

Programación Lógica I

INSTITUCIONALES	1
¿Cómo está organizado este texto?	4
Introducción	5
Esquema	6
Situación profesional 1: Algoritmia computacional elemental	7
SP1 / H1: Algoritmia y secuencia de comandos	8
REFERENCIAS 1	16
SP1 / Autoevaluación 1	17
SP1 / H2: Tipos de datos elementales. Constantes. Variables	20
REFERENCIAS 2	26
SP1 / Autoevaluación 2	31
SP1 / H3: Forma general de los pseudocódigos	34
SP1 / Autoevaluación 3	38
SP1 / Ejercicio resuelto	41
SP1 / Ejercicio por resolver	42
SP1 / Evaluación de paso	43
Situación profesional 2: Estructuras Alternativas	46
SP2 / H1: Tipos de estructuras alternativas	47
SP2 / Autoevaluación 1	64
SP2 / H2: Diagrama de flujo	67
SP2 / Autoevaluación 2	71
SP2 / H3: Regla de estructuras y control de flujos	73
SP2 / Autoevaluación 3	76
SP2 / Ejercicio resuelto	78
SP2 / Ejercicio por resolver	80
SP2 / Evaluación de paso	81
Situación profesional 3: Operadores	84
SP3 / H1: Clasificación de los operadores	85
REFERENCIAS 3	93
SP3 / Autoevaluación 1	97
SP3 / Ejercicio resuelto	100

<i>SP3 / Ejercicio por resolver</i>	102
<i>SP3 / Evaluación de paso</i>	103
Situación profesional 4: Estructuras Repetitivas y Recursivas	106
<i>SP4 / H1: Estructuras repetitivas</i>	107
<i>SP4 / Autoevaluación 1</i>	114
<i>SP4 / H2: Estructura de funciones o procedimientos recursivos</i>	117
<i>SP4 / Autoevaluación 2</i>	122
<i>SP4 / Ejercicio resuelto</i>	124
<i>SP4 / Ejercicio por resolver</i>	126
<i>SP4 / Evaluación de paso</i>	127
Situación profesional 5: Estructuras de Datos Homogéneas: Vectores	130
<i>SP5 / H1: Vectores</i>	131
<i>SP5 / Autoevaluación 1</i>	142
<i>SP5 / H2: Búsqueda secuencial y binaria</i>	144
<i>SP5 / Autoevaluación 2</i>	148
<i>SP5 / H3: Métodos de ordenamiento</i>	151
<i>SP5 / Autoevaluación 3</i>	165
<i>SP5 / Ejercicio resuelto</i>	167
<i>SP5 / Ejercicio por resolver</i>	170
<i>SP5 / Evaluación de paso</i>	171
Situación profesional 6: Estructuras de Datos Homogeneas: Matrices	173
<i>SP6 / H1: Estructura de dato homogénea bidimensional</i>	174
<i>SP6 / Autoevaluación 1</i>	188
<i>SP6 / Ejercicio resuelto</i>	190
<i>SP6 / Ejercicio por resolver</i>	193
<i>SP6 / Evaluación de paso</i>	197
Situación profesional 7: Programación Orientada a Objetos	199
<i>SP7 / H1: Clases</i>	200
<i>SP7 / Autoevaluación 1</i>	215
<i>SP7 / H2: Construcción de una Clase</i>	218
<i>SP7 / Autoevaluación 2</i>	221
<i>SP7 / H3: Modelo de programa completo usando clases</i>	224

SP7 / Autoevaluación 3	229
SP7 / Ejercicio resuelto	231
SP7 / Ejercicio por resolver	238
SP7 / Evaluación de paso	239
Cierre	242
Bibliografía	243

AGRADECIMIENTOS

Agradecemos a todos aquellos que han aportado su investigación, su experiencia y su tiempo para la elaboración de este TID.

Especialmente a Érica BONGIOVANNI, Alejandra FARNETI y Diego OBREGÓN por haber ofrecido el texto que se utilizó como base para hacer este.

Agradecemos, finalmente, a Fernando FRÍAS, Director de la Carrera de INFORMÁTICA del Colegio Universitario IES..

AUTORES



Pablo Javier Romo

- Analista de Sistemas de Computación. Egresado del Colegio Universitario IES.
 - Programador de aplicaciones visuales en la empresa CDSI Argentina S.A.
 - Docente del Colegio Universitario IES en las materias: *Estructura de Datos, Matemática Discreta, Programación Lógica I y Programación Lógica II.*
-

EQUIPO DE PRODUCCIÓN

Producción y dirección general

- Director general: Alberto Rabbat
- Directora Académica: María Fernanda Sin
- Vicedirectora Académica: María Teresa de las Casas

Planificación y coordinación general

- Coordinador de estudios a distancia: Érica Bongiovanni

Producción Multimedial

- Sebastian Benito

- Nicolás Irusta
- Diego Oliva

Producción Académica

- Ana Giró
- Telmo Torres

Coordinación de sistemas

- Marcela Giraudo

Diseño y Desarrollo

- G. Alejandro Zabala
-

USO DE MARCAS

Cláusulas de uso de marcas y derechos de autor de terceros.

A. Uso atípico de marca ajena: Exclusión de los usos no comerciales del Derecho

Marcario.PERSPECTIVAS S.A. en su carácter de titular de los derechos intelectuales sobre la presente obra declara por esta vía que el uso que realiza de marcas comerciales de terceros lo es sólo a los fines informativos y didácticos, para mejor comprensión de los lectores y alumnos del contenido de la obra, siendo el mismo de carácter atípico (uso atípico de marca ajena) y lícito. Este uso, a tenor de la jurisprudencia vigente queda fuera del ius prohibendi, que detenta el titular de cada marca registrada, atento no ser el mismo de carácter comercial en relación al producto que distinguen las referidas marcas, y por ende de índole marcario.-

B. Uso de derechos de autor en videos, diskettes, imágenes y audio: Libre utilización -Uso privado- de obras protegidas. PERSPECTIVAS S.A. en su carácter de titular de los derechos intelectuales sobre la presente obra declara por esta vía que el uso que realiza de determinadas grabaciones (audio y video), e imágenes (fotografías) de terceros, lo es a los fines informativos y didácticos, para mejor comprensión de los alumnos del contenido de la obra, siendo el mismo de carácter privado y no comercial, y desde ya respetando el derecho de cita, esto es declarando en toda ocasión la cita o fuente (obra y autor) de la cual se toman los fragmentos de obras de terceros para incorporarlos a la presente (Convenio de Berna, Acta de París, 1971 – Art. 10, § 2 y § 3).-

C. Modificación de obras literarias por el titular de los derechos patrimoniales: PERSPECTIVAS S.A. en su carácter de titular de los derechos intelectuales (patrimoniales) sobre la presente obra, "Programación Lógica 1", aclara que ha autorizado su modificación a Pablo Javier ROMO respecto de la obra original, "Programación Lógica" publicada en los talleres gráficos de Editorial Copiar, en enero de 2009, bajo N° de ISBN 978-987-600-062-8, constituyéndose en autores morales Érica BONGIOVANNI, Alejandra FARNETI y Diego OBREGÓN de la obra referida y modificada. Se declara a todo efecto, que los derechos intelectuales se ceden y mantienen a favor de su titular PERSPECTIVAS S.A.

COPYRIGHT

Romo, Pablo Javier

Programación lógica 1 / Pablo Javier Romo; coordinado por María Teresa de las Casas; dirigido por José Alberto Rabbat. - 1a ed. - Córdoba : IES Siglo 21, 2012.

E-Book.

ISBN 978-987-600-207-3

1. Programación. 2. Enseñanza Superior. I. De las Casas, María Teresa, coord. II. Rabbat, José Alberto, dir.

III. Título

CDD 005.107 11

1er Edición

© 2012 – Editorial IES Siglo 21

Buenos Aires 563

TE: 54-351-4211717

5000 - Córdoba

Queda hecho el depósito que establece la Ley 11723

Libro de edición argentina

No se permite la reproducción parcial o total, el almacenamiento, el alquiler, la transmisión o la transformación de este libro, en cualquier forma o por cualquier medio, sea electrónico o mecánico, mediante fotocopias, digitalización u otros métodos, sin el permiso previo y escrito del editor. Su infracción está penada por las leyes 11723 y 25446.

Se terminó de replicar durante el mes de mayo de 2012 en el departamento de Logística en Editorial IES Siglo 21.



¿Cómo está organizado este texto?

Usted está en presencia de este texto que los autores proponen para la comprensión y estudio de la asignatura. Ha sido preparado y diseñado para facilitarle el acceso al conocimiento, a partir de una secuencia uniforme cuyo punto de partida es la práctica profesional cotidiana y no la teoría alejada de la realidad.

Está organizado de la siguiente manera:

- **Introducción.** Indica qué papel desempeña la asignatura dentro de la carrera y los conceptos básicos que usted conocerá.
- **Esquema.** Muestra los enlaces que unen los conceptos centrales de la asignatura entre sí.
- **Situación profesional.** Lo ubica frente a un problema de la práctica profesional cotidiana que puede ser resuelto, ya que existe al menos una solución para ello, a través de conocimientos específicos que en cada caso se aportan.
- **Herramientas.** Son los conocimientos necesarios para resolver la situación profesional planteada.
- **Autoevaluación.** Para que usted compruebe si ha comprendido correctamente lo que se explicó en una herramienta, los autores proponen la resolución de actividades y le ofrecen las respuestas.
- **Ejercicio resuelto.** Bajo este título encontrará una manera de resolver los problemas de práctica profesional planteados, con la selección de las herramientas pertinentes.
- **Ejercicio por resolver.** Ahora le toca a usted. Es el momento de aplicar las herramientas a una Situación profesional nueva o similar a la ya expuesta. Todas las dudas que le aparezcan podrán ser planteadas a su docente/tutor.
- **Evaluación de paso.** Para que usted compruebe si ha comprendido correctamente lo que se explicó en las distintas herramientas que hasta el momento se han presentado, los autores proponen la resolución de actividades y le ofrecen las respuestas.
- **Bibliografía.** Se indican los textos, revistas y links de consulta a los que podrá recurrir para complementar o ampliar algunos temas.

Introducción

Como autor me ha tocado el desafío de poder resumir en un texto de estudio, las herramientas necesarias para formar a futuros programadores, con el perfil que el Colegio Universitario IES le ofrece a las empresas informáticas del mercado actual.

En este texto encontrará como contenido herramientas que se han trabajado por varios años en nuestra institución, por profesores como Érica Bongiovanni y Diego Obregón. Herramientas que hoy tomo y continúo sumándole mi experiencia como alumno y docente de la institución y lo que el ejercicio de la profesión le ha dado a mi formación.

Programación Lógica 1 es el cimiento que necesita un profesional de la informática para lograr una fuerte base que le sirva para afrontar cualquier tipo de desarrollo que se le presente en su vida profesional.

Para formar esta base nos introduciremos en el diseño de programas basados en interfaz gráfica, desarrollaremos la lógica computacional, reglas de estructuras y control de flujo de datos y algunas estructuras de datos elementales, como así también tendremos una introducción a los que es la programación orientada a objetos.

Con todas estas herramientas, estaremos en condiciones de realizar soluciones a situaciones profesionales de baja complejidad, que usted podrá transformar en programas reales y ejecutables utilizando cualquier lenguaje de programación.

Ahora es tiempo de ponerse a trabajar, espero que este material sea lo suficientemente completo y claro, para que entienda todo su contenido.

¡Le deseo mucha suerte!

El autor



Situación profesional 1: Algoritmia Computacional Elemental

Camino de las Sierras

La empresa Camino de las Sierras, antes de habilitar las cabinas de peaje, realiza una estadística del movimiento de vehículos de la ruta. Para ello, dos semanas antes de comenzar con el cobro de peaje, el personal de las cabinas, solo se limita a contabilizar los vehículos que pasan. Se necesita un programa que facilite esta tarea de cuenta vehicular.

El programa debe ser simple y rápido de operar ya que los usuarios del mismo no tendrán tiempo de abrir muchas ventanas.

Además, deberá considerar que el contador comenzará en cero (0) y por cada vehículo que pase, (no importa el tipo), se incrementará en 1.

Se le ha pedido a usted, que colabora con el área de sistemas, que diseñe una interfaz gráfica y realice el pseudocódigo del programa que resuelva esta situación.

SP1 / H1: Algoritmia y secuencia de comandos

1. Algoritmos

Un Analista de Sistemas, básicamente, se dedicará a lo largo del ejercicio de su profesión a resolver problemas informáticos. Por lo tanto, antes de entrar a la etapa de programación, será conveniente plantear los pasos que se deben seguir para la resolución de una situación problemática, pasos que se respetarán en cada situación durante el desarrollo de los contenidos de esta materia y de las que le siguen.

1.1. Análisis del Problema

El análisis consiste en estudiar el problema planteado para obtener el punto de partida y la forma en que se llegará a la solución. Para tener conocimiento del problema que se debe resolver es necesario leer atentamente el enunciado del mismo. Si no se sabe "qué es lo que se pide", no se puede dar una solución.

Es necesario formularse las siguientes preguntas:

- ¿Qué se pide?
- ¿Qué se conoce?
- ¿Qué se debe obtener?

Para resolver el problema nos concentraremos básicamente en los procesos que se deben realizar para obtener el resultado. Estos procesos son los que conforman el contenido principal de la materia, ya que requieren de la aplicación de una metodología específica para la construcción de los **algoritmos** * 1.1 computacionales.

Básicamente, podemos decir que el algoritmo es la "fórmula" que nos permite resolver el problema. Pero... ¡No piense que las fórmulas están hechas! Cada problema requiere un algoritmo único y, aunque muchos algoritmos son parecidos, un programador requiere de mucha habilidad, práctica y técnica para resolver cada problema.

Un algoritmo debe tener las siguientes características:

- Debe ser **preciso** e indicar el orden en que se realizará cada paso. Algo así como una receta de cocina. Nunca pondremos la comida en el horno antes de mezclar los ingredientes.
- Debe estar **definido**. Esto significa que si seguimos el algoritmo dos veces, o tres, siempre llegaremos al mismo resultado. Volvamos a la receta de cocina, si dos fines de semanas seguidos realizamos la misma receta, y la seguimos al pie de la letra, obtendremos el mismo menú.
- Debe ser **finito**. Esto significa que cada algoritmo debe tener un número determinado de pasos (en algún momento debe terminar).

Típicamente, un algoritmo tiene tres partes:

Entrada: Corresponde a los datos con los que se cuenta. Por ejemplo, si el problema consiste en realizar una comida, los datos de entrada serían ingredientes y utensilios necesarios.

Proceso: Consiste en los pasos a seguir para resolver el problema. Si seguimos con el mismo ejemplo, constituyen la elaboración de la receta.

Salida: Es lo que se obtiene como resultado del proceso. En definitiva, el plato terminado.

Debido a que el ordenador es incapaz de tomar una determinación, es imprescindible que un algoritmo sea absolutamente claro, sin ningún tipo de ambigüedad y, además, que contemple todas y cada una de las posibles situaciones que puedan presentarse durante la resolución del mismo.

En general, así como el ejemplo del plato de comida, cualquier situación de la vida cotidiana se puede resolver mediante un algoritmo. Por ejemplo: Si a un experimentado conductor se le preguntase cómo pone en marcha su automóvil seguramente contestaría: "Es fácil, se pone en marcha el auto y primera"

Efectivamente es fácil. Pero... ¿Qué sucedería si un individuo que no ha conducido nunca siguiera estas instrucciones? Lo más probable es que si el automóvil tiene una marcha puesta al momento de darle arranque no se obtenga el resultado que se espera. La conclusión inmediata a la que se debe llegar es que este conductor no ha tenido en cuenta todas las posibilidades que se pueden presentar.

El algoritmo adecuado debería desglosar el problema en instrucciones simples y concretas, comprensibles para cualquier persona y que indique claramente el orden en que se ejecutarán dichas instrucciones.

Analice el siguiente ejemplo:

Algoritmo PonerElAutoEnMovimiento

Inicio

Pisar el embrague con el pie izquierdo.

Poner punto muerto.

Soltar el embrague.

Girar la llave de contacto.

Pisar el embrague con el pie izquierdo.

Poner primera.

Levantar el pie izquierdo del embrague a medida que se pisa el acelerador con el pie derecho.

Fin

Este es un ejemplo típico de un algoritmo de tipo secuencial. Esto significa que todas las instrucciones deben realizarse, una a continuación de la otra, sin posibilidad de saltar una sola. Veremos más adelante que existen otras dos estructuras de programación además de la secuencial: las estructuras alternativas y las repetitivas.

Volviendo a nuestra situación profesional, nuestros datos de entrada serán los autos que pasen por la casilla. El proceso, será el sumarle a la cantidad que tengamos contado, el valor de 1. Y, como salida, será el conteo de todos los autos que pasaron por la casilla.

1.2. Documentación

El objetivo de documentar un programa es que cualquier programador sea capaz de entenderlo (e incluso modificarlo en caso de necesidad) aunque no haya sido el autor del mismo.

Una correcta documentación debe contener:

- Descripción del problema
- Descripción de los datos de entrada
- Descripción de los datos de salida
- Descripción del algoritmo

1.3. Codificación del algoritmo en un lenguaje

Para que el algoritmo diseñado en la fase anterior pueda ser introducido en el ordenador, debe ser codificado en un lenguaje de programación (C#, Visual Basic, Java, etc.), siguiendo las reglas de sintaxis que ese lenguaje exija.

Si bien este tema no se desarrollará en esta materia, todo lo aquí aprendido posibilitará la codificación de algoritmos de manera exitosa en las asignaturas correspondientes.

1.4. Características de los nombres de Objetos en un programa

En un programa, cada elemento debe ser identificado con un nombre, ya sea un componente de la interfaz o un componente de los procesos o **procedimientos**. La definición de los nombres, y su posterior referencia, es el único medio que tiene el ordenador para acceder a esos elementos.

Es importante destacar que los nombres están constituidos por una secuencia de caracteres, que pueden ser letras mayúsculas o minúsculas (o, eventualmente, números) pero no pueden tener espacios en blanco.

Cada formulario o ventana se identifica con un nombre único. Por ejemplo: "**Consulta**". Pero el problema es que pueden existir botones de comandos o cajas de texto que puedan denominarse de forma similar. Por eso es conveniente utilizar tres caracteres en la parte inicial del nombre que haga referencia al tipo de control, como se detalló precedentemente en este texto. Un formulario, podría identificarse como **frmConsulta** (se tomaron las tres primeras consonantes de la palabra "formulario"). Una caja de texto podría denominarse **txtConsulta**, y un botón de comando **cmdConsulta**. Observe que, de esa forma, no solo se accede al elemento de la interfaz sino que, además, se indica qué tipo de control es el que se está utilizando.

Si el programador considera que es conveniente que el nombre de un control esté compuesto por varias palabras, no podrá dejar espacio entre las mismas, por lo que se sugiere comenzar con mayúscula cada palabra del nombre, por ejemplo: **frmConsultaDeSaldos**. De este modo, identificar las palabras constituyentes del nombre, es más simple que si no utilizamos las mayúsculas (**frmconsultadesaldos**).

A continuación se listan los prefijos utilizados en la nomenclatura de los diferentes objetos de interfaz, según detallaremos anteriormente al estudiar cada uno de ellos:

Control	Prefijo	Ejemplo
Formulario o Ventana	frm	frmConsulta
Etiqueta	lbl	lblSaldoPendiente
Caja de texto	txt	txtNombre
Botón de Comando	cmd	cmdSalir
Lista Desplegable	lst	lstNombre
Cuadro de Lista	lst	lstLocalidadOrigen
Botones de Opción	opt	optCuenta
Casillas de Verificación	chk	chkNegrita
Grilla	grl	grlClientes
Marco	mrc	mrcFormato

Tenga en cuenta que un formulario contiene muchos controles GUI. Cada uno de esos componentes debe documentarse indicando el tipo de control y el nombre que se le asigna. Esta documentación se denomina

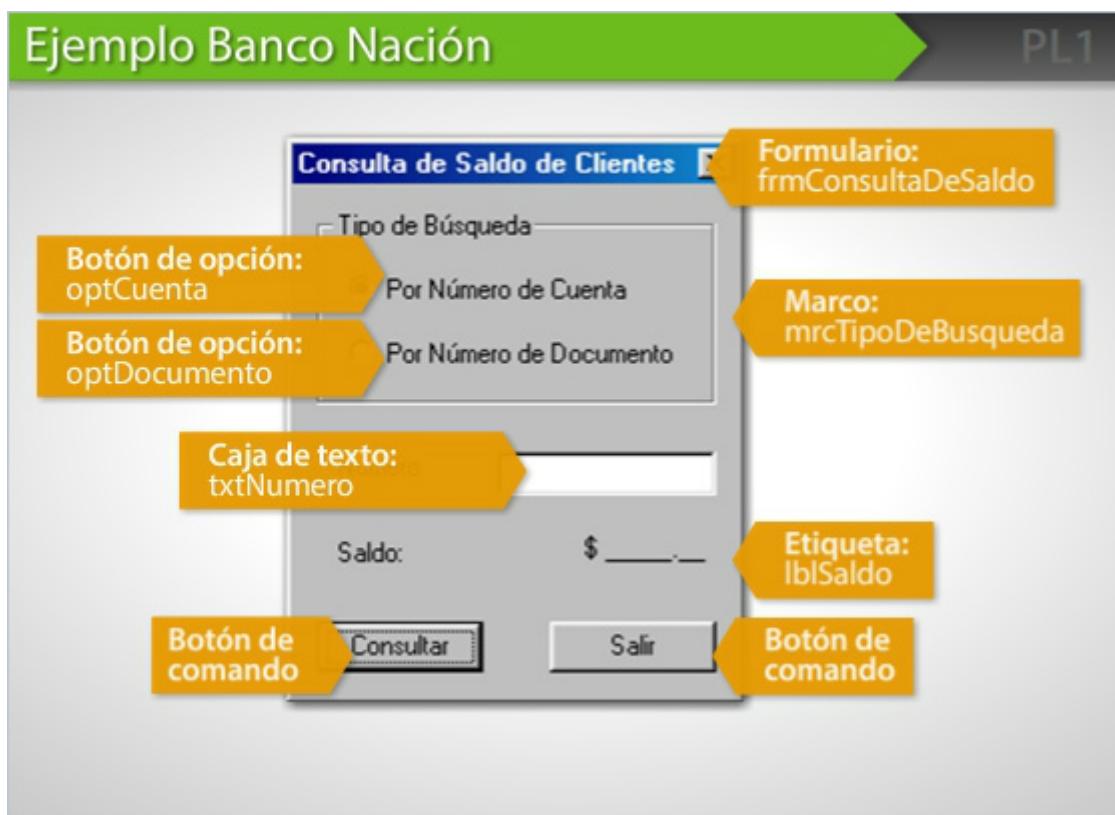
"definición de la interfaz" y forma parte de todo programa, junto con los procedimientos (algoritmos propiamente dichos) que ejecutará.

Veamos, a partir del siguiente ejemplo, la forma de definir los componentes de una interfaz.

1.5. Ejemplo Banco Nación

El Banco Nación necesita un programa que les permita a sus empleados consultar los saldos de los clientes. El sistema le permitirá realizar la consulta por Número de Cuenta o por Número de Documento, el saldo deberá mostrarse en la misma interfaz.

Para solucionar este problema, sería adecuada la implementación de una interfaz como la del siguiente ejemplo:



A partir de la misma el programador deberá definir cada uno de los componentes. Esto es parte de la documentación del programa.

PSEUDO: Pseudo frmConsultaDeSaldoDeClientes

```
'Declaración del Formulario
Formulario frmConsultaDeSaldoDeClientes
Marco mrcTipoDeBúsqueda
    'Es el que está seleccionado
    BotonDeOpcion optCuenta = VERDADERO
    'Es el que no está seleccionado
    BotonDeOpcion optDocumento = FALSO
Fin Marco
```

```

CajaDeTexto txtNumero
Etiqueta lblSaldo
BotonComando cmdConsultar
BotonComando cmdSalir
Fin Formulario

```

Observe que se respeta la tabulación de las estructuras. El formulario es el que contiene todos los controles, se menciona el tipo de control y se asigna el nombre. En un nivel más interno se mencionan los demás controles con sus respectivos nombres.

Los controles que pertenecen al marco están en un nivel más interno, para que quede bien establecido que son elementos del marco y no del formulario.

Cada botón de comando es encargado de disparar una acción. Quiere decir que cuando el usuario oprima uno de ellos se ejecutará un procedimiento.

2. Procedimientos

Un procedimiento es un subalgoritmo. La mayoría de los procedimientos se ejecutan cuando el usuario solicita algo al ordenador. Por ejemplo, cuando oprime un botón de comando con el mouse.

Los programas con entorno visual tienen procedimientos relacionados con los controles GUI. Eso significa que cuando un usuario oprime un botón, el ordenador lee y ejecuta el procedimiento asociado a dicho botón.

Cabe destacar que existen dos tipos de procedimientos:

- Procedimientos de Evento
- Procedimientos Definidos por el Programador

2.1. Procedimientos de Evento

Son los procedimientos que se ejecutan como consecuencia de una acción del usuario sobre el control GUI. Generalmente el evento es el **clic** con el mouse. Si en una interfaz gráfica existen muchos controles encargados de ejecutar acciones, el programador deberá escribir un procedimiento para cada uno.

Estos **procedimientos**, al igual que los controles GUI, deben identificarse con un nombre único (no debe repetirse), y se debe indicar el nombre del control GUI y el nombre del evento del usuario que lo activa, como se detalló al principio de este texto.

Suponga que en el momento que el usuario oprime un botón denominado **cmdSaludar** el programa muestra un mensaje. El programa será el siguiente. Es un programa que utiliza solo procedimientos de eventos:

PSEUDO: Pseudo frmBienvenida

```

Programa Bienvenida
    'Definición del formulario
    Formulario frmBienvenida
        BotonComando cmdSaludar
    Fin Formulario
    'Desarrollo de los Procedimientos de Eventos de la clase
    Procedimiento cmdSaludar_Click
        Mensaje
            Título: "Bienvenida"

```

```
Icono: Personalizado  
Texto: "Bienvenido al Sistema"  
Fin Mensaje  
Fin Procedimiento  
Fin Programa
```

Cajas de Mensaje

El sistema operativo permite a las aplicaciones implementar Cajas de Mensajes estándar con una sola instrucción, que tiene la siguiente estructura:

PSEUDO: Pseudo Cajas de Mensajes

```
Mensaje  
Título: "Título de la Caja de Mensajes"  
Icono: (Información, Alerta, Pregunta, Error Fatal, Personalizado)  
Texto: "Texto que presentará la Caja de Mensajes"  
Fin Mensaje
```

Al ejecutar esta estructura se presenta al usuario una caja de mensajes como las que siguen:



2.2. Procedimientos Definidos por el Programador

Este tipo de procedimientos es referido en muchos textos como "Procedimientos Definidos por el Usuario". En estos casos se considera al Programador como Usuario de la herramienta de desarrollo de software.

A diferencia de los Procedimientos de Evento, estos son procedimientos que no están relacionados con controles GUI. Los procedimientos de este tipo son ejecutados cuando se invocan desde otro procedimiento (de cualquier tipo).

Al igual que los procedimientos anteriores, deben tener un nombre único que los identifique. Ese nombre se utilizará para poder ejecutar el mismo.

Estos procedimientos facilitan la tarea de programación ya que se pueden reutilizar. Por ejemplo, el mismo se

podrá ejecutar desde un ítem de menú, un botón de comando y desde una barra de herramientas, si fuese necesario. El programador solo lo programa una vez y lo convoca con la palabra reservada **Ejecutar** todas las veces que lo necesite.

Veamos el siguiente ejemplo. Es un programa que utiliza procedimientos de eventos y procedimientos creados por el programador:

PSEUDO: Pseudo Palabra reservada "Ejecutar"

```
Programa Bienvenida
    'Definición del Formulario
    Formulario frmBienvenida
        BotonComando cmdSaludar
    Fin Formulario
    'Desarrollo de los Procedimientos de Eventos
    Procedimiento cmdSaludar_Click
        'Se convoca al procedimiento "Bienvenida"
        Ejecutar Bienvenida()
    Fin Procedimiento
    'Desarrollo de los Procedimientos creados por el Programador
    'Este procedimiento no responde a un evento
    Procedimiento Bienvenida()
        Mensaje
            Título : "Bienvenida"
            Icono : Personalizado
            Texto : "Bienvenido al Sistema"
        Fin Mensaje
    Fin Procedimiento
Fin Programa
```

Del mismo modo que en este programa se invoca, desde el evento **clic** sobre un botón, a un procedimiento definido por el programador que ejecuta una tarea puntual, así también podría resolverse, volviendo a nuestra Situación profesional, el caso de desarrollar los cálculos para la cuenta de los vehículos en un procedimiento y limitarnos a invocarlo desde algún evento del usuario sobre la interfaz, como por ejemplo, clic sobre un botón de comando que indique "Contar Vehículo".

2.3. Pasaje de Parámetros a los Procedimientos

Un procedimiento puede recibir parámetros para su ejecución. Es decir, al invocarlo, pueden pasársele ciertos valores (variables o constantes) para que sean procesados por el procedimiento.

En el ejemplo anterior se observa un procedimiento definido por el programador llamado Bienvenida, el que se encarga de mostrar una caja de mensaje con un saludo al usuario. Si observa detenidamente la línea de código en que el procedimiento es invocado para su ejecución, verá que la misma dice:

Ejecutar Bienvenida()

Los paréntesis que se abren y cierran indican que este procedimiento simplemente es invocado sin indicarle al mismo ningún parámetro para su funcionamiento. Si se hubiere querido pasar algún parámetro (como por ejemplo el nombre del usuario, para poder saludarlo por su nombre) hubiera hecho falta incluir tal parámetro entre los paréntesis, tanto en la llamada al procedimiento, como en el procedimiento propiamente dicho.

Veamos a continuación el mismo ejemplo, pero considerando que en la interfaz de usuario contará con una caja de texto en la que el usuario ingresará su nombre. Querremos pasar ese parámetro al procedimiento de Bienvenida para que el mismo pueda saludar al usuario adecuadamente.

Observe que se han subrayado las modificaciones sobre el pseudocódigo anterior, a los efectos de que pueda compararlos y comprender las diferencias resultantes del pasaje de parámetros a un procedimiento.

En nuestra Situación profesional nos encontraremos con un procedimiento de evento que responderá al clic del botón de comando que ejecutará el usuario del sistema para el conteo.

REFERENCIAS 1

1.1 : Algoritmo

Un algoritmo es un conjunto de instrucciones que especifican el orden en que se realizan las operaciones necesarias para resolver un problema.



¡Vamos a comprobar cuánto aprendiste!

1. Indique la opción correcta

Los algoritmos cuentan con 3 partes: Entrada, Visualización y Salida.

- Verdadero
- Falso

2. Indique la opción correcta

Un procedimiento es un subalgoritmo o subprograma.

- Verdadero
- Falso

3. Indique la opción correcta

Existen 2 tipos de procedimientos, los de eventos y los definidos por el programador.

- Verdadero
- Falso

4. Indique la opción correcta

Los procedimientos de eventos son los que responden a lo que piensa el usuario.

- Verdadero
- Falso

5. Indique la opción correcta

Los procedimientos solo pueden recibir un solo parámetro.

- Verdadero
- Falso

6. Indique la opción correcta

Para llamar a un procedimiento en pseudocódigo, se usa la palabra reservada "Llamar".

- Verdadero

Falso

7. Indique la opción correcta

Una caja de texto, puede ser una entrada a un algoritmo.

Verdadero

Falso

Respuestas de la Autoevaluación

1. Indique la opción correcta

Los algoritmos cuentan con 3 partes: Entrada, Visualización y Salida.

Verdadero

Falso

2. Indique la opción correcta

Un procedimiento es un subalgoritmo o subprograma.

Verdadero

Falso

3. Indique la opción correcta

Existen 2 tipos de procedimientos, los de eventos y los definidos por el programador.

Verdadero

Falso

4. Indique la opción correcta

Los procedimientos de eventos son los que responden a lo que piensa el usuario.

Verdadero

Falso

5. Indique la opción correcta

Los procedimientos solo pueden recibir un solo parámetro.

Verdadero

Falso

6. Indique la opción correcta

Para llamar a un procedimiento en pseudocódigo, se usa la palabra reservada "Llamar".

Verdadero

Falso

7. Indique la opción correcta

Una caja de texto, puede ser una entrada a un algoritmo.

Verdadero

Falso

SP1 / H2: Tipos de datos elementales. Constantes. Variables

1. Acceso y Almacenamiento de Datos

Hemos visto que los botones de comando disparan procesos, pero tenemos que tener en cuenta que la mayoría de las veces operan con Datos que deben estar almacenados en algún lugar del ordenador.

Desde el punto de vista lógico, cuando en un programa se ingresan caracteres en una caja de texto, estos se almacenan en la memoria del ordenador. Es importante considerar que la memoria almacena cada uno de estos caracteres uno a continuación del otro, formando una cadena.

Cada posición de la memoria permite almacenar un carácter. Así, si en la caja de texto se ingresó el nombre: "José", en memoria se habrán grabado estos caracteres, formando una sucesión, como vemos en el siguiente ejemplo.

Pero un programador no trabaja directamente con los datos almacenados en una posición de la memoria. Nunca va a trabajar con un programa donde trate de leer la posición de memoria absoluta AF024B16. Al programador se le facilita el acceso a los datos desde el momento que puede acceder por medio de un "nombre simbólico" al cada dato almacenado. Es el propio lenguaje de programación el que se encarga de convertir cada Nombre de un dato en la Dirección de Memoria donde el dato está almacenado, para poder accederlo y realizar la operación que se requiera con él.

Como ya comentamos anteriormente, un programa es un conjunto de instrucciones ejecutables en una computadora, que permiten cumplir una función específica. Los programas requieren de **Datos** y **Algoritmos**. En los apartados anteriores definimos el concepto de algoritmo. Ahora desarrollaremos la definición de datos y tipos de datos.

Los datos son los valores que se necesita tener disponibles para que un programa pueda cumplir su objetivo: resolver un problema. Son, en otros términos, los parámetros de dicho problema.

Definiremos dos **clases de datos** simples: variable y constante.

- **Variable.** Es un espacio de memoria que permite almacenar un valor que puede variar durante la ejecución del programa. Esta característica es la que le da su nombre. Por ejemplo, podemos crear una variable llamada A y hacer que valga 5. Luego, siguiendo el programa, la misma variable puede recibir un nuevo valor.

A = 5 → Primero le asignamos el valor 5

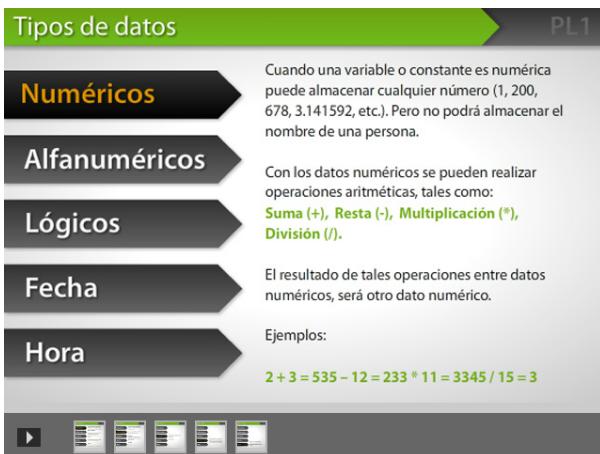
...

A = 8 → Luego le asignamos 8 dentro del mismo programa

- **Constante.** Al igual que la variable, es un espacio en memoria que puede almacenar un valor. Pero a diferencia de aquella, el valor de la constante permanecerá invariable durante la ejecución del programa. En otras palabras, no puede modificarse el valor de una constante durante el tiempo de ejecución del programa.

Tanto las variables como las constantes pueden almacenar determinados tipos de datos. Definiremos "Tipo de Dato" como el conjunto de valores que puede tomar una variable o el valor que almacena una constante.

En esta materia trabajaremos con los siguientes **tipos de datos**:



* 2.1 Imágenes

Imágenes "Tipos de datos"

Tanto los datos constantes como variables deben guardarse en la memoria de datos de una computadora. En ambos casos estarán representados simbólicamente con un nombre que se asocia con una dirección única de memoria. En esa dirección de memoria se almacena el valor que tomará la constante o los diferentes valores que tomará la variable.

Como vimos en los ejemplos anteriores, con la mayoría de los datos se realizan operaciones. Supongamos que necesitamos sumar dos números. Para sumar 2 más 3 realizamos lo siguiente:

$$2 + 3$$

En los procedimientos tenemos que seguir algunas reglas. Por ejemplo, el dos y el tres seguramente son datos contenidos en alguna variable, constante o control GUI (caja de texto, etiqueta, etc.).

Por ejemplo, si contamos con dos variables, una denominada **valorA**, que contiene al 2, y la otra denominada **valorB**, que contiene al 3, la suma se puede realizar de la siguiente forma: **valorA + valorB**.

Tenga en cuenta que en un programa los resultados de los cálculos siempre se almacenan en algún lugar, ya sea en una variable, o en un control GUI. Pero, para que estos componentes tomen el valor resultante de la operación, es necesario realizar una asignación del resultado. La asignación siempre se realiza de derecha a izquierda. Por ejemplo, supongamos que contamos con una etiqueta en una interfaz gráfica denominada: **lblResultado**, la asignación del resultado a la etiqueta se realiza de la siguiente forma:

$$\text{lblResultado} = \text{valorA} + \text{valorB}$$

El "**=**" es el operador de asignación.

Se asigna el resultado de la operación de la derecha a la variable de la parte izquierda de la expresión.

El resultado de la ejecución de la línea de programa anterior será la suma de los valores de las variables **valorA** y **valorB**, y el dato resultante de dicha suma será establecido como título de una etiqueta llamada **lblResultado**, con lo que el usuario será capaz de ver el resultado de la suma en la ventana que contiene la etiqueta.

Contador

Cuando a una variable se le asigna el mismo valor que ya tenía más un valor constante, se la denomina contador:

A = A + 1 → 1 es un valor constante

En esta operación a la variable **A** se le asigna el mismo valor que tenía más 1, lo que significa que se incrementará de uno en uno. Normalmente, antes de incrementar un contador es necesario asignarle un valor inicial, que generalmente suele ser 0.

Un contador puede contar de uno en uno, de diez en diez, e incluso de -2 en -2. Lo que importa para considerarlo como contador es que el incremento o decremento que sufre la variable sea constante.

Acumulador

Cuando a la variable o control GUI se le asigna lo mismo que tiene más un valor variable, se denomina acumulador. El incremento depende del contenido de otra variable:

Total = Total + Venta → Venta es un valor variable

En esta operación, a la variable **Total** se le suma (acumula) el valor de la variable **Venta**. Es decir, **Total** valdrá luego de la ejecución de esta línea lo mismo que valía antes más **Venta**.

Como **Venta** es a su vez una variable (que puede variar su contenido a lo largo de la ejecución del programa) el incremento sobre **Total** será variable, por eso se denomina Acumulador.

En nuestra Situación profesional, necesitaremos una variable del tipo contador que empiece en cero y que al final del día nos devolverá la cantidad de autos contados en el día.

Nomenclatura de Variables y Controles GUI

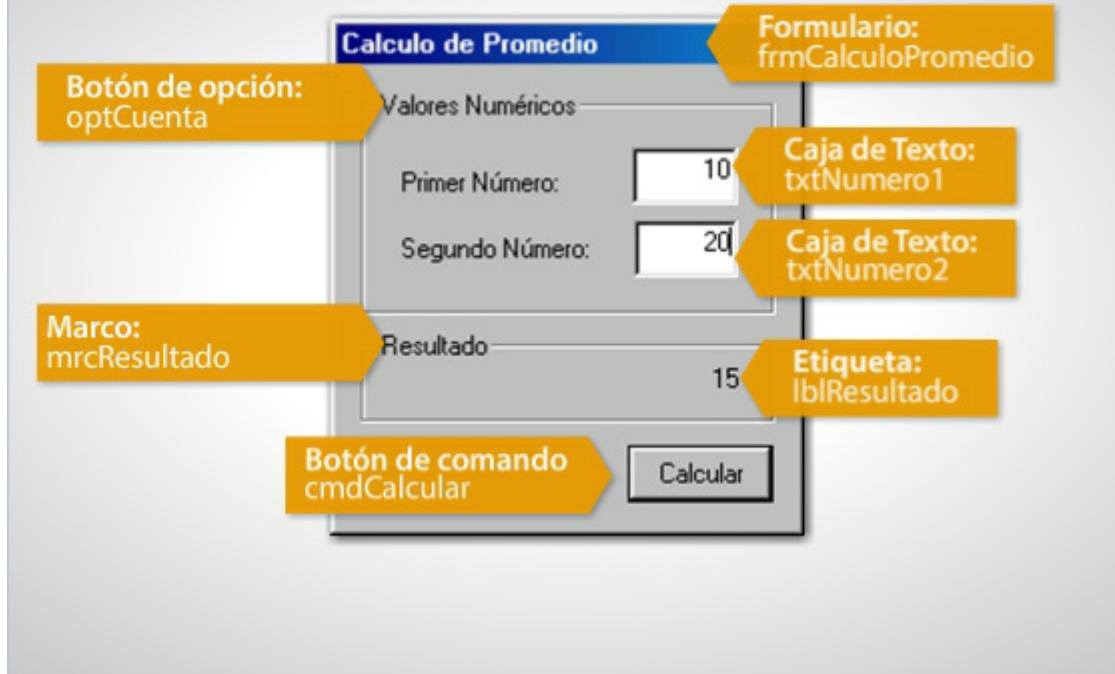
Cuando en un programa se trabaja con datos será necesario, antes del desarrollo del algoritmo, definir el nombre de los datos que se utilizarán y el tipo de dato, así como también definir todos los componentes de la interfaz gráfica.

Por ejemplo, si se necesita mostrar en una etiqueta el promedio de dos números, que el usuario ingresará en cajas de texto, tendremos una interfaz gráfica con los siguientes componentes básicos:

- Dos cajas de texto: para ingresar los números.
- Una etiqueta: para mostrar el resultado.
- Un botón de comando: que tiene relacionado el procedimiento encargado de realizar el cálculo.

Cálculo de promedio: Interfaz

PL1



La definición de la interfaz anterior se realiza del siguiente modo:

PSEUDO: Pseudo Definición de interfaz Cálculo de promedio

```
'Definimos el formulario
Formulario frmCalculoPromedio
    Marco mrcValoresNumericos
        CajaDeTexto txtNumero1
        CajaDeTexto txtNúmero2
    Fin Marco
    Marco mrcResultado
        Etiqueta lblResultado
    Fin Marco
    BotonComando cmdCalcular
Fin Formulario
```

El botón de comando "cmdCalcular" tiene asociado un procedimiento, que es el que se encargará de tomar los datos de entrada de la interfaz, calcular el promedio y mostrar el resultado en la misma interfaz gráfica.

En nuestra Situación profesional, definiremos una pequeña interfaz que da respuesta a lo solicitado.

Y tendrá su nomenclatura de componentes como se muestra a continuación:

Cálculo de vehículos: Interfaz



2. Operaciones Básicas

En un programa se requieren muchos operadores. Existen diferentes tipos, que se utilizan según la necesidad del programador.

Ahora veremos algunos:

Operador de asignación

Se utiliza para asignar a una variable su valor, o a un control GUI su contenido. El operador de asignación lo podemos ver en las bibliografías representado por una flecha ← de derecha a izquierda, o con un signo igual =.

Operador	Utilización	Ejemplos
=	Para asignar un valor a una variable o control GUI. Recuerde que la asignación siempre se realiza de derecha a izquierda. Muchas veces se remplaza el signo igual: “=” por una flecha: “←”.	Nombre = “Ana” Cantidad = Cantidad + 1 lblResultado = Número Total = Total + Importe

Operadores aritméticos

Se utilizan para realizar operaciones de cálculo en un algoritmo.

Operador	Utilización	Ejemplos
+	Para sumar .	Cantidad = Cantidad + 1
-	Para Restar .	Total = Precio – Descuento
*	Para Multiplicar .	SubTotal = Precio * Cantidad
/	Para Dividir .	Promedio = Suma / Cantidad

Operadores Jerárquicos. Expresiones

Se utilizan para establecer jerarquías en las operaciones de cálculo y en las comparaciones lógicas de más de dos expresiones. Se utilizan para esto los paréntesis (), en tantos niveles como sea necesario.

Operador	Utilización	Ejemplos
()	Para establecer jerarquías.	PromedioParcial = (Primero + Segundo) / 2

2.1 Imágenes: Tipos de datos

Tipos de datos

PL1

Numéricos

Cuando una variable o constante es numérica puede almacenar cualquier número (1, 200, 678, 3.141592, etc.). Pero no podrá almacenar el nombre de una persona.

Alfanuméricos

Con los datos numéricos se pueden realizar operaciones aritméticas, tales como:

Suma (+), Resta (-), Multiplicación (*), División (/).

Fecha

El resultado de tales operaciones entre datos numéricos, será otro dato numérico.

