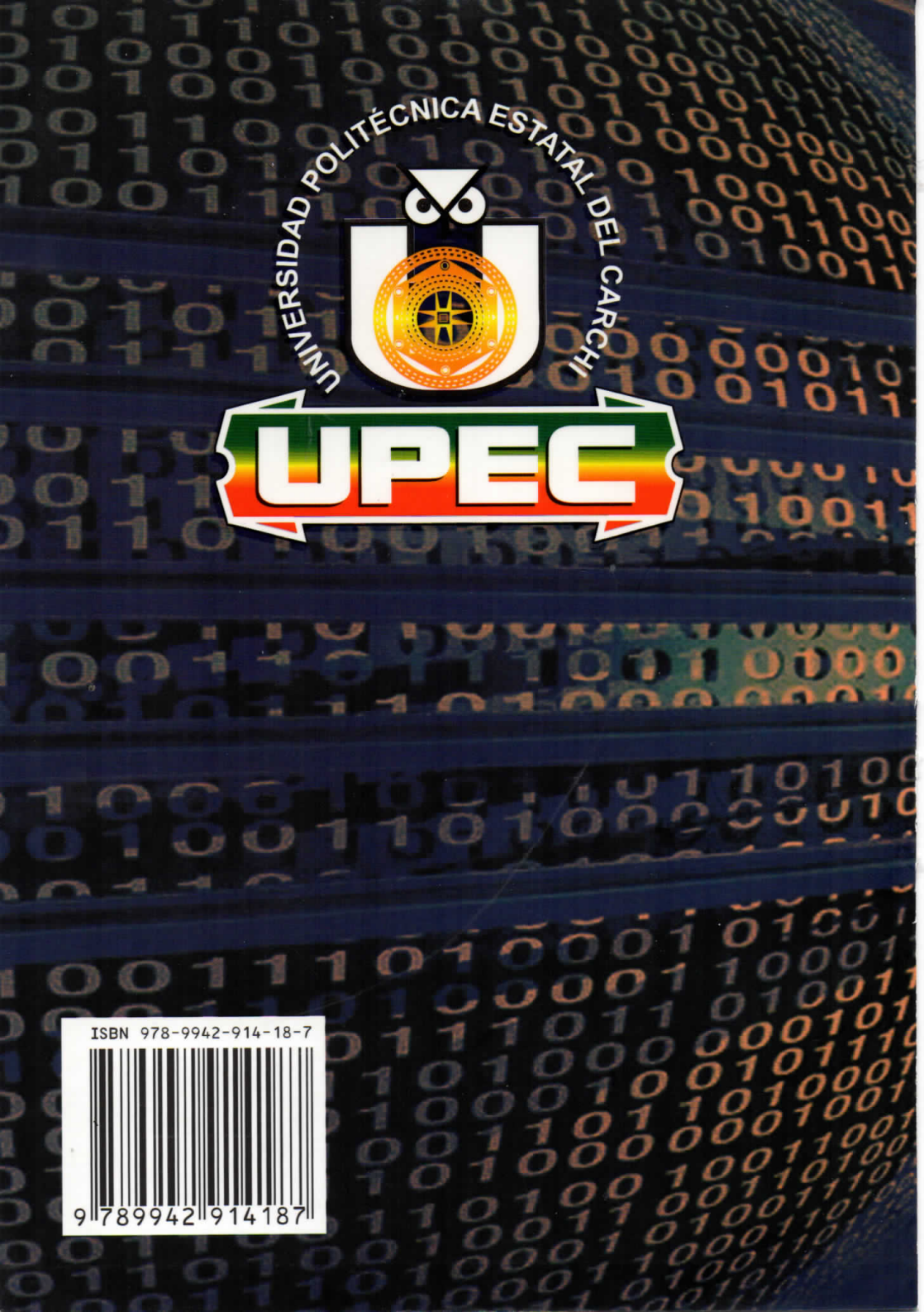
Joyanes, L. (2008). *Fundamentos de programación. Algoritmos, estructuras de datos y objetos*. Cuarta edición. España: McGraw-Hill.

López J. (2009). *Algoritmos y programación*. Colombia: Eduteka. Segunda edición. Disponible en http://www.eduteka.org/pdfdir/AlgoritmosProgramacion.pdf

López, L. (2011). *Programación estructurada y orientada a objetos un enfoque algorítmico*. Tercera edición. México: Alfaomega

Novara, P. (2014). *Sitio Web oficial de PSeInt*. Consultado el 21-09-2014. Disponible en http://pseint.sourceforge.net/

Sznajdleder, P. (2012). *Algoritmos a fondo con implementaciones en C y java*. Argentina: Algaomega.



ISBN 978-9942-914-18-7