Hora

Ejemplos:

$$2 + 3 = 5 \quad 35 - 12 = 23 \quad 3 * 11 = 33 \quad 45 / 15 = 3$$

Numéricos

En este caso la variable o constante podrá almacenar un carácter, una palabra, una frase, o incluso una dirección.

Alfanuméricicos

La única operación posible con cadenas de caracteres es la concatenación (&).

Lógicos

Por ejemplo:

Si concatenamos la cadena "Hola" con la cadena "Juan", lo que obtendremos será la cadena "Hola Juan".

En símbolos:

"Hola" & "Juan" = "Hola Juan"

Fecha

Hora

Numéricos

Alfanuméricos

Lógicos

Fecha

Hora

Son los datos que pueden tomar solo dos valores posibles:

Verdadero o Falso.

Numéricos

Alfanuméricos

Lógicos

Fecha

Hora

Son un tipo especial de dato numérico que almacena los tres componentes de una fecha (día, mes y año) en un solo tipo de datos, con el formato "**DD/MM/AA**" o "**DD/MM/AAAA**".

Numéricos

Alfanuméricos

Lógicos

Fecha

Hora

Similar a los datos de tipo fecha, estos
almacenan los componentes de una hora
(hora, minutos y segundos) en el formato
"HH:MM:SS".



¿Te animás a medir cuánto aprendiste?

1. Indique la opción correcta

La expresión "1234" es un tipo de dato:

- Numérico.
- Alfanumérico.
- Constante.

2. Indique la opción correcta

En las siguientes líneas de código, la variable A es un:

```
1 variable A tipo numerica  
2  
3 A = A + 1
```

- Contador.
- Sumador.
- Sumando.

3. Indique la opción correcta

Tras ejecutar el siguiente código, el programa:

```
1 variable A tipo alfanumerica  
2 variable B tipo alfanumerica  
3  
4 A = "2"  
5 B = "9"  
6 Imprimir A + B
```

- Imprimirá 11.
- Imprimirá 29.
- Dará error en la línea 6.

4. Indique la opción correcta

Al siguiente código le falta una línea. ¿A qué número de línea nos referimos?

```
1  'Definimos el formulario
2  Formulario frmCalculoPromedio
3      Marco mrcValoresNumericos
4          CajaDeTexto txtNumero1
5          CajaDeTexto txtNúmero2
6
7      Marco mrcResultado
8          Etiqueta lblResultado
9      Fin Marco
10     BotonComando cmdCalcular
11 Fin Formulario
```

- La línea 2.
- La línea 11.
- La línea 6.

5. Indique la opción correcta

A un acumulador, se le asigna lo mismo que tenía más:

- Cero.
- Uno.
- Un valor variable.

Respuestas de la Autoevaluación

1. Indique la opción correcta

La expresión "1234" es un tipo de dato:

- Numérico.
- Alfanumérico.
- Constante.

2. Indique la opción correcta

En las siguientes líneas de código, la variable A es un:

- Contador.
- Sumador.
- Sumando.

3. Indique la opción correcta

Tras ejecutar el siguiente código, el programa:

- Imprimirá 11.
- Imprimirá 29.
- Dará error en la línea 6.

4. Indique la opción correcta

Al siguiente código le falta una línea. ¿A qué número de línea nos referimos?

- La línea 2.
- La línea 11.
- La línea 6.

5. Indique la opción correcta

A un acumulador, se le asigna lo mismo que tenía más:

- Cero.
- Uno.
- Un valor variable.

SP1 / H3: Forma general de los pseudocódigos

1. Pseudocódigo

Un pseudocódigo es un lenguaje de especificación de algoritmos. La utilización de este lenguaje hace que el paso del algoritmo al lenguaje de máquina sea muy fácil.

La ventaja de utilizar pseudocódigos es que el programador se desentiende de la sintaxis propia de un lenguaje (Visual Basic, Visual C++, Cobol, Java, etc.), lo que le permite ocuparse y concentrarse plenamente en la lógica que el algoritmo requiere.

Por ejemplo: Imprimir una línea que diga "Hola" es una tarea simple, pero el modo de implementarla en los diferentes **lenguajes de programación es diferente** "Hola Mundo" en diferentes lenguajes de programación
http://es.wikipedia.org/wiki/Anexo:Ejemplos_de_implementaci%C3%B3n_del_%C2%ABHola_mundo%C2%BB

En C++: printf("Hola");

En COBOL: DISPLAY "Hola"

En BASIC: PRINT "Hola"

En JavaScript: document.write("Hola");

En todos los casos anteriores, lo que importa es el concepto de imprimir la palabra "Hola", por lo que en nuestros pseudocódigos, independientes del lenguaje de implementación, lo decimos simplemente así:

Imprimir "Hola"

El siguiente es un pseudocódigo que resuelve el cálculo del promedio:

PSEUDO: Pseudo que resuelve Cálculo del promedio

```
Procedimiento cmdCalcular_Click
    'Declaramos las variables a utilizar
    variable Suma tipo numerica
    variable Resultado tipo numerica
    'Realizamos los cálculos
    Suma = txtNumero1 + txtNumero2
    Resultado = Suma / 2
    'Mostramos el resultado
    lblResultado = Resultado
Fin Procedimiento
```

Las variables utilizadas, pueden definirse fuera del procedimiento. Eso significa que estos datos estarán disponibles para todos los procedimientos pertenecientes a la clase. Tenga en cuenta que en los formularios no se tiene un solo botón de comando (se pueden tener varios), lo que significa que tendrá varios procedimientos, al menos uno por cada botón de comando. Para estos casos puede ser útil definir las variables de este modo:

PSEUDO: Pseudo Definición de variables para Cálculo del promedio

```
variable Suma tipo numerica
variable Resultado tipo numerica
Procedimiento cmdCalcular_Click
    Suma = txtNumero1 + txtNumero2
    Resultado = Suma / 2
```

```
lblResultado = Resultado  
Fin Procedimiento
```

Cuando los datos se definen dentro de un procedimiento, se dicen que son locales al mismo. Solo ese procedimiento puede hacer uso de esos datos. El resto de los procedimientos del programa no tiene noción de la existencia de estas variables.

Cuando los datos se definen en forma independiente, fuera de los procedimientos, se dice que son globales, ya que cualquier procedimiento del programa podrá hacer uso de los mismos.



Operadores jerárquicos

Video "Operadores jerárquicos"

Estos procedimientos no siempre requieren de tantas variables. Si se utilizan operadores jerárquicos se pueden simplificar los pasos e incluso hasta se puede llegar a evitar la definición de algunos datos, que es muy importante a la hora de economizar memoria y optimizar los recursos de la computadora.

Veamos el mismo procedimiento de cálculo del promedio anterior, pero sin derrochar variables:

2. Estructura general de un programa en Pseudocódigo

Siempre que trabajemos con pseudocódigo, respetaremos una estructura como se detalla a continuación.

El programa, que al igual que todos los objetos con los que hemos venido trabajando se identifica con un nombre, tendrá las siguientes partes:

- **Definición de los datos globales** (*si los hubiere*). Se definen al principio de la clase. Estarán disponibles para todos los procedimientos de la misma, conservando su valor entre ellos.
- Definición de la **Interfaz Gráfica**. Se define en la estructura Formulario - Fin Formulario, detallando todos los controles GUI que la componen.
- **Desarrollo de los procedimientos**. Se desarrollan los procedimientos necesarios para la respuesta de la clase. Toda la lógica de la programación es conducida por los procedimientos, según la ocurrencia de eventos en el sistema. Primero se desarrollan los procedimientos de evento, como parte constituyente de la Interfaz. Luego los procedimientos definidos por el programador.

Veamos estos componentes en el pseudocódigo completo de nuestro ejemplo:

PSEUDO: Pseudo Programa completo de Cálculo promedio

```
Programa frmCalculoPromedio
    'Definición de los Datos Globales
    variable Suma tipo numerica
    variable Resultado tipo numerica
    'Definición del Formulario
    Formulario frmCalculoPromedio
        Marco mrcValoresNumericos
            CajaDeTexto txtNúmero1
            CajaDeTexto txtNúmero2
        Fin Marco
        Marco mrcResultado
            Etiqueta lblResultado
        Fin Marco
        BotonComando cmdCalcular
    Fin Formulario
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdPromedio_Click
        Suma = txtNúmero1 + txtNúmero2
        Resultado = Suma / 2
        lblResultado = Resultado
    Fin Procedimiento
Fin Programa
```

Tenga en cuenta que este mismo programa resolverá perfectamente el problema si tuviera la siguiente forma, ahorrándonos la declaración de variables innecesarias:

PSEUDO: PseudoPrograma Cálculo promedio (sin declaración de variables)

```
Programa CalculoPromedio
    'Definición del Formulario
    Formulario frmCalculoPromedio
        Marco mrcValoresNumericos
            CajaDeTexto txtNúmero1
            CajaDeTexto txtNúmero2
        Fin Marco
        Marco mrcResultado
            Etiqueta lblResultado
        Fin Marco
        BotonComando cmdCalcular
    Fin Formulario
```

```
'Desarrollo de los Procedimientos de Evento
Procedimiento cmdPromedio_Click
    lblResultado = (txtNumero1 + txtNumero2) / 2
    Fin Procedimiento
Fin Programa
```

En este último ejemplo, la sección correspondiente a la definición de datos globales no está presente, ya que se utiliza directamente los valores almacenados en los controles de la interfaz gráfica.

Además, el cálculo propiamente dicho es más eficiente, pues no se utiliza ninguna variable intermedia (variables Suma y Resultado de la solución anterior), sino que en una sola expresión se resuelve el promedio requerido y se asigna dicho resultado como título a la etiqueta **lblResultado**.

En la resolución de nuestra situación profesional, veremos cómo queda una estructura general realizada en pseudocódigo.



¿Estás listo para un desafío?

1. Indique la opción correcta

La estructura general de un pseudocódigo, respeta la siguiente declaración:

Desarrollo de los Procedimientos

Definición de los Datos Globales

Definición del Formulario

- Verdadero
- Falso

2. Indique la opción correcta

El siguiente código corresponde a un procedimiento de evento.

```
1     variable Suma tipo numerica
2     variable Resultado tipo numerica
3
4     Procedimiento cmdCalcular_Click
5         Suma = txtNumero1 + txtNumero2
6         Resultado = Suma / 2
7         lblResultado = Resultado
8     Fin Procedimiento
```

-
- Verdadero
 - Falso

3. Indique la opción correcta

El evento click de un botón de comando se captura con un procedimiento que se declara:

Procedimiento cmdCalcular_Click

.....

Fin Procedimiento

- Verdadero
- Falso

4. Indique la opción correcta

Los datos globales estan disponibles solo dentro del formulario donde se declaran.

- Verdadero
- Falso

5. Indique la opción correcta

¿Cuál de las siguientes declaraciones de variables es válida en pseudocódigo para una variable numérica?

- variable numerica tipo Suma.
- variable Suma tipo numerica.
- variable suma.

Respuestas de la Autoevaluación

1. Indique la opción correcta

La estructura general de un pseudocódigo, respeta la siguiente declaración:

Desarrollo de los Procedimientos
Definición de los Datos Globales
Definición del Formulario

- Verdadero
 Falso

2. Indique la opción correcta

El siguiente código corresponde a un procedimiento de evento.

- Verdadero
 Falso

3. Indique la opción correcta

El evento click de un botón de comando se captura con un procedimiento que se declara:

Procedimiento cmdCalcular_Click

.....
Fin Procedimiento

- Verdadero
 Falso

4. Indique la opción correcta

Los datos globales estan disponibles solo dentro del formulario donde se declaran.

- Verdadero
 Falso

5. Indique la opción correcta

¿Cuál de las siguientes declaraciones de variables es válida en pseudocódigo para una variable numérica?

- variable numerica tipo Suma.
 variable Suma tipo numerica.
 variable suma.

SP1 / Ejercicio resuelto

El problema de la Situación profesional era el siguiente

Camino de las Sierras

La empresa Camino de las Sierras, antes de habilitar las cabinas de peaje, realiza una estadística del movimiento de vehículos de la ruta. Para ello, dos semanas antes de comenzar con el cobro de peaje, el personal de las cabinas, solo se limita a contabilizar los vehículos que pasan. Se necesita un programa que facilite esta tarea de cuenta vehicular.

El programa debe ser simple y rápido de operar ya que los usuarios del mismo no tendrán tiempo de abrir muchas ventanas.

Además, deberá considerar que el contador comenzará en cero (0) y por cada vehículo que pase, (no importa el tipo), se incrementará en 1.

Se le ha pedido a usted, que colabora con el área de sistemas, que diseñe una interfaz gráfica y realice el pseudocódigo del programa que resuelva esta situación.

.

El programa que resuelve la situación tiene desarrollado los procedimientos de eventos a partir de la siguiente interfaz gráfica:

PSEUDO: Ejercicio resuelto de Camino de las Sierras

Programa ContarVehiculos

```
'Definición del Formulario
Formulario frmContarVehiculos
    Marco     mrcCantidad
    Etiqueta  lblCantidad
    Fin Marco
    BotonComando cmdContar
Fin Formulario
'Definición de los Datos Globales
variable Cantidad = 0 tipo numerica
'Desarrollo de los Procedimientos de Evento
Procedimiento cmdContar_Click
    Cantidad = Cantidad + 1
    lblCantidad = Cantidad
Fin Procedimiento
Fin
```

SP1 / Ejercicios por resolver

Desarrolle las soluciones de los ejercicios planteados a continuación:

1. Se le solicita que realice un programa que obtenga el resultado de evaluar una función de 2^{do} grado. El usuario del programa ingresará el valor de la variable X, y el programa deberá mostrarle el valor correspondiente al resultado (variable Y). La fórmula que define la función a evaluar es la siguiente:

$$Y = 3X^2 + 7X - 15$$

2. El Servicio Meteorológico de la Provincia de Córdoba, le solicita un programa que lea una temperatura en grados Fahrenheit y muestre su equivalente de grados Celsius, Kelvin y Rankine.

- Para convertir a Celsius a la temperatura Fahrenheit se le resta 32 y se multiplica por 5/9.
- Para convertir a Kelvin, se le suma 273 a los grados Celsius.
- Para convertir a Rankine, a los grados Fahrenheit se le suma 460.

3. La agencia de publicidad Rombo Velox está promocionando el último video clip del grupo "La Mosca". Una de las estrategias publicitarias, fue la de instalar varias computadoras en los principales shopping de la ciudad de Córdoba en las que se puede ver dicho video.

Para esta empresa es muy importante saber cuantas personas se acercaron a ver el mismo. Se le solicita que desarrolle un programa que permita resolver esta situación.



¡Vamos a comprobar cuánto aprendiste!

1. Indique la opción correcta

Un contador es un valor variable al que se le suma otro valor variable.

- Verdadero
- Falso

2. Indique la opción correcta

Los datos globales son los que pueden ser accedidos desde todos los procedimientos.

- Verdadero
- Falso

3. Indique la opción correcta

Entrada, Proceso y salida, son las partes de un algoritmo.

- Verdadero
- Falso

4. Indique la opción correcta

El prefijo utilizado para los botones de comando es frm.

- Verdadero
- Falso

5. Indique la opción correcta

Si ejecutamos el siguiente programa:

Se imprimirá:

```

1 Programa Evaluacion
2
3     variable i tipo numerica
4     variable j tipo numerica
5
6     j = 5
7     i = j
8
9     Procedimiento A()
10    j = 3
11    Ejecutar B()
12    Imprimir i
13    Fin Procedimiento
14
15    Procedimiento B()
16    j = j + 3
17    Ejecutar C()
18    Fin Procedimiento
19
20    Procedimiento C()
21    j = j + 1
22    Fin Procedimiento
23
24 Fin Programa

```

- 7
- 3
- 5

Respuestas de la Autoevaluación

1. Indique la opción correcta

Un contador es un valor variable al que se le suma otro valor variable.

Verdadero

Falso

2. Indique la opción correcta

Los datos globales son los que pueden ser accedidos desde todos los procedimientos.

Verdadero

Falso

3. Indique la opción correcta

Entrada, Proceso y salida, son las partes de un algoritmo.

Verdadero

Falso

4. Indique la opción correcta

El prefijo utilizado para los botones de comando es frm.

Verdadero

Falso

5. Indique la opción correcta

Si ejecutamos el siguiente programa:

Se imprimirá:

7

3

5

Situación profesional 2: Estructuras Alternativas

BELLO CORPO

La empresa “Bello Corpo”, que se dedica a la estética corporal, tiene un grupo de profesionales que trabajan en las diferentes áreas: gimnasia, modelación, cosmética, nutrición, entre otras.

Los profesionales que trabajan en la sección de nutrición, antes de recomendar una dieta, deben realizar un diagnóstico del paciente. Para ello, entre otros indicadores, obtienen el índice de masa corporal.

Este índice es un valor que permite determinar si la persona está dentro del peso normal, o si está excedido, o si pesa menos de lo que se considera saludable.

El índice de masa corporal se calcula de la siguiente manera:

$$\text{Índice de Masa Corporal} = \frac{\text{Peso}}{\text{Altura}^2}$$

Este índice permite saber si el paciente tiene el peso adecuado para su contextura. A partir de este valor, el nutricionista, puede recomendar una dieta alimenticia adecuada.

Para facilitar el diagnóstico, se le ha solicitado a usted, que es colaborador del área de sistemas, que diseñe un programa que obtenga el índice de masa corporal y que diagnostique al paciente.

SP2 / H1: Tipos de estructuras Alternativas

En esta herramienta le presentaremos dos tipos de estructuras alternativas:

1. Las SIMPLES (Si - Fin Si)
2. Las COMPUESTAS (Según sea)

1. Estructuras Alternativas Simples: Si – Fin Si

Para la situación que se plantea, las herramientas que hemos visto hasta ahora resultan insuficientes. Se pide mostrar un mensaje que dependerá totalmente del resultado obtenido. Ahora bien, note que no se mostrarán todos los mensajes, sino solo el que corresponda, por lo que el algoritmo deberá discriminar, o si se prefiere "decidir" qué mensaje mostrar.

Para solucionar casos como este, existen las estructuras alternativas (también llamadas condicionales). Para poder comprender las mismas primero plantearemos un caso de la vida cotidiana y, luego, haremos la resolución de un problema que se resuelva con un programa.

Considere que debe indicarle a alguien, que nunca utilizó un teléfono en su vida, lo que debe hacer para realizar una llamada.

Seguramente usted le dirá que debe realizar los siguientes pasos:

Descolgar el auricular.
Esperar la señal de línea.
Marcar el número.
Si hay señal de llamada Entonces
Esperar a ser atendido
Hablar
Colgar Auricular.

Observe que, a diferencia de los algoritmos vistos en la situación profesional anterior, en este, una de sus instrucciones no es imperativa. Es decir, que no obliga a ejecutar una orden determinada, sino que esa orden se ejecutará dependiendo de la veracidad de una condición. Esto se denomina "estructura alternativa".

En un pseudocódigo, la estructura alternativa debe respetar la siguiente sintaxis:

PSEUDO: Pseudo Si

```
Si (condición) Entonces
    Instrucción 1
    ...
    Instrucción n
Fin Si
```

Donde las instrucciones sólo se ejecutarán en caso de cumplirse la condición.

Observe que en el modelo la condición se encerró entre paréntesis, y que las acciones correspondientes están en un nivel interior. Esto es lo que se denomina indentación (tabulación) de un programa. Además tenga en cuenta que al hacerse válida una condición, se ejecutan desde una (1) a **n** instrucciones. La cantidad solo dependerá de la situación a resolver. Por eso es que se verán muchos ejemplos que irán de casos muy simples

(como este), a casos muy complejos (que se desarrollarán más adelante).

En una primera instancia haremos un pseudocódigo con el caso de la llamada telefónica:

PSEUDO: Pseudo llamada telefónica

```
Procedimiento LlamadaTelefónica
    Descolgar el auricular
    Esperar la señal de línea
    Marcar el número
    Si (hay señal de llamada) Entonces
        Esperar a ser atendido
        Hablar
    Fin Si
    Colgar Auricular
Fin Procedimiento
```

Cuando se realiza el pseudocódigo de un programa, las condiciones deben respetar una sintaxis mínima. Para ello es necesario utilizar operadores que permiten realizar comparaciones de diferentes tipos. Estos operadores se denominan **operadores relacionales**. Los detallamos en la siguiente tabla:

Operador	Utilización	Ejemplos
=	Para comparar si dos elementos de un programa son iguales.	Si (txtNúmero = 50) ... Fin si
>	Para comparar si el primer elemento es mayor al segundo.	Si (txtNúmero > 50) ... Fin si
>=	Para comparar si el primer elemento es mayor o igual al segundo.	Si (txtNúmero >= 50) ... Fin si
<	Para comparar si el primer elemento es menor al segundo.	Si (txtnombre < "Ana") Fin si
<=	Para comparar si el primer elemento es menor o igual al segundo.	Si (txtNúmero <= 50) ... Fin si
<> ó □	Para comparar si dos elementos son diferentes.	Si (txt nombre <> "") ... Fin si Las dos comillas juntas se utilizan, en este caso, para preguntar si el contenido de la caja de texto no está vacío

1.1 Estructura alternativa simple

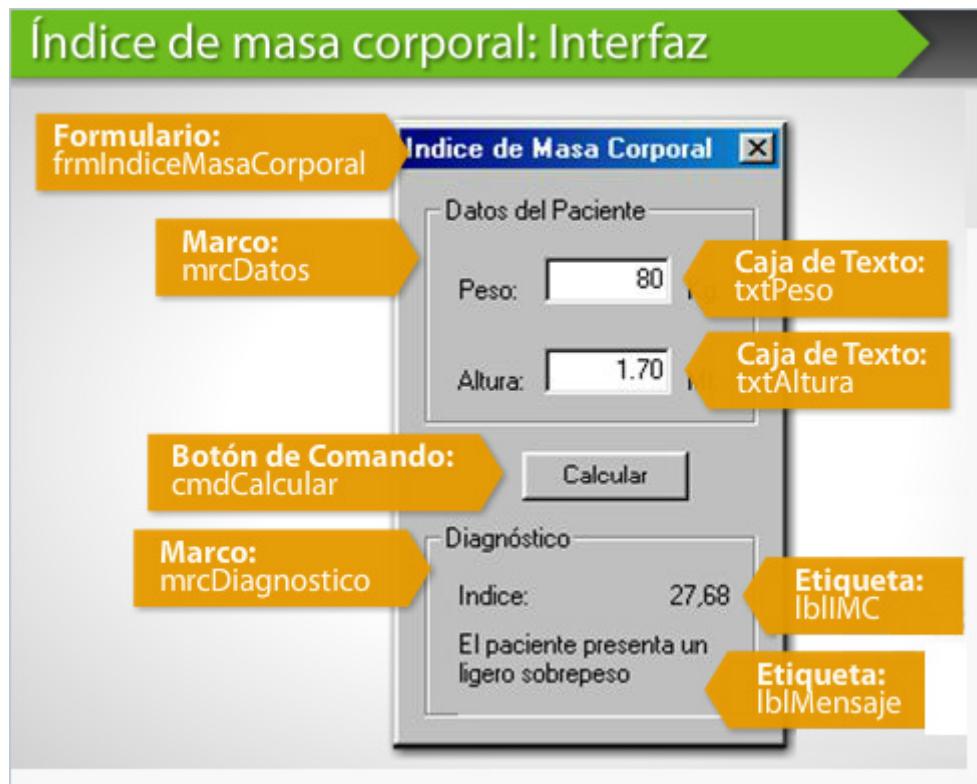
En el instituto “Bello Corpo”, se le solicita un programa que calcule el índice de masa corporal de los pacientes. Para realizar el cálculo, el nutricionista a cargo, deberá ingresar el peso y la altura del paciente y el programa mostrará el IMC (índice de Masa Corporal) y un diagnóstico orientativo.

Si el diagnóstico que se espera que se muestre al paciente fuera solo aquel que da cuenta de que tiene

sobrepeso, entonces, lo que habría que programar es que, cuando el índice de masa corporal supere el valor 25, se muestre el mensaje: "El paciente presenta un ligero sobrepeso". A esta situación la podríamos solucionar utilizando una estructura alternativa simple, tal como lo indicamos a continuación.

Seguramente, se está imaginando que el programa requiere una interfaz gráfica que cuenta mínimamente con:

- Dos cajas de texto, una para el ingreso de la altura y otro para el ingreso del peso
- Una etiqueta para mostrar el resultado del cálculo y otra para el mensaje
- Un botón de comando que dispara el procedimiento.



Realizaremos la definición de la misma:

PSEUDO: Pseudo Interfaz IMC

```
Formulario frmCalculoDeIMC
  Marco mrcDatos
    CajaDeTexto txtPeso
    CajaDeTexto txtAltura
  Fin Marco
  Marco mrcDiagnostico
    Etiqueta lblIMC
    Etiqueta lblMensaje
  Fin Marco
  BotonComando cmdCalcular
Fin Formulario
```

El pseudocódigo correspondiente al procedimiento representado en el diagrama es el siguiente:

PSEUDO: Pseudo del procedimiento de la interfaz IMC

```
Procedimiento cmdCalcular_Click
    lblIMC = txtPeso / (txtAltura * txtAltura)
    Si (lblIMC > 25) Entonces
        lblMensaje = "El paciente presenta un ligero sobrepeso"
    Fin Si
Fin Procedimiento
```

El programa completo debe incluir todos los elementos señalados anteriormente de manera correcta, como sigue:

PSEUDO: Pseudo de una estructura alternativa simple

```
Programa CalculoDeIMC
    'Definición del Formulario
    Formulario frmCalculoDeIMC
        Marco mrcDatos
            CajaDeTexto txtPeso
            CajaDeTexto txtAltura
        Fin Marco
        Marco mrcDiagnostico
            Etiqueta lblIMC
            Etiqueta lblMensaje
        Fin Marco
        BotonComando cmdCalcular
    Fin Formulario
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdCalcular_Click
        lblIMC = txtPeso / (txtAltura * txtAltura)
        Si (lblIMC > 25) Entonces
            lblMensaje = "El paciente presenta un ligero sobrepeso"
        Fin Si
    Fin Procedimiento
Fin Programa
```

Volvamos al caso de las instrucciones para realizar una llamada telefónica. Seguramente este algoritmo se puede realizar teniendo en cuenta más posibilidades. Por ejemplo, si la llamada es atendida, puede suceder que la persona con la que se desea hablar esté o no esté. Por consiguiente, en un caso se podrá establecer el diálogo con la misma, mientras que en el otro caso, se podrá dejar un mensaje.

Veamos los pasos:

Descolgar el auricular.

Esperar la señal de línea.

Marcar el número.

Esperar a ser atendido

Preguntar por la persona con la que se quiere hablar

Si está la persona con la que se desea hablar **Entonces**

Conversar con ella

Si No

Dejar mensaje

Colgar Auricular.

En el pseudocódigo, la estructura alternativa representada en el diagrama, deberá respetar el siguiente modelo:

PSEUDO: Pseudo SI NO

```
Si (condición) Entonces
    Instrucción 1
    ...
    Instrucción n
Si No
    Instrucción 1
    ...
    Instrucción n
Fin Si
```

El pseudocódigo será el siguiente:

PSEUDO: Pseudo llamada telefónica 2

```
Procedimiento LlamadaTelefónica
    Esperar la señal de línea
    Marcar el número
    Esperar a ser atendido
    Preguntar por la persona con la que se quiere hablar
    Si (está la persona con la que se desea hablar) Entonces
        Conversar
    Si No
        Dejar mensaje
    Fin Si
    Colgar Auricular
Fin Procedimiento
```

1.2 Estructura alternativa doble

Si el diagnóstico que se espera que se muestre al paciente en "Bello Corpo" fuera no solo aquel que da cuenta de que tiene sobrepeso, sino también aquel que le avisa al paciente que está en su peso normal, entonces, lo que habría que programar es que, cuando el índice de masa corporal supere el valor 25, se muestre el mensaje: "El paciente presenta un ligero sobrepeso" y que, en caso contrario, el mensaje sea: "El peso del paciente es normal". A esta situación la podríamos solucionar utilizando una estructura alternativa doble, tal como lo indicamos a continuación.

Ahora ya podemos armar el pseudocódigo del programa completo que resuelve el problema de este ejemplo:

PSEUDO: Pseudo de cálculo de IMC con una estructura alternativa doble

```
Programa CalculoDeIMC
    'Definición del Formulario
    Formulario frmCalculoDeIMC
        Marco mrcDatos
            CajaDeTexto txtPeso
            CajaDeTexto txtAltura
        Fin Marco
        Marco mrcDiagnostico
            Etiqueta lblIMC
            Etiqueta lblMensaje
        Fin Marco
        BotonComando cmdCalcular
```

```

Fin Formulario
'Desarrollo de los Procedimientos de Evento
Procedimiento cmdCalcular_Click
    lblIMC = txtPeso / (txtAltura * txtAltura)
    Si (lblIMC > 25) Entonces
        lblMensaje = "El paciente presenta un ligero sobrepeso"
    Si No
        lblMensaje = "El peso del paciente es normal"
    Fin Si
Fin Procedimiento
Fin Programa

```

1.3 Estructuras Anidadas

Hemos mencionado anteriormente que un algoritmo debe tener en cuenta la mayor cantidad de situaciones que se pueden presentar para disminuir los márgenes de error. Si usted es observador habrá notado que en el primer instructivo de la llamada telefónica se tienen en cuenta situaciones que en el segundo instructivo no. Por consiguiente, haremos a continuación un breve listado de los pasos a seguir para realizar una llamada telefónica que contemple todas las consideraciones posibles:

Descolgar el auricular

Esperar la señal de línea

Marcar el número

Si hay señal de llamada **Entonces**

Esperar a ser atendido

Si se es atendido **Entonces**

Preguntar por la persona con la que se quiere hablar

Si está la persona con la que se desea hablar **Entonces**

Conversar con ella

Si no

Dejar mensaje

Colgar Auricular

Observe que existen acciones que se realizan cuando se cumple la condición y otras que se realizan cuando no. Pero, cuando se hace válida la primera alternativa, es necesario validar otra condición. Quiere decir que existe una alternativa dentro de otra.

Esto es lo que se conoce en programación como Anidamiento de Estructuras o Estructuras Anidadas. Para entenderlo mejor, imagine que va por un camino y de pronto se divide en dos, usted opta por uno (no puede andar por ambos caminos simultáneamente). Después de un tramo, el camino elegido se vuelve a dividir y es necesario volver a tomar una decisión. Sucesivamente, los caminos pueden dividirse tantas veces como destinos existan.

En pseudocódigo, una estructura alternativa anidada debe respetar la siguiente sintaxis:

PSEUDO: Pseudo de una estructura alternativa anidada

Si (condición) **Entonces**

 Instrucciones

 ...

```

Si (condición) Entonces
    Instrucciones
    ...
Si No
    Instrucciones
    ...
Fin Si
Si No
    Instrucciones
    ...
Si (condición) Entonces
    Instrucciones
    ...
Si No
    Instrucciones
    ...
Fin Si
Fin Si

```

Tenga presente que el modelo es orientativo (los puntos suspensivos pueden ser más instrucciones o más alternativas). En un programa pueden existir muchas condiciones (aún más de las indicadas en el modelo)

El pseudocódigo de la situación presentada será el siguiente:

PSEUDO: Pseudo ejemplo llamada telefónica

```

Procedimiento RealizarLlamadaTelefónica
    Descolgar el auricular
    Esperar la señal de línea
    Marcar el número.
    Si (hay señal de llamada) Entonces
        Esperar a ser atendido
        Si (atienden) Entonces
            Preguntar por la persona con la que se quiere hablar
            Si (está la persona con la que se desea hablar) Entonces
                Conversar
            Si No
                Dejar mensaje
            Fin Si
        Fin Si
        Colgar Auricular
    Fin Procedimiento

```

Volviendo a la situación de "Bello Corpo", tenemos que el diagnóstico que se espera que se muestre al paciente no tiene ni uno ni dos mensajes posibles, sino que tiene tres:

- Cuando el índice de masa corporal supere el valor 25 se mostrará el mensaje “El paciente presenta un ligero sobrepeso”,
- En caso que el índice sea menor a 20 el mensaje será: “El paciente tiene un peso inferior a lo normal”
- y en caso contrario el mensaje: “El peso del paciente es normal”.

El pseudocódigo del programa que resuelve esta situación es el siguiente:

PSEUDO: Pseudo de cálculo de IMC con estructuras anidadas

```

Programa frmCalculoDeIMC
    'Definición del Formulario
    Formulario frmCalculoDeIMC
        Marco mrcDatos
            CajaDeTexto txtPeso
            CajaDeTexto txtAltura
        Fin Marco
        Marco mrcDiagnostico
            Etiqueta lblIMC
            Etiqueta lblMensaje
        Fin Marco
        BotonComando cmdCalcular
    Fin Formulario
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdCalcular_Click
        lblIMC = txtPeso / (txtAltura * txtAltura)
        Si (lblIMC > 25) Entonces
            lblMensaje = "El paciente presenta un ligero sobrepeso"
        Si No
            Si (lblIMC < 20) Entonces
                lblMensaje = "El paciente tiene un peso inferior a lo normal"
            Si No
                lblMensaje = "El peso del paciente es normal"
            Fin Si
        Fin Si
    Fin Procedimiento
Fin Programa

```

1.4 Estructuras Alternativas Independientes No Anidadas

Siempre que se utilicen estructuras alternativas se debe tener en cuenta que se pueden presentar casos de lo más variados, como por ejemplo:

- Que se ejecuten instrucciones cuando se cumple una condición y cuando no se cumple que no se ejecute nada. Esto significa que son alternativas que carecen de un: "si no".

PSEUDO: Pseudo SI..NO

```

Si (condición) Entonces
    ...
    ...
Fin Si

```

- Que se ejecuten instrucciones cuando la condición se cumple y se ejecuten otras instrucciones diferentes cuando la condición no se cumple. Esto significa que son alternativas que tienen instrucciones para el "si" y para el "si no".

PSEUDO: Pseudo Si...Sino

```

Si (condición) Entonces
    ...
    ...
Si No
    ...
    ...
Fin Si

```

- Que tanto cuando se cumple o cuando no se cumple una condición, pueda haber otra alternativa que, a su vez, puede presentar cualquiera de estas situaciones. Esto significa que los anidamientos de alternativas se pueden ver en el "si", en el "si no", o en ambas partes de la estructura.

PSEUDO: Pseudo Anidadas

```

Si (condición) Entonces
  Si (condición) Entonces
    ...
    ...
    Si No
    ...
    Fin Si
    ...
Si No
    ...
    Si (condición) Entonces
    ...
    Fin Si
    ...
Fin Si
```

Existen muchos casos donde en un programa es necesario utilizar alternativas que son absolutamente independientes unas de otras. En estos programas las alternativas no están anidadas.

En el pseudocódigo de un programa, esta situación queda representada del siguiente modo:

PSEUDO: Pseudo de alternativas no Anidadadas

```

Si (condición) Entonces
  Instrucciones...
  ...
Si No
  Instrucciones...
  ...
Fin Si
  ...
  ...
Si (condición) Entonces
  Instrucciones...
  ...
Si No
  Instrucciones...
  ...
Fin Si
```

Este tipo de estructuras son necesarias, por ejemplo, en el caso de una inmobiliaria que necesita un programa que le permita obtener rápidamente el precio de los terrenos que tiene para ofertar. Para ellos es necesario que el usuario del mismo ingrese los siguientes datos: largo, ancho y precio por metro cuadrado. Una vez obtenido el precio se deberá considerar lo siguiente:

- Si el terreno tiene más de 400 m² se hace un descuento del 10%.
- En caso que el valor del terreno supere los \$25.000 se hará un descuento del 5%.

Tenga presente que se pueden presentar las siguientes situaciones:

- Que el terreno tenga una superficie mayor a 400m² y su valor supere los \$25.000.

- Que el terreno tenga una superficie mayor a $400m^2$ y su valor no supere los \$25.000.
- Que el terreno no tenga una superficie mayor a $400m^2$ y su valor supere los \$25.000.
- Que el terreno no tenga una superficie mayor a $400m^2$ y su valor no supere los \$25.000.

Cuando se muestre la información al usuario, debe detallarse cada uno de los descuentos que se haya realizado.

Considere que se utilizará la siguiente interfaz:

Medidas del Terreno		Precio del Terreno	
Largo:	20 Mts.	Precio sin Descuento:	9.000,00
Ancho:	30 Mts.	Descuento por Superficie:	900,00
Precio del Metro Cuadrado		Descuento por Valor del Terreno	0,00
\$	15	Precio a Pagar:	8.100,00

"Interfaz precio de terrenos" | *Elaboración propia*

La cual tendrá la siguiente definición:

```
PSEUDO: Pseudo Definición interfaz inmobiliaria
Formulario frmPrecioTerrenos
  Marco mrcMedidas
    CajaDeTexto txtLargo
    CajaDeTexto txtAncho
  Fin Marco
  Marco mrcPrecioMetro
    CajaDeTexto txtPrecio
  Fin Marco
  Marco mrcPrecioTerreno
    Etiqueta lblPrecioSinDescuento
    Etiqueta lblDescuentoSuperficie
    Etiqueta lblDescuentoPrecio
    Etiqueta lblPrecioFinal
  Fin Marco
  BotonComando cmdCalcular
Fin Formulario
```

Este es un caso donde conviene utilizar alternativas independientes.

Pseudocódigo del programa completo:

PSEUDO: Pseudo de programa completo de alternativas independientes del ejemplo inmobiliaria

```
Formulario frmPrecioTerrenos
    Marco mrcMedidas
        CajaDeTexto txtLargo
        CajaDeTexto txtAncho
    Fin Marco
    Marco mrcPrecioMetro
        CajaDeTexto txtPrecio
    Fin Marco
    Marco mrcPrecioTerreno
        Etiqueta lblPrecioSinDescuento
        Etiqueta lblDescuentoSuperficie
        Etiqueta lblDescuentoPrecio
        Etiqueta lblPrecioFinal
    Fin Marco
    BotonComando cmdCalcular
Fin Formulario

Clase frmPrecioTerrenos

'Desarrollo de los Procedimientos de Evento
Procedimiento cmdCalcular:Click
    variable Superficie = 0 tipo numerica
    variable Precio = 0 tipo numerica
    variable DescuentoPrecio = 0 tipo numerica
    variable DescuentoSuperficie = 0 tipo numerica
    Superficie = txtLargo * txtAncho
    Precio = Superficie * txtPrecio
    lblPrecioSinDescuento = Precio
    Si (Superficie > 400 ) Entonces
        DescuentoSuperficie = Precio * 10 / 100
    Fin Si
    lblDescuentoSuperficie = DescuentoSuperficie
    Si (Precio > 25.000) Entonces
        DescuentoPrecio = Precio * 5 / 100
    Fin Si
    lblDescuentoPrecio = DescuentoPrecio
    lblPrecioFinal = Precio - (DescuentoSuperficie + DescuentoPrecio)
Fin Procedimiento
Fin Clase
```

2. Estructuras alternativas compuestas: SEGÚN SEA

Existen situaciones en la programación donde la estructura alternativa vista hasta el momento (simple) no es suficiente para la resolución de los problemas. Y en caso que resulte suficiente puede suceder que no sea la solución más adecuada.

Consideremos un problema: "Un usuario ingresa el número de un mes en una interfaz gráfica y el programa deberá mostrar el nombre del mes".



¿Qué tendremos que tener en cuenta? Seguramente que, dependiendo del número que se ingrese, se mostrará el nombre del mes.

¿El siguiente algoritmo soluciona el problema?:

PSEUDO: Pseudo Solución complicada de programar (número de mes)

```

Procedimiento cmdMostrarMes:Click
  Si txtNúmero = 1 Entonces
    LblNombre = "Enero"
  Si No
    Si txtNúmero = 2 Entonces
      LblNombre = "Febrero"
    Si No
      Si txtNúmero = 3 Entonces
        LblNombre = "Marzo"
      Si No
        Si txtNúmero = 4 Entonces
          LblNombre = "Abril"
        Si No
          Si txtNúmero = 5 Entonces
            LblNombre = "Mayo"
          Si No
            Si txtNúmero = 6 Entonces
              LblNombre = "Junio"
            Si No
              ...
            Fin Si
          Fin Si
        Fin Si
      Fin Si
    Fin Si
  Fin Si
Fin Procedimiento

```

Definitivamente, este algoritmo resuelve la situación. Pero es un poco complicado programarlo, considerando que tiene muchos "si" anidados. Además, el programa, en el momento de ejecución, debe evaluar muchos de ellos cuando el usuario ingrese un valor alto. Por ejemplo, para mostrar el nombre de mes "Agosto", se hacen ocho validaciones.

Una manera más fácil de programar esto es trabajando con las alternativas en forma independiente. Por ejemplo:

PSEUDO: Pseudo Solución lenta: utilizando alternativas independientes (meses)

```

Procedimiento cmdMostrarMes:Click
  Si txtNúmero = 1 Entonces
    LblNombre = "Enero"
  Fin Si
  Si txtNúmero = 2 Entonces
    LblNombre = "Febrero"
  Fin Si
  Si txtNúmero = 3 Entonces
    LblNombre = "Marzo"
  Fin Si
  Si txtNúmero = 4 Entonces
    LblNombre = "Abril"
  Fin Si
  Si txtNúmero = 5 Entonces
    LblNombre = "Mayo"
  Fin Si
  Si txtNúmero = 6 Entonces
    LblNombre = "Junio"
  Fin Si
  Si ...
    ...
    ...
    ...
  Si txtNúmero = 12 Entonces
    LblNombre = "Diciembre"
  Fin Si
Fin Procedimiento

```

Este algoritmo es más fácil de programar, pues no tiene tantas estructuras anidadas. No obstante es largo y la ejecución es lenta, ya que si el usuario ingresa el número 2, por ejemplo, la segunda validación se hace verdadera y muestra el resultado esperado, pero sigue ejecutando las líneas siguientes, con lo que realizará las doce validaciones por más que ya no sea necesario.

Para resolver una situación como esta es conveniente utilizar una estructura alternativa que se denomina “Según Sea”. Esta estructura permite validar el contenido de una variable, y dependiendo del contenido se ejecutará una línea del programa. Es más rápido y fácil de programar, por consiguiente resulta mucho más eficiente que las dos soluciones anteriores.

Imagine que está en el pasillo de un hotel, parado en un punto donde puede divisar la puerta de todas las habitaciones del piso, y usted tiene la llave de solo una de ellas. Seguramente se dirigirá a la puerta que le corresponde y no intentará abrir las demás. Esta estructura de programación actúa de un modo similar.

En los pseudocódigos, la estructura alternativa “Según Sea” puede implementarse de acuerdo a los siguientes modelos:

MODELO 1

```
Según Sea (Expresión a evaluar)
    Cuando sea valor 1
        Grupo de Instrucciones 1
    Cuando sea valor 2
        Grupo de Instrucciones 2
    Cuando sea valor 3
        Grupo de Instrucciones 3
    ...
    ...
    Cuando sea valor N
        Grupo de Instrucciones N
    En cualquier otro caso
        Grupo de Instrucciones ...
Fin según sea
```

MODELO 2

```
Según Sea
    Cuando condición 1
        Grupo de Instrucciones 1
    Cuando condición 2
        Grupo de Instrucciones 2
    Cuando condición 3
        Grupo de Instrucciones 3
    ...
    ...
    Cuando condición N
        Grupo de Instrucciones N
    En cualquier otro caso
        Grupo de Instrucciones ...
Fin según sea
```

Siguiendo este modelo, el pseudocódigo que resuelve el ejemplo planteado es:

PSEUDO: Pseudo completo del ejemplo meses

```
Procedimiento cmdMostrarMes_Click
    Segun Sea txtNúmero Hacer
        Cuando Sea 1
            lblNombre = "Enero"
        Cuando Sea 2
            lblNombre = "Febrero"
        Cuando Sea 3
            lblNombre = "Marzo"
        Cuando Sea 4
            lblNombre = "Abril"
        Cuando Sea 5
            lblNombre = "Mayo"
        Cuando Sea 6
            lblNombre = "Junio"
        Cuando Sea 7
            lblNombre = "Julio"
        Cuando Sea 8
            lblNombre = "Agosto"
        Cuando Sea 9
            lblNombre = "Septiembre"
        Cuando Sea 10
            lblNombre = "Octubre"
        Cuando Sea 11
            lblNombre = "Noviembre"
        Cuando Sea 12
            lblNombre = "Diciembre"
    Fin Segun Sea
Fin Procedimiento
```

En este tipo de estructura alternativa el programa no ejecuta todas las validaciones sino que, directamente, se

dirige a la que corresponde. Es casi como el proceso mental que realizamos muchos de nosotros cuando consideramos el número de un mes. Si vemos escrita la fecha 12/06/2012, directamente traducimos el número 6 como "Junio"; no vamos recorriendo secuencialmente desde el mes 1 (correspondiente a Enero) hasta llegar al 6 (Junio). Acá sucede lo mismo. Igual que cuando tengo la llave de una habitación que está en un piso de un hotel: nos dirijimos directamente a la puerta que corresponde sin pasar por las otras.

Habrá observado que en los dos modelos de Pseudocódigo para el uso de esta estructura se incluye, al final de todos los "Cuando", una opción más denominada "En cualquier otro caso". Esta condición está presente solo en los casos que el programador lo considera conveniente y su funcionamiento es complementario a todas las condiciones planteadas en los "Cuando". Esto quiere decir que en el caso de que ninguno de los "Cuando" se haya verificado, el flujo del programa ingresará por esta opción.

Veámoslo en el siguiente ejemplo:

Supongamos que nuestro programa deberá imprimir la Categoría de IVA correspondiente según el código de cada cliente. Una posible solución sería la siguiente (solo detallaremos el núcleo del pseudocódigo):

PSEUDO: Pseudo Segun Sea con ejemplo de Categoría de IVA

```
...
Segun Sea Categoría Hacer
    Cuando Sea "I"
        Imprimir "Responsable Inscripto"

    Cuando Sea "N"
        Imprimir "Responsable No Inscripto"

    Cuando Sea "M"
        Imprimir "Responsable Monotributista"

    'Estas dos líneas tratan cualquier otro código no contemplado inicialmente
    En Cualquier Otro Caso
        Imprimir "Categoría Incorrecta"
Fin Segun Sea
...
```

La principal ventaja de utilizar este tipo de estructura en vez de estructuras Si - Si No - Fin Si "anidadas" consiste en que los niveles de tabulación del pseudocódigo son mucho menores (no se hace tan profundo hacia la derecha) y, por ende, la legibilidad del código es sensiblemente mayor.

Por otra parte, al cumplirse una condición del Según Sea y ejecutarse todas las instrucciones que estén asociadas a la misma, el programa continúa su ejecución en la línea siguiente al Fin Según Sea.

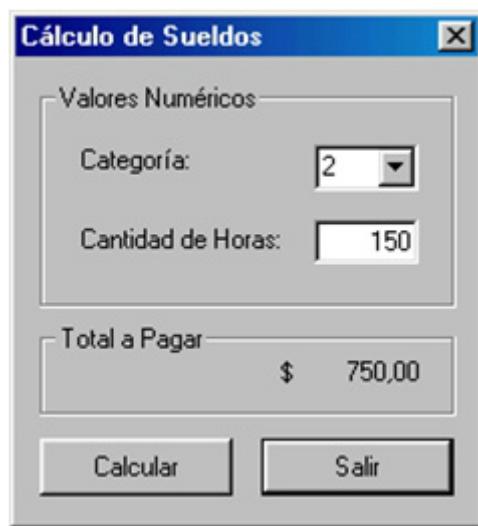
Estos elementos simplifican la lógica del programa y facilitan su seguimiento y análisis por parte del programador.

Como ejemplo, utilizaremos esta estructura para determinar la lógica a seguir en la solución de la siguiente situación:

La empresa Venturi le solicita a usted la confección y programación de una interfaz gráfica que permita el cálculo del importe a pagar a los empleados por sus horas de trabajo. Debe tener en cuenta que la empresa tiene diferentes tarifas de acuerdo a la categoría del empleado que pasamos a detallar:

- Categoría 1: \$3 por hora
- Categoría 2: \$5 por hora

- Categoría 3: \$6 por hora
- Categoría 4: \$7 por hora



"Interfaz cálculo de sueldos (SEGÚN SEA)" | Elaboración propia

La solución es:

PSEUDO: Pseudo del Ejemplo Venturi de estructura SEGÚN SEA

```

Programa Venturi
'Definición del formulario
Formulario frmCálculoSueldos
Marco mrcDatos
    ListaDesplegable lstCategoría
    CajaDeTexto txtHoras
Fin Marco
Marco mrcResultados
    Etiqueta lblTotal
Fin Marco
BotonComando cmdCalcular
BotonComando cmdSalir
Fin Formulario
'Desarrollo de los Procedimientos de Evento que calcula total a pagar
Procedimiento cmdCalcular_Click
    'Variable numérica que calcula total a pagar
    variable Total tipo numerica
    'Alternativa que valida el valor de la categoría. Calcula el total a pagar de acuerdo a la
    categoría'
        Segun Sea lstcategoría Hacer
            Cuando Sea: 1
                Total = 3 * txtHoras
            Cuando Sea: 2
                Total = 5 * txtHoras
            Cuando Sea: 3
                Total = 6 * txtHoras
            Cuando Sea: 4
                Total = 7 * txtHoras

```

```

Fin Segun Sea
'Asigna el valor de la variable Total a la etiqueta lblTotal
lblTotal = Total
Fin Procedimiento
'Cierra la interfaz. Está asociado al botón cmdSalir
Procedimiento cmdSalir_Click
    Cerrar frmCálculoSueldos
Fin Procedimiento
Fin Clase

```

Si volvemos a nuestra Situación profesional podremos observar que, de las dos estructuras alternativas que hemos descriptos, optaremos por la estructura Si. Quedando planteado de la siguiente manera:

PSEUDO: Pseudo Solución a situación de Bello Corpo

```

'Validación del índice para el diagnóstico
Si (IMC <= 22) Entonces
    lblDiagnóstico = "El paciente presenta menos del peso ideal"
Si No
    Si (IMC <= 28) Entonces
        lblDiagnóstico = "El paciente está en su peso ideal"
    Si No
        Si (IMC <= 35) Entonces
            lblDiagnóstico = "El paciente presenta un ligero sobrepeso"
        Si No
            lblDiagnóstico = "El paciente está excedido de peso"
    Fin Si
Fin Si
Fin Si

```

Si vemos con atención el pseudocódigo, podrá advertir que, además de la utilización de la estructura Si, nos encontramos con estructuras simples, estructuras dobles y con anidamiento de estructuras.



¿Te animás a medir cuánto aprendiste?

1. Indique la opción correcta

La declaración de la estructura Si - Fin Si se escribe de la siguiente manera:

```
1  Si (condición) Entonces
2    Instrucción 1
3    ...
4    Instrucción n
5  Fin Si
```

- Verdadero
- Falso

2. Indique la opción correcta

La siguiente es una estructura anidada válida: Si (Si (condición) Instrucciones Fin Si) Fin si

- Verdadero
- Falso

3. Indique la opción correcta

Cuando tenemos que analizar una variable entre 7 alternativas, ¿Qué estructura alternativa nos conviene utilizar?

- Si - Fin Si
- Segun Sea - Fin Segun Sea

4. Indique la opción correcta

En una estructura Segun Sea, ¿Cuantas alternativas se pueden tener?

- Solo 10.
- Solo 2.
- Ninguna de las anteriores

5. Indique la opción correcta

El siguiente anidamiento de estructuras, ¿es válido o inválido?

```
1 Segun Sea x
2   Caso 1
3     Si (condición) Entonces
4       .....
5       Fin Si
6   Caso 2
7     Si (condición) Entonces
8       .....
9       Fin Si
10  Caso 3
11    Si (condición) Entonces
12      .....
13      Fin Si
14  Fin Segun Sea
```

- Válido
- Inválido

Respuestas de la Autoevaluación

1. Indique la opción correcta

La declaración de la estructura Si - Fin Si se escribe de la siguiente manera:

Verdadero

Falso

2. Indique la opción correcta

La siguiente es una estructura anidada válida: Si (Si (condición) Instrucciones Fin Si) Fin si

Verdadero

Falso

3. Indique la opción correcta

Cuando tenemos que analizar una variable entre 7 alternativas, ¿Qué estructura alternativa nos conviene utilizar?

Si - Fin Si

Segun Sea - Fin Segun Sea

4. Indique la opción correcta

En una estructura Segun Sea, ¿Cuantas alternativas se pueden tener?

Solo 10.

Solo 2.

Ninguna de las anteriores

5. Indique la opción correcta

El siguiente anidamiento de estructuras, ¿es válido o inválido?

Válido

Inválido

SP2 / H2: Diagrama de flujo

Concepto

Un diagrama de flujo es la manera de representar visualmente el flujo de datos a través de sistemas informáticos. Los diagramas de flujo describen qué operaciones y, en qué secuencia, se requieren para solucionar un problema dado.

Los diagramas de flujo se dibujan antes de la programación en un lenguaje de programación que se elija. Facilitan la comunicación entre los programadores y los encargados de las políticas de negocio. Estos diagramas de flujo desempeñan un papel vital en la programación de un problema y facilitan la comprensión de problemas complicados y, sobre todo, muy largos.

Una vez que se dibuja el diagrama de flujo, llega a ser fácil escribir el programa. Nuestra experiencia indica que los diagramas de flujo son de mucha utilidad al momento de explicar el programa a otros. Por lo tanto, está correcto decir que un diagrama de flujo es una necesidad para la documentación de un programa complejo.

Componentes

A continuación, detallaremos los componentes gráficos que utilizaremos en esta materia:

	Inicio o fin del algoritmo
	Pasos, procesos o líneas de instrucción de programa de cómputo
	Operaciones de entrada y salida
	Toma de decisiones y Ramificación
	Conejero para unir el flujo a otra parte del diagrama
	Conejero de página
	Líneas de flujo
	Envía datos a la impresora
	Llamada a Procedimiento

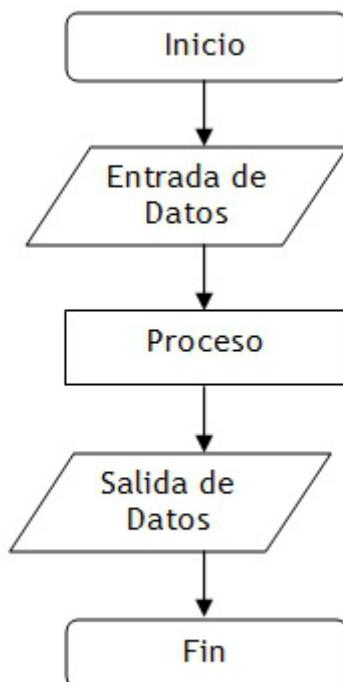
Reglas para la definición de diagramas

- Los Diagramas de flujo deben escribirse de arriba hacia abajo, y/o de izquierda a derecha.
- Los símbolos se unen con líneas, las cuales tienen en la punta una flecha que indica la dirección que fluye la información procesos. Se deben utilizar solamente líneas de flujo horizontal o verticales (nunca diagonales).
- Se debe evitar el cruce de líneas. Si se quiere separar el flujo del diagrama a un sitio distinto, se pudiera realizar utilizando los conectores. Se debe tener en cuenta que solo se van a utilizar conectores cuando sea estrictamente necesario.

- No deben quedar líneas de flujo sin conectar.
- Todo texto escrito dentro de un símbolo debe ser legible, preciso, evitando el uso de muchas palabras.
- Todos los símbolos pueden tener más de una línea de entrada, a excepción del símbolo final.
- Solo los símbolos de decisión pueden y deben tener más de una línea de flujo de salida.

Forma general de un Diagrama de Flujo

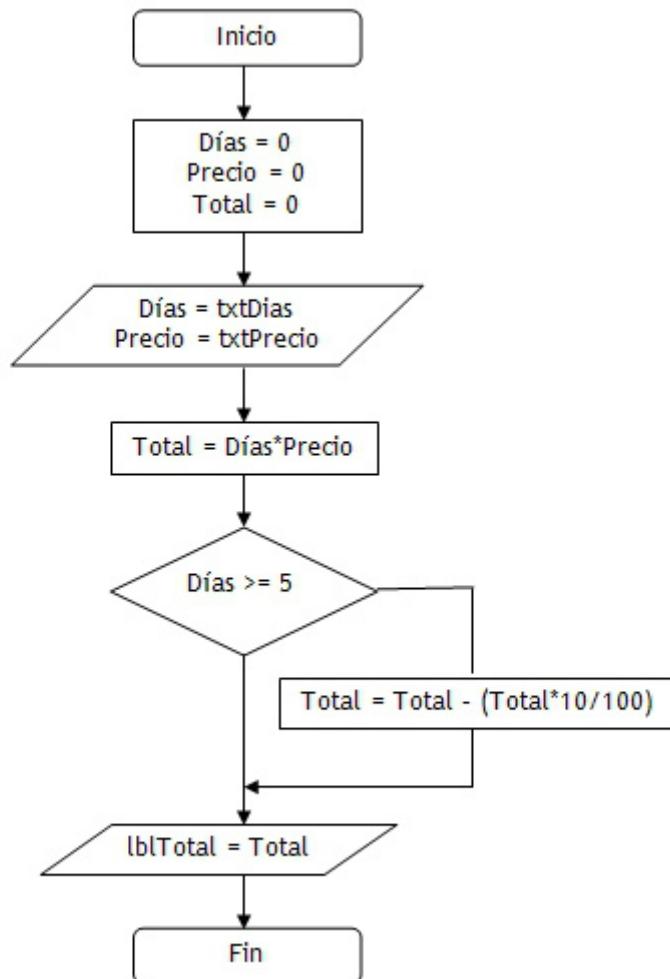
Al igual que en la estructura general de pseudocódigo, en los diagramas de flujos también se debe respetar una estructura como la siguiente:



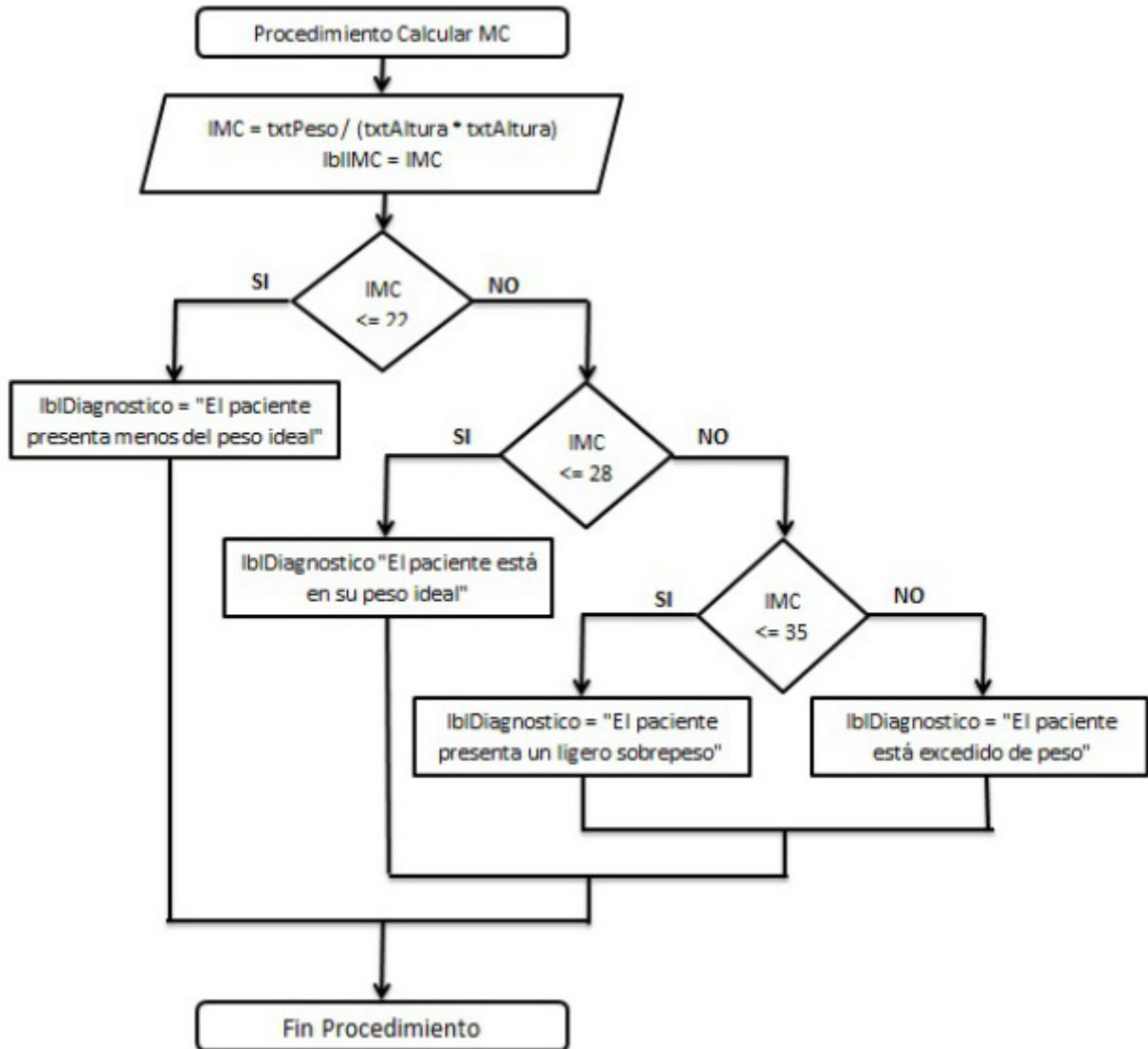
Vemos un ejemplo:

Un Hotel de la ciudad de Córdoba necesita un programa que resuelva la situación en donde el recepcionista ingresa la cantidad de días en que se alojará un pasajero y el precio por día de la habitación, y el sistema devuelva el total del hospedaje teniendo en cuenta que, si el pasajero se aloja 5 días o más, se le hará un descuento del 10% en su estadía.

El diagrama de flujo de esta situación quedará como se muestra a continuación:



Si nos hubieran solicitado en nuestra Situación profesional generar el diagrama de flujo del procedimiento Calcular, tendríamos el siguiente diagrama:





¿Estás listo para un desafío?

1. Indique la opción correcta

Un diagrama de flujos nos permite apreciar el uso de la memoria que hace un programa.

- Verdadero
- Falso

2. Indique la opción correcta

Cuando tenemos que graficar un flujo alternativo en un diagrama de flujos, utilizamos un rombo en cuyo interior tenemos una condición.

- Verdadero
- Falso

3. Indique la opción correcta

Un diagrama de flujos puede representar más de un programa.

- Verdadero
- Falso

4. Indique la opción correcta

Si al dibujar un diagrama necesitamos más de una hoja, utilizamos un símbolo de conexión para unir las mismas.

- Verdadero
- Falso

5. Indique la opción correcta

Los símbolos utilizados en un diagrama de flujos pueden o no unirse con líneas continuas.

- Verdadero
- Falso

Respuestas de la Autoevaluación

1. Indique la opción correcta

Un diagrama de flujos nos permite apreciar el uso de la memoria que hace un programa.

Verdadero

Falso

2. Indique la opción correcta

Cuando tenemos que graficar un flujo alternativo en un diagrama de flujos, utilizamos un rombo en cuyo interior tenemos una condición.

Verdadero

Falso

3. Indique la opción correcta

Un diagrama de flujos puede representar más de un programa.

Verdadero

Falso

4. Indique la opción correcta

Si al dibujar un diagrama necesitamos más de una hoja, utilizamos un símbolo de conexión para unir las mismas.

Verdadero

Falso

5. Indique la opción correcta

Los símbolos utilizados en un diagrama de flujos pueden o no unirse con líneas continuas.

Verdadero

Falso

SP2 / H3: Regla de estructuras y control de flujos

En el ámbito actual del desarrollo de sistemas, la manera en que se escribe y se organiza el código es una de las características más valiosas en las técnicas de programación. Que los algoritmos y los programas sean claros de entender y que el flujo lógico sea fácil de seguir, permiten una mejor lectura y un mejor mantenimiento.

En un programa estructurado, el flujo lógico se organiza utilizando las estructuras básicas de control:

1. Secuenciales
2. Alternativas
3. Repetitivas

Estas estructuras se denominan estructuras de control, porque son las que controlan el modo o flujo de ejecución del algoritmo. Su importancia es tal, que una vez que se entienda su estructura y funcionamiento, puede decirse que en esencia es todo lo que hay que saber respecto al control y flujo de los algoritmos.

Determinar cuándo usar cada una de estas estructuras, dependerá de la situación que se esté resolviendo. El nombre de cada estructura nos dice mucho sobre cuál es su función y uso.

1. Estructuras secuenciales. Las estructuras secuenciales son las más simples. En estas estructuras, cada acción o sentencia se ejecuta una detrás de la otra. El flujo del algoritmo es el mismo orden físico en las que se han puesto las secuencias del algoritmo.

2. Estructuras alternativas. Son las que vimos al comienzo de esta situación profesional. Como su nombre lo dice, este tipo de estructuras optan por una alternativa de sentencias a ejecutar dependiendo de si se cumple o no una determinada condición.

La condición que rige el control de flujo puede ser compuesta en base a la necesidad del problema, pero se deberá asegurar que el valor final de lo evaluado será un valor lógico, como verdadero o falso.

Las estructuras de control alternativas pueden ser: simples, dobles o múltiples.

3. Estructuras repetitivas. Son estructuras que cuentan con un ciclo que se repite mientras se cumpla una determinada condición, cuya característica es similar a la descripta en las estructuras alternativas. Más adelante las describiremos con más detalle.

En la situación que se nos planteó, hacemos uso de estructura alternativa, ya que se nos presentan situaciones caminos alternativos, como se observa a continuación:

PSEUDO: Estructura alternativa ejemplo de "Bello Corpo"

```
'Desarrollo de los Procedimientos
Procedimiento cmdCalcular_Click

    'Definición de una variable local
    variable IMC = 0 tipo numerica

    'Cálculo del índice de masa corporal
    IMC = txtPeso / (txtAltura * txtAltura)
    lblIMC = IMC

    'Validación del índice para el diagnóstico
    Si (IMC <= 22) Entonces
        lblDiagnostico = "El paciente presenta menos del peso ideal"
    Si No
```

```

Si (IMC <= 28) Entonces
    lblDiagnostico = "El paciente está en su peso ideal"
Si No
    Si (IMC <= 35) Entonces
        lblDiagnostico = "El paciente presenta un ligero sobrepeso"
    Si No
        lblDiagnostico = "El paciente está excedido de peso"
    Fin Si
Fin Si
Fin Si
Fin Procedimiento

```

Diagrama de Warnier

Los diagramas de Warnier son también conocidos como "construcción lógica de programas" o "construcción lógica de sistemas".

Es considerado como un método que ayuda al diseño de estructuras de programas identificando la salida y resultado del procedimiento. Trabaja hacia atrás para determinar los pasos y combinaciones de entrada necesarios para producirlos. Los sencillos métodos gráficos usados en los diagramas de Warnier hacen evidentes los niveles en un sistema y más claros los movimientos de los datos en dichos niveles.

Estos diagramas muestran los procesos y la secuencia en que se realizan. Cada proceso se define de una manera jerárquica, es decir, consta de conjuntos de subprocesos que lo definen. En cada nivel, el proceso se muestra en una llave que agrupa a sus componentes. Puesto que un proceso puede tener muchos subprocesos distintos, un diagrama de Warnier usa un conjunto de llaves para mostrar cada nivel del sistema.

Elementos básicos

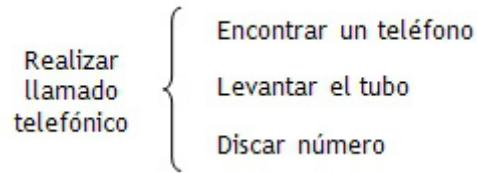
Para diseñar un diagrama de Warnier se necesita de los siguientes elementos:

- Conjuntos: {
- Subconjuntos: {
- Cardinalidad: (1, n)
- Condisionalidad: (0,1)
- Secuencia de acciones mutuamente excluyentes: +
- (+) Se utiliza para disyuntivas
- + Se utiliza en el caso de concurrencia de sentencias

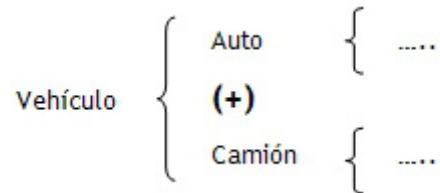
Representación de las estructuras de control de flujos

Representaremos a continuación, como se grafican las estructuras de control de flujo utilizando Diagramas de Warnier en cada uno de los tipos de estructuras:

Estructuras secuenciales. Las secuenciales son las estructuras más simples de un diagrama de Warnier. Dentro de un nivel de una jerarquía, las características listadas son presentadas en el orden en que ocurren.



Estructuras alternativas. Como dijimos, esta estructura representa una alternativa en base a una condición.



Estructuras repetitivas. Recordemos que una estructura repetitiva es una estructura en donde el mismo conjunto de acciones se repiten muchas veces.

La repetición es indicada colocando un par ordenado de números entre paréntesis debajo del conjunto repetitivo.



El par de números representa el mínimo y máximo número de veces que ocurre la repetición.



¡Vamos a comprobar cuánto aprendiste!

1. Indique la opción correcta

El flujo de datos en un programa se controlan con la estructura Hacer hasta

- Verdadero
- Falso

2. Indique la opción correcta

Los elementos básicos para diseñar un Diagrama de Warnier son las llaves ({}).

- Verdadero
- Falso

3. Indique la opción correcta

Los diagramas de Warnier muestran los procesos y la secuencia en que se realizan.

- Verdadero
- Falso

4. Indique la opción correcta

Las estructuras que ejecutan cada instrucción una detrás de otra se denominan:

- Estructura secuencial.
- Estructura alternativa.
- Estructura repetitiva.

5. Indique la opción correcta

Las estructuras que se repiten en un bucle hasta tanto cumplir con una condición se denominan:

- Estructura secuencial.
- Estructura alternativa.
- Estructura repetitiva.

Respuestas de la Autoevaluación

1. Indique la opción correcta

El flujo de datos en un programa se controlan con la estructura Hacer hasta

Verdadero

Falso

2. Indique la opción correcta

Los elementos básicos para diseñar un Diagrama de Warnier son las llaves ({}).

Verdadero

Falso

3. Indique la opción correcta

Los diagramas de Warnier muestran los procesos y la secuencia en que se realizan.

Verdadero

Falso

4. Indique la opción correcta

Las estructuras que ejecutan cada instrucción una detrás de otra se denominan:

Estructura secuencial.

Estructura alternativa.

Estructura repetitiva.

5. Indique la opción correcta

Las estructuras que se repiten en un bucle hasta tanto cumplir con una condición se denominan:

Estructura secuencial.

Estructura alternativa.

Estructura repetitiva.

SP2 / Ejercicio resuelto

El pseudocódigo completo del programa para resolver la situación profesional planteada

BELLO CORPO

La empresa "Bello Corpo", que se dedica a la estética corporal, tiene un grupo de profesionales que trabajan en las diferentes áreas: gimnasia, modelación, cosmética, nutrición, entre otras.

Los profesionales que trabajan en la sección de nutrición, antes de recomendar una dieta, deben realizar un diagnóstico del paciente. Para ello, entre otros indicadores, obtienen el índice de masa corporal.

Este índice es un valor que permite determinar si la persona está dentro del peso normal, o si está excedido, o si pesa menos de lo que se considera saludable.

El índice de masa corporal se calcula de la siguiente manera:

$$\text{Masa Corporal} = \text{Peso} / \text{Altura}^2$$

Este índice permite saber si el paciente tiene el peso adecuado para su contextura. A partir de este valor, el nutricionista, puede recomendar una dieta alimenticia adecuada.

Para facilitar el diagnóstico, se le ha solicitado a usted, que es colaborador del área de sistemas, que diseñe un programa que obtenga el índice de masa corporal y que diagnostique al paciente.

se detalla a continuación:

PSEUDO: Pseudo Bello Corpo

```
Programa BelloCorpo
    'Definición del Formulario
    Formulario frmCalculoDeIMC
        Marco mrcDatos
            CajaDeTexto txtPeso
            CajaDeTexto txtAltura
        Fin Marco
        Marco mrcDiagnostico
            Etiqueta lblIMC
            Etiqueta lblDiagnostico
        Fin Marco
        BotonComando cmdCalcular
        BotonComando cmdNuevaConsulta
        BotonComando cmdSalir
    Fin Formulario
    'Desarrollo de los Procedimientos. Procedimiento relacionado al botón cmdCalcular
    Procedimiento cmdCalcular_Click
        ' Definición de una variable local
        variable IMC tipo numerica
        ' Cálculo del índice de masa corporal
        IMC = txtPeso / (txtAltura * txtAltura)
        lblIMC = IMC
        ' Validación del índice para el diagnóstico
        Si (IMC <= 22) Entonces
            lblDiagnostico = "El paciente presenta menos del peso ideal"
        Si No
```

```

Si (IMC <= 28) Entonces
    lblDiagnostico = "El paciente está en su peso ideal"
Si No
    Si (IMC <= 35) Entonces
        lblDiagnostico = "El paciente presenta un ligero sobrepeso"
    Si No
        lblDiagnostico = "El paciente está excedido de peso"
    Fin Si
    Fin Si
Fin Si
Fin Procedimiento
Procedimiento cmdNuevaConsulta_Click
    ' Se le asigna 0 a las cajas de texto
    txtPeso = 0
    txtAltura = 0
    lblIMC = 0
    ' Se borra el contenido de la etiqueta
    lblDiagnostico = ""
    txtPeso.PonerFoco
Fin Procedimiento
Procedimiento cmdSalir_Click
    ' Se cierra el formulario y finaliza el programa
    Cerrar frmCalculoDeIMC
Fin Procedimiento
Fin Programa

```

SP2 / Ejercicios por resolver

Desarrolle las soluciones de los ejercicios planteados a continuación:

1. Realice un programa que calcule y muestre el precio de un terreno del cual se necesitan los siguientes datos: largo, ancho y precio por metro cuadrado. Si el terreno tiene más de 400m^2 se hace un descuento del 10%, si tiene más de 500m^2 el descuento es del 17% y si tiene más de 1000m^2 el descuento es de 25%.
2. La empresa de Cálculo Numérico "Calculín" le solicita a usted la confección y la posterior programación de una interfaz gráfica que permita el ingreso de dos datos numéricos. A continuación se mostrará un mensaje indicando si los datos son iguales, si el primero es menor al segundo o si el primero es mayor al segundo.
3. El Hotel Guamúchil realiza un descuento del 10% a los clientes que se hospedan más de cinco días, de 15% a los que se hospedan por más de diez días, y de 20% a los que se hospedan más de 15 días. Realice un programa que solicite al usuario el número de días que se hospedará el cliente, y el precio diario de la habitación y muestre, el subtotal a pagar, el descuento y el total a pagar. Tener en cuenta lo siguiente para realizar los cálculos:
 - Subtotal a pagar = Precio diario * Cantidad de días.
 - Descuento = Subtotal a pagar * Valor Descuento / 100.
 - Total a Pagar = Subtotal a pagar – Descuento.
4. Un instituto de enseñanza primaria le solicita a un analista una aplicación orientada a alumnos de cuarto grado que plantee problemas de geometría y aritmética.

De dicha aplicación, se le solicita a Ud., como miembro de un equipo de trabajo, que diseñe una interfaz que solicite al niño las medidas de la base y de la altura de un triángulo. El niño deberá ingresar, también, el área del triángulo (valor que deberá calcular). El objetivo del programa es que se evalúe el resultado que el niño ingresó, de tal modo que si el resultado es incorrecto el programa le indique el resultado real, y si el niño ingresó bien el resultado se lo debe incentivar con un mensaje de felicitación.

Deberá tener en cuenta que: Área de un Triángulo = $(\text{Base} * \text{Altura}) / 2$

También se le solicita que realice el algoritmo que resuelve dicha situación.



¿Te animás a medir cuánto aprendiste?

1. Indique la opción correcta

En un programa estructurado, el flujo lógico se organiza utilizando las estructuras Secuenciales, Alternativas y Repetitivas

- Verdadero
- Falso

2. Indique la opción correcta

En el diseño de un diagrama de flujos, las líneas de puntos representan el flujo de datos

- Verdadero
- Falso

3. Indique la opción correcta

Las estructuras alternativas se clasifican en Simple y Compuestas.

- Verdadero
- Falso

4. Indique la opción correcta

Que clase de operadores son utilizados para formar las condiciones en una estructura Si - Fin Si

- Operadores Matemáticos.
- Operadores Relacionales.
- Ninguna de las anteriores.

5. Indique la opción correcta

Las estructuras que optan por una alternativa de sentencias a ejecutar dependiendo de si se cumple o no una determinada condición, se denomina:

- Estructura secuencial.
- Estructura alternativa.
- Estructura repetitiva.

Respuestas de la Autoevaluación

1. Indique la opción correcta

En un programa estructurado, el flujo lógico se organiza utilizando las estructuras Secuenciales, Alternativas y Repetitivas

- Verdadero
- Falso

2. Indique la opción correcta

En el diseño de un diagrama de flujos, las líneas de puntos representan el flujo de datos

- Verdadero
- Falso

3. Indique la opción correcta

Las estructuras alternativas se clasifican en Simple y Compuestas.

- Verdadero
- Falso

4. Indique la opción correcta

Que clase de operadores son utilizados para formar las condiciones en una estructura Si - Fin Si

- Operadores Matemáticos.
- Operadores Relacionales.
- Ninguna de las anteriores.

5. Indique la opción correcta

Las estructuras que optan por una alternativa de sentencias a ejecutar dependiendo de si se cumple o no una determinada condición, se denomina:

- Estructura secuencial.
- Estructura alternativa.
- Estructura repetitiva.

Situación profesional 3: Operadores

Instituto San José

El instituto de enseñanza primaria San José le solicita a Ud., como su analista, una aplicación orientada a alumnos de cuarto grado que plantee problemas de geometría y aritmética.

De dicha aplicación, se le solicita a usted, como miembro de un equipo de trabajo, que: desarrolle el programa en pseudocódigo y que diseñe una interfaz donde el niño pueda ingresar las medidas de los tres lados de un triángulo. El niño deberá indicar si esas medidas corresponden a triángulos isósceles, equiláteros o escalenos.

Tenga en cuenta que:

- Isósceles tiene dos lados iguales.
- Equilátero tiene los tres lados iguales.
- Escaleno tiene los tres lados diferentes.

Es imprescindible que el programa verifique que las medidas correspondan a un triángulo. Para ello debe considerar que la suma de dos lados debe ser siempre mayor a la medida del lado restante. No se podrá evaluar al alumno si las medidas no corresponden a un triángulo.

El objetivo del programa es que se evalúe el resultado que el niño seleccionó, de tal modo que si el resultado es incorrecto el programa le indique el resultado real. Y si el niño ingresó bien el resultado se lo debe incentivar con un mensaje de felicitación.

SP3 / H1: Clasificación de los Operadores

En las situaciones profesionales anteriores hemos utilizado y nombrado algunos operadores como los matemáticos y los relacionales, y hemos hecho uso de ellos en la resolución de distintas situaciones planteadas.

En esta situación profesional veremos en detalle su uso y las consideraciones a tener en cuenta al momento de desarrollar expresiones que den soluciones a nuestros planteos.

Tipos de Operadores

Los programas que hasta el momento desarrollamos, constan de datos, instrucciones y expresiones. Las expresiones son una combinación de constantes, variables, paréntesis y símbolos de operaciones. Cada una de las expresiones, toma el valor resultante de aplicar las operaciones indicadas sobre las variables y constantes del problema que se analiza.

Dependiendo el tipo de datos que manipulan, las expresiones se clasifican en:

- Aritméticas
- Lógicas
- De cadena
- Relacionales

Las últimas mencionadas, ya las vimos en la situación profesional anterior, e hicimos uso de las mismas.

En base a esta clasificación de expresiones, obtendremos la clasificación de los operadores en las categorías que se enumeran a continuación:

Tipos de Operadores

PL1

• ARITMÉTICOS

• LÓGICOS

• DE CADENA

• RELACIONALES

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
^	Exponentiación
div	División entera
mod	Módulo (Resto)

* 3.1 Imágenes

Imágenes "Tipos de operadores"

Habrá situaciones en donde se nos presentará la necesidad de construir condiciones en donde hagamos uso de una combinación de operadores.

Si nos situamos en nuestra situación profesional, tendremos casos en donde analizaremos a 3 ángulos, para determinar que tipo de triángulos forman.

En caso de la expresión que controla si el triángulo es equilátero, se tendrá:

PSEUDO: Psedu que controla si el triángulo es equilátero

```
Si (a = b AND b = c) Entonces
.....
Si No
.....
Fin Si
```

Como vemos en esta expresión hicimos uso de operadores relacionales (=) y también de un operador lógico (AND), que veremos en detalle a continuación.

Operadores Aritméticos

Las expresiones aritméticas en un algoritmo, son similares a las operaciones aritméticas. Los operadores aritméticos los utilizaremos para armar la expresión dependiendo de la situación o problema que estemos analizando.

Los operadores aritméticos que veremos en esta materia son:

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
^	Exponenciación
div	División entera
mod	Módulo (Resto)

En algunos lenguajes de programación, no encontraremos los operadores **div** y **mod**, pero igualmente repasaremos su funcionalidad.

Reglas de Prioridad

Al igual que en las operaciones matemáticas, cuando tenemos más de dos operandos, necesitamos regirnos por reglas para determinar el orden en que se realizaran las operaciones.

El orden que usaremos para la resolución de operaciones aritméticas será:

1) Las operaciones que están entre paréntesis serán las primeras en ser evaluadas. En caso de que tengamos expresiones con paréntesis dentro de otros paréntesis, siempre se resolverá en primera instancia las expresiones internas.

2) Y en cuanto a las operaciones, se respetará el siguiente orden:

- Operador exponencial (^).
- Operadores * y /.

- Operadores **div** y **mod**.
- Operadores **+** y **-**.

Ejemplos:

Suma +

Usaremos la suma cuando tengamos la necesidad de obtener la suma de dos números considerados **operando**s, el valor que obtenemos de esta operación se conoce como **resultado**.

Tomando como referencia nuestra situación profesional, tendremos en un punto la necesidad de preguntarnos si la suma de dos ángulos es menor a un tercer ángulo. Aquí haremos uso del operador como se muestra a continuación:

PSEUDO: Pseudo Operadores Suma

```
Si (( b + c ) > a AND ( a + c ) > b AND ( a + b ) > c) Entonces
.....
Si No
.....
Fin Si
```

Siendo $a = \text{ángulo 1}$, $b = \text{ángulo 2}$ y $c = \text{ángulo 3}$ que son ingresados por interfaz por un alumno.

Resta -

Usaremos la resta para obtener la diferencia entre dos valores.

Ya que en nuestra situación profesional no necesitamos usar la resta, tomemos como ejemplo el siguiente:

Variable A, B y Resultado

$A = 5$

$B = 3$

$\text{Resultado} = A - B$

$\text{Resultado} = 2$

Multiplicación *

Usaremos la multiplicación para obtener el producto de dos valores.

Al igual que en la resta, en nuestra situación no tenemos la necesidad de usar la multiplicación, por lo que tomaremos el siguiente ejemplo:

Variable A, B y Resultado

$A = 5$

$B = 3$

$\text{Resultado} = A * B$

$\text{Resultado} = 15$

Exponenciación ^

Usaremos la exponenciación para obtener como resultado el valor de elevar un número a una potencia dada. Como en nuestra situación no tenemos la presencia de exponentiales, usaremos el siguiente ejemplo:

Variable A, B y Resultado

A = 2

B = 3

Resultado = A ^ B

Resultado = 8

División /, División Entera div y Módulo mod

Estos tres operadores son variantes de la división y, como mencionamos anteriormente, los dos últimos no están incluidos en todos los lenguajes.

Ninguno de estos operadores será necesario para la resolución de nuestra situación, por lo que veremos la aplicación de los mismos en el siguiente ejemplo:

10,5 / 3 = 3,5.....división real.

10,5 div 3 = 3.....división entera.

10,5 mod 3 = 1,5.....resto de la división.

Dependiendo de la situación que esté analizando, verá conveniente el uso de uno de estos operadores.

Operador de Cadenas

Cuando tenemos la necesidad de unir o concatenar distintos valores alfanuméricos, entonces hacemos uso del operador de cadenas.

El operador de cadena es:

Operador	Significado
&	Concatenación

Definiremos **concatenación** a la acción de unir expresiones alfanuméricas como si fueran partes de una misma cadena-

Dado que en nuestra situación no usaremos este operador, veamos los siguientes ejemplos:

Expresión Resultado

"Pseudo" & "Código" "PseudoCódigo"

"3" & "." & "1416" "3.1416"

En caso de que concatenemos valores almacenados en variables, podemos tener casos como el siguiente:

A = "Buenos"

B = "días."

si colocamos
imprimir A & B
se mostrará
"Buenos días."

Operadores Lógicos

Estos operadores los utilizaremos en expresiones en las que necesitamos obtener como resultado un valor verdadero o un valor falso. Estos operadores son tambien conocidos como Operadores Booleanos, en honor a George Boole, creador del álgebra de Boole.

Los operadores básicos que veremos son:

Operador	Significado
OR	Suma Lógica
AND	Producto Lógico
NOT	Negación

Veamos en detalle el significado de cada operador:

OR u O

Es un operador binario que afecta a dos operandos. La expresión en la que interviene será **verdadera(distinto de cero)** si cuaqueira de los operandos es **verdadero(distinto de cero)**, y produce **falso(cero)** sólo si ambos operadores son falsos.

AND o Y

Es un operador binario. La expresión que forma será **verdadera(distinto de cero)** solo si ambos operandos son **verdaderos(distintos de cero)**; en caso de que cualquier operando sea **falso(cero)**, la expresión dará como resultado **falso(cero)**.

NOT o NO

Es un operador unitario. Modifica la expresión en la que se encuentra, produciendo **falso(cero)** si su operando es **verdadera(distinto de cero)** y **verdadera(distinto de cero)** cuando el operando es **falso(cero)**.

Si nos situamos en nuestra situación profesional, tendremos el uso de los operadores lógicos en distintas situaciones.

Un ejemplo de esto es cuando verificamos si el triángulo que analizamos es equilátero, en donde todos sus ángulos deben ser iguales, por lo que formaremos las expresiones de la siguiente manera:

PSEUDO: Ejemplo análisis triángulo equilátero

```
Si ( a = b AND b = c ) Entonces  
  Si (optEquilatero = VERDADERO)
```

```

lblEvaluacion = "Felicitaciones...tu opción es la correcta"
Si No
    lblEvaluacion = "Las medidas corresponden a un triángulo Equilátero"
Fin Si
Si No
.....
Fin Si

```

Como vemos en este fragmento del código, se usa el operador **AND** para comprobar que se cumple tanto que $a = b$ y que $b = c$.

Tablas de Verdad

Una tabla de verdad es una herramienta que nos sirve para determinar los posibles valores de verdad de una expresión o proposición. La tabla de verdad está compuesta por una o más variables (por lo general 2) y los operadores lógicos con los que se quiere combinar.

Si tomamos los operadores lógicos vistos anteriormente (OR, AND y NOT), podemos representar las siguientes tablas de verdad para cada operador:

- Tabla de Verdad del Operador OR

Operandos		Operador
a	b	a OR b
1 (verdadero)	1 (verdadero)	1 (verdadero)
1 (verdadero)	0 (falso)	1 (verdadero)
0 (falso)	1 (verdadero)	1 (verdadero)
0 (falso)	0 (falso)	0 (falso)

- Tabla de Verdad del Operador AND

Operandos		Operador
a	b	a AND b
1 (verdadero)	1 (verdadero)	1 (verdadero)
1 (verdadero)	0 (falso)	0 (falso)
0 (falso)	1 (verdadero)	0 (falso)
0 (falso)	0 (falso)	0 (falso)

- Tabla de Verdad del Operador NOT

Operando a	Operador NOT a
1 (verdadero)	0 (falso)
0 (falso)	1 (verdadero)

Diagramas de Karnaugh

Los diagramas de Karnaugh son un método gráfico que sirve para simplificar una expresión lógica eliminando condiciones redundantes.

Su diseño trata de una representación bidimensional de la tabla de verdad de la operación a simplificar.

Operadores Relacionales

Las expresiones relacionales se usan para realizar comparaciones de valores de tipo numéricos y de cadena. Los operadores de relación las usaremos para expresar condiciones en un algoritmo, como parte de una expresión.

Los operadores de relación son:

Operador	Significado
<	Menor que
>	Mayor que
=	Igual que
\leq	Menor o igual que
\geq	Mayor o igual que
\neq	Distinto de

La sintaxis para las comparaciones es:

expresión1 **operador de relación** expresión2

Y el resultado siempre será **verdadero o falso**. Por ejemplo en nuestra situación profesional, para comparar si dos ángulos son iguales, tendremos una expresión como la siguiente:

Angulo1 = Angulo2

Si se da que Angulo1 y Angulo2 son iguales, entonces el resultado de la expresión será verdadera, en caso contrario falsa.

Veamos otros ejemplos:

Expresión Resultado

34<=34 verdadero

34<>34 falso

34<>12 verdadero

Cuando los valores que se analizan son alfanuméricos, se realiza una comparación carácter por carácter, en sentido de izquierda a derecha. Si uno de los datos es más corto que el otro, si la comparación hasta que se termina el término más corto de los caracteres de ambos valores es igual, entonces se dice que la variable más corta es menor que la más larga. Se determinará que ambas variables son iguales solo si son iguales todos sus componentes y son iguales en longitud.

La última consideración a tener en cuenta es que las letras minúsculas serán mayores que las mayúsculas.

Expresión Resultado

"A" < "B" verdadero

"AA" < "AAAA" verdadero

"C" > "BBB" verdadero

"C" < "c" verdadero

"2" < "12" falso

3.1 Imágenes: Tipos de operadores

Tipos de Operadores

PL1

- ARITMÉTICOS

- LÓGICOS

- DE CADENA

- RELACIONALES

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
^	Exponenciación
div	División entera
mod	Módulo (Resto)

- ARITMÉTICOS

- LÓGICOS

- DE CADENA

Operador	Significado
OR	Suma Lógica
AND	Producto Lógico
NOT	Negación

- RELACIONALES

- ARITMÉTICOS

- LÓGICOS

Operador	Significado
&	Concatenación

- DE CADENA

- RELACIONALES

- ARITMÉTICOS

- LÓGICOS

- DE CADENA

- RELACIONALES

Operador	Significado
<	Menor que
>	Mayor que
=	Igual que
\leq	Menor o igual que
\geq	Mayor o igual que
\neq	Distinto de



¿Estás listo para un desafío?

1. Indique la opción correcta

El operador "=" es un:

- Operador Relacional.
- Operador Aritmético.
- Operador Lógico.

2. Indique la opción correcta

Si realizamos la operación "5 MOD 2" el resultado será:

- 2,5.
- 2.
- 1.

3. Indique la opción correcta

Si realizamos la operación "VERDADERO AND VERDADERO AND FALSO" el resultado será:

- VERDADERO.
- FALSO.
- Dará error en la ejecución.

4. Indique la opción correcta

Si realizamos la operación "(VERDADERO AND VERDADERO) OR FALSO" el resultado será:

- VERDADERO.
- FALSO.
- Dará error en la ejecución.

5. Indique la opción correcta

Las operaciones "+" y "-" se ejecutarán en el orden:

- En primer orden.
- En el último orden.

- Siempre antes de la multiplicación.

Respuestas de la Autoevaluación

1. Indique la opción correcta

El operador "=" es un:

- Operador Relacional.
- Operador Aritmético.
- Operador Lógico.

2. Indique la opción correcta

Si realizamos la operación "5 MOD 2" el resultado será:

- 2,5.
- 2.
- 1.

3. Indique la opción correcta

Si realizamos la operación "VERDADERO AND VERDADERO AND FALSO" el resultado será:

- VERDADERO.
- FALSO.
- Dará error en la ejecución.

4. Indique la opción correcta

Si realizamos la operación "(VERDADERO AND VERDADERO) OR FALSO" el resultado será:

- VERDADERO.
- FALSO.
- Dará error en la ejecución.

5. Indique la opción correcta

Las operaciones "+" y "-" se ejecutarán en el orden:

- En primer orden.
- En el último orden.
- Siempre antes de la multiplicación.

SP3 / Ejercicio resuelto

El objetivo del programa que se nos solicitó en esta situación profesional

Instituto San José

El instituto de enseñanza primaria San José le solicita a Ud., como su analista, una aplicación orientada a alumnos de cuarto grado que plantee problemas de geometría y aritmética.

De dicha aplicación, se le solicita a usted, como miembro de un equipo de trabajo, que: desarrolle el programa en pseudocódigo y que diseñe una interfaz donde el niño pueda ingresar las medidas de los tres lados de un triángulo. El niño deberá indicar si esas medidas corresponden a triángulos isósceles, equiláteros o escalenos.

Tenga en cuenta que:

- Isósceles tiene dos lados iguales.
- Equilátero tiene los tres lados iguales.
- Escaleno tiene los tres lados diferentes.

Es imprescindible que el programa verifique que las medidas correspondan a un triángulo. Para ello debe considerar que la suma de dos lados debe ser siempre mayor a la medida del lado restante. No se podrá evaluar al alumno si las medidas no corresponden a un triángulo.

El objetivo del programa es que se evalúe el resultado que el niño seleccionó, de tal modo que si el resultado es incorrecto el programa le indique el resultado real. Y si el niño ingresó bien el resultado se lo debe incentivar con un mensaje de felicitación.

es que se evalúe el resultado que el niño seleccionó, de tal modo que, si el resultado es incorrecto el programa le indique el resultado real y, si el niño ingresó bien el resultado, se lo debe incentivar con un mensaje de felicitación.

El problema se resuelve con el siguiente programa:

PSEUDO: Pseudo del ejercicio resuelto del Instituto San José (triángulos)

```
Programa EjerciciosConTriángulos
    'Definición del Formulario
    Formulario Triangulos
        Marco mrcMedidas
            CajaDeTexto txtA
            CajaDeTexto txtB
            CajaDeTexto txtC
        Fin Marco
        Marco mrcTipo
            BotonDeOpcion optIsosceles
            BotonDeOpcion optEquilatero
            BotonDeOpcion optEscaleno
        Fin Marco
        Marco mrcEvaluación
            Etiqueta lblEvaluacion
        Fin Marco
        BotonDeOpcion cmdEvaluar
    Fin Formulario
    'Desarrollo de los Procedimientos de Eventos
    Procedimiento cmdEvaluar_Click
        variable a = txtA tipo numerica
```

```

variable b = txtB tipo numerica
variable c = txtC tipo numerica
'Verifica que las medidas correspondan a las de un triangulo
Si (( b + c ) > a AND ( a + c ) > b AND ( a + b ) > c) Entonces
    ' Pregunta si los tres lados son iguales
    Si ( a = b AND b = c ) Entonces
        Si ( optEquilatero = VERDADERO ) Entonces
            lblEvaluacion = "Felicitaciones...tu opción es la correcta"
            'Se verifica la elección del alumno para evaluarlo
        Si No
            lblEvaluacion = "Las medidas corresponden a un triángulo Equilatero"
        Fin Si
    Si No
        'Verifica si dos lados son iguales
        Si ( a = b OR a = c OR b = c ) Entonces
            Si ( optIsosceles = VERDADERO ) Entonces
                lblEvaluacion = "Felicitaciones...tu opción es la correcta"
                ' Se verifica la elección del alumno para evaluarlo
            Si No
                lblEvaluacion = "Las ... a un triángulo Isósceles"
            Fin Si
        Si No
            'Si no es Equilátero ni Isósceles, entonces es Escaleno
            Si ( optEscaleno = VERDADERO ) Entonces
                lblEvaluacion = "Felicitaciones...tu opción es la correcta"
                'Se verifica la elección del alumno para evaluarlo
            Si No
                lblEvaluacion = "Las ... a un triángulo Escaleno"
            Fin Si
        Fin Si
    Si No
        Mensaje
            Título : "Medidas Incorrectas"
            Icono : Advertencia
            Texto : "La suma de dos lados debe ser siempre mayor al restante"
        Fin Mensaje
    Fin Si
Fin Procedimiento
Fin Programa

```

SP3 / Ejercicio por resolver

Un instituto de enseñanza primaria le solicita a un analista una aplicación orientada a alumnos de 4º grado que plantee problemas de geometría y aritmética.

De dicha aplicación, se le solicita a usted, como miembro de un equipo de trabajo, que desarrolle el programa en pseudocódigo y que diseñe una interfaz donde el niño pueda ingresar las medidas de los tres ángulos de un triángulo. El niño también deberá indicar qué tipo de triángulo es.

Es imprescindible que el programa verifique que las medidas de los ángulos correspondan a un triángulo. Para ello debe considerar que la suma de los tres ángulos debe ser siempre igual a 180° . No se podrá evaluar al alumno si las medidas no corresponden a las de un triángulo.

El objetivo del programa es que se evalúe el resultado que el niño seleccionó, de tal modo que si el resultado es incorrecto el programa le indique el resultado real, y si el niño ingresó bien el resultado se lo debe incentivar con un mensaje de felicitación.

Tenga en cuenta que los triángulos, según las medidas de los ángulos, se clasifican en:

- Rectángulo: cuando tienen un ángulo recto (igual a 90°).
- Acutángulo: Cuando tiene los tres ángulos agudos (menor a 90°).
- Obtusángulo: Cuando tienen un ángulo obtuso (mayor a 90° y menor a 180°).



¡Vamos a comprobar cuánto aprendiste!

1. Indique la opción correcta

El operador "MOD" es un:

- Operador Relacional.
- Operador Aritmético.
- Operador Lógico.

2. Indique la opción correcta

Si realizamos la operación "10 DIV 3" el resultado será:

- 3.
- 3,3333.
- 1.

3. Indique la opción correcta

Si realizamos la operación "VERDADERO OR FALSO OR FALSO" el resultado será:

- VERDADERO.
- FALSO.
- Dará error en la ejecución.

4. Indique la opción correcta

Si realizamos la operación "(FALSO OR VERDADERO) OR FALSO" el resultado será:

- VERDADERO.
- FALSO.
- Dará error en la ejecución.

5. Indique la opción correcta

El operador "&" es un operador:

- Operador Relacional.
- Operador de Cadena.

o Operador Lógico.

Respuestas de la Autoevaluación

1. Indique la opción correcta

El operador "MOD" es un:

- Operador Relacional.
- Operador Aritmético.
- Operador Lógico.

2. Indique la opción correcta

Si realizamos la operación "10 DIV 3" el resultado será:

- 3.
- 3,3333.
- 1.

3. Indique la opción correcta

Si realizamos la operación "VERDADERO OR FALSO OR FALSO" el resultado será:

- VERDADERO.
- FALSO.
- Dará error en la ejecución.

4. Indique la opción correcta

Si realizamos la operación "(FALSO OR VERDADERO) OR FALSO" el resultado será:

- VERDADERO.
- FALSO.
- Dará error en la ejecución.

5. Indique la opción correcta

El operador "&" es un operador:

- Operador Relacional.
- Operador de Cadena.
- Operador Lógico.

Situación profesional 4: Estructuras Repetitivas y Recursivas

Colegio Castel Franco

El Colegio Castel Franco, dedicado a la educación primaria requiere de un programa que contribuya con el aprendizaje de matemática.

En uno de los módulos el alumno ingresará dos números y obtendrá el resultado de la suma de todos los valores comprendidos entre ambos números.

Por ejemplo, si el alumno ingresa los números 2 y 7, se deberá calcular la siguiente suma:

$$2 + 3 + 4 + 5 + 6 + 7 = 27$$

El alumno también deberá ingresar el resultado. El objetivo del programa es evaluar al alumno. Por consiguiente el programa lo felicitará cuando haya ingresado el resultado correcto, y en caso contrario le indicará cual es el resultado que debió obtener.

Esta última parte es fundamental, ya que un programa educativo no está completo si no verifica los resultados para una posterior evaluación.

Se le pide que, como colaborador del Departamento de Informática del colegio, desarrolle el programa que resuelva dicha situación.

SP4 / H1: Estructuras Repetitivas

Para resolver una situación como la planteada, un programa con estructura secuencial y/o alternativa no resulta suficiente. Por lo que veremos el uso de las estructuras repetitivas que son comunes en la mayoría de los lenguajes de programación.

En informática, es muy frecuente que a las estructuras repetitivas se las denomine "Bucles", ya que las líneas de programa que están dentro de la estructura se ejecutarán muchas veces. Si se sigue el programa paso a paso, cuando se ejecuta la última línea de la repetitiva, se debe verificar la condición: Si es válida, se repite todo, si no, se continúa con las instrucciones que sigan a continuación de la última línea de la estructura repetitiva. En otras palabras: Continúa ejecutando las instrucciones que siguen después de la estructura repetitiva.

En el momento de regresar para controlar la condición se tiene la percepción de estar dando "giros" en el programa como si fuesen "bucles".

En esta materia veremos tres tipos de estructuras repetitivas, y su uso dependerá de la situaciones y de las condiciones previas que existan en la misma.

Las estructuras son:

- Estructura **Mientras**: la condición de salida se realiza al comienzo del bucle.
- Estructura **Hacer hasta**: la condición se realiza al final del bucle, el mismo se ejecuta hasta tanto se cumpla cierta condición.
- Estructura **Para**: la condición de salida se realiza con un contador que controla el número de iteraciones o giros.

Estructura Mientras

La estructura repetitiva **Mientras** es aquella en que las instrucciones se repiten mientras se cumple cierta condición. En esta estructura, lo primero que se evalúa es la condición de salida, que siempre será una expresión Lógica o booleana. De ser falsa la condición, no se ingresará a la estructura por lo que no se ejecutará ninguna de las instrucciones contenida en la misma, y se resolverán las instrucciones que se encuentran a posterior del bucle.

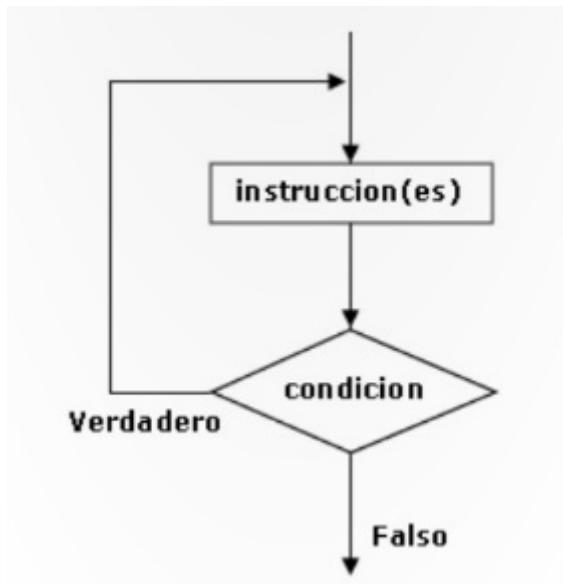
Sin embargo, si la condición es verdadera, se ejecutarán todas las instrucciones contenidas en el bucle y luego de esto se evaluará nuevamente la condición. Este proceso se repetirá tantas veces mientras condición de salida sea Verdadera.

En pseudocódigo, la estructura repetitiva mientras debe respetar el siguiente modelo:

PSEUDO: Pseudo estructura MIENTRAS

```
Mientras (condición) Hacer
    Instrucción 1
    ...
    Instrucción n
Fin Mientras
```

Y el diagrama de flujo será como el siguiente:



Volviendo a nuestra situación profesional, se nos pide que sumemos los números comprendidos entre dos números que se ingresaran por interfaz, pero no sabemos cuantos números tendremos que sumar, por lo que controlaremos con una estructura **mientras** la cantidad de repeticiones necesarias para cumplir con la situación. Dicha estructura quedará de la siguiente manera:

PSEUDO: Pseudo estructura MIENTRAS en ejemplo de coelgio Castel Franco

```

'Asigna a la variable Numero el valor inicial a sumar
Numero = txtDesde
'Suma toma como valor inicial el 0 (cero)
Suma = 0
'Calcula mientras Numero sea menor o igual a txtHasta
Mientras (Numero <= txtHasta) Hacer
    'Se acumulan los valores en la sumatoria
    Suma = Suma + Numero
    'Se obtiene el próximo número que se sumará
    Numero = Numero + 1
Fin Mientras

```

Estructura Hacer hasta

Tendremos situaciones en donde necesitaremos que las instrucciones contenidas en una estructura repetitiva, se realicen al menos una vez, independientemente de la condición de corte. Esta estructura se denomina **Hacer hasta**.

Esta estructura es similar a la **Mientras** en donde las instrucciones se ejecutan tantas veces mientras la condición de salida sea verdadera.

En pseudocódigo, la estructura repetitiva **Hacer hasta** debe respetar el siguiente modelo:

PSEUDO: Pseudo estructura Hacer hasta

```

Hacer
    Instrucción 1

```

```
...
Instrucción n
Hasta (condición)
```

Recordando nuestra situación profesional, realizaremos la misma solución utilizando la estructura **Hacer hasta**:

PSEUDO: Pseudo Hacer hasta en ejemplo de colegio Castel Franco

```
'Asigna a la variable Numero el valor inicial a sumar
Número = txtDesde
'Suma toma como valor inicial el 0 (cero)
Suma = 0
'Calcula mientras Número sea menor o igual a txtHasta
Hacer
    'Se acumulan los valores en la sumatoria
    Suma = Suma + Número
    'Se obtiene el próximo número que se sumará
    Número = Número + 1
Hasta (Número <= txtHasta)
```

En este caso se acumulará primero el valor de Número (txtDesde) a suma, incrementamos nuestro control de salida de la repetitiva, y a posterior controlamos si la misma se cumple, de ser así, hacemos una vez al menos el bloque de instrucciones y luego salimos de la estructura.

Estructura Para

Se nos presentarán situaciones en donde conocemos de antemano el número de veces que deseamos ejecutar las instrucciones de la estructura repetitiva. En este caso, en donde el número de iteraciones es conocido, usaremos la estructura **Para**.

En pseudocódigo, la estructura repetitiva **Para** debe respetar el siguiente modelo:

PSEUDO: Pseudo estructura PARA

```
Para variable = valor inicial Hasta valor final inc/dec valor Hacer
    Instrucción 1
    ...
    Instrucción n
Fin Para
```

inc/dec: incremento / decremento

Si el valor variable es menor que el valor final, los incrementos deben ser positivos para que se ejecuten las instrucciones. Si el valor final es menor al inicial se debe usar un decremento.

Veamos el siguiente ejemplo, en donde se suman los primeros 10 números pares comenzando desde cero:

PSEUDO: Pseudo ejemplo de estructura PARA

```
Procedimiento SumarPares
    variable n tipo numerica
    variable suma tipo numerica
    Para n = 2 Hasta 20 INC 2 Hacer
        suma = suma + n
    Fin Para
    Imprimir "La suma de los 10 primeros pares es " & suma
```

Consideraciones en el uso de estructuras repetitivas

Diferencia entre las estructuras repetitivas Mientras y Hacer Hasta

Mientras	Hacer hasta
Primero valida la condición de corte, si es verdadera se ejecutan las instrucciones que están dentro de la estructura.	Primero ejecuta, al menos una vez, las instrucciones que están dentro de la estructura, luego valida la condición de corte para seguir ejecutándolas en el caso de que dicha condición No se cumpla.
Cuando el programador arma la condición de corte debe formularse la siguiente pregunta: ¿Cuándo entra a la estructura?	Cuando el programador arma la condición de corte debe formularse la siguiente pregunta: ¿Cuándo sale de la estructura?

La elección de una u otra estructura estará dada por la situación que nos encontramos analizando.

Programas que no ingresan a una estructura repetitiva Mientras

Muchas veces un programador se encuentra frente a una estructura repetitiva que no se ejecuta. Pueden suceder dos cosas:

- que el programa esté mal diseñado,
- o que los datos de entrada no cumplan con la condición que se requiere.

El siguiente ejemplo muestra una parte de un programa resuelto incorrectamente, donde hay una estructura repetitiva a la que nunca se ingresará. Se lo conoce como "Bucle Cero", ya que se ejecutará cero veces.

PSEUDO: Pseudo ejemplo de programa resuelto incorrectamente "bucle cero"

```

Procedimiento cmdCalcular_Click
    variable cantidad = 0 tipo numerica
    Mientras (cantidad > 10) Hacer
        ...
        cantidad = cantidad + 1
    Fin Mientras
    ...
Fin Procedimiento

```

Como se puede observar, si a la variable cantidad se le asigna 0 en la línea anterior al **Mientras**, entonces nunca cumplirá la condición de ingreso a la estructura ya que la variable nunca será mayor que 10.

Estos son errores de programación que ocurren muy frecuentemente. Otras veces sucede que la lógica del programa es correcta, pero los datos de ingreso no cumplen con la condición, en esos casos sucede que circunstancialmente no se entra a la estructura repetitiva. Supongamos que tenemos el siguiente algoritmo que resuelve la potencia de un número. Supongamos el caso de que se calcule la potencia 1 de un número.

PSEUDO: Pseudo ejemplo de resolución incorrecta por datos de ingreso que no cumplen con la condición

```
Procedimiento cmdCalcular_Click
    variable Resultado tipo numérica
    variable Cantidad tipo numérica
    Resultado = txtBase
    Cantidad = 1
    Mientras (cantidad < txtExponente) Hacer
        Resultado = Resultado * txtBase
        Cantidad = cantidad + 1
    Fin Mientras
    LblResultado = Resultado
Fin Procedimiento
```

Cantidad es igual a 1. txtExponente también es igual a 1. Por consiguiente Cantidad no puede ser menor a txtExponente. Con estos valores no se ingresará a la estructura repetitiva. No obstante, el valor obtenido para Resultado es correcto

Tenga en cuenta que este ejemplo demuestra un caso donde circunstancialmente no se ingresa a la estructura, pero el programa está correctamente resuelto.

Programas que nunca salen de una estructura repetitiva

Este caso también es muy frecuente, pero siempre se debe a un error del programador. Se conoce como "bucle infinito". A veces el error está en la condición de corte, otras veces está en las instrucciones internas. Veamos dos ejemplos, uno para cada situación.

El siguiente es un programa que tiene incorrecta la condición de corte:

PSEUDO: Pseudo ejemplo de "bucle infinito" (MIENTRAS que no salen)

```
Procedimiento cmdCalcular_Click
    variable Cantidad = 0 tipo numérica
    Mientras (cantidad >= 0) Hacer
        ...
        Cantidad = cantidad + 1
    Fin Mientras
    ...
Fin Procedimiento
```

Cantidad es una variable que toma el valor inicial cero. Dentro de la estructura se incrementa de uno en uno.

Como se puede verificar, la condición de corte nunca será falsa, ya que Cantidad nunca será menor que cero, por lo tanto nunca dejará de ejecutar el bucle.

Otro error frecuente, es el caso en donde la condición de corte es correcta, pero no existe la posibilidad de que se salga del bucle ya que la condición nunca dejará de ser cierta. Tomemos como ejemplo la repetitiva de nuestra situación:

PSEUDO: Pseudo ejemplo que muestra que nunca sale del bucle ya que la condición nunca dejará de ser cierta (MIENTRAS que no salen)

```
'Asigna a la variable Numero el valor inicial a sumar
Numero = txtDesde
'Suma toma como valor inicial el 0 (cero)
Suma = 0
'Calcula mientras Número sea menor o igual a txtHasta
Mientras (Número <= txtHasta) Hacer
    'Se acumulan los valores en la sumatoria
    Suma = Suma + Numero
    'Se obtiene el próximo número que se sumará
    Numero = Numero
Fin Mientras
```

Como se observa en este caso, nuestra condición de corte se dará cuando Numero se menor o igual que txtDesde, pero dentro de la estructura, nunca modificamos esta variable, por lo que nunca será falsa la condición de corte.

Uso de operadores lógicos para unir varias condiciones en las estructuras repetitivas

En la mayoría de los programas (un poco más grandes que los vistos hasta el momento) se utilizan condiciones de corte compuestas. Para esto uniremos las condiciones por medio de los operadores lógicos ya vistos AND, OR y NOT.

Por ejemplo:

Para validar que dos o más condiciones se cumplan simultáneamente:

PSEUDO: Pseudo para validar que dos o más condiciones se cumplan simultáneamente

```
Mientras (txtNombre = "Ana" AND txtZona = "Sur") Hacer
    ...
Fin Mientras
Hacer
...
Hasta (txtNombre <> "Ana" AND txtNombre <> "ANA")
```

Para validar que al menos una de dos o más condiciones se cumplan:

PSEUDO: Pseudo para validar que AL MENOS una de dos o más condiciones se cumplan

```
Mientras (txtNombre = "Ana" OR txtNombre = "ANA") Hacer
    ...
Fin Mientras
Hacer
...
Hasta (txtNombre <> "Ana" OR txtZona <> "Sur")
```

Para negar una o más condiciones:

PSEUDO: Pseudo para negar una o más condiciones

```
Mientras (NOT txtNombre = "Ana") Hacer
    ...
Fin Mientras
Hacer
...
```

Hasta (NOT txtNombre <> "Ana")



¿Te animás a medir cuánto aprendiste?

1. Indique la opción correcta

La estructura "Para - Fin Para" es una extuctura repititiva.

- Verdadero.
- Falso.

2. Indique la opción correcta

Si ejecutamos el siguiente código, el resultado será:

```
1     Numero = 1
2     Mientras (Numero <= 5 Hacer) Hacer
3         Suma = Suma + Numero
4     Fin Mientras
```

- 5.
- 15.
- Ninguno de los anteriores.

3. Indique la opción correcta

Si ejecutamos el siguiente código, el resultado será:

```
1     Numero = 1
2     Mientras (Numero >= 5) Hacer
3         Suma = Suma + Numero
4         Numero 0 Numero + 1
5     Fin Mientras
6     Escribir Numero
```

- 15.
- 1.
- 5.

4. Indique la opción correcta

Si ejecutamos el siguiente código, ¿qué sucederá con la estructura repetitiva?

```
1     Número = 1
2     Mientras (Número <= 5) Hacer
3         Suma = Suma + Número
4         Número = Número
5     Fin Mientras
```

- No ingresará nunca.
- No saldrá nunca de la estructura.
- Se ejecutará normalmente y el resultado será 15.

5. Indique la opción correcta

La condición de la estructura Mientras, es válida o inválida:

```
1     Mientras (txtNombre = "Ana" AND txtZona = "Sur") Hacer
2     ...
3     Fin Mientras
```

- Válida.
- Inválida .

Respuestas de la Autoevaluación

1. Indique la opción correcta

La estructura "Para - Fin Para" es una estructura repititiva.

Verdadero.

Falso.

2. Indique la opción correcta

Si ejecutamos el siguiente código, el resultado será:

5.

15.

Ninguno de los anteriores.

3. Indique la opción correcta

Si ejecutamos el siguiente código, el resultado será:

15.

1.

5.

4. Indique la opción correcta

Si ejecutamos el siguiente código, ¿qué sucederá con la estructura repetitiva?

No ingresará nunca.

No saldrá nunca de la estructura.

Se ejecutará normalmente y el resultado será 15.

5. Indique la opción correcta

La condición de la estructura Mientras, es válida o inválida:

Válida.

Inválida .

SP4 / H2: Estructura de funciones o procedimientos recursivos

Recursividad

Hasta el momento hemos visto distintos tipos de estructuras, entre las cuales encontramos las repetitivas con contador (Para) y las que utilizan condición (Mientras, Hacer Hasta), muy importantes para recorrer estructuras de datos, para ingresar un conjunto de datos o simplemente para repetir un grupo de instrucciones. Pero se nos presentaran situaciones en donde se requiere de una repetición con algunas características especiales.

Por ejemplo, en un diccionario cuando buscamos el significado de una palabra, es posible que tengamos que reccurrir al significado de otra o varias para entender totalmente su significado; pero no podemos determinar cuántas palabras serán visitadas, si siempre pasará lo mismo; por lo tanto, el algoritmo se hace más complejo y se necesita simplificarlo para hacerlo más eficiente. Para este tipo de situaciones usaremos el concepto de recursividad.

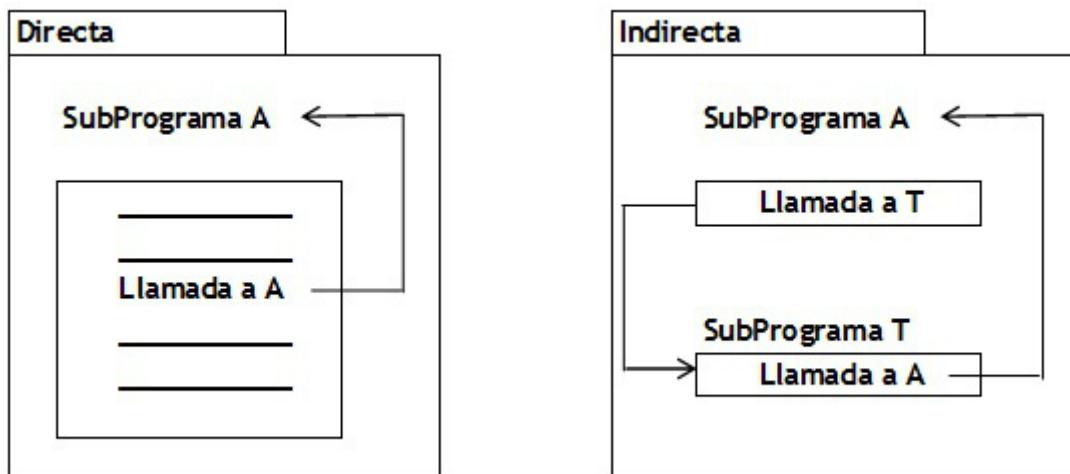
Este caso descripto no es el de nuestra situación profesional por lo que no aplicaremos esta herramienta a la misma, y utilizaremos otras situaciones que nos ayudarán a comprender los nuevos conceptos.

Existen distintas formas de definir lo que es la recursividad. Podemos decir que es: "un método, procedimiento o función parcialmente definido en términos de si mismo recibe el nombre de recursivo", o bien, "un subprograma que se llama a si mismo se dice recursivo". Es una herramienta de programación potente que en muchos casos puede producir algoritmos cortos y eficientes, pero se debe tener cuidado de no quedar en un bucle infinito, o sea, sin salida del programa.

La recursión en los programas puede darse de dos formas distintas:

- **Directa:** el procedimiento se llama directamente a si mismo ya que algún pase de ejecución encuentra una llamada a si mismo.
- **Indirecta:** un procedimiento A llama a otro procedimiento B, y este a su vez llama a A. Esta recursividad se puede dar con más de dos procedimientos.

Veamos estas dos situaciones gráficamente:



Para poder entender cómo funciona un programa recursivo, es conveniente compararlo con una estructura

repetitiva. Por ejemplo, si queremos calcular la suma de 5 números naturales, la repetitiva la planteamos desde 1 hasta 5, mientras que la recursividad la planteamos desde 5 hasta 1.

Queda claro que el resultado es el mismo, pero el procedimiento es al revés; esto se corresponde con las características de los procesos recursivos.

Dentro de las reglas de los procedimientos recursivos, podemos destacar:

- **Estado Básico:** en todo programa recursivo se debe definir un estado básico, o sea, un estado en el cual las solución no se presente de manera recursiva, sino directamente el resultado.
- **Progreso:** la entrada de datos del programa debe ir acercándose al estado básico.

En nuestro ejemplo, el estado básico es cuando $N=1$ entonces $Suma=1$, no hace falta la recursividad, mientras que en la entrada $N=5$ $Suma=5+4+3+2+1$.

Comparativa entre cálculo recursivo e iterativo

En base al ejemplo descripto anteriormente, veamos ahora la resolución algorítmica iterativa y recursiva para apreciar las diferencias.

Solución iterativa

Esta solución es la que hemos utilizado a lo largo de esta materia. La estructura del programa quedará de esta manera:

PSEUDO: Ejemplo de solución ITERATIVA

```
Programa SumaIterativa
    Procedimiento cmdSumar_Click
        'En esta etiqueta mostramos lo que nos devuelve
        'el procedimiento Sumar
        lblResultado = Sumar (txtNumero)
    Fin Procedimiento
    Procedimiento Sumar(variable N tipo numerica)
        variable i tipo numerica
        variable sum tipo numerica
        i = 1
        sum = 0
        Mientras i <= N Hacer
            'Realizo la suma, y la almaceno en una variable
            'para acumular los valores sumados
            sum = sum + i
            i = i + 1
        Fin Mientras
        Sumar = sum
    Fin Procedimiento
Fin Programa
```

Analicemos paso a paso lo realizada en el procedimiento Sumar del programa Sumalterativa:

- Se llama una vez al procedimiento.
- Se define una variable para el resultado, la cual va almacenando el valor calculado,
- Cuando sumo todoslos número y se cumple la condición de salida de la estructura Mientras, devuelvo el valos sumado.

Solución recursiva

PSEUDO: Ejemplo de solución RECURSIVA

```
Programa SumaRecursiva
    variable nro tipo numerica
    Procedimiento cmdSumar_Click
        nro = txtNumero
        'En esta etiqueta mostramos lo que nos devuelve
        'el procedimiento Sumar
        lblResultado = Sumar (nro)
    Fin Procedimiento
    Procedimiento Sumar(variable N tipo numerica)
        Si N = 1 Entonces
            'Caso básico, que corta la recursividad
            Sumar(1) = 1
        Si No
            'El procedimiento Sumar toma el mismo valor
            'que lo que retorna ella misma
            Sumar(N) = N + Sumar(N - 1)
        Fin Si
    Fin Procedimiento
Fin Programa
```

Analicemos paso a paso lo realizada en el procedimiento Sumar del programa SumaRecursiva:

Recibe el valor N, en nuestro ejemplo recibe 5, o sea, la entrada está dada por el mayor valor.

Establece el caso básico, donde controla que si N = 1, entonces el resultado es 1 y puede salir de la recursión, formalmente Sumar(1) = 1.

Se define el procedimiento recursivo que da el resultado buscado, Sumar(N) = N + Sumar(N - 1), como se recibió un 5, sería: Sumar(5) = 5 + Sumar(4), pero esto no nos da el resultado; por lo tanto debe volver, como recibe ahora un cuatro hace lo mismo, acercándose al resultado básico, para poder resolver la situación.

Las llamadas sucesivas dan la siguiente secuencia:

$$\text{Sumar}(5) = 5 + \text{Sumar}(4)$$

$$\text{Sumar}(4) = 4 + \text{Sumar}(3)$$

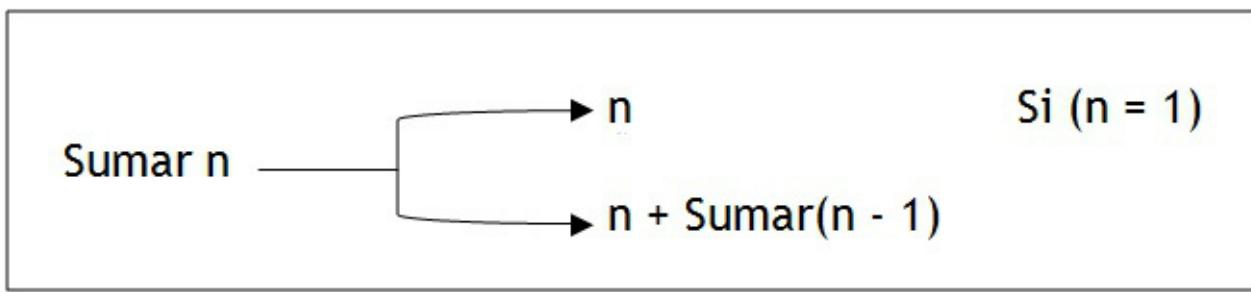
$$\text{Sumar}(3) = 3 + \text{Sumar}(2)$$

$$\text{Sumar}(2) = 2 + \text{Sumar}(1)$$

$$\text{Sumar}(1) = 1$$

- Como Sumar(1) está resuelto (porque no hay más números por sumar), entonces comienza el proceso regresivo, y devuelve lo que dejó pendiente:
 - Sumar(1) = 1
 - Sumar(2) = 2 + 1
 - Sumar(3) = 3 + 3
 - Sumar(4) = 4 + 6
 - Sumar(5) = 5 + 10

Por lo que el procedimiento Sumar devolverá 15.

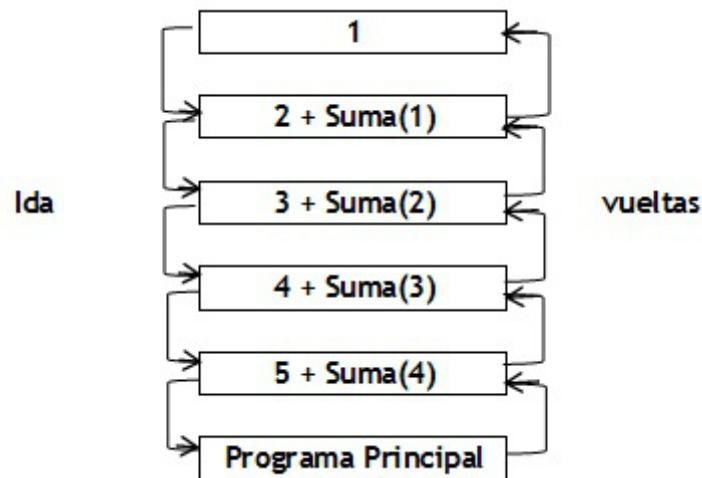


A esta altura usted quizás se este preguntando, ¿Realmente funciona?, la respuesta es "Sí", esa definición recursiva funciona. Siempre que la definición esté correctamente definida, la recursividad funcionará.

Ahora estamos en condiciones de platear otra regla de recursividad:

- **Puede creerlo:** asuma siempre que toda llamada recursiva interna funciona correctamente. Esta regla es difícil de creer, pues no estamos acostumbrados a aceptar con los ojos cerrados sin verificar que realmente funcione, lo que lleva a preguntarnos: *¿Cómo funciona la recursividad?* La implementación de la recursividad requiere la realización de tareas de bajo nivel del microprocesador; en realidad, la implementación de cualquier programa requiere de tareas de bajo nivel y la recursividad también. Los lenguajes de programación utilizan la memoria RAM para almacenar los programas que llamamos y las variables que son definidas por los operadores constructores, apilándolos una encima de otro, similar a una pila de platos.

Volvamos a la situación antes vista. Por cada llamado que se realiza al procedimiento `Sumar()`, se asigna un espacio de memoria nueva, generando un clon del procedimiento `Sumar()` y de todas sus variables locales. Por lo que, si realizamos 5 llamados, habrá 5 espacios asignados ubicados, como dijimos anteriormente, en manera de pila, guardando una referencia al anterior. En la cima de la pila, encontraremos el último llamado, por lo que una vez concluido el proceso la devolución se realiza en sentido inverso a la invocación. Si representamos este funcionamiento en manera gráfica, tendremos:



Ahora, imaginémonos que N^o es demasiado grande, hará N^o de llamadas al procedimiento... ¿cuál será el límite

de llamados?. El límite es físico no lógico, dado por el tamaño de la memoria RAM. Entonces, ¿qué pasa cuando no tenemos más memoria?, el programa no puede seguir con el proceso y no obtendremos el resultado.

Esta última demostración nos hace plantear la última regla de recursividad:

- **Regla de interés compuesto:** nunca duplique trabajo resolviendo instancias de un problema en llamadas recursivas separadas. Esto nos indica que, si bien un programa se puede resolver con método recursivos (como el que nosotros planteamos), no siempre es conveniente, pues ocupa mucha memoria y, quizás, no sea necesario. Tal vez el caso de nuestro ejemplo, tiene validez solo para explicar el funcionamiento de la recursividad, pero para nada sería eficiente su implementación.



¿Estás listo para un desafío?

1. Indique la opción correcta

Un procedimiento recursivo es el que se llama a sí mismo.

- Verdadero
- Falso

2. Indique la opción correcta

La recursividad en un programa puede darse de dos formas: directa o pausada.

- Verdadero
- Falso

3. Indique la opción correcta

La recursividad directa es cuando un programa se llama a sí mismo directamente.

- Verdadero
- Falso

4. Indique la opción correcta

El estado básico en un procedimiento recursivo es cuando se obtiene una solución de manera no recursiva.

- Verdadero
- Falso

5. Indique la opción correcta

Los procedimientos recursivos tienen dos reglas: estado básico y estado compuesto.

- Verdadero
- Falso

Respuestas de la Autoevaluación

1. Indique la opción correcta

Un procedimiento recursivo es el que se llama a sí mismo.

Verdadero

Falso

2. Indique la opción correcta

Las recursividad en un programa puede darse de dos formas: directa o pausada.

Verdadero

Falso

3. Indique la opción correcta

La recursividad directa es cuando un programa se llama a sí mismo directamente.

Verdadero

Falso

4. Indique la opción correcta

El estado básico en un procedimiento recursivo es cuando se obtiene una solución de manera no recursiva.

Verdadero

Falso

5. Indique la opción correcta

Los procedimientos recursivos tienen dos reglas: estado básico y estado compuesto.

Verdadero

Falso

SP4 / Ejercicio resuelto

Recuerde que nuestra situación profesional era

Colegio Castel Franco

El Colegio Castel Franco, dedicado a la educación primaria requiere de un programa que contribuya con el aprendizaje de matemática.

En uno de los módulos el alumno ingresará dos números y obtendrá el resultado de la suma de todos los valores comprendidos entre ambos números.

Por ejemplo, si el alumno ingresa los números 2 y 7, se deberá calcular la siguiente suma:

$$2 + 3 + 4 + 5 + 6 + 7 = 27$$

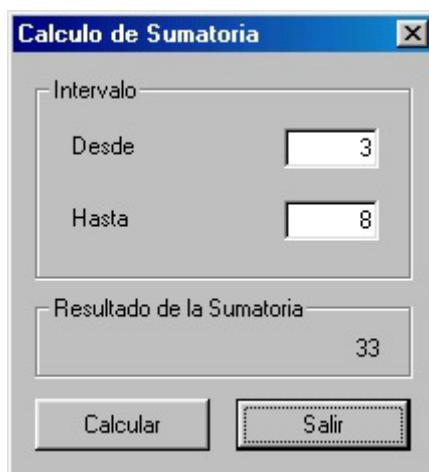
El alumno también deberá ingresar el resultado. El objetivo del programa es evaluar al alumno. Por consiguiente el programa lo felicitará cuando haya ingresado el resultado correcto, y en caso contrario le indicará cual es el resultado que debió obtener.

Esta última parte es fundamental, ya que un programa educativo no está completo si no verifica los resultados para una posterior evaluación.

Se le pide que, como colaborador del Departamento de Informática del colegio, desarrolle el programa que resuelva dicha situación.

.

La interfaz del cálculo de la sumatoria sería:



El pseudocódigo sería:

PSEUDO: Pseudocódigo del ejercicio del colegio Castel Franco

```
Programa Sumando
  'Definición del Formulario
  Formulario frmSumando
    Marco mrcRango
```

```

Caja de texto txtDesde
Caja de texto txtHasta
Fin Marco
Marco mrcResultado
    Etiqueta lblResultado
Fin Marco
BotonComando cmdCalcular
BotonComando cmdSalir
Fin Formulario
'Desarrollo de los Procedimientos de Evento
Procedimiento cmdCalcular_Click
    'Variable que se incrementará de uno
    variable Numero tipo numerica
    'Variable que almacenará la sumatoria
    variable Suma tipo numerica
    'Asigna a la variable Numero el valor inicial a sumar
    Numero = txtDesde
    'Suma toma como valor inicial el 0 (cero)
    Suma = 0
    'calcula mientras Número sea menor o igual a txtHasta
    Mientras (Numero <= txtHasta) Hacer
        'Se acumulan los valores en la sumatoria
        Suma = Suma + Numero
        'Se obtiene el próximo número que se sumará
        Numero = Numero + 1
    Fin Mientras
    lblResultado = Suma
Fin Procedimiento
Procedimiento cmdSalir_Click
    Cerrar frmSumando
Fin Procedimiento
Fin Programa

```

SP4 / Ejercicio por resolver

Realice el pseudocódigo del programa completo que resuelva la siguiente situación:

La empresa "Sumando" dedicada al cálculo numérico le solicita a usted la confección y programación de una interfaz gráfica que permita obtener la sumatoria de los números comprendidos entre 0 y un valor final que será ingresado por el usuario.

El usuario deberá tener la opción de elegir si la sumatoria tomará:

- Todos los números.
- Sólo números pares.
- Sólo múltiplos de cinco.
- Sólo múltiplos de diez.



¡Vamos a comprobar cuánto aprendiste!

1. Indique la opción correcta

Las recursividad puede darse en un programa de dos formas, directa o indirecta.

- Verdadero
- Falso

2. Indique la opción correcta

Los procedimientos recursivos tienen dos reglas: estado básico y progreso.

- Verdadero
- Falso

3. Indique la opción correcta

Si ejecutamos el siguiente código, el resultado será:

```
1  Numero = 5
2  Mientras (Numero <= 5) Hacer
3      Suma = Suma + Numero
4      Numero = Numero + 1
5  Fin Mientras
6  Imprimir Suma
```

- 5.
- 6.
- 15.

4. Indique la opción correcta

Si ejecutamos el siguiente código, el resultado será:

```
1     Numero = 5
2     Mientras (Numero <= 5) Hacer
3         Suma = Suma + Numero
4         Numero = Numero + 1
5     Fin Mientras
6     Imprimir Suma
```

- 15.
- 5.
- 6.

5. Indique la opción correcta

Cuantas veces se ejecuta la línea 3 en el siguiente código:

```
1     Numero = 2
2     Mientras (Numero < 3) Hacer
3         Numero = Numero + 1
4     Fin Mientras
```

- 3.
- 2.
- 1.

Respuestas de la Autoevaluación

1. Indique la opción correcta

Las recursividad puede darse en un programa de dos formas, directa o indirecta.

Verdadero

Falso

2. Indique la opción correcta

Los procedimientos recursivos tienen dos reglas: estado básico y progreso.

Verdadero

Falso

3. Indique la opción correcta

Si ejecutamos el siguiente código, el resultado será:

5.

6.

15.

4. Indique la opción correcta

Si ejecutamos el siguiente código, el resultado será:

15.

5.

6.

5. Indique la opción correcta

Cuantas veces se ejecuta la línea 3 en el siguiente código:

3.

2.

1.

Situación profesional 5: Estructuras de Datos Homogéneas: Vectores

Empresa Mocona

La Empresa Mocona, dedicada a la venta de repuestos de automóviles, necesita obtener diferentes tipos de informes correspondientes a la facturación.

- Importe de la facturación anual.
- Promedio de la facturación.
- Máxima facturación mensual.
- Cantidad de meses con facturación inferior a \$ 3.000.

Para obtener dicha información es imprescindible contar con los importes de la facturación mes a mes.

Se le solicita a usted, que es colaborador del área de sistemas, que haga un programa que permita emitir por pantalla dichos reportes. Tenga en cuenta que el usuario podrá acceder a un informe por vez.

Vector o Array

Hasta este momento hemos visto ejemplos de programas que requieren el empleo de datos simples, pero esta forma de almacenar información no resulta suficiente para resolver las situaciones con las que se enfrenta un programador.

Si analizamos nuestra situación profesional, imagine que tenemos que almacenar las ventas realizadas en todo el año mes a mes.

Ahora haga la siguiente reflexión:

- ¿Habrá que definir doce variables?
- ¿Cómo se definen esas variables?
- ¿Qué nombre es conveniente asignarle para no confundirlas y recordarlas... Ventas1, Ventas2, Ventas3.....Ventas12?
- ¿Y si se inscribe un alumno más, hay que modificar el programa para incorporar una nueva variable?
- ¿Existe un modo más simple para manejar una situación como esta?

Hasta ahora habíamos trabajado con variables y constantes, que son datos simples. Pero para resolver situaciones como la planteada, estos datos no resultan suficientes, es por eso que se crearon los datos estructurados.

Imagine qué fácil sería poder trabajar con los datos correspondientes a todas las ventas y referenciarlos con un único nombre. Esto es absolutamente posible si se define lo que se conoce como una estructura de datos.

Las estructuras de datos son muchas y tienen diferentes características. El programador decide que estructura de datos utilizará de acuerdo a las características de la estructura y de las operaciones que se pueden realizar con la misma. Para resolver una situación como la planteada es necesario usar una estructura denominada **Vector o Array**.

Un vector es un conjunto de datos homogéneos (que pueden ser simples o estructurados). Al conjunto se le asigna un nombre y a cada elemento del conjunto se lo puede procesar en forma independiente.

Para trabajar con un vector en un programa es fundamental definirlo previamente. Se debe tener en cuenta que el vector tiene las siguientes características:

- **Nombre:** si no se le asigna un nombre es imposible trabajar con el mismo en el programa. Siempre es conveniente asignarle un nombre significativo, por ejemplo en nuestra situación será **Ventas**.
- **Tipo de dato:** recuerde que hemos visto diferentes tipos de datos: numéricos, alfanuméricos, lógicos, fecha y hora. Todos los elementos constituyentes del vector serán del mismo tipo de dato, por eso es que se dice que los vectores son homogéneos. En nuestra situación deseamos almacenar en el vector las ventas realizadas en un mes, este valor se de tipo numérico, pero si se desea almacenar los nombres de los meses en que se realizaron las mismas, el tipo de dato deberá ser alfanumérico.
- **Dimensión:** indica la cantidad de elementos que tendrá el vector. Para almacenar las ventas de un año mes a mes, se necesitarán 12 variables elementos que forman el mismo. Según las

características del problema, pueden definirse vectores más pequeños, y vectores más grandes. Cuando en un programa se define el vector, debe indicarse siempre el nombre, el tipo de dato y la dimensión. Veamos como quedará la definición del vector de nuestra situación:

Estructura Vector Alumnos [10] tipo alfanumérico



Cada vez que se define un vector, se reserva en memoria la cantidad de espacio que necesita. Si se define un vector de diez posiciones, imagine que se verá en memoria como un conjunto de diez variables una a continuación de otra.

Estructura Vector Alfanumérico Alumnos[10]

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

Índices

Cada elemento de un vector, debe referenciarse por medio de un índice o subíndice, que debe tener la siguiente característica:

$$1 \leq i \leq n$$

En donde i es el índice y n la dimensión del vector. i es una variable que toma el valor de la posición del elemento que deseamos manejar.

Vale la pena aclarar que :

- La **posición** es el lugar que ocupa cada celda que compone el vector.
- El **elemento** es el dato que está almacenado en la celda de una posición determinada.

Ejemplo

$$i = 3$$

imprimir Ventas[i]

Primero nos situamos en la posición 3 y luego imprimimos el valor de la celda que se encuentra en esta posición.

En un vector se pueden realizar diferentes operaciones, tales como:

- Ingresar datos.
- Listar datos.
- Buscar datos.
- Modificar parte del vector o su totalidad.
- Borrar algún dato o todos.

Ingresar datos en un vector

Para ingresar datos en un vector, como dijimos anteriormente, es necesario contar con un índice para hacer manejo de las posiciones del mismo.

Así, si queremos ingresar en nuestro vector de ventas de la situación profesional, un algoritmo de carga sería como el siguiente:

PSEUDO: Ejemplo de algoritmo de carga

Procedimiento Carga

Ventas[1] = 1250

Ventas[2] = 1000

Ventas[3] = 1600

.....

Ventas[12] = 1760

Fin Procedimiento

Listar el contenido completo de un vector

Seguramente, listar el contenido completo del vector, utilizando índices constantes no es muy práctico, y menos aún cuando la estructura tiene una gran dimensión. Por ese motivo es que, generalmente, los índices de los vectores son variables numéricas que pueden tomar el valor de cualquiera de las posiciones del mismo.

Volviendo a nuestra situación, supongamos que después de la definición del vector se define una variable numérica que se utilizará como índice.

Estructura Vector Ventas[12] tipo numérico

Variable Numérica i = 1

Cuando "i" vale 1, Ventas [i] hace referencia al contenido de la primera posición del vector.

Cuando "i" vale 2, Ventas [i] hace referencia al contenido de la segunda posición del vector.

Cuando "i" vale 3, Ventas [i] hace referencia al contenido de la tercera posición del vector.

Y así sucesivamente, hasta que i tenga el valor de la ultima posición del vector.

De este modo se podrán acceder a todas las posiciones del vector con sólo modificar el valor de la variable utilizada como índice.

Imagine que el índice actúa como un apuntador de cada posición:

Estructura Vector numérico Ventas[12]

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

Ahora sólo falta pensar un algoritmo que genere el movimiento del índice a medida que muestra el contenido del vector. Ahora bien, será necesario contestar a las siguientes preguntas:

- ¿En qué posición se debe comenzar?

Generalmente se comienza el recorrido en la primera posición del vector, o sea, la posición 1.

- ¿Hasta cuándo se debe avanzar?

Si el vector está completo, se listarán los datos hasta llegar a la última posición. Si tenemos en cuenta el ejemplo que estamos manejando, será hasta la diez.

- ¿Cómo se modifica el valor del índice?

Para modificar el valor del índice deberá trabajarse el mismo como si fuese un contador ((AL)).

i = i + 1

La variable "i" incrementa su valor en 1

Por ejemplo: Si "i" vale 4, la línea de programa i = i + 1, cuando se ejecuta, lo que está haciendo en realidad es i = 4 + 1, de este modo, i quedará con el valor 5.

- ¿Es necesario modificar el valor del índice muchas veces?

Considerando que el índice comenzará con valor 1, y se lo incrementará de 1 en 1, para que llegue a doce será necesario incrementar el valor once veces. Esto sugiere la utilización de una estructura repetitiva en el algoritmo.

Ejemplo 1

Mostrar los datos del vector de nuestra situación profesional utilizando una estructura repetitiva "Mientras".

PSEUDO: Muestra datos del vector utilizando estructura repetitiva MIENTRAS

```
Programa ListarVentas
    'Definición de Datos Globales
    Estructura Vector Ventas[12] tipo numerico
    'Definición del Formulario
    Formulario frmListarAlumnos
        BotonComando cmdListar
    Fin Formulario
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdListar_Click
        variable i tipo numerica
        i = 1
        Mientras ( i <= 12 ) Hacer
            Imprimir Ventas[i]
            i = i + 1
        Fin Mientras
    Fin Procedimiento
Fin Programa
```

Este programa producirá un desplazamiento del índice que irá de la posición 1 a la posición 12. La estructura repetitiva controla que i sea menor o igual a 12, por lo tanto llegará un momento, durante la ejecución del programa, donde i valdrá 13. En ese momento será falsa la condición del "Mientras" y finalizará la ejecución del mismo, con lo que el último elemento impreso del vector Alumnos será el 12. Observe que primero se imprime Ventas[i] y recién después se incrementa i.

Ejemplo 2

Mostrar los datos del vector de nuestra situación profesional utilizando una estructura repetitiva "Para".

PSEUDO: Muestra los datos del vector utilizando la estructura repetitiva PARA

```
Programa ListarVentas
    'Definición de Datos Globales
    Estructura Vector Ventas[12] tipo numerico
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdListar_Click
        variable i tipo numerica
        i = 1
        Para i = 1 Hasta 12 Hacer
```

```

    Imprimir Ventas[i]
    Fin Para
    Fin Procedimiento
    Fin Programa

```

Como podemos observar, usando esta estructura repetitiva no necesitas de incrementar el índice ya que la misma estructura lo realiza. Solo debes colocar desde y hasta qué posición recorreremos (en este caso todo el vector), y dentro de la repetitiva imprimir el valor contenido en el vector.

Ejemplo 3

Mostrar los datos del vector de nuestra situación profesional utilizando una estructura repetitiva "Mientras" y que controle la existencia de datos en el vector, esto último significa que deje de listar cuando no encuentre datos cargados.

PSEUDO: Muestra datos del vector utilizando la estructura repetitiva MIENTRAS y controla la existencia de datos en el vector

```

Programa ListarAlumnos
    'Definición de Datos Globales
    Estructura Vector Ventas[12] tipo alfanumerico
    'Definición del Formulario
    Clase frmListarAlumnos
        BotónComando cmdListar
    Fin Clase
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdListar_Click
        variable i tipo numerica
        i = 1
        Mientras ( i <= 11 AND Ventas[i] > "" ) Hacer
            Imprimir Ventas [i]
            i = i + 1
        Fin Mientras
    Fin Procedimiento
Fin Programa

```

Carga de datos en un vector

La carga de datos en un vector tiene muchas características diferentes. Todo depende de la situación a resolver. Veremos algunos casos, los más frecuentes, en sucesivos ejemplos.

Ejemplo 4

Se necesita un programa que permita ingresar los nombres de los alumnos desde una interfaz gráfica, almacenándolos consecutivamente en un vector. Luego veremos el caso de nuestra situación en otra variante.

Para resolver este problema se requerirá una interfaz gráfica con una caja de texto y un botón de comando. Cada vez que el usuario del programa ingrese un nombre en la caja de texto, deberá oprimir el botón de comando para que el nombre se cargue en el vector.

¿Cuántos nombres se cargan cuando el usuario oprime el botón de comando? La respuesta es uno, un sólo nombre por cada clic, por consiguiente no es necesario utilizar una estructura repetitiva.

PSEUDO: Pseudo que permite ingresar los nombres de los alumnos desde una interfaz gráfica, almacenándolos consecutivamente en un vector

```

Programa IngresarAlumnos
    'Definición de Datos Globales

```

```

Estructura Vector Alumnos[10] tipo alfanumerico
variable i = 1 tipo numerico
'Definición del Formulario
Clase frmIngresarAlumnos
    CajaDeTexto txtNombre
    BotonComando cmdIngresar
Fin Formulario
'Desarrollo de los Procedimientos de Evento
Procedimiento cmdIngresar_Click
    Alumnos[i] = txtNombre
    i = i + 1
    txtNombre = ""
Fin Procedimiento
Fin Programa

```

Como habrá observado, luego de almacenarse en Alumnos[i] el nombre, la variable i se incrementa para que quede apuntando a la siguiente posición del vector.

Ejemplo 5

Disponemos de un Vector Numérico destinado a almacenar las Ventas de nuestra situación profesional. Pero antes de usarlo, se requiere inicializarlo. Para ello, se solicita poner un valor de 0 (cero) en cada una de las posiciones del vector.

Para resolver este problema se requerirá una interfaz gráfica con sólo un botón de comando. Cada vez que el usuario del programa oprima el botón de comando se cargará un cero en cada una de las posiciones del vector.
¿Cuántos ceros se cargan cuando el usuario oprime el botón de comando?.

La respuesta es: tantos ceros como posiciones tenga el vector, por lo que resulta necesario utilizar una estructura repetitiva.

PSEUDO: Pseudo para que cada vez que el usuario del programa oprima el botón de comando se cargará un cero en cada una de las posiciones del vector

```

Programa InicializarVector
'Definición del Formulario
Formulario frmInicializarVector
    BotonComando cmdInicializar
Fin Formulario
'Definición de Datos Globales
Estructura Vector Ventas[12] tipo numerico
variable i = 1 tipo numerica
'Desarrollo de los Procedimientos de Evento
Procedimiento cmdInicializar_Click
    i = 1
    Mientras ( i <=10 ) Hacer
        Ventas[i] = 0
        i = i + 1
    Fin Mientras
Fin Procedimiento
Fin Programa

```

Buscar datos en un vector

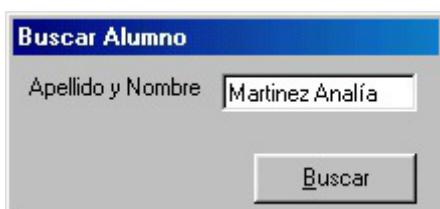
Uno de los algoritmos que más se utiliza, cuando se trabaja con vectores, es el de búsqueda.

Suponga que queremos buscar en un vector el nombre de un alumno, por ejemplo: "Franco".

Estructura Vector Alfanumérico Alumnos [10]

Ana	José	Franco	Gonzalo						
1	2	3	4	5	6	7	8	9	10

Con sólo observar el vector sabemos que el nombre que nos interesa está en la tercera posición. Pero el ordenador no puede tener esta visión de los datos del vector. Por este motivo, es que será necesario, recorrer el vector buscando el dato.



PSEUDO: Pseudocódigo para interfaz buscar alumno

Programa BuscarAlumno

'Definición de Datos Globales

Estructura Vector Alumnos[10] tipo alfanumerico

variable i tipo numerica

'Definición del Formulario **

Clase frmBuscarAlumno

CajaDeTexto txtNombre

BotonComando cmdBuscar

Fin Clase

'Desarrollo de los Procedimientos de Evento

Procedimiento cmdBuscar_Click

i = 1

'Se ejecutará el "mientras" siempre que el índice sea menor a diez y que en la posición del vector no esté el dato que se busca.

Mientras (i < 10 AND Alumnos[i] <> txtNombre) Hacer

 i = i + 1

Fin Mientras

'La condición de corte del mientras indica que existen dos motivos por los cuales se detiene la búsqueda:

 ' Por que se encontró el dato

 ' Por que se terminó el vector.

'Por eso se realizan los controles necesarios antes de mostrar los mensajes.

Si (Alumnos [i] = txtNombre) Entonces

 Mensaje "Alumno encontrado en la posición" & i

Si No

 Mensaje "No se encontró el alumno solicitado"

Fin Si

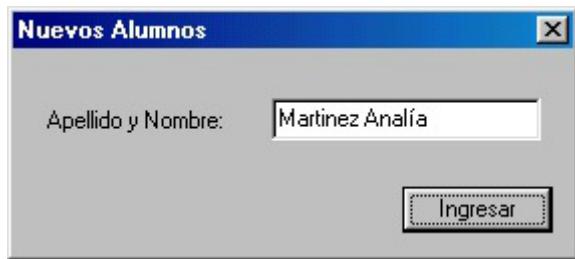
Fin Procedimiento

Fin Programa

Estos algoritmos de búsqueda se pueden utilizar para controlar que los datos que se ingresan no se repitan. En ese caso, antes de ingresar un nuevo dato se debe verificar su existencia con una búsqueda.

Ejemplo 6

Se necesita ingresar en un vector el nombre de diez alumnos, pero ningún nombre debe estar repetido.



PSEUDO: Pseudocódigo para interfaz nuevos alumnos

```

Programa IngresarAlumnos
    'Definición del Formulario
    Formulario frmIngresarAlumnos
        CajaDeTexto txtNombre
        BotonComando cmdIngresar
    Fin Formulario
    'Definición de Datos Globales
    Estructura Vector Alumnos[10] tipo alfanumerico
    variable i = 1 tipo numerica
    'El índice que se utiliza en la carga es el denominado i,
    'éste está indicando la próxima posición disponible, por
    'consiguiente no debe modificarse. Es por eso que la búsqueda
    'utiliza otro índice denominado j. Que se define en forma local.
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdIngresar_Click
        variable j = 1 tipo numerica
        Mientras (j < 10 AND Alumnos[j] <> txtNombre) Hacer
            j = j + 1
        Fin Mientras
        'Si el nombre está se muestra un mensaje que informa al usuario de la situación.
        'Si el nombre no está se carga el nuevo nombre y se incrementa el valor del índice i
        Si (Alumnos[j] = txtNombre) Entonces
            Mensaje "Se encontró el alumno en la posición" & j
        Si No
            Alumnos[i] = txtNombre
            i = i + 1
        Fin Si
    Fin Procedimiento
Fin Programa

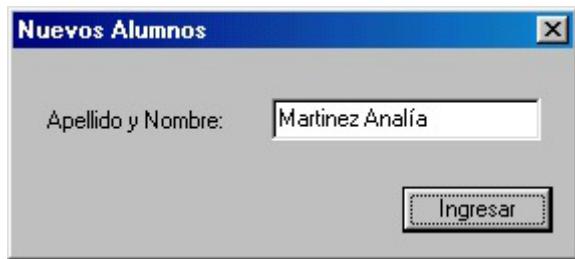
```

Ejemplo 7

Veamos, ahora, una variante del ejemplo anterior en donde al usuario se le informa si el vector ya no tiene más posiciones.

Cada vez que se ingresa un dato el índice se incrementa, cuando el índice tenga un valor más alto que la dimensión del vector significa que éste ya está completo. Por ejemplo, si el vector tiene diez posiciones, estará completo cuando el índice sea igual a 11.

Se necesita ingresar en un vector el nombre de diez alumnos, pero ningún nombre debe estar repetido, además el programa deberá avisarle al usuario en qué momento se completó el vector.



PSEUDO: Pseudocódigo para ingresar un vector en nombre, que ninguno esté repetido y que avise cuando se completó el vector

```

Programa IngresarAlumnos
    'Definición del Formulario
    Formulario frmIngresarAlumnos
        CajaDeTexto txtNombre
        BotonComando cmdIngresar
    Fin Formulario
    'Definición de Datos Globales
    Estructura Vector Alumnos[10] tipo alfanumerico
    variable i = 1 tipo numerica
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdIngresar_Click
        variable j = 1 tipo numerica
        Si (i < 11) Entonces
            Mientras (j < 11 AND Alumnos[j] <> txtNombre) Hacer
                j = j + 1
            Fin Mientras
            Si (Alumnos[j] = txtNombre) Entonces
                Mensaje "Se encontró el alumno en la posición:" & j
            Si No
                Alumnos[i] = txtNombre
                i = i + 1
            Fin Si
            Si No
                Mensaje "No hay más lugar para cargar datos"
            Fin Si
        Fin Procedimiento
    Fin Programa

```

Manejo de más de un vector

En un programa se puede trabajar con la cantidad de vectores que sea necesaria. Mientras haya memoria disponible en el ordenador no hay límite. Lo importante de trabajar con varios vectores es definir nombres significativos para cada vector e índices diferentes para los mismos.

Ejemplo 8

Se necesita realizar un programa que imprima las notas de los exámenes finales de un curso. Tenga en cuenta que los nombres de los alumnos están en un vector ya cargado pero no necesariamente completo, y las notas en otro vector. La relación existente entre los dos vectores es que el primer alumno de un vector tiene la primera nota del otro vector, el segundo alumno, la segunda nota, y así sucesivamente.

Alumnos [10]		Notas [10]	
a → 1	Migotti, Carolina	n → 1	5
2	Martinez, Analía	2	8
3	Ortiz, Franco	3	6
4	Diller, Gonzalo	4	7
5	Nicolodi, Lucas	5	5
6	Rossi, Daniel	6	4
7	Gonzalez, Marcos	7	2
8	Perez, Javier	8	10
9		9	
10		10	

PSEUDO: Pseudocódigo para el manejo de más de un vector

```

Programa ListarAlumnosNotas
    'Definición del Formulario
    Formulario frmListarAlumnosNotas
        BotonComando cmdListar
    Fin Formulario
    'Definición de Datos Globales
    Estructura Vector Alumnos[10] tipo alfanumerico
    variable numerica a = 1
    Estructura Vector Notas[10] tipo numerico
    variable n = 1 tipo numerica
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdListar_Click
        Imprimir "Listado General de Alumnos", SaltoDePagina
        Mientras (a <= 10 AND Alumnos[a] <> "") Hacer
            Imprimir Alumnos[a], Notas[n], SaltoDePagina
            a = a + 1
            n = n + 1
        Fin Mientras
    Fin Procedimiento
Fin Programa

```

Ejemplo 9

Se necesita realizar un programa que permita consultar la nota del examen final de un alumno. La consulta se realizará por pantalla.

PSEUDO: Pseudocódigo para un programa que permite consultar la nota del examen final de un alumno

```

Programa ConsultarNotas
    'Definición del Formulario
    Formulario frmConsultarNotas
        ListaDesplegable lstNombre
        Etiqueta lblNota
        BotonComando cmdConsultar
    Fin Formulario

```

```

'Definición de Datos Globales
Estructura Vector Alumnos[10] tipo alfanumerico
variable a = 1 tipo numerica
Estructura Vector Notas[10] tipo numerico
variable n = 1 tipo numerica
'Desarrollo de los Procedimientos de Evento
Procedimiento cmdConsultar_Click
    variable i = 1 tipo numerica
    Mientras (i < a AND Alumnos[i] <> lstNombre) Hacer
        i = i + 1
    Fin Mientras
    Si (Alumnos [i] = lstNombre) Entonces
        lblNota = Notas[i]
    Si No
        Mensaje "No se encontró el alumno solicitado"
    Fin Si
Fin Procedimiento
Fin Programa

```



¿Te animás a medir cuánto aprendiste?

1. Indique la opción correcta

Un vector es un conjunto de datos homogéneos.

- Verdadero
- Falso

2. Indique la opción correcta

Con la siguiente línea de código mostramos el contenido de la posición 3 de un vector: "Imprimir Vector = 3".

- Verdadero
- Falso

3. Indique la opción correcta

La dimensión de un vector es la cantidad de elementos que tendrá el mismo.

- Verdadero
- Falso

4. Indique la opción correcta

La única manera de recorrer un vector de manera completa es con un estructura repetitiva Mientras - Fin Mientras.

- Verdadero
- Falso

5. Indique la opción correcta

Para recorrer un vector utilizaremos una variable de tipo numérica denominada posición.

- Verdadero
- Falso

Respuestas de la Autoevaluación

1. Indique la opción correcta

Un vector es un conjunto de datos homogéneos.

Verdadero

Falso

2. Indique la opción correcta

Con la siguiente línea de código mostramos el contenido de la posición 3 de un vector: "Imprimir Vector = 3".

Verdadero

Falso

3. Indique la opción correcta

La dimensión de un vector es la cantidad de elementos que tendrá el mismo.

Verdadero

Falso

4. Indique la opción correcta

La única manera de recorrer un vector de manera completa es con un estructura repetitiva Mientras - Fin Mientras.

Verdadero

Falso

5. Indique la opción correcta

Para recorrer un vector utilizaremos una variable de tipo numérica denominada posición.

Verdadero

Falso

SP5 / H2: Búsqueda secuencial y binaria

Hay actividades cotidianas y profesionales que son importantes. Una de ellas es: la búsqueda. Continuamente buscamos elementos que son necesarios para alguna actividad.

Una de las operaciones más importantes en el procesamiento de datos, en los sistemas de información, es la búsqueda, pues con ella podemos recuperar datos almacenados muchas veces indispensable para la toma de decisiones.

Estas búsquedas, a veces, se realizan sobre elementos que ya están ordenados, pero no siempre es así; por ello existen métodos de búsqueda para cada caso.

Desarrollaremos a continuación dos métodos de búsqueda:

- Búsqueda Secuencial
- Búsqueda Binaria

Búsqueda Secuencial

Comienza su recorrido en el lugar que se le indique, normalmente la primera posición y se detiene cuando encuentra el dato o cuando no hay más datos almacenados.

Tomemos el siguiente ejemplo. Dado los siguientes datos almacenados en un vector (podrían ser los valores de venta de nuestra situación):

Vector A

5	8	2	9	4
				i = 6

Si buscamos el valor 2:

PSEUDO: Pseudocódigo para hacer una búsqueda secuencial

```
Estructura Vector numérico A[6]
variable i tipo numérica
Procedimiento BusquedaSecuencial
    variable j tipo numérica
    j = 1
    Mientras (j < i AND A[j] <> 2) Hacer
        j = j + 1
    Fin Mientras
    Si (A[j] = 2) Entonces
        Mensaje: "Lo encontramos en la posición: " & j
    Si No
        Mensaje: "El elemento no está"
    Fin Si
Fin Procedimiento
```

Como tiene dos condiciones en el corte de control de la estructura repetitiva, cuando termina su recorrido no sabemos si lo encontró o no. Una forma de verificarlo es controlando el índice, si $j = i$ es que no está, si paró antes es que lo encontró.

Búsqueda Binaria

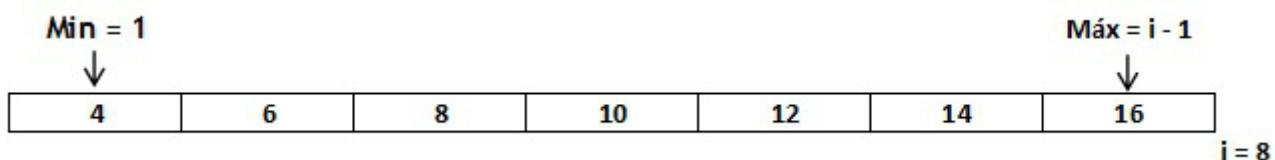
Los requisitos principales para la búsqueda binaria son:

- El vector debe estar ordenado en un orden específico.
- Debe conocerse el número de componentes.

El método consiste en partir en dos al vector y quedarnos con el intervalo factible de encontrar el dato buscado (de allí el nombre binario).

Comencemos con un ejemplo (ya que nuestra situación no es propicia para este tema). Tenemos almacenados en el vector los siguientes datos:

Vector A



Y queremos buscar el valor 8.

La variable Min guarda la primera posición, desde donde comienza el recorrido.

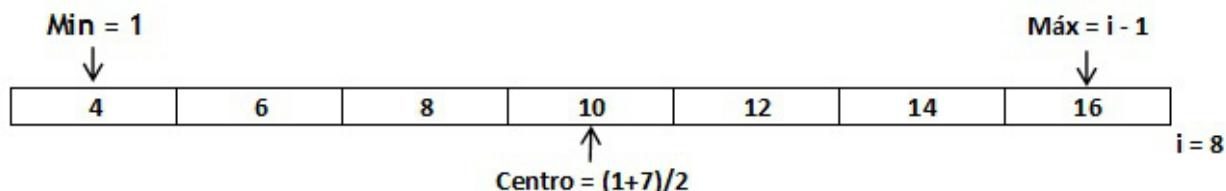
La variable Max guarda la posición del último elemento.

La búsqueda procede mediante una serie de pruebas sucesivas en el vector. El algoritmo partitiona en dos al vector y compara el elemento que está en el centro con el buscado. Si lo encuentra, termina el algoritmo. Si no, queda con la posición que corresponda por mayor o menor elemento.

$$\text{Centro} = (\text{Min} + \text{Max}) / 2$$

La variable Centro será un número entero, pues actuará como índice.

Vector A

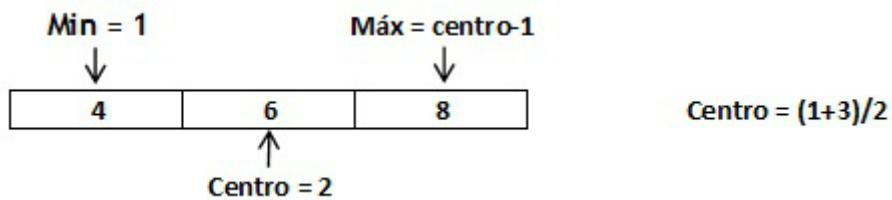


Compara:

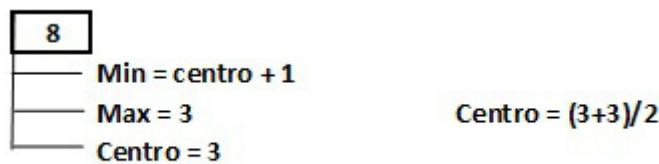
A[centro] = 8 por verdadero, finaliza.

A[centro] > 8 por verdadero, se queda con el intervalo izquierdo.

A[centro] < 8 por verdadero, se queda con el intervalo derecho.



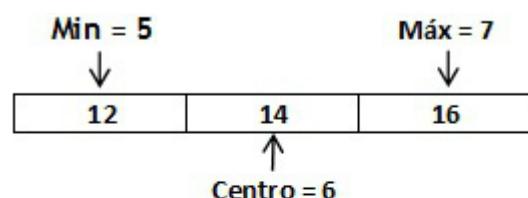
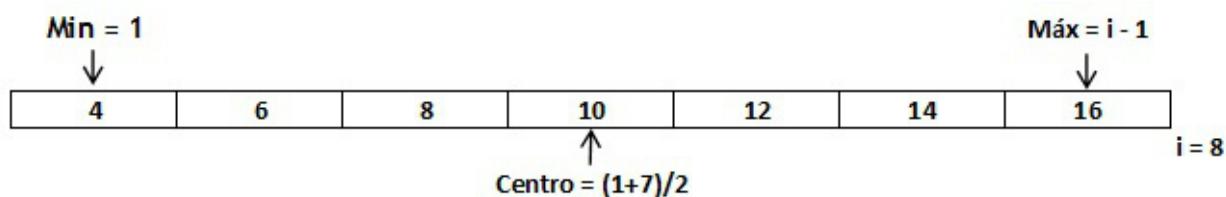
Y por último,



Lo encuentro en Centro.

Si el elemento por buscar es el 13, actuará de la siguiente manera:

Vector A



13 > 12 entonces incrementa $\text{Min} = \text{Centro}+1$, quedando $\text{Min} > \text{Max}$, indicando que no tiene mas

componentes y que no loecntró.

Presentamos a continuación el pseudocódigo de un procedimiento de búsqueda binaria que busca un nombre. Si el mismo existe en el vector, la búsqueda retorna la posición donde fue encontrado; de lo contrario retorna un -1 (posición nula).

PSEUDO: Pseudocódigo de un procedimiento de búsqueda binaria que busca un nombre

```
Procedimiento BusquedaBinaria(variable Nom tipo alfanumerico, variable ini tipo numerica, variable fin tipo numerica)
    variable med tipo numerica
    med = (ini + fin)/2
    Mientras (ini <= fin AND A[med]<>Nom) Hacer
        Si (A[med] < Nom) Entonces
            ini = med + 1
        Si No
            fin = med - 1
        Fin Si
    Fin Mientras
    Si (A[med] = Nom) Entonces
        BusquedaBinaria = med
    Si No
        BusquedaBinaria = -1
    Fin Si
Fin Procedimiento
```



¿Estás listo para un desafío?

1. Indique la opción correcta

La búsqueda binaria en vectores se realiza sólo si los elementos que contiene son números.

- Verdadero
- Falso

2. Indique la opción correcta

En la búsqueda binaria en vectores se debe conocer cuántos elementos hay.

- Verdadero
- Falso

3. Indique la opción correcta

Para la búsqueda binaria en vectores necesitaremos dos índices que nos ayuden con este trabajo.

- Verdadero
- Falso

4. Indique la opción correcta

La posición media de un vector en la búsqueda binaria se obtiene con: Centro = (Min + Max) / 2

- Verdadero
- Falso

5. Indique la opción correcta

En qué línea falla el código de Búsqueda Binaria:

```

1 Procedimiento BusquedaBinaria(variable Nom tipo alfanumerico, variable ini tipo numerica,
2     variable med tipo alfanumerica
3     med = (ini + fin)/2
4     Mientras (ini <= fin AND A[med]<>Nom)
5         Si A[med] < Nom
6             ini = med + 1
7             Si no
8                 fin = med - 1
9                 Fin Si
10            Fin Mientras
11            Si A[med] = Nom
12                BusquedaBinaria = med
13            Si no
14                BusquedaBinaria = -1
15            Fin Si
16        Fin Procedimiento

```

- 4.
- 3.
- No falla en ninguna línea.

Respuestas de la Autoevaluación

1. Indique la opción correcta

La búsqueda binaria en vectores se realiza sólo si los elementos que contiene son números.

Verdadero

Falso

2. Indique la opción correcta

En la búsqueda binaria en vectores se debe conocer cuántos elementos hay.

Verdadero

Falso

3. Indique la opción correcta

Para la búsqueda binaria en vectores necesitaremos dos índices que nos ayuden con este trabajo.

Verdadero

Falso

4. Indique la opción correcta

La posición media de un vector en la búsqueda binaria se obtiene con: Centro = (Min + Max) / 2

Verdadero

Falso

5. Indique la opción correcta

En qué línea falla el código de Búsqueda Binaria:

4.

3.

No falla en ninguna línea.

SP5 / H3: Métodos de ordenamiento

Generalmente, se considera **clasificar u ordenar** al proceso de organizar un conjunto de dato de objetos en una secuencia especificada. Es claro que el objetivo de este proceso es facilitar la búsqueda subsiguiente de los elementos del conjunto ordenado. Por lo tanto, podemos afirmar que es una actividad fundamental que se realiza universalmente.

Normalmente toda operación de búsqueda de datos, se realiza sobre elementos ordenados para que dicha información se obtenga en el menor tiempo posible.

Ordenar significa permutar elementos para que Iso mismos queden de acuerdo a una distribución preestablecida, o sea, de acuerdo a un criterio dado.

Existen varios métodos de ordenacióm, teniendo en cuenta la influencia de la estructura de datos en la elección de un algoritmo. Los metodos de ordenación se dividen en dos categorías:

- Ordenación de vectores.
- Ordenación de archivos.

La ordenacion de vectores se denomina ordenación interna, ya que se almacena en la memoria interna de la computadora. La ordenación de archivos se realiza casi siempre sobre soportes externos como discos, cintas, etc. y por eso se denomina ordenación externa.

Ordenación interna

Cuando se piensa en un algoritmo para ordenar los componentes de un vector surgen dos métodos:

- los que transportan elementos del vector inicial a otro vector resultado
- los que trabajan con un solo vector en el cual quedan sus elementos ordenados, lo que posibilita utilizar menos memoria de la disponible, implicando que las permutaciones de ítems, para obtener un vector ordenado, deben realizarse utilizando el espacio ocupado por el vector (ordenamiento in situ) lo cual es de mayor interés.

Otra característica a tener en cuenta en la clasificación de los algoritmos de ordenación, es su eficiencia, es decir su economía de tiempo; por ello se tiene en cuenta la cantidad de comparaciones de claves y de movimientos de items que se debe realizar.

La eficiencia es el factor que mide la calidad y rendimiento de un algoritmo.

Hay dos criterios que se suelen seguir a la hora de decidir que algoritmo es más eficiente:

- Tiempo menor de ejecución en computadora.
- Menor número de instrucciones.

Se clasifican en dos tipos:

- Métodos directos o cuadráticos (n^2).
- Métodos logarítmicos ($n * \log n$).

En ambos casos n representa la cantidad de elementos a ordenar.

Si bien los Métodos logarítmicos son considerados más rápidos, hay 3 buenas razones para estudiar en primera instancia los Métodos cuadráticos:

1. Representar las características de los principios de ordenamiento más usados.

2. Son algoritmos cortos y fáciles de comprender, teniendo en cuenta que los programas también ocupan memoria.

3. Son más rápidos para una cantidad pequeña o media de elementos a ordenar, no así cuando se tiene un gran número de elementos.

En conclusión, para la elección de un método de un ordenamiento entonces se tendrán en cuenta la estructura de datos a utilizar, el espacio en memoria requerido, el tiempo de demora, la complejidad del algoritmo y la cantidad de componentes a ordenar.

Los métodos de ordenación in situ (interna), pueden clasificarse en 3 categorías principales de acuerdo al método o forma de resolución:

- a) Ordenamiento por INTERCAMBIO.
- b) Ordenamiento por INSERCIÓN.
- c) Ordenamiento por SELECCIÓN.

a) Ordenamiento por Intercambio Directo (método de la Burbuja)

La característica dominante en este método es el intercambio entre pares de ítems, basado en el principio de comparar e intercambiar pares de ítems adyacentes, según un criterio de mayor o menor, hasta que todos estén ordenados.

Por ejemplo, dado los siguientes elementos cargados en este orden (tomamos este ejemplo ya que el vector de nuestra situación no se puede ordenar ya que cada posición equivale a un mes en particular):

vector A

pos =	5	16	3	23	40	i = 6
	1	2	3	4	5	

Estableciendo que los elementos de menor valor se ubican en el extremo derecho del vector, el algoritmo primero compara el elemento almacenado en la posición 1 con el elemento de la posición 2; si es menor lo intercambia; si no, queda como está.

5 < 16, por lo tanto intercambiamos los elementos quedando:

16 5 3 23 40

Compara el segundo con el tercero, quedando:

16 5 3 23 40

Compara el tercero con el cuarto, intercambiando:

16 5 23 3 40

Compara el cuarto con el quinto, intercambiando:

16 5 23 40 3

Al terminar las primera pasada podemos asegurar que acomodamos en la posición correcta al de menor valor, pero el resto no, por lo tanto se realizan repetidas pasadas por el vector, moviendo en cada una el elemento clave menor hacia el extremo elegido.

Para nuestro ejemplo, ¿Cuántas pasadas tendríamos?

Observemos que para ordenar 5 elementos necesitamos recorrer 4 veces el vector.

Si miramos en posición vertical, los ítems como burbujas en un depósito de agua ascienden hasta el nivel de peso que le corresponde; por ello el nombre de método de la burbuja.

Para realizar los intercambios en el algoritmo, nos ayudamos con una variable auxiliar definida del mismo tipo del vector.

Lógicamente y expresado en pseudocódigo, el procedimiento es:

PSEUDO: Declaración para realizar los intercambios en el algoritmo. Nos ayudamos con una variable auxiliar definida del mismo tipo del vector

```
Estructura Vector A[5] tipo numerico  
variable i tipo numerica
```

Recordemos que nuestro índice *i* está indicando la posición siguiente a la del último dato. En otras palabras: la próxima posición libre. Además, tengamos presente que este procedimiento se utiliza cuando el vector ya tiene elementos cargados.

PSEUDO: Pseudocódigo de algoritmo sencillo pero que realiza comparaciones innecesarias

```
Procedimiento Ordenar  
    variable j, k tipo numerica  
    variable aux tipo numerica  
    'j se utiliza para contar la cantidad de pasadas  
    j = 0  
    Mientras (j < (i - 2)) Hacer  
        k = 1  
        Mientras (k < i) Hacer  
            Si (A[k] > A[k + 1]) Entonces  
                aux = A[k]  
                A[k] = A[k + 1]  
                A[k + 1] = A[k]  
            Fin Si  
            k = k + 1  
        Fin Mientras  
        j = j + 1  
    Fin Mientras  
Fin Procedimiento
```

Se puede reemplazar la estructura repetitiva **Mientras** utilizada por la **Para** obteniéndose el mismo resultado.

Observando el algoritmo podemos considerarlo sencillo pero se ralizan comparaciones innecesarias, ya que por cada pasada asguramos un valor colocado en la posición que corresponde.

Este algoritmo admite fácilmente algunas mejoras. Además, podríamos preguntarnos ¿que pasaría si estuviesen ya ordenados los elementos almacenados? La respuesta es simple: no es óptimo en cuanto a tiempo porque no controla los cambios que realiza en las posiciones de sus componentes.

Para ello, se puede optimizar el algoritmo de dos formas:

- **Método de la burbuja con señal.** Consiste en controlar que los ciclos repetitivos varíen, disminuyendo en uno cada pasada no siendo necesario controlar el total de elementos (ya que se acomoda uno por vez). Por ejemplo, si tiene que ordenar 10 elementos, en la primera pasada controlará los 10, acomodando según el criterio. En la segunda pasada es necesario controlar sólo 9, en la tercera sólo 8, en la cuarta sólo 7, y así sucesivamente hasta controlar sólo 3 elementos. Por otro lado, si se coloca un contados en el lugar del algoritmo donde se realiza la permutación, se controla la cantidad de cambios que se realizaron. Cuando el contador almacena el valor cero significa que no hubo intercambios, se termina el algoritmo, pues está ordenado.

- **Método de la sacudida o "Shaker Sort".** Este algoritmo consiste en mezclar las formas en que puede realizar el Método de la Burbuja cada pasada tiene dos etapas. En la primera, "de derecha a izquierda", se trasladan los elementos más pequeños hacia la izquierda del vector, guardando la posición del último elemento intercambiado en una variable auxiliar. En la segunda etapa, "de izquierda a derecha", se trasladan los elementos más grandes hacia la derecha del vector, almacenando la posición del último intercambio en otra variable auxiliar. En las siguientes pasadas se trabaja con los elementos comprendidos entre las posiciones almacenadas. Los elementos del vector están ordenados cuando en una pasada no se han realizado intercambios, o bien cuando los valores almacenados en las posiciones auxiliares, de izquierda y derecha, cumplen con la condición "el elemento del extremo izquierdo es menor que el elemento del extremo derecho".

Por ejemplo, supongamos que tenemos los siguientes elementos en el vector A:

vector A

pos =	20	60	15	7	32	1	i = 7
	1	2	3	4	5	6	

Primera pasada: de derecha a izquierda

A[6] > A[5] intercambio 1, 32

A[5] > A[4] intercambio 1, 15, 32

A[4] > A[3] intercambio 1, 7, 15, 32

A[3] > A[2] intercambio 1, 60, 7, 15, 32

A[2] > A[1] intercambio 1, 20, 60, 7, 15, 32

Última posición de intercambio = 2

Segunda pasada: de izquierda a derecha.

A[2] > A[3] no hay intercambio 20, 60

A[3] > A[4] intercambio 20, 7, 60

A[4] > A[5] intercambio 20, 7, 15, 60

A[5] > A[5] intercambio 20, 7, 15, 32, 60

Última posición de intercambio = 5

Al repetir las dos etapas, los elementos quedan de la siguiente forma:

1, 7, 29, 15, 32, 60 posición izq=3, posición der=4

Luego:

1, 7, 15, 20, 32, 60

PSEUDO: Método de la sacudida

```
Estructura Vector A[6] tipo numerico
Procedimiento Sacudida
    variable posIzq tipo numerica
    variable posDer tipo numerica
    variable k tipo numerica
    variable aux tipo numerica
```

```

variable j tipo numerica
posIzq = 1
posDer = i - 1

Mientras (posIzq <= posDer) Hacer
    j = posDer
    Mientras (j >= posIzq) Hacer
        Si (A[j - 1] > A[j]) Entonces
            aux = A[j - 1]
            A[j - 1] = A[j]
            A[j] = aux
            k = j
        Fin Si
        j = j - 1
    Fin Mientras
    posIzq = k + 1
    j = posIzq
    Mientras (j <= posDer) Hacer
        Si (A[j - 1] > A[j]) Entonces
            aux = A[j - 1]
            A[j - 1] = A[j]
            A[j] = aux
            k = j
        Fin Si
        j = j + 1
    Fin Mientras

    posDer = k - 1
Fin Mientras
Fin Procedimiento

```

Recordemos que la eficacia de un método está dada matemáticamente por la fórmula que combina los tres factores que afectan al tiempo de ejecución: la cantidad de comparaciones, la cantidad de intercambios y las pasadas que se realizan. En función de ello, podemos establecer que este algoritmo es más eficiente que el de la Burbuja y puede serlo con relación a otros métodos, si los elementos están parcialmente ordenados; si no, debe preferirse los otros métodos directos: Inserción y Selección.

b) Ordenación por Inserción Directa (método de la Baraja)

Llamado tambien método de la Baraja, ya que es el que utilizan los jugadores de cartas para ordenarlas. Consiste en colocar el elemento menor que encuentra, a la izquierda, aunque se haya considerado ordenado anteriormente.

Por ejemplo: si tenemos los elementos del Vector A del ejemplo anterior.

vector A

pos =	20	60	15	7	32	1	i = 7
	1	2	3	4	5	6	

1^{ra} pasada:

A[2] < A[1] no hay intercambio 20, 60

2^{da} pasada:

A[3] < A[2] intercambio 7, 60

A[2] < A[1] intercambio 1, 20, 60

3^{ra} pasada:

A[4] < A[3] intercambio 1, 20, 7, 60

A[3] < A[2] intercambio 1, 7, 20, 60

A[2] < A[1] no hay intercambio 1, 7, 20, 60

4^{ta} pasada:

A[5] < A[4] intercambio 1, 7, 20, 15, 60

A[4] < A[3] intercambio 1, 7, 15, 20, 60

A[3] < A[2] no hay intercambio 1, 7, 15, 20, 60

5^{ta} pasada:

A[6] < A[5] intercambio 1, 7, 15, 20, 32, 60

A[5] < A[4] no hay intercambio 1, 7, 15, 20, 32, 60

Veamos el pseudocódigo completo de este método:

PSEUDO: Inserción Directa

```
Procedimiento Inserción
    variable i tipo numérica
    variable k tipo numérica
    variable aux tipo numérica
    j = 1
    Mientras (j < i - 1) Hacer
        Aux = A[j]
        K = j - 1
        Mientras ((k >= 1) AND (aux < A[k])) Hacer
            A[k + 1] = A[k]
            k = k - 1
        Fin Mientras
        A[k + 1] = aux
        j = j + 1
    Fin Mientras
Fin Procedimiento
```

El tiempo necesario para ordenar los elementos de un arreglo con este método, es proporcional a N^2 , donde N representa el número de elementos del vector. Sólo es recomendable cuando N es pequeño.

Método de Inserción Binaria

Su mejoramiento consiste en recurrir a una Búsqueda Binaria, en vez de la Secuencial, para insertar un elemento en la parte izquierda del vector, donde los elementos ya se encuentran ordenados. Al igual que el algoritmo anterior, el proceso toma los elementos ubicados desde la posición 2 hasta la última ocupada.

Por ejemplo: utilizando los elementos del Vector A.

vector A

	20	60	15	7	32	1	
pos =	1	2	3	4	5	6	i = 7

1^{ra} pasada:

A[2] < A[1] no hay intercambio 20, 60

2^{da} pasada:

A[3] < A[2] intercambio 15, 60, 20

A[3] < A[1] intercambio 15, 20, 60

3^{ra} pasada:

A[4] < A[1] intercambio 7, 15, 20, 60

A[4] < A[2] no hay intercambio 7, 15, 20, 60

A[4] < A[3] no hay intercambio 7, 15, 20, 60

4^{ta} pasada:

A[5] < A[1] intercambio 1, 7, 15, 20, 60

A[5] < A[2] no hay intercambio 1, 7, 15, 20, 60

A[5] < A[3] no hay intercambio 1, 7, 15, 20, 60

5^{ta} pasada:

A[6] < A[2] no hay intercambio 1, 7, 15, 20, 60, 32

A[6] < A[3] no hay intercambio 1, 7, 15, 20, 60, 32

A[6] < A[4] no hay intercambio 1, 7, 15, 20, 60, 32

A[6] < A[5] intercambio 1, 7, 15, 20, 32, 60

Veamos el pseudocódigo completo de este método:

PSEUDO: Inserción Binaria

```

Procedimiento InsercionBinaria
    variable j tipo numerica
    variable aux tipo numerica
    variable izq tipo numerica
    variable der tipo numerica
    variable med tipo numerica
    variable k tipo numerica
    j = 2
    Mientras (j < i) Hacer
        aux = A[j]
        izq = 1
        der = j - 1
        Mientras (izq <= der) Hacer
            med = (izq + der) / 2
            Si (aux < A[med]) Entonces
                der = med - 1

```

```

    Si No
        izq = med +1
    Fin Si
Fin Mientras
k = j - 1
Mientras (k >= izq) Hacer
    A[k + 1] = A[k]
    k = k - 1
Fin Mientras
A[izq] = aux
j = j + 1
Fin Mientras
Fin Procedimiento

```

Si analizamos el método, observamos un comportamiento antinatural que se efectúa el menor número de comparaciones cuando el arreglo está totalmente desordenado y el máximo cuando está ordenado. En este método, la cantidad de comparaciones será la mitad que en el método de inserción, pero no reduce la cantidad de movimientos, por lo tanto el tiempo de ejecución del algoritmo sigue siendo proporcional a N^2 .

c) Ordenamiento por Selección Directa

Este método consiste en buscar el elemento de menor valor e intercambiarlo con el elemento que está en la primera posición; luego se busca el siguiente menor y se intercambia con el que está en la segunda posición y así sucesivamente hasta que quede el de mayor valor.

Siguiendo con el ejemplo del Vector A tenemos.

vector A

	20	60	15	7	32	1	i = 7
pos =	1	2	3	4	5	6	

1^{ra} pasada:

menor = A[1] menor = A[1] menor = 20

menor < A[2] verdadero

menor < A[3] falso menor = A[3] menor = 15

menor < A[4] falso menor = A[4] menor = 7

menor < A[5] verdadero

menor < A[6] falso menor = A[6] menor = 1

Se intercambia: 1, 60, 15, 7, 32, 20.

2^{da} pasada:

menor < A[2] menor = A[2] menor = 60

menor < A[3] falso menor = A[3] menor = 15

menor < A[4] falso menor = A[4] menor = 7

menor < A[5] verdadero

menor < A[6] verdadero

Se intercambia: 1, 7, 15, 60, 32, 20.

3^{ra} pasada:

menor < A[3] menor = A[3] menor = 15

menor < A[4] verdadero

menor < A[5] verdadero

menor < A[6] verdadero

Se intercambia: 1, 7, 15, 60, 32, 20.

4^{ta} pasada:

menor < A[4] menor = A[4] menor = 60

menor < A[5] falso menor = A[5] menor = 32

menor < A[6] falso menor = A[6] menor = 20

Se intercambia: 1, 7, 15, 20, 32, 60.

5^{ta} pasada:

menor < A[5] menor = A[5] menor = 32

menor < A[6] verdadero

Se intercambia: 1, 7, 15, 20, 32, 60.

Veamos el pseudocódigo completo de este método:

PSEUDO: Selección Directa

```
Procedimiento SelecciónDirecta
    variable j tipo numérica
    variable k tipo numérica
    variable m tipo numérica
    variable menor tipo numérica
    j = 1
    Mientras (j < i - 1) Hacer
        Menor = A[j]
        k = j
        m = j + 1
        Mientras (m < i) Hacer
            Si (A[m] < menor) Entonces
                menor = A[m]
                k = m
            Fin Si
            m = m + 1
        Fin Mientras
        A[k] = A[j]
        A[j] = menor
        j = j + 1
    Fin Mientras
Fin Procedimiento
```

El número de comparaciones que se realiza entre los elementos es independiente de los valores de los mismos, siempre será $(n - 1)$ en la primera pasada, $(n - 2)$ en la segunda, y así sucesivamente hasta una comparación en la última pasada. En cuanto a los intercambios, son $(n - 1)$. Es más eficiente que los anteriores, aunque no es recomendado para cantidades medianas o grande de datos y el tiempo del algoritmo es N^2 .

Veremos a continuación una optimización del método que vimos anteriormente.

Método Shell

Este método tambien es conocido como **Inserción con incrementos decrecientes**. Para que los elementos del arreglo queden ordenados más rápidamente, Shell (su creador) propuso que las comparaciones que realiza un elemento con su grupo a la izquierda se realicen con saltos mayores.

En primer lugar, se agrupan y ordenan por separado los elementos que distan cuatro posiciones, "ordenación de a cuatro en cuatro". Luego se agrupan los elementos que distan de dos posiciones y se ordenan por separado, "Ordenación de dos en dos". Por último, en la tercera pasada, se ordena de uno en uno.

Este método presenta ciertas características:

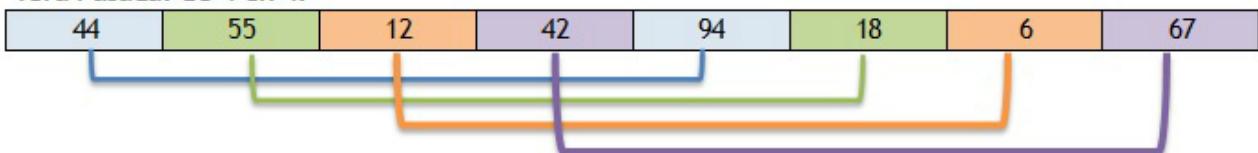
- Cada pasada se beneficia de las anteriores, ya que combina grupos previamente ordenados.
- Cualquier salto es aceptable, siempre que la última pasada sea de uno en uno.
- Se recomienda que los incrementos sean potencias de dos. Si el vector es de 16 elementos, se realiza para la primera pasada, 8 grupos de a dos, comparando el 1º con el 9º, el 2º con el 10º, y así sucesivamente hasta el 8º con el 16º.

Por ejemplo, si tenemos el vector A con 8 elementos.

Vector A

44	55	12	42	94	18	6	67
1	2	3	4	5	6	7	8

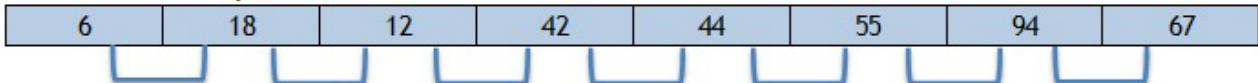
1era Pasada: de 4 en 4.



2da Pasada: de 2 en 2.



3era Pasada: uno por uno



Resultado

6	12	18	42	44	55	67	94
---	----	----	----	----	----	----	----

Veamos el pseudocódigo de este método:

PSEUDO: Pseudocódigo del método SHELL

```
Procedimiento Shell
    variable n tipo numerica
    variable j tipo numerica
    variable aux tipo numerica
    variable band tipo numerica
    n = i
    Mientras (n > 1) Hacer
        n = n / 2
        band = VERDADERO
        Mientras (band = VERDADERO) Hacer
            band = FALSO
            j = 1
            Mientras ((j + n) <= (i - 1)) Hacer
                Si (A[j] > A[j + n]) Entonces
                    Aux = A[j]
                    A[j] = A[j + n]
                    A[j + n] = aux
                    band = VERDADERO
                Fin Si
                j = j + 1
            Fin Mientras
        Fin Mientras
    Fin Mientras
Fin Procedimiento
```

La eficiencia de este método está en duda, pues no se ha podido establecer cuál es la secuencia de intervalos más apropiada, sobre todo cuando existe una gran cantidad de elementos almacenados en un vector. Algunas pruebas arrojaron como resultado que la mejor secuencia es uno en uno, lo cual corresponde al método de insección directa.

Ordenación por partición

Es sorprendente como la mejora, basada en intercambios, que describimos a continuación produce el mejor método de ordenar vectores, conocido hasta ahora.

Este método fue creado por Hoare, quien lo denominó "**Ordenación Rápida (QuickSort)**", ya que la cantidad de código necesario es pequeño comparado con la excelente velocidad que proporciona. Es una optimización sustancial del intercambio directo.

El algoritmo consiste, básicamente, en:

- Elegir un elemento cualquiera del vector denominado **Pivot**.
- Dividir el vector inicial en dos mitades, de tal forma que en una de ellas queden los elementos menores y en la otra mitad, todos los elementos mayores al pivot.

Para lograr esto tenemos que:

- Tomar arbitrariamente un elemento.
- Inspeccionar el vector de izquierda a derecha hasta encontrar un elemento menor que el pivot.
- Inspeccionar el vector de derecha a izquierda encontrando un elemento mayor al pivot.
- Intercambiar los dos componentes y se continúa este proceso de inspección e intercambio hasta que los recorrido en ambas direcciones se encuentren en algún punto, cercano al centro por lo

general.

- Como resultado se obtiene un vector partido en dos, la parte izquierda con elementos menores que el pivot y la derecha con elementos mayores al pivot.

Por ejemplo, si tenemos los siguientes números 18, 11, 27, 13, 9, 4, 16, los índices i (izquierdo), d (derecho) y m (medio) nos ayudarán a recorrerla.

Posiciones	1	2	3	4	5	6	7	8
	18	11	27	13	9	4	16	
i = 1				m = 4			d = 7	
				$m = (i + d) / 2$				
Intercambio								
	4	11	27	13	9	18	16	
i = 1							d = 7	
Intercambio								
	4	11	9	13	27	18	16	

Su algoritmia se puede representar de dos formas posibles, una iterativa y una recursiva.

Forma Iterativamente del QuickSort

PSEUDO: Pseudocódigo 1 de QuickSort

```
Estructura Vector A[N] tipo numerica
Procedimiento QuickSort
    variable izq tipo numerica
    variable der tipo numerica
    variable med tipo numerica
    variable aux tipo numerica
    izq = 1
    der = i - 1
    med = (izq + der) / 2
    Hacer
        Mientras (A[izq] < A[med]) Hacer
            izq = izq +1
        Fin Mientras
        Mientras (A[der] > A[med]) Hacer
            der = der +1
        Fin Mientras
```

```

Si (izq <= der) Entonces
    aux = A[izq]
    A[izq] = A[der]
    A[der] = aux
    izq = izq + 1
    der = der - 1
Fin Si
Hasta izq > der
Fin Procedimiento

```

En el momento en que $izq > der$, se ha terminado la partición. Se han generado 2 subvectores, que tiene las propiedades citadas anteriormente, pero no está ordenado el vector.

La ordenación de los subvectores implica el mismo proceso que antes, excepto que los índices izquierdo y derecho no tienen los mismos valores.

$4 \quad 11 \quad 9$ $i = 1 \quad d = 3$	$13 \quad 27 \quad 18 \quad 16$ $i = 4 \quad d = 7$
---	--

Se aplica el mismo proceso a cada una de las partes y a continuación a las partes resultantes, hasta que cada partición contenga un único ítem.

Forma Recursivo del QuickSort

PSEUDO: Pseudocódigo recursivo QuickSort

```

Procedimiento QuickSort (variable Izquierda tipo numerica, variable Derecha tipo numerica)
    variable izq tipo numerica
    variable der tipo numerica
    variable med tipo numerica
    variable aux tipo numerica
    izq = Izquierda
    der = Derecha
    med = A[(izq + der) / 2]
    Hacer
        Mientras (A[izq] < med) Hacer
            izq = izq + 1
        Fin Mientras
        Mientras (A[der] > med) Hacer
            der = der - 1
        Fin Mientras
        Si (izq <= der) Entonces
            aux = A[izq]
            A[izq] = A[der]
            A[der] = aux
            izq = izq + 1
            der = der - 1
        Fin Si
    Hasta izq > der
    Si (Izquierda > der) Entonces
        Ejecutar QuickSort(Izquierda, der)
    Fin Si
    Si (izq < Derecha) Entonces

```

```
Ejecutar QuickSort(izq, Derecha)
Fin Si
Fin Procedimiento
```

Primero se deberá determinar la cantidad de elementos a ordenar y, luego, se iniciará el proceso con la primera llamada al procedimiento QuickSort (1, N), donde N es la cantidad de elementos.

El procedimiento QuickSort se activa "él mismo recursivamente". Un subprograma recursivo recordemos que es aquel que se llama a sí mismo, pudiendo pasar parámetros en la llamada si fuese necesario, tal como sucede en nuestro algoritmo.

Cada llamada recursiva toma el elemento medio del vector como referencia y coloca todos los elementos menores a su izquierda y todos los elementos mayores a su derecha; esto permite subdividir el vector en dos partes sobre las cuales se aplica el mismo procedimiento hasta que todo el vector esté ordenado.

La recursividad es una alternativa a la iteración o repetición y, aunque es más eficiente en casos claros de resolución recursiva, también tenemos que tener en cuenta que ocupa más espacio en memoria y corremos el riesgo de quedar en un ciclo o bucle infinito del cual no podremos salir; por ello requiere de un corte de control.

Ordenación externa

Como ya lo explicamos al principio de la herramienta, existen métodos para ordenar fuera de la memoria RAM. Son los casos en que se tienen gran cantidad de elementos, los cuales no pueden ser almacenados en memoria. Estos datos se encuentran en dispositivos de memoria auxiliar, como por ejemplo en discos duros, DVD, ZIP, organizados en estructuras Archivos.

Estos métodos de ordenación, serán tema de la materia Programación Lógica 2, por lo que no desarrollaremos el mismo ahora y sólo nos quedaremos con los vistos en Ordenación Interna.



¡Vamos a comprobar cuánto aprendiste!

1. Indique la opción correcta

Los métodos de ordenación se clasifican en dos categorías: secuenciales y binarios.

- Verdadero
- Falso

2. Indique la opción correcta

El método de ordenamiento "Burbuja con señal" es una optimización del método de la Burbuja.

- Verdadero
- Falso

3. Indique la opción correcta

El método de inserción directa también es conocido como el "Método de la Baraja".

- Verdadero
- Falso

4. Indique la opción correcta

El método QuickSort es un algoritmo que sólo se resuelve recursivamente.

- Verdadero
- Falso

5. Indique la opción correcta

La eficiencia es el factor que mide la calidad y rendimiento de un algoritmo.

- Verdadero
- Falso

Respuestas de la Autoevaluación

1. Indique la opción correcta

Los métodos de ordenación se clasifican en dos categorías: secuenciales y binarios.

Verdadero

Falso

2. Indique la opción correcta

El método de ordenamiento "Burbuja con señal" es una optimización del método de la Burbuja.

Verdadero

Falso

3. Indique la opción correcta

El método de inserción directa también es conocido como el "Método de la Baraja".

Verdadero

Falso

4. Indique la opción correcta

El método QuickSort es un algoritmo que sólo se resuelve recursivamente.

Verdadero

Falso

5. Indique la opción correcta

La eficiencia es el factor que mide la calidad y rendimiento de un algoritmo.

Verdadero

Falso

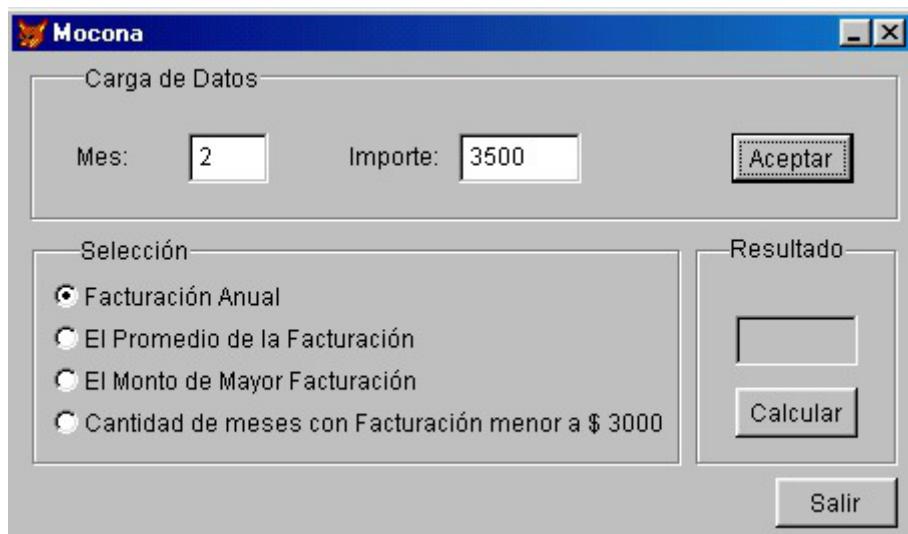
SP5 / Ejercicio resuelto

La empresa Mocona tiene cargado en memoria el importe total de la facturación mensual, en un vector de doce posiciones. Cada elemento del vector almacena el importe de la facturación mensual. El mes indica la posición índice dentro del vector.

Desarrolle el Pseudocódigo que permita calcular:

- La Facturación Anual
- El Promedio de la Facturación
- El Monto de Mayor Facturación
- Cantidad de Meses con Facturación menor a \$3.000

Modelo de la Interfaz Gráfica para resolver la situación planteada:



PSEUDO: Pseudocódigo de ejercicio resuelto de Mocona

```
Programa Mocona
  'Definición de Datos Globales
  Estructura Vector facturacion[12] tipo numerico
  'Definición del Formulario
  Formulario frmFacturacion
    Marco mrcCargaDatos
      CajaDeTexto txtMes
      CajaDeTexto txtImporte
      BotonComando cmdAceptar
    Fin Marco
    Marco mrcSelección
      BotonDeOpcion optFactAnual = VERDADERO
      BotonDeOpcion optPromedio = FALSO
      BotonDeOpcion optMayorFact = FALSO
      BotonDeOpcion optCantidad = FALSO
    Fin Marco
    Marco mrcResultado
      Etiqueta lblResultado
```

```

        BotonComando cmdCalcular
    Fin Marco
    BotonComando cmdSalir
Fin Formulario

'Desarrollo de los procedimientos de Eventos
Procedimiento cmdAceptar_Click
    'Validación de los datos de entrada
    Si (txtMes = 0 OR txtImporte = 0) Entonces
        Mensaje
            Título : "Error en Datos de Ingreso"
            Icono : Advertencia
            Texto : "El mes debe ser una valor comprendido entre 1 y 12, y
                    el importe debe ser mayor a 0"
        Fin Mensaje
    Si No
        'Se carga el importe en la posición indicada
        facturacion[txtMes] = txtImporte
        'Se inicializan los controles para la carga de otro dato
        txtMes = 0
        txtImporte = 0
    Fin Si
Fin Procedimiento
Procedimiento cmdCalcular_Click
Segun Sea
    Cuando optFactAnual.valor = VERDADERO
        Ejecutar Procedimiento FactAnual()
    Cuando optPromedio = VERDADERO
        Ejecutar Procedimiento Promedio()
    Cuando optMayorFact = VERDADERO
        Ejecutar Procedimiento Mayorfact()
    Cuando optCantidad = VERDADERO
        Ejecutar Procedimiento Cantidad()
    Fin Segun Sea
Fin Procedimiento
Procedimiento cmdSalir_Click
    Ejecutar Procedimiento FinPrograma
Fin Procedimiento

Procedimiento FinPrograma
    Salir
Fin Procedimiento
'Definición de los Procedimientos Definidos por el Usuario
Procedimiento FactAnual( )
    variable mes = 1 tipo numerica
    variable suma = 0 tipo numerica
    'Recorremos todo el arreglo y acumulamos el importe almacenado en cada elemento
    Mientras (mes <= 12) Hacer
        suma = suma + facturacion[mes]
        mes = mes + 1
    Fin Mientras
    lblResultado = suma
Fin Procedimiento
Procedimiento Promedio()
    variable mes = 1 tipo numerica

```

```

variable suma = 0 tipo numerica
variable promedio = 0 tipo numerica
Mientras (mes <= 12) Hacer
    Suma = suma + facturacion[mes]
    mes = mes + 1
Fin Mientras
promedio = suma / 12
lblResultado = promedio
Fin Procedimiento
Procedimiento Cantidad()
    variable mes,cantidad tipo numerica
    mes = 1
    cantidad = 0
    'Recorremos todo el arreglo y contamos cada elemento que sea menor a 3000
    Mientras (mes <= 12) Hacer
        Si (facturacion[mes] < 3000) Entonces
            cantidad = cantidad + 1
        Fin Si
        Mes = mes + 1
    Fin Mientras
    lblResultado = cantidad
Fin Procedimiento
Procedimiento MayorFact()
    variable mes = 1 tipo numerica
    variable ImpoMayor = 0 tipo numerica
    'Para determinar el importe de mayor de facturación utilizamos una variable
    'auxiliar que almacena el importe mayor y que se utiliza para comparar el
    'valor con cada elemento del arreglo, de dicha comparación surgirá el importe mayor
    Mientras (mes <=12) Hacer
        Si (facturacion[mes] > ImpoMayor) Entonces
            ImpoMayor = facturacion[mes]
        Fin Si
        mes = mes + 1
    Fin Mientras
    lblResultado = ImpoMayor
Fin Procedimiento
Procedimiento Inicializar-Interfaz()
    variable mes = 1 tipo numerica
    txtMes = 0
    txtImporte = 0
    lblResultado = 0
    'Blanqueamos el arreglo antes de comenzar a trabajar
    Mientras (mes <= 12) Hacer
        facturacion[mes] = 0
        mes = mes + 1
    Fin Mientras
    optFactAnual = VERDADERO
Fin Procedimiento
Fin Programa

```

SP5 / Ejercicios por resolver

1. Realice un programa que permita mostrar el promedio de ventas anual de un comercio. La información ya está cargada en un vector de doce posiciones. Es necesario que también se muestre el monto mayor de venta y el mes en el que se realizó (considere una relación directa entre número de mes y posición del vector).
2. Realice un programa que permita mostrar el promedio de ventas anual de un comercio. La información ya está cargada en un vector de doce posiciones. Es necesario que también se muestre el mes en el que se realizó la venta mayor y el mes en el que se realizó la menor venta (considere una relación directa entre número de mes y posición del vector).



¿Te animás a medir cuánto aprendiste?

1. Indique la opción correcta

Un vector es un conjunto de datos homogéneos. Homogéneo porque todos los datos son numéricos.

- Verdadero
- Falso

2. Indique la opción correcta

La manera más óptima de recorrer un vector de manera completa es con un estructura repetitiva.

- Verdadero
- Falso

3. Indique la opción correcta

En la búsqueda binaria en vectores se debe conocer el valor de todos los elementos.

- Verdadero
- Falso

4. Indique la opción correcta

Los métodos de ordenación se clasifican en dos categorías: Internos y Externos.

- Verdadero
- Falso

5. Indique la opción correcta

La memoria RAM disponible, mide la calidad y rendimiento de un algoritmo de ordenamiento.

- Verdadero
- Falso

Respuestas de la Autoevaluación

1. Indique la opción correcta

Un vector es un conjunto de datos homogéneos. Homogéneo porque todos los datos son numéricos.

Verdadero

Falso

2. Indique la opción correcta

La manera más óptima de recorrer un vector de manera completa es con un estructura repetitiva.

Verdadero

Falso

3. Indique la opción correcta

En la búsqueda binaria en vectores se debe conocer el valor de todos los elementos.

Verdadero

Falso

4. Indique la opción correcta

Los métodos de ordenación se clasifican en dos categorías: Internos y Externos.

Verdadero

Falso

5. Indique la opción correcta

La memoria RAM disponible, mide la calidad y rendimiento de un algoritmo de ordenamiento.

Verdadero

Falso

Situación profesional 6: Estructuras de Datos Homogéneas: Matrices

Empresa La Ganadora

La empresa La Ganadora S.A. dispone de 4 sucursales y 3 empleados en cada sucursal. Le solicita a usted la confección y la programación de los controles de una interfaz que permita el ingreso de las ventas realizadas, indicando código de sucursal, código de vendedor y el importe de la venta.

Además, se deberá dar la posibilidad de emitir los siguientes informes:

- Listado General: imprimir código de sucursal, código de vendedor y total vendido.
- Listado por Sucursal: imprimir código de sucursal y total vendido. En este informe deberá informarse, además la sucursal que más vendió.

SP6 / H1: Estructura de dato homogénea bidimensional

Matriz o Array Bidimensional

Hasta este momento hemos visto ejemplos de programas que requieren el empleo de datos simples y de vectores y habíamos comentado que existen diferentes estructuras de datos que el programador conoce y decide utilizar según las características del programa.

Si nos ubicamos en nuestra situación profesional, en donde tenemos que manejar las ventas de las sucursales de la empresa. Cada sucursal tiene 3 empleados de los cuales registramos las ventas que realiza cada empleado.

Ahora haga la siguiente reflexión:

- ¿Habrá que definir doce variables? Teniendo en cuenta que son 4 sucursales y cada una tiene 3 vendedores.
- ¿Será mejor definir 4 vectores? Un vector para cada sucursal. Y que cada vector tenga la capacidad necesaria para almacenar las ventas de los tres empleados.
- ¿O es mejor definir tres vectores? Un vector para cada vendedor. Y que cada uno almacene las 4 ventas de la sucursal.
- ¿Existe un modo más simple para manejar una situación como esta?

La respuesta a la última pregunta es "sí". Efectivamente existe una estructura de datos indicada para la resolución de esta situación: Vector Bidimensional o Matriz, también llamado "vector de vector".

Este último nombre resulta más ilustrativo con respecto a la definición de una matriz.

Rescatemos dos de las preguntas hechas anteriormente:

- ¿Será mejor definir 4 vectores? ¿O es mejor definir 3 vectores?
Le diríamos, defina un vector de 4 posiciones, pero que cada posición no almacene una variable, sino que cada posición del vector contenga un vector de dimensión tres.

De este modo tenemos un vector con 4 lugares, y en cada lugar se almacenarán tres datos.

En realidad se observa como una cuadrícula o tabla de 4 filas por 3 columnas.

En la Situación Profesional anterior dijimos que un vector es un conjunto de datos homogéneos (que pueden ser simples o estructurados). Al conjunto se le asigna un nombre y a cada elemento del conjunto se lo puede procesar en forma independiente.

Para trabajar con un vector bidimensional (o matriz) en un programa es fundamental definirlo previamente. Se debe tener en cuenta que la matriz tiene las siguientes características:

- **Nombre:** si no se le asigna un nombre es imposible trabajar con la misma en el programa. Recuerde que es conveniente asignarle un nombre significativo, por ejemplo: Notas.
- **Tipo de dato:** todos los elementos constituyentes del vector bidimensional serán del mismo tipo de dato. Ya hemos dicho en la definición que es una estructura homogénea. Para el ejemplo que estamos trabajando lo ideal es que el vector sea tipo numérico.

- **Dimensión:** indica la cantidad de elementos constituyentes del vector bidimensional. Pero se debe prestar especial atención al modo en que se realiza esta definición, ya que si estamos hablando de un vector bidimensional, es obvio que tiene dos dimensiones. Esto significa que cada elemento del vector será accedido indicando el número de la fila y de la columna a la que pertenece. Observe en el siguiente diagrama:



Cuando en un programa se define un vector bidimensional, debe indicarse siempre el nombre, el tipo de dato y las dimensiones. Veamos algunos ejemplos de definición de vectores bidimensionales:

En nuestra situación la definición del vector que almacenará las ventas de 3 empleados que trabajan en 4 sucursales:

Estructura Vector Ventas [4] [3] tipo numérico

_____ { Nombre Dimension Tipo de Dato }

¿ Cómo se hace para ingresar un dato en el vector bidimensional o matriz?

Para ingresar datos en el vector es fundamental trabajar con las posiciones de los elementos constituyentes del mismo. Para lograrlo es imprescindible trabajar con dos índices.

Ventas[1][2] Señala las ventas del segundo vendedor de la sucursal 1.

Ventas[3][1] Señala las ventas del primer vendedor de la sucursal 3.

Ventas[4][3] Señala las ventas del tercer vendedor de la sucursal 4.

Combinación de estructuras de procesamiento para el tratamiento de Matrices

Acceso a los datos almacenados en una matriz

Para recorrer una matriz será necesario definir dos variables, que se utilizarán como índices, una para subindicar la fila, y otra para la columna.

Observe la siguiente definición:

Estructura Vector Ventas[4][3] tipo numerico

Variable Numérica f = 1

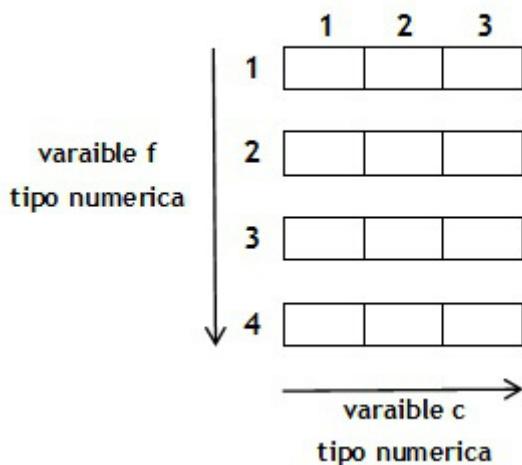
Variable Numérica c = 1

Ahora sólo falta pensar un algoritmo que genere el movimiento del índice a medida que muestra el contenido de la matriz. Ahora bien, será necesario contestar a las siguientes preguntas:

- ¿En qué posición se debe comenzar?
Generalmente se comienza el recorrido en la primer posición. Fila 1 y columna 1.
- ¿Hasta cuando se debe avanzar?
Si la estructura está completa, se listarán los datos hasta llegar a la última posición. Si tenemos en cuenta la situación profesional, será la celda de la cuarta fila y tercer columna.
- ¿Cómo se modifica el valor de los índices para recorrer todas las celdas de la matriz?
Para modificar el valor de los índices se tendrá que tener en cuenta cómo se recorrerá la matriz.
Esto significa que previamente se debe determinar si se muestra primero una fila, y luego la siguiente hasta llegar a la última, o si se muestra primero una columna y luego las demás.

Observe el siguiente diagrama:

Estructura Vector Ventas[4][3] tipo numérico



Veamos los siguientes recorridos:

Ejemplo 1

Imprimir los datos de la matriz de la situación profesional, fila por fila, utilizando una estructura repetitiva "Mientras". La matriz ya está cargada, y es la siguiente:

Estructura Vector Ventas[4][3] tipo numérico

	c ↓	1	2	3
f →	1	1300	5342	6324
	2	923	103	2342
	3	3452	9323	843
	4	432	4342	3423

Para recorrerla utilizamos las siguientes variables como subíndices para acceder, respectivamente, a sus filas y columnas:

variable f tipo numerica

variable c tipo numerica

Para mostrar las ventas de una fila, por ejemplo la primera, se podría hacer lo siguiente:

PSEUDO: Procedimiento Listar

```

Procedimiento Listar ()
    f = 1
    c = 1
    Mientras ( c <=3) Hacer
        Imprimir Ventas[f][c]
        c = c + 1
    Fin Mientras
Fin Procedimiento

```

Pero para listar todas las filas, este procedimiento se deberá repetir diez veces. En ese caso hace falta una repetitiva más.

El programa siguiente controla que se listen las 3 filas, para eso requiere de un índice con el que apuntamos el número de fila que comienza con valor uno y termina con valor 5. Por cada fila otro índice debe moverse desde la columna 1 hasta la 3 inclusive para que sea posible mostrar todas las notas de cada alumno. Como son dos los índices que se deben utilizar será necesario emplear dos estructuras repetitivas.

PSEUDO: Programa ListarVentas

```

Programa ListarVentas
    ' Definición del Formulario
    Formulario frmListarVentas
        BotonComando cmdListar
    Fin Formulario
    ' Definición de Datos Globales
    Estructura Vector Ventas[4][3] tipo numerico
    variable f tipo numerica
    variable c tipo numerica
    ' Desarrollo de los Procedimientos de Evento
    Procedimiento cmdListar_Click
        f = 1
        Imprimir "Ventas de las Sucursales de La Ganadora"
        Imprimir SaltoDeLinea
        'Controla que se listen las cuatro filas

```

```

Mientras (f <= 4) Hacer
    ' Por cada cambio de fila es necesario volver a la columna 1
    c = 1
    ' Controla que se listen las tres ventas de la fila f
    Mientras (c <= 3) Hacer
        Imprimir Ventas[f][c]
        ' cambio de columna
        c = c + 1
    Fin Mientras
    Imprimir SaltoDeLinea
    ' Cambio de fila
    f = f + 1
Fin Mientras
Fin Procedimiento
Fin Programa

```

Ejemplo 2

Mostraremos los datos del vector de nuestra situación utilizando una estructura repetitiva "Para".

PSEUDO: Programa ListarVentas

```

Programa ListarVentas
    ' Definición del Formulario
    Formulario frmListarVentas
        BotonComando cmdListar
    Fin Formulario
    ' Definición de Datos Globales
    Estructura Vector Ventas[4][3] tipo numerico
    variable f tipo numerica
    variable c tipo numerica
    ' Desarrollo de los Procedimientos de Evento
    Procedimiento cmdListar_Click
        Imprimir "Ventas de las Sucursales de La Ganadora"
        Imprimir SaltoDeLinea
        Para f = 1 Hasta 4 Hacer
            Para c = 1 Hasta 3 Hacer
                Imprimir Ventas[f][c]
            Fin Para
        Fin Para
    Fin Procedimiento
Fin Programa

```

Veremos un ejemplo en donde trabajaremos con un recorrido columna por columna. Ya que nuestra situación no maneja datos relacionados en las columnas usaremos otra situación.

Ejemplo 3

Se dispone de un vector en donde se almacenan 3 notas de 10 alumnos de un curso. En la columna 1 se almacena la nota del parcial 1, en la columna 2 se almacena la nota del parcial 2 y en la tercera columna la nota del recuperatorio.

Se necesita realizar una programa que permita imprimir el promedio del curso en cada uno de los exámenes.

Es importante tener en cuenta que para obtener el promedio del curso del primer parcial se necesita recorrer la primera columna completamente, acumulando las notas, para después dividir por la cantidad de alumnos.

Posteriormente se realizará lo mismo con la segunda columna, y por último con la tercera.

En este caso el recorrido de la matriz se realiza “columna por columna”, a diferencia de los ejemplos anteriores, donde se recorrió: “fila por fila”.

PSEUDO: Programa ObtenerPromedio

```
Programa ObtenerPromedio
    ' Definición del Formulario
    Formulario frmObtenerPromedio
        BotonComando cmdCalcular
    Fin Formulario
    ' Definición de Datos Globales
    Estructura Vector Notas[10][3] tipo alfanumerico
    variable f tipo numerica
    variable c tipo numerica
    ' Desarrollo de los Procedimientos de Evento
    Procedimiento cmdCalcular_Click
        variable Sumatoria = 0 tipo numerica
        variable Cantidad = 0 tipo numerica
        variable Promedio = 0 tipo numerica
        c = 1
        Imprimir "Promedio del por Examen"
        Imprimir SaltoDeLinea
        Imprimir "Primer Parcial    Segundo Parcial      Recuperatorio"
        Imprimir SaltoDeLinea
        ' Controla que se listen las tres columnas
        Mientras ( c <= 3 ) Hacer
            ' Por cada cambio de columna es necesario volver a la fila 1
            f = 1
            ' Por cada cambio de columna el acumulador se inicializa
            Sumatoria = 0
            ' Por cada cambio de columna el contador se inicializa
            Cantidad = 0
            ' Controla que se acumulen todas las notas de la columna c
            Mientras ( f <=10 ) Hacer
                ' Controla que haya nota
                Si (Notas[f][c] >= 0 AND Notas[f][c] <= 10) Entonces
                    Sumatoria = Sumatoria + Notas [f][c]
                    Cantidad = Cantidad + 1
                Fin Si
                ' Cambio de fila
                f = f + 1
            Fin Mientras
            Promedio = Sumatoria / Cantidad
            Imprimir Promedio
            ' Cambio de columna
            c = c + 1
        Fin Mientras
    Fin Procedimiento
Fin Programa
```

A este mismo programa también lo resolveremos utilizando la repetitiva “Hacer...Hasta”.

Ejemplo 4

Mostrar el promedio del curso por examen utilizando la estructura de programación “Para”.

PSEUDO: Programa ObtenerPromedio

```
Programa ObtenerPromedio
    ' Definición del Formulario
    Formulario frmObtenerPromedio
        BotonComando cmdCalcular
    Fin Formulario
    ' Definición de Datos Globales
    Estructura Vector Notas[10][3] tipo alfanumerico
    variable f tipo numerica
    variable c tipo numerica
    ' Desarrollo de los Procedimientos de Evento
    Procedimiento cmdCalcular_Click
        variable Sumatoria = 0 tipo numerica
        variable Cantidad = 0 tipo numerica
        variable Promedio = 0 tipo numerica
        c = 1
        Imprimir "Promedio del por Examen"
        Imprimir SaltoDeLinea
        Imprimir "Primer Parcial    Segundo Parcial      Recuperatorio"
        Imprimir SaltoDeLinea
        Para c = 1 Hasta 3 Hacer
            ' Por cada cambio de columna es necesario volver a la fila 1
            f = 1
            ' Por cada cambio de columna el acumulador se inicializa
            Sumatoria = 0
            ' Por cada cambio de columna el contador se inicializa
            Cantidad = 0
            Para f = 1 Hasta 10 Hacer
                ' Controla que haya nota
                Si ( Notas[f][c] >= 0 AND Notas[f][c] <= 10 ) Entonces
                    Sumatoria = Sumatoria + Notas[f][c]
                    Cantidad = Cantidad + 1
                Fin si
                ' Cambio de fila
                f = f + 1
            Fin para
            ' Cambio de columna
            c = c + 1
        Fin Para
    Fin Procedimiento
Fin Programa
```

¿Para cargar los datos en una matriz también se utilizan las dos repetitivas?

La carga de datos en una matriz tiene muchas características diferentes. Todo depende de la situación a resolver. Veremos algunos casos, los más frecuentes, en sucesivos ejemplos.

Ejemplo 5

Se necesita un programa que almacene las ventas realizadas por los vendedores de 3 sucursales, pero antes de realizar la carga de las mismas es imprescindible que en cada posición de la matriz haya un cero, valor que se tomará por defecto. Se le pide realizar un procedimiento que inicialice la matriz.

Para resolver este problema se requerirá una interfaz gráfica con sólo un botón de comando. Cada vez que el usuario del programa oprima el botón de comando se cargará un cero en todas las posiciones de la matriz.

¿Cuántos ceros se cargan cuando el usuario oprime el botón de comando?. La respuesta es: tantos ceros como posiciones existan en la matriz, por lo que resulta necesario utilizar un algoritmo muy parecido a los anteriores, pero esta vez se asignarán valores en lugar de imprimirse.

PSEUDO: Programa InicializarMatrizDeVentas

```
Programa InicializarMatrizDeVentas
    ' Definición del Formulario
    Formulario frmInicializarVenas
        BotonComando cmdInicializar
    Fin Formulario
    ' Definición de Datos Globales
    Estructura Vector Ventas[4][3] tipo numerico
    variable f tipo numerica
    variable c tipo numerica
    ' Desarrollo de los Procedimientos de Evento
    Procedimiento cmdInicializar_Click
        f = 1
        ' Controla que se inicialicen las cuatro filas
        Mientras (f <= 4) Hacer
            ' Por cada cambio de fila es necesario volver a la columna 1
            c = 1
            ' Controla que se inicialicen las tres ventas de la fila f
            Mientras (c <= 3) Hacer
                Ventas[f][c] = 0
                ' Cambio de columna
                c = c + 1
            Fin Mientras
            ' Cambio de fila
            f = f + 1
        Fin Mientras
    Fin Procedimiento
Fin Programa
```

Ejemplo 6

Se necesita un programa que permita ingresar la venta de los vendedores de 3 sucursales, el usuario ingresará en una interfaz gráfica el número de la sucursal, el número del vendedor y la venta realizada.

Para resolver este problema se requerirá una interfaz gráfica con tres cajas de texto y un botón de comando. Cada vez que el usuario del programa ingrese los datos en las cajas de texto, deberá oprimir el botón de comando para que la nota se cargue en la matriz.

Tenga presente que por cada vez que el usuario oprima el botón de comando, sólo se cargará una nota. Es fundamental a la hora de realizar el algoritmo, porque está indicando que no es necesario la utilización de estructuras repetitivas.

PSEUDO: Programa IngresarVentas

```
Programa IngresarVentas
    ' Definición del Formulario
    Formulario frmIngresarVentas
        CajaDeTexto txtSucursal
        CajaDeTexto txtVendedor
        CajaDeTexto txtVenta
        BotonComando cmdIngresar
    Fin Formulario
```

```

' Definición de Datos Globales
Estructura Vector Ventas[4][3] tipo numerico
variable f tipo numerica
variable c tipo numerica
' Desarrollo de los Procedimientos de Evento
Procedimiento cmdIngresar_Click
    ' Los índices f y c toman las posiciones ingresadas por el usuario.
    f = txtSucursal
    c = txtVendedor
    ' Se asigna a la matriz la venta ingresada en la caja de texto
    Ventas[f][c] = txtVenta
Fin Procedimiento
Fin Programa

```

Búsqueda secuencial

¿Es posible buscar datos en una matriz? Sí, se puede acceder a cualquier dato almacenado en la matriz, pero es importante considerar que las búsquedas dependen mucho del requerimiento. Por consiguiente, cuando se trata de matrices el programador tiene que emplear todo el conocimiento que tiene con respecto a estructuras de programación.

Explicaremos algunas búsquedas a partir de problemas de ejemplos. La numeración de los mismos continúa de la Herramienta anterior:

Ejemplo 7

Se necesita un programa que permita saber cuantos vendedores vendieron \$1000.

Este programa consiste en buscar todos los 1000 de la matriz y a medida que se los encuentra contarlos.

PSEUDO: Matrices Ejemplo 7

```

Programa BuscarLosMil
    'Definición del Formulario
    Formulario frmBuscarLosMil
        BotonComando cmdBuscar
        Etiqueta lblCantidad
    Fin Formulario
    'Definición de Datos Globales
    Estructura Vector Ventas[4][3] tipo alfanumerico
    variable f tipo numerica
    variable c tipo numerica
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdBuscar_Click
        variable cantidad = 0 tipo numerica
        f = 1
        Mientras (f <= 4) Hacer
            'Por cada cambio de fila es necesario volver a la columna 1
            c = 1
            'Controla que se busque en los tres vendedores de la fila f
            Mientras (c <= 3) Hacer
                'Se controla si el dato es igual a 1000
                Si (Ventas[f][c] = 1000) Entonces
                    'Se incrementa el contador
                    cantidad = cantidad + 1

```

```

        Fin Si
        'Cambio de columna
        c = c + 1
    Fin Mientras
    'Cambio de fila
    f = f + 1
Fin Mientras
lblCantidad = cantidad
Fin Procedimiento
Fin Programa

```

Ejemplo 8

Volvamos al ejemplo del curso con 10 alumnos. Se necesita obtener el promedio del curso en un examen elegido por el usuario.

En este caso se requiere una interfaz gráfica con una lista o botones de opción (para elegir el tipo de examen), una etiqueta (para mostrar en ella el promedio) y un botón de comando (para disparar el proceso de cálculo).

En el programa será necesario definir un acumulador para obtener la suma de todas las notas. El resultado final de la suma será dividido por diez para obtener el promedio.

Esta vez el programa no debe recorrer la matriz completa, sólo una columna en particular, que dependerá de la selección del usuario. Observe el programa siguiente:

PSEUDO: Matrices Ejemplo 8

```

Programa CalcularPromedio
    'Definición del Formulario
    Formulario frmCalcularPromedio
        Marco mrcTipoDeExamen
            BotonDeOpcion optParcial1 = VERDADERO
            BotonDeOpcion optParcial2 = FALSO
            BotonDeOpcion optRecuperatorio = FALSO
        Fin Marco
        Marco mrcPromedio
            Etiqueta lblPromedio
        Fin Marco
        BotonComando cmdCalcular
    Fin Formulario
    'Definición de Datos Globales
    Estructura Vector Notas[10][3] tipo alfanumerico
    variable f tipo numerica
    variable c tipo numerica
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdCalcular_Click
        variable sumatoria = 0 tipo numerica
        'Controla los botones de opción para determinar la columna de búsqueda
        Segun Sea
            Cuando optParcial1 = VERDADERO
                'Si fue elegido el primer botón se buscará en la columna 1
                c = 1
            Cuando optParcial2 = VERDADERO
                'Si fue elegido el segundo botón se buscará en la columna 2
                c = 2
            Cuando optRecuperatorio = VERDADERO
                'Si fue elegido el tercer botón se buscará en la columna 3

```

```

c = 3
Fin Segun Sea
f = 1
'Controla que se busque en las diez filas
Mientras (f <= 10) Hacer
    'Se incrementa el acumulador
    sumatoria = sumatoria + Notas[f][c]
    f = f + 1
Fin Mientras
lblPromedio = sumatoria / 10
Fin Procedimiento
Fin Programa

```

Combinación de vectores y matrices

En un programa se puede trabajar con la cantidad de variables, vectores y matrices que se necesiten para resolver la situación que se presente. Esta en la experiencia del programador, el elegir las estructuras adecuadas, consumiendo la menor cantidad de recursos y obteniendo los resultados en el menor tiempo posible.

Observemos la siguiente situación en donde se puede observar un caso en donde se utilizan distintas estructuras de datos.

Para una mayor claridad de la herramienta, no usaremos nuestra situación profesional, sino que retomaremos el ejemplo del curso con alumno utilizado en ejemplos anteriores (la numeración de los ejemplos continúa de las Herramientas anteriores).

Ejemplo 9

Se necesita realizar un programa que imprima las notas de los exámenes parciales de un curso. Tenga en cuenta que los nombres de los alumnos están en un vector ya cargado completamente, y las notas en una matriz. La relación existente entre las dos estructuras es que el primer alumno del vector tiene las notas en la primer fila de la matriz, el segundo alumno, en la segunda fila, y así sucesivamente.

Alumnos [10]										Notas [10][3]										
i →	1	Ana								f →	1	5	6				c ↓	1	2	3
	2	José									2	9	10							
	3	Franco									3	3	9	8						
	4	Gonzalo									4	4	4	3						
	5	Lucas									5	1	3	4						
	6	Daniel									9	8	5							
	7	Carolina									7	7	6							
	8	Javier									8	5	3	6						
	9	Carlos									9	2	5	6						
	10	Enrique									10	1	3	4						

Para indicar el vector y las filas de la matriz se podría utilizar un solo índice, pero para que sea más simple su comprensión se utilizarán todos los índices representados en el diagrama.

PSEUDO: Matrices Ejemplo 9

```

Programa ListarAlumnosYNotas
    'Definición del Formulario
    Formulario frmListarAlumnosYNotas
        BotonComando cmdListar
    Fin Formulario
    'Definición de Datos Globales
    Estructura Vector Alumnos[10] tipo alfanumerico
    variable i tipo numerica
    Estructura Vector numerico Notas[10][3]
    variable f tipo numerica
    variable c tipo numerica
    'Desarrollo de los Procedimientos de Evento
    Procedimiento cmdListar_Click
        i = 1
        f = 1
        Imprimir "Listado General de Alumnos"
        Imprimir SaltoDePagina
        'Controla que se listen los diez alumnos
        Mientras (i <= 10) Hacer
            'Se imprime el nombre del alumno
            Imprimir Alumnos[i]
            'Por cada alumno se debe comenzar en la primer columna de la matriz
            c = 1
            'Controla que se listen las tres notas del alumno
            Mientras (c <= 3) Hacer
                Imprimir Notas[f][c]

```

```

    'Se pasa a la siguiente columna
    c = c + 1
  Fin Mientras
  Imprimir SaltoDePagina
  'El índice pasa al próximo alumno
  i = i + 1
  'El índice pasa a la próxima fila
  f = f + 1
  Fin Mientras
Fin Procedimiento
Fin Programa

```

Ejemplo 10

Se necesita realizar un programa que permita imprimir las notas de los exámenes parciales de un alumno.

Lo más indicado es emplear una lista en la interfaz gráfica, para que el usuario pueda seleccionar el nombre del alumno.

PSEUDO: Matrices Ejemplo 10

```

Programa ConsultarNotas
  'Definición del Formulario
  Formulario frmConsultarNotas
    ListaDesplegable lstNombre
    BotonComando cmdImprimir
  Fin Formulario
  'Definición de Datos Globales
  Estructura Vector Alumnos[10] tipo alfanumerico
  variable i = 1 tipo numerica
  Estructura Vector Notas[10][3] tipo alfanumerico
  variable f tipo numerica
  variable c tipo numerica
  'Desarrollo de los Procedimientos de Evento
  Procedimiento cmdImprimir_Click
    i = 1
    'Busca el nombre en el vector
    Mientras (i < 10 AND Alumnos [i] >= lstNombre) Hacer
      i = i + 1
    Fin Mientras
    'si encuentra el alumno realizará el listado
    Si (Alumnos [i] = lstNombre) Entonces
      Imprimir "Listado de Notas Parciales"
      Imprimir SaltoDeLinea
      Imprimir "Alumno: " & Alumnos[i]
      Imprimir SaltoDeLinea

      Imprimir "Primer Parcial      Segundo Parcial     Recuperatorio"
      c = 1
      'Controla que se listen las tres notas
      Mientras (c <= 3) Hacer
        'Imprime la lista
        Imprimir Notas[f][c]
        c = c + 1
      Fin Mientras
      'Finaliza el listado
      Imprimir SaltoDePagina

```

```
Si No
    Mensaje: "No se encontró el alumno solicitado"
Fin Si
Fin Procedimiento
Fin Programa
```



¿Estás listo para un desafío?

1. Indique la opción correcta

Una Matriz, es un vector de vectores.

- Verdadero
- Falso

2. Indique la opción correcta

La siguiente es una declaración válida de una matriz llamada Ventas: Estructura Vector Ventas[4, 3] tipo numérico.

- Verdadero
- Falso

3. Indique la opción correcta

Para recorrer una matriz de manera completa necesitamos de 2 índices, uno que apunte a las filas y otro que apunte a las columnas.

- Verdadero
- Falso

4. Indique la opción correcta

Para recorrer una fila de una Matriz, solo necesitamos de una estructura repetitiva.

- Verdadero
- Falso

5. Indique la opción correcta

Utilizando 2 índices, podemos recorrer una Matriz y dos Vectores.

- Verdadero
- Falso

Respuestas de la Autoevaluación

1. Indique la opción correcta

Una Matriz, es un vector de vectores.

Verdadero

Falso

2. Indique la opción correcta

La siguiente es una declaración válida de una matriz llamada Ventas: Estructura Vector Ventas[4, 3] tipo numérico.

Verdadero

Falso

3. Indique la opción correcta

Para recorrer una matriz de manera completa necesitamos de 2 índices, uno que apunte a las filas y otro que apunte a las columnas.

Verdadero

Falso

4. Indique la opción correcta

Para recorrer una fila de una Matriz, solo necesitamos de una estructura repetitiva.

Verdadero

Falso

5. Indique la opción correcta

Utilizando 2 índices, podemos recorrer una Matriz y dos Vectores.

Verdadero

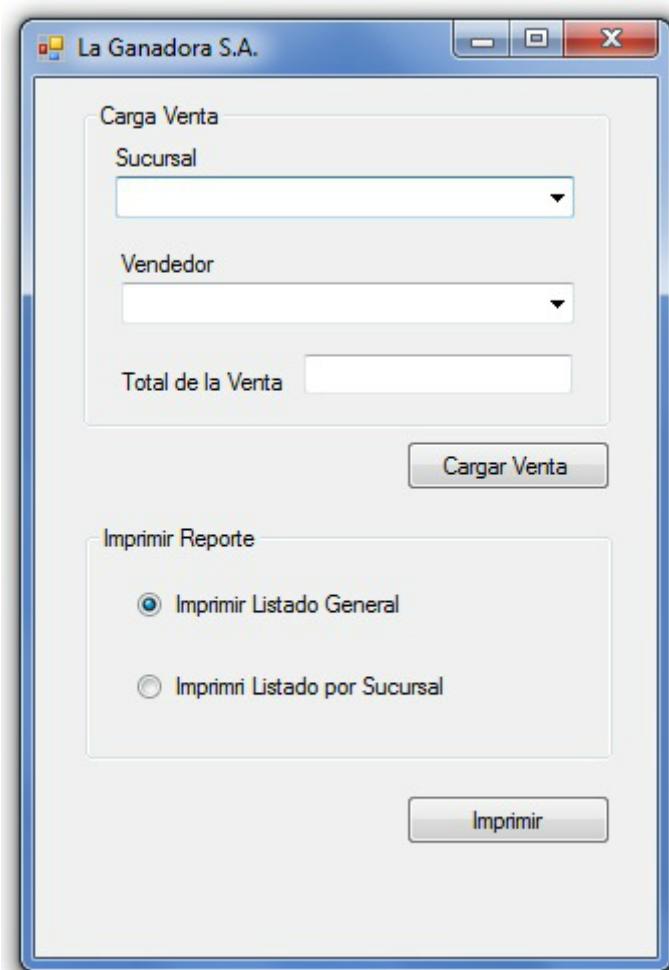
Falso

SP6 / Ejercicio resuelto

La matriz tendrá la siguiente forma:

		c
		↓
		1 2 3
f →	1	1300 5342 6324
	2	923 103 2342
	3	3452 9323 843
	4	432 4342 3423

Y la interfaz gráfica:



Veamos, a continuación, los controles de la interfaz que resuelven la situación planteada:

PSEUDO: SP6 Resolucion

```
Programa LaGanadora
    Estructura Vector Ventas[4][3] tipo numerico
    'Declaración del Formulario
    Formulario Ventas
        Marco Ventas
            ListaDesplegable lstSucursal
            ListaDesplegable lstVendedor
            CajaDeTexto txtVenta
        Fin Marco
        BotonComando cmdCargar
        Marco Reportes
            BotonDeOpcion optGral
            BotonDeOpcion optSuc
        Fin Marco
        BotonComando cmdImprimir
    Fin Formulario
    Procedimiento cmdCargar_Clink
        'El código de la sucursal nos indica la fila en donde grabaremos
        'Mientras que el código de vendedor la columna
        Vector[lstSucursal][lstVendedor] = txtVenta
    Fin Procedimiento
    Procedimiento cmdImprimir_Clink
        'Indice a las filas (sucursales)
        variable suc tipo numerica
        'Indice a las columnas (vendedores)
        variable ven tipo numerica
        variable TotalSucursal tipo numerica
        variable may tipo numerica
        variable sucMay tipo numerica
        Si (optGral = VERDADERO) Entonces
            Imprimir "LISTADO GENERAL"
            Imprimir "Suc. Vend. Total "
            Mientras (suc <= 4) Hacer
                Mientras (ven <= 3) Hacer
                    Imprimir suc & " " & ven & " " & Ventas[suc][ven]
                    ven = ven + 1
                Fin Mientras
                suc = suc + 1
            Fin Mientras
        Si No
            may = 0
            sucMay = 0
            TotalSucursal = 0
            Imprimir "LISTADO POR SUCURSAL"
            Imprimir "Suc. Total "
            Mientras (suc <= 4) Hacer
                Mientras (ven <= 3) Hacer
                    TotalSucursal = TotalSucursal + Ventas[suc][ven]
                    ven = ven + 1
                Fin Mientras
                'Controlamos si la venta de esta sucursal es la mayor
                Si (TotalSucursal > may) Entonces
                    may = TotalSucursal
                    sucMay = suc
```

```
Fin Si
    Imprimir suc & "    " & TotalSucursal
    TotalSucursal = 0
    suc = suc + 1
Fin Mientras
Imprimir "La sucursal que más vendió fue la " & sucMay
Fin Si
Fin Procedimiento
Fin Programa
```

SP6 / Ejercicios por resolver

1. Una empresa de transporte tiene almacenado en un vector el nombre de las localidades correspondientes a los itinerarios que realiza. Además, en una matriz se almacenan los costos para cada viaje.

El vector almacena diez nombres de localidades. La matriz es de 10×10 . A cada fila de la matriz le corresponde una localidad. Pero a cada columna de la matriz, también le corresponde una localidad.

Consideré, por ejemplo, que el vector y la matriz son los siguientes:

1	Córdoba
2	Jesús María
3	Cosquín
4	Alta Gracia
5	Arroyito
6	Río IV
7	Oncativo
8	Carlos Paz
9	Villa Dolores
10	Colonia Caroya

	1	2	3	4	5	6	7	8	9	10
1	0	2	2	1	5	6	4	3	8	2
2	2	0	3	0	6	10	5	0	0	1
3	2	3	0	4	0	0	0	0	8	0
4	1	0	4	0	8	9	7	0	9	2
5	5	6	0	8	0	5	11	12	10	0
6	6	10	0	9	5	0	9	17	21	7
7	4	5	0	7	11	9	0	8	0	9
8	3	0	0	0	12	17	8	0	20	11
9	8	0	8	9	10	21	0	20	0	23
10	2	1	0	2	0	7	9	11	23	0

- Se necesita realizar un programa que permita consultar, por pantalla, el costo de un viaje para un origen y un destino. Por ejemplo, si el usuario necesita el precio para un viaje que tiene como origen Cosquín (posición 3) y como destino: Villa Dolores (posición 9), se le deberá mostrar al usuario \$8 (fila 3, columna 9). Esto significa que cada fila de la matriz se corresponde con los orígenes. Y cada columna, con los destinos. Pero las tarifas cargadas en la matriz corresponden sólo a la carga mínima (1 bulto). El usuario deberá también indicar la cantidad de paquetes (bultos) que serán transportados. Entonces, si de Cosquín a Villa Dolores se transportaran 4 paquetes, el costo total será: \$ 32 (\$8 * 4). Es necesario también que diseñe la interfaz gráfica necesaria. Tenga en cuenta que el usuario deberá ingresar: Origen, Destino y Cantidad de Bultos; y el programa deberá mostrarle: Tarifa Mínima y Costo Total.
- Se necesita un programa que procese consultas por pantalla de los viajes que se realizan hasta una ciudad en particular. Por ejemplo: si el usuario quiere saber desde dónde llegan viajes a la ciudad de Arroyito, se le deberá mostrar el nombre de todos los orígenes y sus correspondientes precios (Tarifas mínimas). Vea ejemplo: Tenga en cuenta que si en la matriz aparece como precio un cero, eso significa que es un viaje que no se realiza; por lo tanto no deberá mostrarse en el listado. Es necesario que diseñe la interfaz gráfica necesaria. Tenga en cuenta que el usuario deberá ingresar un Destino y el programa deberá mostrarle muchos: Orígenes y Tarifas Mínimas.

2. En los sistemas de una empresa constructora se cuenta con una matriz llamada Insumos, de 9 filas por 12 columnas, y un vector llamado Precios, de 12 posiciones.

Matriz: Insumos (9 x 12)

	Ins1	Ins2	Ins3	Ins4	Ins5	Ins6	Ins7	Ins8	Ins9	Ins10	Ins11	Ins12
Prod1												
Prod2												
Prod3												
Prod4												
Prod5												
Prod6												
Prod7												
Prod8												
Prod9												

Esta es una matriz de Insumo / Producto, donde en las filas (9) se encuentran los nueve modelos de vivienda que la firma desarrolla, y en las columnas (12) las cantidades de materiales requeridos para cada una de ellas, considerando de importancia las cantidades de los doce principales materiales de cada construcción.

Vector : Precios (12)

Ins1	Ins2	Ins3	Ins4	Ins5	Ins6	Ins7	Ins8	Ins9	Ins10	Ins11	Ins12

Este vector tiene los precios individuales de los doce insumos considerados en la construcción de una vivienda por parte de la empresa.

Se solicita desarrollar el Pseudocódigo Completo de los procedimientos que generen los siguientes resultados:

- Consultar el costo de una vivienda (producto). El usuario podrá ingresar el código de la misma y se le mostrará solamente el precio final.
- Listar por pantalla (en una grilla) los precios de todas las viviendas que construye la empresa.
- Mostrar el costo de una vivienda en particular detallando en una grilla la cantidad de cada insumo que requiere, el precio por unidad y el costo del insumo requerido.
- Imprimir los costos por vivienda detallando el costo de cada insumo que se utiliza para la construcción de la misma.
- Actualizar los precios de los insumos. El usuario podrá ingresar un porcentaje de aumento y serán actualizados los precios de los doce insumos almacenados en el vector.
- Consultar la cantidad que se requiere de un insumo para cada vivienda. El usuario deberá ingresar el código de insumo y se listará la información por pantalla (en una grilla).

3. En los sistemas de una fábrica de calzado se cuenta con un vector que almacena el nombre de los veinte obreros de la empresa y una matriz que almacena la producción de los mismos.

Vector: Obreros (20)

01 02 03 04 05 06 07 08 09 010 011 012 013 014 015 016 017 018 019 020
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []

Matriz : Producción (20 x 12)

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
Obrero 1												
Obrero 2												
Obrero 3												
Obrero 4												
Obrero 5												
Obrero 6												
Obrero 7												
Obrero 8												
Obrero 9												
Obrero 10												
Obrero 11												
Obrero 12												
Obrero 13												
Obrero 14												
Obrero 15												
Obrero 16												
Obrero 17												
Obrero 18												
Obrero 19												
Obrero 20												

Esta matriz tiene una fila para cada obrero y una columna para cada mes. En la matriz se almacenará la cantidad producida por cada obrero en cada mes.

Se solicita desarrollar el pseudocódigo completo de los procedimientos que generen los siguientes resultados:

- Consultar la producción de un obrero en particular. El usuario seleccionará el nombre del obrero en una lista y se detallará por pantalla: Nombre del mes y cantidad producida por mes del obrero seleccionado.
- Listar por pantalla (en una grilla) los totales producidos por cada obrero. La grilla tendrá las siguientes columnas: Nombre obrero y Total producido.
- Mostrar la producción de cada obrero en un mes en particular. El usuario seleccionará el nombre del mes en una lista y se desplegará en la grilla: Nombre del obrero y cantidad producida.

- Imprimir detalladamente la producción de cada obrero, especificando: Nombre de obrero, nombre de mes, producción mensual y producción anual.
- Consultar el mejor obrero de un mes. El usuario seleccionará de una lista el nombre de un mes y se le mostrará por pantalla el nombre del obrero y la cantidad producida.
- Consultar el promedio mensual de producción por obrero. En una grilla se mostrará: Nombre de obrero y promedio.

Nota: Tenga en cuenta que la matriz se completará en la medida que avanza el año. Por lo tanto se completarán las doce columnas al finalizar diciembre. Por ejemplo, si estamos en el mes de octubre, las columnas correspondientes a noviembre y diciembre almacenarán cero.

SP6 / Evaluación de paso



¡Vamos a comprobar cuánto aprendiste!

1. Indique la opción correcta

Una de las diferencias entre Matrices y Vectores, es que en una matriz no se puede buscar un valor debido a la dimensión que tiene.

- Verdadero
- Falso

2. Indique la opción correcta

En una Matriz nunca tendremos datos alfanuméricos.

- Verdadero
- Falso

3. Indique la opción correcta

Para recorrer por completo una matriz, necesitamos de un índice.

- Verdadero
- Falso

4. Indique la opción correcta

Es imposible sumar la diagonal de una matriz cargada con números, debido a la manera en que se recorre una matriz.

- Verdadero
- Falso

5. Indique la opción correcta

Para encontrar el valor máximo de una matriz, necesitamos recorrer todas sus posiciones.

- Verdadero
- Falso

Respuestas de la Autoevaluación

1. Indique la opción correcta

Una de las diferencias entre Matrices y Vectores, es que en una matriz no se puede buscar un valor debido a la dimensión que tiene.

- Verdadero
- Falso

2. Indique la opción correcta

En una Matriz nunca tendremos datos alfanuméricos.

- Verdadero
- Falso

3. Indique la opción correcta

Para recorrer por completo una matriz, necesitamos de un índice.

- Verdadero
- Falso

4. Indique la opción correcta

Es imposible sumar la diagonal de una matriz cargada con números, debido a la manera en que se recorre una matriz.

- Verdadero
- Falso

5. Indique la opción correcta

Para encontrar el valor máximo de una matriz, necesitamos recorrer todas sus posiciones.

- Verdadero
- Falso

Situación profesional 7: Programación Orientada a Objetos (POO)

Una institución bancaria le solicita a usted, como pasante del área de desarrollo de software, que realice un modelo de objetos que permita realizar las acciones básicas para depositar y extraer dinero de las cajas de ahorro de los clientes, desde un cajero automático.

Como regla para tener en cuenta se establece que el monto de extracción no puede superar el monto de saldo disponible de la caja de ahorro y, que si el retiro es mayor a \$1500, el titular de la cuenta debe estar al día con los impuestos. Se debe mostrar la información del saldo de cada caja de ahorro, su número y el saldo de cada cliente.

El Paradigma OO asume desde su comienzo una definición totalmente distinta a la tradicional:

Un Programa OO es una red de objetos que interactúan entre sí (cooperando), enviándose mensajes (logrando la comunicación entre ellos)

Este concepto surge de la idea natural de un mundo donde existen infinitos objetos, donde la solución a cualquier problema se realizará en términos de objetos. Por ejemplo, una institución bancaria es un objeto que interactúa con los clientes que son objetos, que tienen cuentas que también son objetos; o en la naturaleza, los animales interactúan con su medio, vegetales, personas, cosas, todos son objetos.

De esta manera:

Programa = objetos + mensajes

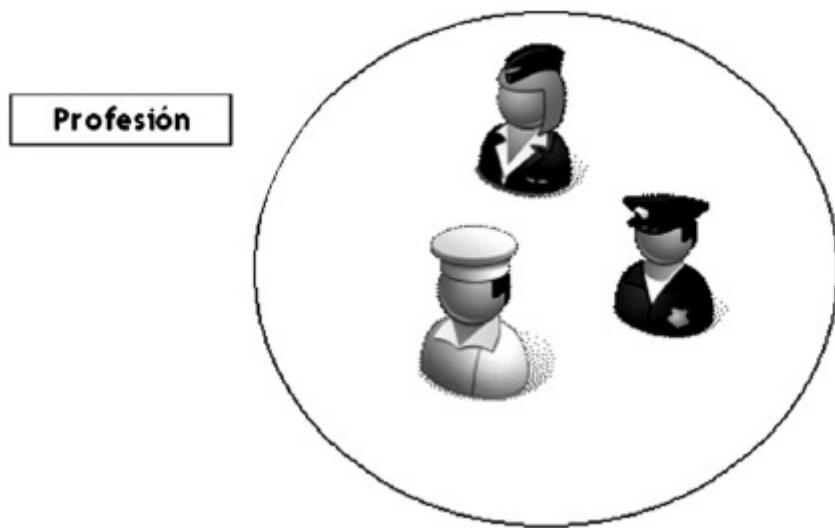
Objetos y clases

La Programación OO principalmente hace uso de:

- clases
- objetos
- relaciones
- instancias
- propiedades
- métodos

Diariamente trabajamos con objetos que representan personas, cosas, hechos, por ejemplo: alumnos, clientes, proveedores, docentes, mesas, sillas, tarjetas de crédito, compras, ventas, inscripciones; todos son objetos.

Cada uno de ellos tiene un propósito que lo identifica y lo diferencia de otros objetos, y pueden clasificarse según sus atributos o funcionalidades. Por ejemplo: cocinero, policía y aviador tienen alguna característica y/o funcionalidad en común, por lo tanto todos pertenecen al conjunto de Profesión.



Esta agrupación es lo que llamamos clase.

Entonces, podemos decir que **clase** es un conjunto de objetos de un mismo tipo (mismos atributos y/o métodos).

En la Programación Orientada a Objetos es necesario primero diseñar un conjunto de clases. Esto permite dar claridad, eficiencia y mantenibilidad al programa resultante, y significa una gran economía en tiempo de desarrollo y mantenimiento de código. La ventaja potencial más importante de un lenguaje OO está en las bibliotecas de clases que se pueden construir para él, lo que permite volver a usarlas en otros sistemas.

Los objetos son las unidades de procesamiento que tienen una aplicación OO y no hay ningún tipo de código fuera de ellos.

Los objetos se crean a partir de una serie de especificaciones o normas que los definen, y esto es lo que en POO se conoce como una clase.

Las clases definen la estructura que van a tener los objetos que se crean a partir de ella, indicando qué propiedades y métodos tendrán los objetos.

Por lo tanto, las clases son definiciones a partir de las que se crean objetos. Los objetos son ejemplares de una clase determinada y, como tal, disponen de los datos y funcionalidades definidos en ella.

- Objeto: entidad presente en la representación del problema.
- Comportamiento: definido por un conjunto de atributos y métodos que pueden ser privados o públicos.

Los atributos o propiedades definen los datos o información del objeto, permitiendo modificar o consultar su estado, mientras que los métodos son las rutinas que definen el comportamiento del objeto.

Por lo tanto:

- Los atributos dan el estado del objeto.
- Los métodos dan el comportamiento del objeto.

Veamos unos ejemplos a continuación:

Ejemplo 1:

- Objeto: Persona
- Atributos: DNI, nombre, apellido, sexo, fecha de nacimiento...
- Métodos: comer, dormir, saltar, caminar, conversar.

Ejemplo 2:

- Objeto: Mesa
- Atributos: alto, tipo, material, color...
- Métodos: sostener.

Si nos basamos en el ejemplo de nuestra situación profesional encontramos objetos como:

- Objeto: CajeroAutomático
- Atributos: idCajero, ubicación, ArrayCajaAhorro
- Métodos: recibirDinero, entregarDinero, getIdCajero, getUbicación, setIdCajero, setUbicación.

y también,

- Objeto: Cliente
- Atributos: código
- Métodos: mostrarDatos, pagoImpuestos, getCodigo, setCodigo.

¿Cuál es la diferencia entre clase y objeto?

- Una clase constituye la representación abstracta de algo.
- Un objeto constituye la representación concreta de lo que la clase define.

Por ejemplo:

Si usted quiere construir una casa, el arquitecto realiza el diseño de la misma y lo plasma en el plano, el cual tiene todos los detalles necesarios: medidas, habitaciones, comodidades, materiales, funcionalidades. Una vez construida la casa, se puede comparar con el plano para ver si responde a las especificaciones del mismo. Llevado a la programación orientada a objetos, el plano es la clase y la casa es el objeto. Si fuese un barrio de casas con las mismas características, se podrían construir N casas con un mismo plano; así, a partir de una clase, se pueden construir N objetos iguales.

Clase: Casa

Objeto: casa

Atributos: materiales, cantidad de dormitorios, dimensiones...

Métodos: proteger, habitar.

Observe que la clase y el objeto tienen el mismo nombre; y por convención, el nombre de la clase se escribe en singular y con la primera letra en mayúscula.

Instancia de una clase

Es la creación de un objeto. En nuestro ejemplo se instancia la clase Casa, cuando se crea cada casa.

Teniendo en cuenta que la Programación OO está pensada para construir objetos que contengan atributos y operaciones de manera tal que cubran nuestras necesidades, los atributos son estructuras de datos que contienen valores del estado de un objeto. Y las operaciones, también conocidas como métodos, funciones y acciones, realizan modificaciones del propio objeto o alguna acción externa a éste.

Para que un objeto realice alguna acción, algún otro objeto se la tiene que solicitar; es por ello que los objetos se comunican entre sí por medio de mensajes.

- Mensaje: solicitud que recibe un objeto para efectuar una tarea.
- Método: algoritmo o subprograma que efectúa la tarea solicitada por medio de un mensaje.

Veamos un ejemplo:

Si el jefe le pide a su secretaria que saque una fotocopia de su DNI, se está estableciendo una comunicación entre el objeto "jefe" y el objeto "secretaria". Es la secretaria quien tiene el conocimiento de cómo sacar una fotocopia, no el jefe. El método "sacarfotocopia" está definido en la clase "Secretaria".

Veamos otro ejemplo:

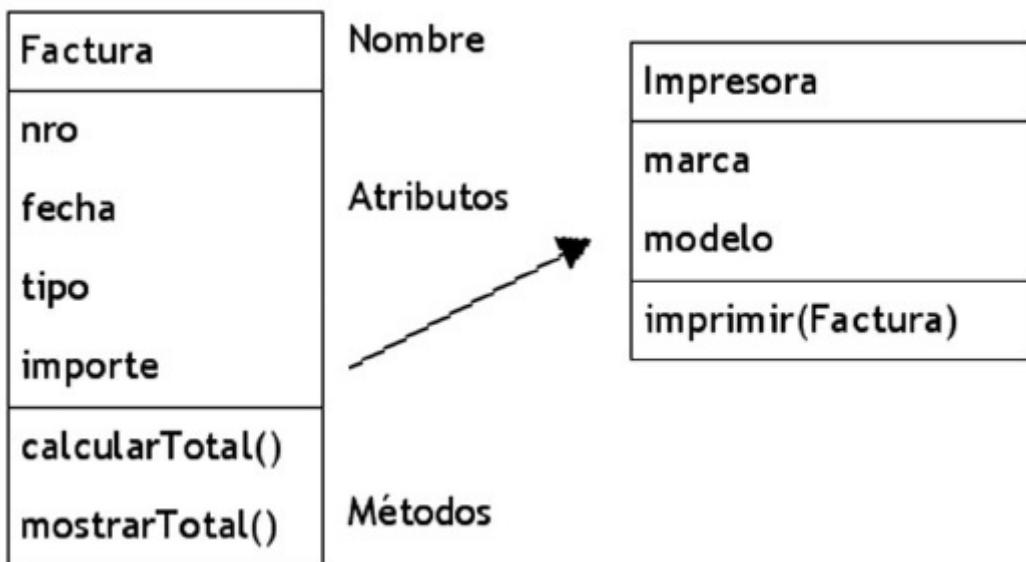
Los objetos factura e impresora se comunican entre ellos cuando es necesario imprimir una factura. Entonces nos preguntamos:

1 ¿Qué objeto tiene el método imprimir()?

2 ¿Qué objeto solicita la impresión?

Parece claro que el objeto impresora posee el método imprimir() ya que es su función, y el objeto factura se comunica solicitando la impresión.

Si representamos estos objetos podríamos hacerlo de la siguiente forma:



De esta manera, reconocemos como características principales de la comunicación entre los objetos:

- La invocación de métodos.
- El medio de colaboración entre objetos.

Un ejemplo de la vida cotidiana sobre esto se nos presenta debido a que estamos acostumbrados a utilizar el control remoto para encender los electrodomésticos y para cambiar su volumen; por lo tanto, comunicamos dos objetos a través de mensajes, una petición que encuentra una respuesta.

La Programación Orientada a Objetos separa estrictamente la noción sobre qué se va a hacer, del cómo se va a hacer.

El "qué" se describe como un conjunto de métodos. Como el contrato entre el diseñador de la clase y el programador que la usa, puesto que dice lo que ocurre cuando se invocan ciertos métodos sobre el objeto.

El "cómo" de un objeto viene dado por su clase, que define la implementación de los métodos que soporta el objeto. Recuerde que "cada objeto es una instancia de una clase".

Comportamiento

Cuando uno piensa en un objeto, tiene que preguntarse ¿para qué sirve?, ¿qué comportamiento tiene?, ¿qué le puedo pedir?, ¿qué servicios provee? y ¿qué sabe hacer el objeto?

Esta es una visión antropomórfica ya que se ve al objeto con cierta capacidad para hacer cosas, porque en el Paradigma de Objetos los que saben hacer algo son ellos.

Cuando se trata de resolver una situación, se deben identificar los objetos como si fueran cosas vivas, aunque al principio se piensa en cosas bastante concretas como un borrador, computadora, cuenta bancaria; pero luego se puede pensar en cosas más abstractas, desde una regla del negocio hasta un algoritmo (pensar en el algoritmo como un objeto al que le puedo pedir que se active y haga algo). La idea de ver a los objetos como entidades con comportamiento es la idea central del Paradigma de Objetos y la Programación Orientada a Objetos.

El comportamiento está expresado por el conjunto de mensajes que el objeto es capaz de responder.

Cuando un objeto recibe un mensaje, ejecuta el código que se llama método (función o procedimiento independiente del lenguaje de programación con el que uno trabaje). Lo importante es que este se activa cada vez que uno de los objetos recibe un mensaje que tiene su interfaz (la interfaz de un objeto está definida por el conjunto de mensajes que puede recibir y ejecutar con sus métodos).

Estado interno/Conocimiento

Los objetos pueden tener un estado interno tal que cuando hablamos de implementación nos podemos referir a que están formados por variables (variables de estado), y que capturan lo que ese objeto conoce (el conocimiento del objeto), o sea sus atributos.

Principios básicos

Los lenguajes OO respetan ciertos principios para poder ser considerados como tales, a saber: Abstracción, Encapsulamiento, Herencia y Polimorfismo.

1. Abstracción

Una clase es una abstracción o una entidad esencial del problema o la solución.

2. Encapsulamiento

Un objeto tiene un estado interno protegido, oculto y privado.

- Ocultación de detalles Vistas: la modularidad y ocultamiento de datos que permite deslindar los cambios de la representación interna del mundo exterior se basan en dos vistas para una clase-objeto.
- La interfaz del objeto (conjunto de métodos) que se ubica en una clase es normalmente pública.
- Independencia: los atributos del objeto que se declaran en una clase son normalmente privados.

Una de las principales ventajas de la Programación OO es el concepto de encapsulamiento, conocido también como protección de datos. Esto significa que solo se pueden modificar los datos de un objeto accediendo a través de sus métodos u operaciones (interfaz del objeto). Nunca se permite modificar directamente desde la aplicación principal.

Como el objeto maneja datos, la funcionalidad está sujeta a ellos. Una ventaja de usar objetos es que podemos modificar su funcionalidad, añadir mejoras o corregir errores sin necesidad de cambiar su interfaz, convirtiéndose en una ventaja, ya que en caso contrario un proyecto estaría sujeto a un mayor número de fallos y los cambios serían más costosos.

En nuestra situación por ejemplo tenemos el objeto **cliente**

- Su atributo es: código
- Sus métodos: mostrarDatos, pagoImpuestos, getCodigo y setCodigo

El atributo es privado, por lo que no se podrá acceder a él desde otra clase, pero si dentro de la misma clase. Sin embargo los métodos son públicos y serán accedidos desde otras clases para manipular los atributos del objeto cliente.

El pseudocódigo de este objeto será:

PSEUDO: Pseudocódigo del objeto cliente de la SP1

Clase Cliente Hereda de Persona

```
variable Privada codigo tipo numerica
'Constructor por defecto
Procedimiento Cliente()
Fin Procedimiento
'Constructor con paso de parámetros.
Procedimiento Cliente(variable DNI tipo numerica, variable nom tipo alfanumerica,
                      variable ape tipo alfanumerica, variable cod numerica)
    super (DNI, cadena, ape)
    Código = cod
Fin Procedimiento
Procedimiento mostrarDatos()
    'Muestra el valor de los atributos de la clase
    Persona.super.mostrarDatos()
    'Muestra el valor de los atributos de la clase
    Cliente.Imprimir ("Código: " + this.getCodigo)
Fin Procedimiento
Procedimiento Propiedad getCodigo() tipo numerico
    Devuelve codigo
Fin Procedimiento
Procedimiento Propiedad setCodigo(variable cod tipo numerica)
    this.codigo = cod
Fin Procedimiento
Procedimiento pagoImpuestos() tipo Logico
    Si pago = VERDADERO
        Devuelve VERDADERO
    Si No
        Devuelve FALSO
    Fin Si
Fin Procedimiento
Fin Clase
```

Veamos otro ejemplo, analicemos el objeto **automóvil**.

- Sus atributos pueden ser: marca, patente, modelo.
- Sus métodos son: arrancar(), frenar(), doblar(), parar()

Los atributos son privados, o sea, se pueden acceder desde adentro de la misma clase, mientras que los métodos son públicos, es decir, se pueden acceder desde otras clases.

Para que pueda accederse a estos atributos desde otras clases, se definen métodos públicos para leer sus valores o para insertarlos. Su definición en pseudocódigo sería:

PSEUDO: Pseudocódigo del objeto automóvil de la SP1

```
Clase Automóvil
    variable Privada marca tipo alfanumerica
    variable Privada patente tipo alfanumerica
    variable Privada modelo tipo alfanumerica
    Procedimiento Publico arrancar()
        Código ...
    Fin Metodo
    Procedimiento Publico frenar()
        Código ...
    Fin Procedimiento
    Procedimiento Publico doblar()
        Código ...
    Fin Procedimiento
    Procedimiento Publico parar()
        Código ....
    Fin Procedimiento
    'Lee el contenido y devuelve el valor al objeto que lo solicitó.
    Procedimiento Propiedad Publico leerMarca() tipo alfanumerico
        Devuelve marca
    Fin Procedimiento
    Procedimiento Propiedad Publico asignarMarca(variable m tipo alfanumerica)
        'Recibe un parámetro y se lo asigna al atributo del objeto.
        marca = m
    Fin Procedimiento
    'Lee el contenido y devuelve el valor al objeto que lo solicitó.
    Procedimiento Propiedad Publico leerPatente() tipo alfanumerico
        Devuelve patente
    Fin Propiedad

    'Recibe un parámetro y se lo asigna al atributo del objeto.
    Procedimiento Propiedad Publico asignarPatente(variable p tipo alfanumerica)
        patente = p
    Fin Procedimiento
    'Lee el contenido y devuelve el valor al objeto que lo solicitó.
    Procedimiento Publico leerModelo() tipo alfanumerico
        Devuelve modelo
    Fin Procedimiento

    'Recibe un parámetro y se lo asigna al atributo del objeto.
    Procedimiento Propiedad Publico asignarModelo(variable mo tipo alfanumerica)
        modelo = mo
    Fin Procedimiento
Fin Clase
```

Es importante desatacar que, por convención, leer es get y asignar es set, por lo tanto se puede usar getModelo() y setModelo (variable cadena mo).

3. Herencia de Clases

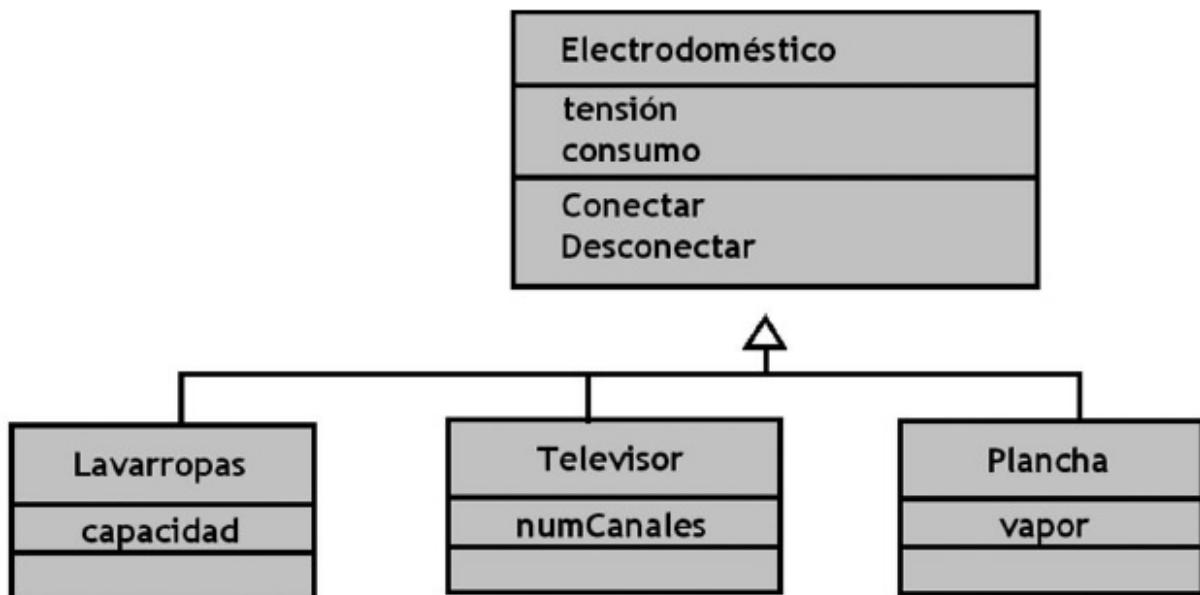
Permite que se puedan definir nuevas clases basadas en otras existentes, lo cual facilita re-utilizar códigos previamente desarrollados. Si una clase deriva de otra, hereda todas sus variables y métodos. La clase derivada puede añadir nuevas variables y métodos, y/o redefinir las variables y métodos heredados.

Permite re-usar código-conocimiento-funcionalidad en una clase-padre que hereda sus atributos y métodos a sus clases-hijas, subclases o clases-tipos derivados.

En el pseudocódigo usado para demostrar el encapsulamiento, también se puede observar que el objeto cliente hereda características del objeto persona, y en su declaración se remarcá la misma de la siguiente manera:

clase Cliente **Hereda de** Persona

Por ejemplo, podemos definir la clase "Electrodoméstico", de la cual heredan las subclases "lavarropas", "televisor", "plancha".



La Super clase "Electrodoméstico" tiene los atributos "tensión" y "consumo", y los métodos "Conectar" y "Desconectar", los cuales heredarán todas las subclases que estén en niveles inferiores. Mientras que las subclases han agregado un atributo que le pertenece a la clase donde están definidas, y que puede heredarse a otra subclase que se desprendiera de alguna de ellas.

La Herencia responde a "es un". En nuestro ejemplo, "lavarropas" es un "Electrodoméstico".

La definición en pseudocódigo de las clases Electrodoméstico y Plancha es:

PSEUDO: Pseudocódigo de las clases electrodoméstico y plancha de la SP1

Clase Electrodoméstico

```
variable Privada tensión tipo numerica  
variable Privada consumo tipo numerica
```

```

Procedimiento conectar()
    Código....
Fin Procedimiento
Procedimiento desconectar()
    Código...
Fin Procedimiento
Fin Clase
Clase Plancha Hereda de Electrodoméstico
    variable vapor tipo Logico
Fin Clase

```

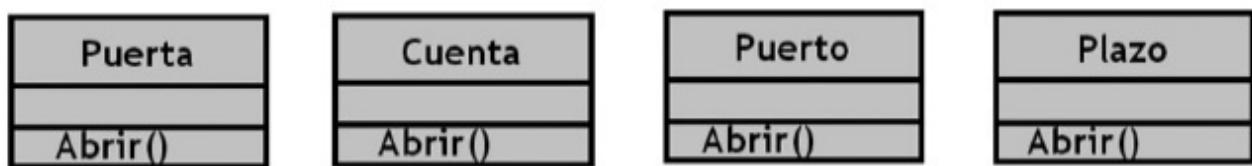
4. Polimorfismo

El Polimorfismo es el grado máximo de excelencia de calidad en POO, permite múltiples implementaciones de métodos, dependiendo del tipo de objeto que se indica al invocar el método correspondiente, lo que posibilita, entonces, que el mismo mensaje enviado a diferentes objetos resulte en acciones dependientes del objeto que recibe el mensaje.

Podemos decir que facilita la adición de capacidades nuevas a un sistema, como lo son el re-uso y la extensión, que proporciona código genérico compuesto por un mensaje polimórfico.

Veamos un ejemplo:

Supongamos que tenemos distintas clases con un mismo método "abrir()"



El mismo nombre de método se encuentra en cada una de las clases pero, dependiendo del tipo de objeto que lo convoque, será el método que se ejecute. Cada uno tendrá su funcionalidad.

También se pueden definir varios métodos con el mismo nombre, aunque con distintas listas de parámetros.

El intérprete utiliza la lista de parámetros para decidir, entre todos los métodos del mismo nombre, cuál debe ejecutar. Comienza por la clase donde se llama y busca en las superclases.

En nuestra situación un ejemplo claro de polimorfismo, es el método mostrarDatos() del objeto cliente y del objeto persona.

Por ejemplo, si definimos el método para que realice la operación suma, este puede recibir dos números enteros, dos números decimales o dos cadenas (para concatenar) y, por lo tanto, tendríamos tres métodos con el mismo nombre, a saber:

Sumar(entero A, entero B),

Sumar(decimal A, decimal B),

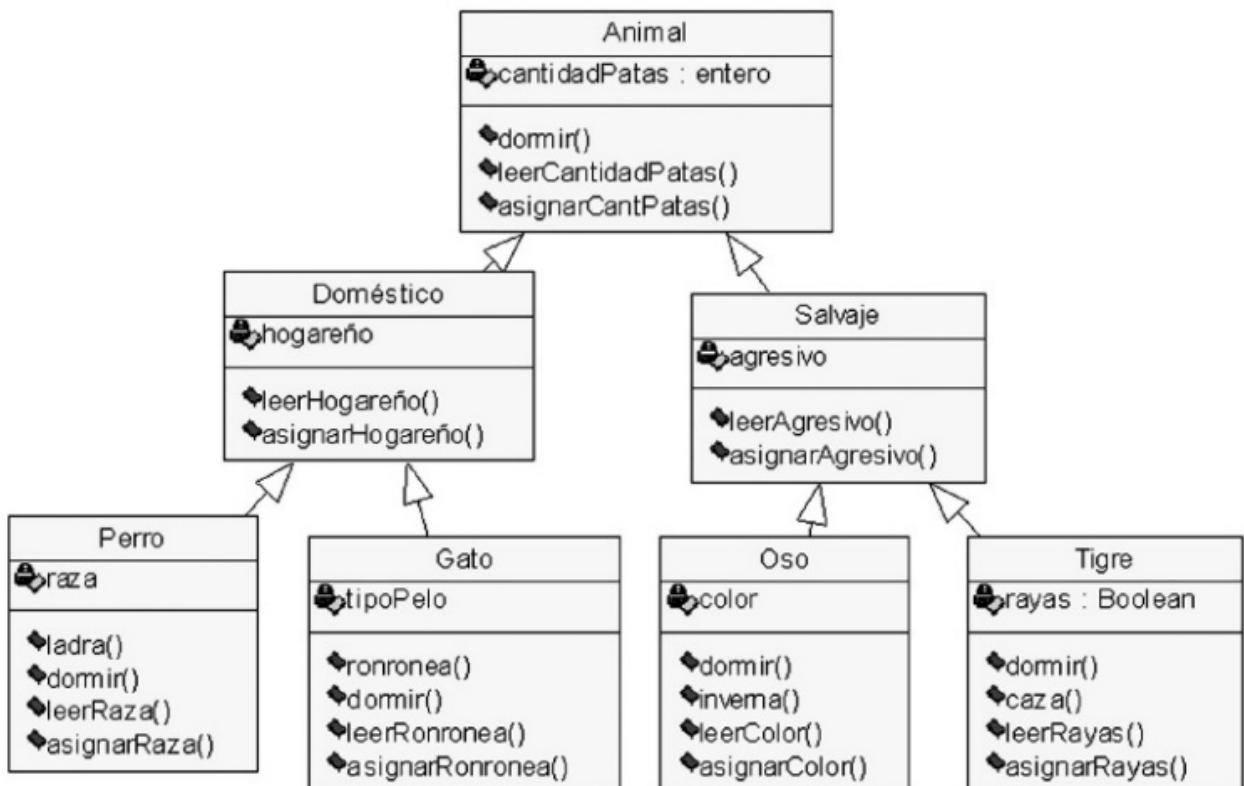
Sumar(cadena A, cadena B)

Se ejecutará el método que corresponda según el tipo y cantidad de parámetros que reciba. Estos métodos pueden estar definidos en una misma clase.

Veamos un ejemplo con herencia, encapsulamiento y polimorfismo. A los animales los podríamos agrupar en

"Domésticos" y "Salvajes"; dentro de los domésticos encontramos al "Perro" y al "Gato"; y dentro del grupo de los salvajes encontramos al "Tigre" y al "Oso". Todos tienen atributos en común (que heredan de Animal) y tienen atributos propios que los diferencian entre sí. Igual razonamiento podemos hacer con los métodos (funciones); por ejemplo, todos duermen, pero cada uno puede hacerlo de una manera distinta.

Gráficamente los podemos representar de la siguiente forma:



Los objetos Perro y Gato heredan los atributos y métodos de Animal y Doméstico.

Los objetos Oso y Tigre heredan los atributos y métodos de Animal y Salvaje.

El método `dormir()` se redefine en cada método, ya que cada uno duerme de una manera distinta.

Los atributos son privados, los métodos son públicos.

A modo de síntesis podemos decir que la mayoría de los lenguajes de programación modernos están orientados a objetos o, en su defecto, se aproximan mucho a éstos, tomando algunas de sus características. Este es el caso de .Net, Java y PHP 5, entre otros.

La Programación Orientada a Objetos permite concebir los programas de una manera bastante intuitiva y cercana a la realidad. La tendencia es que un mayor número de lenguajes de programación adopten esta programación como paradigma para modelar los sistemas. Prueba de ello es la nueva versión de PHP (5), que implanta la programación de objetos como metodología de desarrollo. También Microsoft ha dado un giro hacia la Programación Orientada a Objetos, ya que "punto NET" dispone de varios lenguajes para programar y todos se orientan a objetos. En el caso de Java, nace como orientado a objetos; por lo tanto no se puede trabajar sin definir clases, objetos, atributos y métodos.

Constructor de la clase

Las clases contienen un tipo de función especial que se conoce como constructor, el cual es llamado cuando se crea el objeto.

Las clases solamente son definiciones. Si queremos utilizar la clase, tenemos que crear un ejemplar de dicha clase, lo que corrientemente se le llama instanciar un objeto de una clase. Para ello necesitamos convocar al constructor de la clase.

El constructor se declara de la misma forma que los métodos, pero debe tener el mismo nombre que la clase. Su principal propósito es ser llamado automáticamente cuando un objeto es creado.

El constructor puede ser convocado con o sin parámetros, por defecto no los tiene; por lo tanto, si usted necesita pasarle uno o más parámetros, deberá declarar explícitamente el constructor.

A continuación, veremos cómo se declara el constructor de una clase.

PSEUDO: Declaración del constructor de una clase

```
Clase NombreClase

    variable Privado nombreAtributo tipo .....
    'Constructor de la clase
    Procedimiento NombreClase()
        código
    Fin Procedimiento
Fin Clase

Clase Animal
    'Atributo
    variable Privada cantidadPatas tipo numerica
    'Constructor
    Procedimiento Animal()
        'Inicializa con 0
        cantidadPatas = 0
    Fin Procedimiento
    'Se definen los demás métodos
Fin Clase
```

Toda clase tiene un método constructor por defecto que no recibe parámetros ni inicializa, solo instancia la clase creando el objeto.

Nuestra clase persona tiene un constructor por defecto que recibe parámetros y son declarados de la siguiente manera:

PSEUDO: Constructor por defecto

```
Clase Persona
    'Constructor por defecto
    Procedimiento Persona()
    Fin Procedimiento
    'Constructor con paso de parámetros
    Procedimiento Persona(variable DNI tipo numerica, variable nom tipo alfanumerica,
                           variable ape tipo alfanumerica)
        dni = DNI
        nombre = nom
        apellido = ape
```

Fin Procedimiento
Fin Clase

Para instanciar una clase

Cuando se crean los objetos a partir de las clases, se llama a un constructor que se encarga de inicializar los atributos del objeto y realizar cualquier otra tarea de inicialización que sea necesaria.

No es obligatorio disponer de un constructor, pero resulta muy útil y su uso es muy habitual.

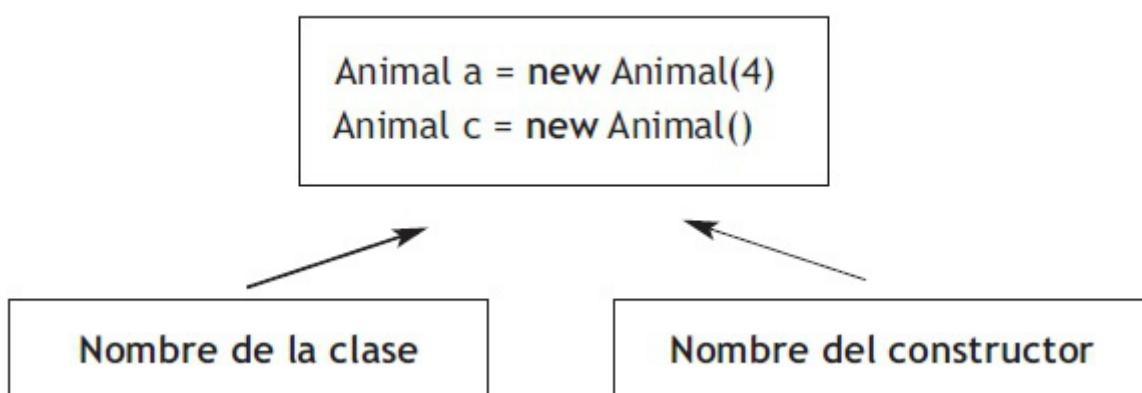
Es muy importante recordar que los objetos son instancias a una clase, por lo tanto cada objeto es único.

En los Programas Orientados a Objetos existe un operador para crear un objeto utilizando una palabra reservada, generalmente new, indicando, además, a qué clase va a instanciar el objeto que creemos y pasarle los parámetros (si los requiere) al constructor.

En nuestra situación, si instanciamos la clase persona con el constructor que recibe parámetros, el código será

```
Persona p = new Persona("21234234", "José", "Mercado")
```

Por ejemplo, instanciamos la clase Animal creando dos objetos, uno con paso de parámetros:



PSEUDO: Objetos 5

```
Clase Animal
    'Constructor con parámetro
    'La variable can recibe el valor 4 y lo asigna al atributo cantidadPatas
    Procedimiento Animal(variable can tipo numerica)
        cantidadPatas = can
    Fin Procedimiento
Fin Clase
```

Cuando creamos un objeto con un constructor por defecto (sin paso de parámetros), tenemos que asignarle los valores en algún momento.

Por ejemplo:

c.cantidadPatas = 4 'Objeto.atributo si está en la misma clase.

Para convocar un método también utilizamos la referencia al objeto. Por ejemplo, para asignar un valor a un atributo desde otra clase se tiene que usar el método asignarCantPatas (var entera cant), de la siguiente forma:

```
c.asignarCantPatas(4)
```

Otra forma de tratar los atributos y métodos de un objeto

Cuando se necesitan atributos o métodos de un objeto recién creado, se puede referenciar con la palabra reservada **this**, por ejemplo:

this.asignarCantPatas(4)

Si instanciamos la clase Perro:

Perro p = new Perro()

Después podemos referenciarlo con **this**:

this.raza = "Pastor alemán"

this.ladra()

Con **this** se referencia al último objeto creado, el activo en memoria; si no, se debe referenciar con su nombre.

Cuando un objeto llama a un método, primero busca en su clase; si no lo encuentra, busca en la superclase hasta que lo encuentra y lo ejecuta.

Cuando el método está en su clase y en la superclase, siempre ejecutará el de su clase, a no ser que se llame directamente al método de la superclase; tal es el caso del método dormir().

Veamos cada caso:

Llama y ejecuta el método dormir de la clase Perro con el objeto p

p.dormir()

Llama y ejecuta el método dormir de la clase Perro con el último objeto creado.

this.dormir()

Llama y ejecuta el método dormir de la superclase Animal con el objeto "p" o this

super.dormir()

Cuando tenemos jerarquía de clases y se definen los constructores con paso de parámetros, la instancia de la subclase recibe todos sus atributos, se dejan los que pertenecen a esa subclase y pasa a la superclase el resto. Por ejemplo, en la jerarquía de Animal y Salvaje, se define de la siguiente forma:

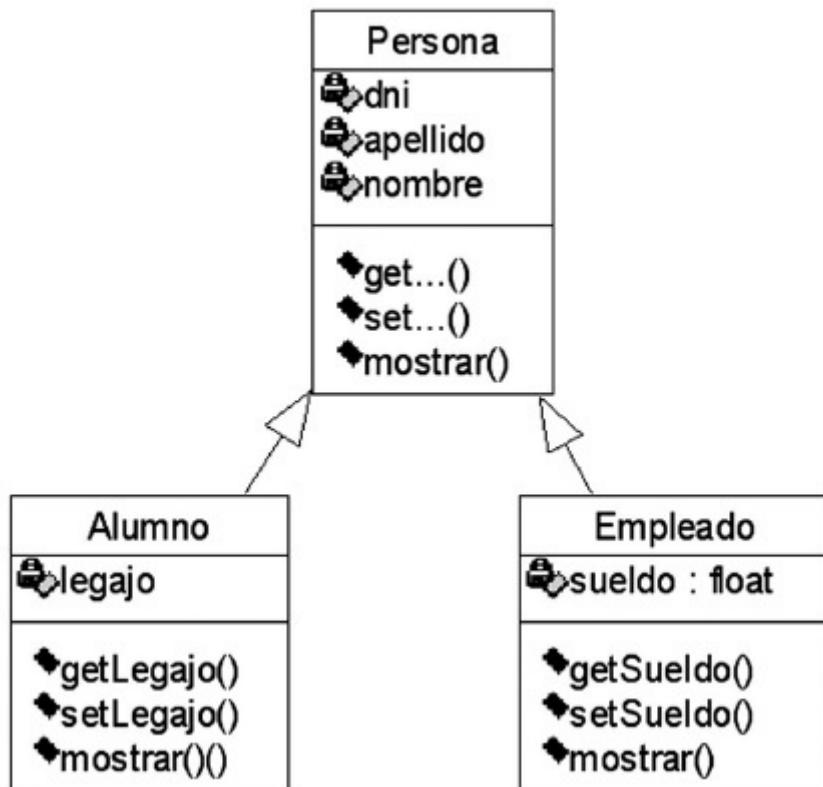
PSEUDO: Definición de constructores con paso de parámetros en la jerarquía de Animal y Salvaje

```
Clase Animal
    variable Privada cantidadPatas tipo numerica
    'Constructor con parámetro. La variable can recibe el valor 4 y lo asigna al
    'atributo cantidadPatas.
    Procedimiento Animal(variable can tipo numerica)
        this.cantidadPatas = can
    Fin Procedimiento
    .....
Fin Clase
Clase Salvaje
    variable Privada agresivo tipo logica
    'Constructor con parámetro. La variable can se envía a Animal y la variable
    'agres la asigna al atributo de la clase, agresivo.
    Procedimiento Salvaje(variable can tipo numerica, variable agres tipo logica)
        super(can)
        this.agresivo = agres
    Fin Procedimiento
```

.....
.....
Fin Clase

Redefinir o sobreescibir métodos y atributos en subclases Cuando en varias clases existe un método con el mismo nombre se trata de referenciar la acción o funcionalidad del objeto. Pero eso no significa que los objetos de distintas clases se comporten de igual manera. Veamos nuestro ejemplo de Animal: todos duermen pero el oso inverna; el gato lo hace quizás panza arriba; el perro, hecho un ovillo y el tigre acostado en la hierba. Dicha definición se realiza en cada método dormir(), o sea, estamos redefiniendo o sobreescribiendo el método.

Veamos otro ejemplo:



El método **mostrar()** se encuentra en las tres clases, pero cada uno tendrá el código necesario para mostrar sus atributos.

El pseudocódigo es:

PSEUDO: Pseudocódigo ejemplo persona (Alumno-Empleado)

```
Clase Persona

    variable Privada dni tipo numerica
    variable Privada apellido tipo alfanumerica
    variable Privada nombre tipo alfanumerica
    Procedimiento mostrar()
        'Muestra el valor del atributo con un método propio del lenguaje
        Imprimir ("DNI: " + dni)
```

```

    Imprimir ("Apellido: " + apellido)
    Imprimir ("Nombre: " + nombre)
Fin Procedimiento
'Código de los get y set.
Fin Clase
Clase Alumno Hereda de Persona
variable Privada legajo tipo numerica
Procedimiento mostrar()

    'Muestra los valores de los atributos heredados de Persona convocando al método mostrar() definido en la superclase.
super.mostrar()
'Muestra el valor del atributo de la clase Alumno.
Imprimir ("Legajo: " + legajo)
Fin Procedimiento
'Código de los get y set.
Fin Clase
Clase Empleado Hereda de Persona

variable Privada sueldo tipo numerica
Procedimiento mostrar()
'Muestra el valor del atributo de la clase Empleado.
Imprimir ("Sueldo: " + sueldo)
Fin Procedimiento
'Código de los get y set.
Fin Clase

```

Todo lo explicado sobre la Programación Orientada a Objetos es factible aplicarlo a los lenguajes que responden al Paradigma OO.

Recuerde dos conceptos, "estado" y "comportamiento" del objeto, ambos definidos en la clase (molde del objeto). Un objeto, a través de su estado (variables), tiene información respecto a sí mismo, y obtiene información respecto de otros objetos con los cuales él coopera (un objeto puede contener otros objetos). Este conocimiento que los objetos tienen entre sí es lo que garantiza que se puedan mandar mensajes. O sea, la única posibilidad de que un objeto pueda mandarle mensajes a otro es porque lo conoce, y la única forma de que un objeto conozca a otro es que de alguna manera en el estado esté eso representado (no es la única forma, sí la más habitual). Cuando el objeto recibe un mensaje (solicitud de que haga algo) se activa algún método, se comporta de alguna forma esperada. El comportamiento de un objeto está dado por sus métodos.

Para algunos no es fácil programar con este paradigma, pero creemos que no es tan así; usted solo tiene que comenzar a pensar en objetos, algo que realiza constantemente en la vida real, identificar sus atributos y sus funcionalidades (¿qué hace?), plantear un pseudocódigo y luego trasladarlo al lenguaje de programación. Eso sí, este entrenamiento requiere de un grado de abstracción que a veces no está adquirido, pero no deja de ser una muy buena práctica.

Tenga en cuenta que este paradigma es válido también para el "análisis" y el "diseño".



¿Te animás a medir cuánto aprendiste?

1. Indique la opción correcta

Una clase constituye la representación abstracta de algo.

- Verdadero
- Falso

2. Indique la opción correcta

¿Cuál de las siguientes características de la Programación Orientada a Objetos promueve el reuso de código, mediante la creación de subclases?

- El Encapsulamiento.
- Ninguna opción es correcta.
- La Herencia.
- El Polimorfismo.

3. Indique la opción correcta

Modelando con Objetos el sistema solar, ¿cuál de las siguientes afirmaciones es la MENOS acertada?

- Tierra, Marte y Jupiter son instancias de la clase Planeta, y Luna de la clase Satélite.
- Diámetro, orbita, satélites son atributos de los objetos de la clase Planeta mientras que giran alrededor del sol es una de las operaciones.
- La Clase Planeta tiene entre sus subclases a la clase Tierra, Marte y Júpiter, y la clase Satélite tiene como una de sus subclases a la clase Luna.

4. Indique la opción correcta

¿Cuál de las siguientes clases podría llegar a ser Subclase de la clase Empleado?

- Apellido.
- Gerente.
- Persona.

5. Indique la opción correcta

¿Cuáles deberían ser objetos en un sistema para administrar un VideoClub?

- Película, Alquiler, Acción.
- Película, Alquiler, Estreno.
- Película, Socio, Alquiler.
- Película, Estreno, Socio.

Respuestas de la Autoevaluación

1. Indique la opción correcta

Una clase constituye la representación abstracta de algo.

Verdadero

Falso

2. Indique la opción correcta

¿Cuál de las siguientes características de la Programación Orientada a Objetos promueve el reuso de código, mediante la creación de subclases?

El Encapsulamiento.

Ninguna opción es correcta.

La Herencia.

El Polimorfismo.

3. Indique la opción correcta

Modelando con Objetos el sistema solar, ¿cuál de las siguientes afirmaciones es la MENOS acertada?

Tierra, Marte y Júpiter son instancias de la clase Planeta, y Luna de la clase Satélite.

Diámetro, orbita, satélites son atributos de los objetos de la clase Planeta mientras que giran alrededor del sol es una de las operaciones.

La Clase Planeta tiene entre sus subclases a la clase Tierra, Marte y Júpiter, y la clase Satélite tiene como una de sus subclases a la clase Luna.

4. Indique la opción correcta

¿Cuál de las siguientes clases podría llegar a ser Subclase de la clase Empleado?

Apellido.

Gerente.

Persona.

5. Indique la opción correcta

¿Cuáles deberían ser objetos en un sistema para administrar un VideoClub?

Película, Alquiler, Acción.

Película, Alquiler, Estreno.

Película, Socio, Alquiler.

Película, Estreno, Socio.

SP7 / H2: Construcción de una Clase

1. ¿Qué hay dentro de una clase?

Los objetos tienen **propiedades** que describen sus atributos, **métodos** que definen sus acciones y **eventos** que definen sus respuestas. Igualmente, la clase que define un objeto tiene sus propias propiedades, métodos y eventos, denominados **miembros de clase** que se pasan a todas las instancias de esa clase.

Por ejemplo, una clase que representa una cuenta bancaria podría tener:

- propiedades (como NroCuenta o Saldo),
- métodos (como CalcularInteres) y
- eventos (como ModificarSaldo)

Una vez creada la instancia de un objeto de cuenta bancaria, puede tener acceso a sus propiedades, métodos y eventos.

Como algunos miembros de la clase son privados, solo se tiene acceso a ellos mediante código dentro de la clase. Por ejemplo, la clase de cuenta bancaria puede tener un método para calcular un saldo. Lo lógico es permitir que otro objeto lea ese balance pero no que pueda cambiarlo directamente.

2. Sintaxis y semántica de los comandos

En base a los conceptos antes mencionados, construiremos el código de nuestra clase utilizando pseudocódigo.

Un pseudocódigo es un lenguaje de especificación de algoritmos. La utilización de este lenguaje, hace que el paso del algoritmo al lenguaje de máquina sea muy fácil.

Repasemos a continuación cómo se compone una clase según lo visto en la SP1.

PSEUDO: Clase CuentaBancaria

```
'Declaramos la clase
Clase NOMBRE_DE_LA_CLASE
    'Declaramos los atributos de la clase
    variable Privada NOMBRE_ATRIBUTO_1 tipo numerica
    variable Privada NOMBRE_ATRIBUTO_2 tipo numerica
    'Declaramos los métodos constructores de la clase
    Procedimiento NOMBRE_DE_LA_CLASE(variable atrิ_1 tipo numerica)
        NOMBRE_ATRIBUTO_1 = atrí_1
        NOMBRE_ATRIBUTO_2 = 0
    Fin Metodo
    'Declaramos los procedimientos de propiedad
    Propiedad Publica getATRIBUTO_1() tipo numerica
        Devuelve NOMBRE_ATRIBUTO_1
    Fin Propiedad
    Propiedad Publica setATRIBUTO_1 (variable atrí_1 tipo numerica)
        NOMBRE_ATRIBUTO_1 = atrí_1
    Fin Propiedad
    Propiedad Publica getATRIBUTO_2 () tipo numerica
        Devuelve NOMBRE_ATRIBUTO_2
    Fin Propiedad
```

```

Propiedad Publica setATRIBUTO_2 (variable atrí_2 tipo numerica)
    NOMBRE_ATRIBUTO_2 = atrí_2
Fin Propiedad
'Declaramos los métodos de la clase
Procedimiento Público NOMBRE_METODO (variable NOMBRE_VAR tipo numerica)
    NOMBRE_ATRIBUTO_2 = NOMBRE_ATRIBUTO_2 + NOMBRE_VAR
Fin Método
Fin Clase

```

Analicemos las secciones que declaramos en este código:

'Declaramos la clase'

Dentro de esta sección declaramos toda la estructura que tendrá la clase. Esta incluirá a todos los miembros de clase (atributos, propiedades y métodos). Se deberán generar tantas declaraciones de clases como clases haya en la situación planteada.

'Declaramos los atributos de la clase'

Aquí incluiremos los atributos del objeto. Tener en cuenta que un atributo de una clase, puede ser un objeto de otra clase. Por ejemplo, si quisiéramos vincular la cuenta a un cliente, dentro de los atributos de **CuentaBancaria**, incluiremos a un objeto **Cliente**.

'Declaramos los métodos constructores de la clase'

Esta sección es opcional ya que, por defecto, toda clase tiene un constructor. En caso de definir un constructor debe incluirse en este apartado.

'Declaramos los métodos de propiedad'

Aquí incluimos todos los métodos que modifiquen o devuelvan el estado de los atributos de la clase. Recordar que denominaremos a la asignación con **set** y a la lectura como **get**.

'Declaramos los métodos de la clase'

En esta sección declaramos todos los procedimientos que necesita la clase para responder a las solicitudes de las demás clases.

3. Formulario

Cuando en un proyecto trabajamos con interfaz gráfica de usuario, el formulario que utilizamos en sí es una clase. El pseudocódigo de esta clase es como cualquier otra clase y tendrá un formato como el siguiente para un formulario con nombre **form1**:

PSEUDO: Pseudo Formulario genérico

```

Clase form1
    Instrucciones
    .....
    .....
Fin Clase

```

Por ejemplo, en el caso de nuestra situación profesional, tendremos una interfaz que nos permita contar los autos que pasan por la casilla de peaje, y su declaración sería:

PSEUDO: Pseudo Form contador de vehículos

Clase frmContarVehiculos

Instrucciones

Fin Clase



¿Estás listo para un desafío?

1. Indique la opción correcta

Una clase está compuesta por objetos a los que pertenece.

- Verdadero
- Falso

2. Indique la opción correcta

Las propiedades de una clase describen los atributos de la misma.

- Verdadero
- Falso

3. Indique la opción correcta

Los miembros de una clase se pasan a todas sus instancias.

- Verdadero
- Falso

4. Indique la opción correcta

La siguiente es una declaración válida de un procedimiento de propiedad.

```
1 Propiedad Publica setATRIBUTO_1 (variable atrิ_1 tipo numerica)
2     NOMBRE_ATRIBUTO_1 = atrí_1
3 Fin Propiedad
```

- Verdadero
- Falso

5. Indique la opción correcta

Una clase solo puede tener como atributos, datos variables de tipo numéricas y alfanuméricas.

```
1 Propiedad Publica setATRIBUTO_1 (variable atrí_1 tipo numerica)
2     NOMBRE_ATRIBUTO_1 = atrí_1
3 Fin Propiedad
```

- Verdadero
- Falso

Respuestas de la Autoevaluación

1. Indique la opción correcta

Una clase esta compuesta por objetos a los que pertenece.

Verdadero

Falso

2. Indique la opción correcta

Las propiedades de una clase describen los atributos de la misma.

Verdadero

Falso

3. Indique la opción correcta

Los miembros de una clase se pasan a todas sus instancias.

Verdadero

Falso

4. Indique la opción correcta

La siguiente es una declaración válida de un procedimiento de propiedad.

Verdadero

Falso

5. Indique la opción correcta

Una clase solo puede tener como atributos, datos variables de tipo numéricas y alfanuméricas.

Verdadero

Falso

SP7 / H3: Modelo de programa completo usando clases

En este apartado, veremos un ejemplo completo de programa utilizando clases.

Para realizar un repaso de los principios de la POO, utilizaremos un ejemplo que nos permitirá ir modificando el código para que usted comprenda y observe su aplicación. No usaremos nuestra situación profesional, la cual será desarrollada de manera completa en la sección Ejercicio Resuelto.

Recuerde que una de las razones por la cual utilizar clase, es que una vez que creamos una clase, a esta la podremos utilizar en cualquier proyecto posterior que nos toque desarrollar.

Es por eso que para este ejemplo utilizaremos la clase Persona, que como puede imaginar la utilizará en innumerables proyectos a lo largo de su vda profesional. Si bien Persona es muy genérico, habrá situaciones en donde le toque manejar empleados, clientes, proveedores; todos estos i características en común que reunen en la clase persona, por ejemplo nombre, edad, dirección, número de teléfono, etc.

En la situación profesional siguiente (Herramienta 4) veremos que las partes del pseudocódigo para generar una clase son:

'Declaramos la clase'

Para nuestra clase Persona la declaración de la clase será:

PSEUDO: Declaración de la Clase persona

```
Clase Persona  
Fin Clase
```

'Declaramos los atributos de la clase'

Ya con la clase Persona creada, ahora declararemos los atributos de la misma. Junto con la declaración del atributo determinaremos que visibilidad tendrá el mismo, indicando si son de acceso **Público** o **Privado**.

En el caso de los atributos públicos, estos podrán ser consultados y modificados directamente desde otras clases. Sin embargo los atributos privados solo podrán ser accedidos y modificados a través de los métodos publicos de la clase persona.

Declararemos en un principio los siguientes atributos:

PSEUDO: Declaración de Atributos de clase persona

```
Clase Persona  
    variable Privada nombre_1 tipo alfanumerico  
    variable Privada nombre_2 tipo alfanumerico  
    variable Privada apellido tipo alfanumerico  
    variable Publica casado tipo logica  
Fin Clase
```

'Declaramos los métodos constructores de la clase'

En esta sección, declararemos los constructores de clases. Los constructores pueden o no existir, por defecto siempre existirá un constructor sin parámetros que inicializará los atributos del objeto a sus valores por defecto.

Agregaremos a nuestra clase Persona un constructor sin parámetros:

PSEUDO: Declaración de los métodos Constructores (sin parámetros) de la clase persona

```
Clase Persona
```

```

variable Privada nombre_1 tipo alfanumerico
variable Privada nombre_2 tipo alfanumerico
variable Privada apellido tipo alfanumerico
variable Publica casado tipo logica

Procedimiento Persona()

Fin Procedimiento
Fin Clase

```

Y para que note la diferencia, ahora agregaremos un constructor que recibe parámetros e inicializa los atributos con los valores recibidos:

PSEUDO: Declaración de Constructor de la clase persona con parámetros

```

Clase Persona
    variable Privada nombre_1 tipo alfanumerico
    variable Privada nombre_2 tipo alfanumerico
    variable Privada apellido tipo alfanumerico
    variable Publica casado tipo logica

    Procedimiento Persona()

    Fin Procedimiento

    Procedimiento Persona(variable primer_nom tipo alfanumerico,variable segundo_nom tipo
alfanumerico,_
        _ variable ape tipo alfanumerico, variable cas tipo logica)
        nombre_1 = primer_nom
        nombre_2 = segundo_nom
        apellido = ape
        casado = cas
    Fin Procedimiento

    Fin Clase

```

'Declaramos los métodos de propiedad'

Ahora nos encargamos de los métodos públicos que utilizaremos para la manipulación de nuestros atributos privados. Estos métodos serán la interfaz con otros objetos que quieran tener acceso a nuestros atributos.

Por cada atributo que tengamos, podremos tener un método **get** y un método **set**, los cuales nos servirán para devolver y asignar el valor de un atributo, respectivamente. Estos métodos existirán, siempre y cuando queramos brindar acceso a nuestros atributos.

PSEUDO: Declaración de los métodos de Propiedades

```

Clase Persona
    variable Privada nombre_1 tipo alfanumerico
    variable Privada nombre_2 tipo alfanumerico
    variable Privada apellido tipo alfanumerico
    variable Publica casado tipo logica

    Procedimiento Persona()

    Fin Procedimiento

    Procedimiento Persona(variable primer_nom tipo alfanumerico,variable segundo_nom tipo

```

```

alfanumerico,_  

    _ variable ape tipo alfanumerico, variable cas tipo logica)  

    nombre_1 = primer_nom  

    nombre_2 = segundo_nom  

    apellido = ape  

    casado = cas  

Fin Procedimiento  
  

Procedimiento Propiedad Publico PrimerNombre() tipo alfanumerico  

Get  

    PrimerNombre = nombre_1  

Fin Get  

Set(variable valor tipo alfanumerica)  

    nombre_1 = valor  

Fin Set  

Fin Procedimiento  

Procedimiento Propiedad Publico SegundoNombre() tipo alfanumerico  

Get  

    SegundoNombre = nombre_2  

Fin Get  

Set(variable valor tipo alfanumerica)  

    nombre_2 = valor  

Fin Set  

Fin Procedimiento  

Procedimiento Propiedad Publico Apellido() tipo alfanumerico  

Get  

    Apellido = apellido  

Fin Get  

Set(variable valor tipo alfanumerica)  

    apellido = valor  

Fin Set  

Fin Procedimiento  
  

Fin Clase

```

'Declaramos los métodos de la clase'

Por último, en nuestro código completo tendremos la declaración de los métodos. Los métodos son simplemente procedimientos declarados dentro de una clase.

Por ejemplo, supongamos que tenemos la necesidad de tener un método que nos devuelva el nombre completo de una persona, el código completo quedará de la siguiente manera.

PSEUDO: Código completo con la declaración de los métodos de la clase

```

Clase Persona  

    variable Privada nombre_1 tipo alfanumerico  

    variable Privada nombre_2 tipo alfanumerico  

    variable Privada apellido tipo alfanumerico  

    variable Publica casado tipo logica  
  

    Procedimiento Persona()  
  

    Fin Procedimiento  
  

    Procedimiento Persona(variable primer_nom tipo alfanumerico,variable segundo_nom tipo

```

```

alfanumerico,_  

    _ variable ape tipo alfanumerico, variable cas tipo logica)  

    nombre_1 = primer_nom  

    nombre_2 = segundo_nom  

    apellido = ape  

    casado = cas  

Fin Procedimiento  
  

Procedimiento Propiedad Publico PrimerNombre() tipo alfanumerico  

Get  

    PrimerNombre = nombre_1  

Fin Get  

Set(variable valor tipo alfanumerica)  

    nombre_1 = valor  

Fin Set  

Fin Procedimiento  

Procedimiento Propiedad Publico SegundoNombre() tipo alfanumerico  

Get  

    SegundoNombre = nombre_2  

Fin Get  

Set(variable valor tipo alfanumerica)  

    nombre_2 = valor  

Fin Set  

FinProcedimiento  

Procedimiento Propiedad Publico Apellido() tipo alfanumerico  

Get  

    Apellido = apellido  

Fin Get  

Set(variable valor tipo alfanumerica)  

    apellido = valor  

Fin Set  

Fin Procedimiento  

Procedimiento Publico NombreCompleto() tipo alfanumerico  

Si nombre_2 <> "" Entonces  

    NombreCompleto = nombre_1 & " " & nombre_2 & " " & apellido  

Si No  

    NombreCompleto = nombre_1 & " " & apellido  

Fin Si  

Fin Procedimiento  
  

Fin Clase

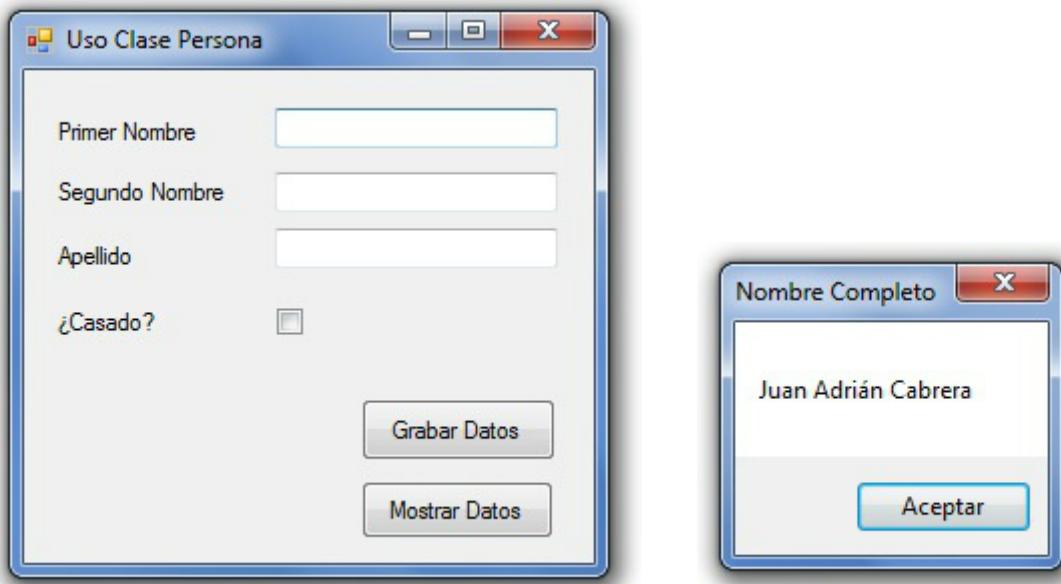
```

Ya con nuestra clase persona creada, lo único que debemos realizar ahora, es instanciar a la misma del modo que ya conocemos:

Persona A = new Persona()

Y, a través de la misma, tendremos accesos a sus métodos.

Para hacer uso de esta clase en un programa completo, nos queda crear una nueva clase, que será un formulario quien manipule objetos de tipo persona.



El primer botón de comando, asignará los datos ingresados por el formulario, mientras que el segundo, nos mostrará los datos almacenados en los atributos de la persona instanciada por medio del método declarado para tal fin en la clase persona.

Para esto escribimos el siguiente código:

PSEUDO: Clase persona Form

```

Clase Formulario
    Persona personal1 = new Persona()
    Procedimiento cmdGrabar_Click
        'Usamos los métodos de propiedad de la clase persona e inicializamos los
        'atributos de la instancia declarada
        personal1.PrimerNombre = txtNombre1
        personal1.SegundoNombre = txtNombre2
        personal1.Apellido = txtApellido
        Si chkCasado = VERDADERO Entonces
            personal1.Casado = VERDADERO
        Si No
            personal1.Casado = FALSO
        Fin Si
    Fin Procedimiento
    'Este método llama al de la clase persona que devuelve el nombre completo de la persona.
    Procedimiento cmdMostrar_Click
        Mensaje personal1.NombreCompleto
    Fin Procedimiento
Fin Clase

```



¡Vamos a comprobar cuánto aprendiste!

1. Indique la opción correcta

Un constructor de una clase debe tener el mismo nombre que la clase y no debe recibir parámetros.

- Verdadero
- Falso

2. Indique la opción correcta

Un método get devuelve el valor que tiene un atributo de un objeto.

- Verdadero
- Falso

3. Indique la opción correcta

Un formulario, en la POO también es una clase.

- Verdadero
- Falso

4. Indique la opción correcta

Una clase puede tener tantos constructores como se desee siempre y cuando tengan distinto nombre.

- Verdadero
- Falso

5. Indique la opción correcta

¿Cuál de las siguientes maneras es la correcta para declarar un objeto?:

- new A =Persona()
- Persona A = new Persona()
- Persona() = new A

Respuestas de la Autoevaluación

1. Indique la opción correcta

Un constructor de una clase debe tener el mismo nombre que la clase y no debe recibir parámetros.

Verdadero

Falso

2. Indique la opción correcta

Un método get devuelve el valor que tiene un atributo de un objeto.

Verdadero

Falso

3. Indique la opción correcta

Un formulario, en la POO también es una clase.

Verdadero

Falso

4. Indique la opción correcta

Una clase puede tener tantos constructores como se desee siempre y cuando tengan distinto nombre.

Verdadero

Falso

5. Indique la opción correcta

¿Cuál de las siguientes maneras es la correcta para declarar un objeto?:

new A =Persona()

Persona A = new Persona()

Persona() = new A

SP7 / Ejercicio resuelto

Recordemos nuestra situación profesional

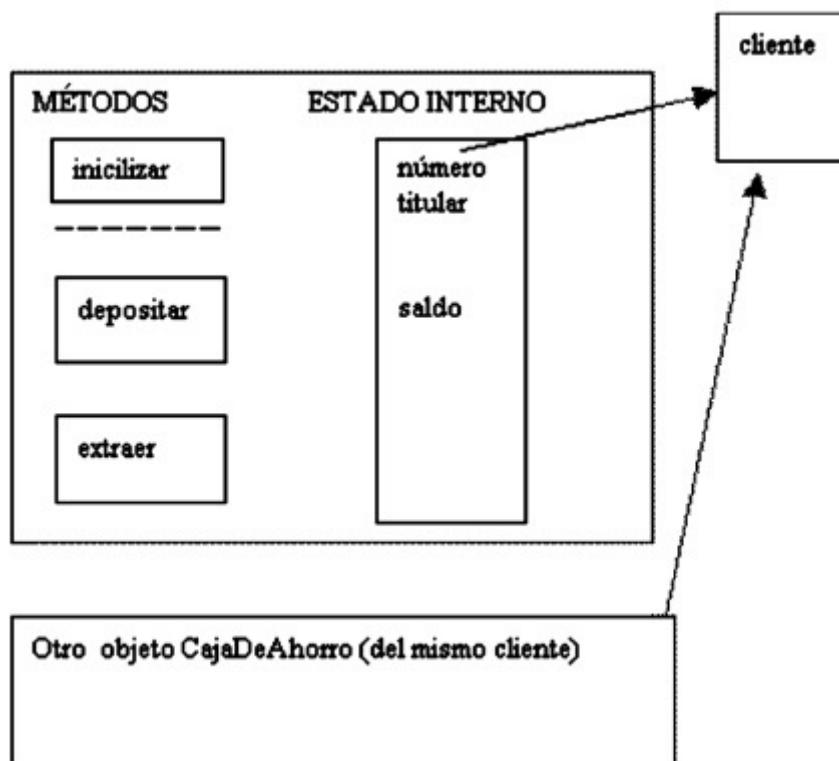
Una institución bancaria le solicita a usted, como pasante del área de desarrollo de software, que realice un modelo de objetos que permita realizar las acciones básicas para depositar y extraer dinero de las cajas de ahorro de los clientes, desde un cajero automático.

Como regla para tener en cuenta se establece que el monto de extracción no puede superar el monto de saldo disponible de la caja de ahorro y, que si el retiro es mayor a \$1500, el titular de la cuenta debe estar al día con los impuestos. Se debe mostrar la información del saldo de cada caja de ahorro, su número y el saldo de cada cliente.

Ahora debemos identificar el estado y comportamiento de cada uno, o sea atributos y métodos; luego, sus cooperaciones (conocimiento entre ellos).

En la figura se observa un objeto que es el CajeroAutomático y hay tres objetos CajaDeAhorro. En esta aplicación, en principio, la forma en que funciona la interacción entre los objetos es tal que el usuario, usando una interfaz que provee el objeto CajeroAutomático, selecciona acciones que dicho objeto transforma en mensajes, enviados a los objetos que representan CajaDeAhorro. Veamos un ejemplo: cuando el usuario selecciona que quiere hacer un depósito, el objeto CajeroAutomático envía el mensaje correspondiente a uno de los objetos CajaDeAhorro que es el que el usuario seleccionó.

Analizaremos desde la óptica de la definición de Objeto ¿cómo es un objeto CajaDeAhorro?



"POO Análisis del objeto CajaDeAhorro" | Un objeto CajaDeAhorro básicamente va a soportar dos operaciones esenciales para el

negocio:• depositar. • extraer. O sea que a un objeto CajaDeAhorro se le puede solicitar que realice un depósito y que haga una extracción. Estas dos operaciones permiten, en principio, pensar cómo va a ser la interacción entre el objeto CajeroAutomático y cada uno de los objetos CajaDeAhorro. El objeto CajeroAutomático enviará mensajes para realizar depósitos o realizará extracciones con algún argumento, que es la cantidad que hay que depositar o extraer.

Un objeto CajaDeAhorro básicamente va a soportar dos operaciones esenciales para el negocio:

- depositar.
- extraer.

O sea que a un objeto CajaDeAhorro se le puede solicitar que realice un depósito y que haga una extracción.

Estas dos operaciones permiten, en principio, pensar cómo va a ser la interacción entre el objeto CajeroAutomático y cada uno de los objetos CajaDeAhorro.

El objeto CajeroAutomático enviará mensajes para realizar depósitos o realizará extracciones con algún argumento, que es la cantidad que hay que depositar o extraer.

Caracterizamos un objeto con el comportamiento más relevante que el objeto tiene, o sea, el comportamiento que lo distingue de otros objetos, en nuestro caso el hecho de que un objeto CajaDeAhorro pueda hacer depósitos y extracciones hace que CajaDeAhorro sea distinto de un objeto PlazoFijo, o distinto de un objeto Cliente del Banco.

Seguramente, depositar y extraer no son las únicas acciones que puede realizar, pero sí son las dos centrales en el objeto CajaDeAhorro desde el punto de vista del negocio del banco que nos interesa en este momento.

¿Qué necesita saber el objeto CajaDeAhorro respecto de sí mismo para poder ejecutar los mensajes?

Lo primero que surge es que precisa conocer su saldo; para ello tendrá que calcularlo, o sea, necesita un método que lo haga y un atributo que contenga el valor. Esto va a modificarse en los métodos depositar y extraer; con el primero se va a incrementar y con el segundo, se va decrementar. Básicamente, entre esa variable y el código de esos dos mensajes está dado el 80% de la funcionalidad central de los objetos CajaDeAhorro.

Ahora tenemos que empezar a pensar que el objeto CajaDeAhorro no es un objeto aislado, sino que coexiste en un contexto de una aplicación más grande, que es el Banco. En ese contexto, las cajas de ahorro están identificadas únicamente por un número.

Muchas veces, cuando se trabaja con base de datos (archivos que almacenan los atributos de las entidades y que se relacionan entre sí por medio de un atributo único) ese número es lo que se conoce como la clave, pero en los Objetos no existe el concepto de clave; los Objetos tienen algo que los identifica que no es su clave, no es un atributo.

Para cada objeto CajaDeAhorro el número se considera igual que el saldo, una parte de su estado. Todas las variables tienen la misma relevancia.

La forma en que el objeto CajeroAutomático va a conocer a un objeto CajaDeAhorro no será por el número de la caja de ahorro, pues ningún usuario va a ingresar al sistema con el número de su cuenta.

De todo lo expuesto, surge que el objeto CajaDeAhorro tiene un número y un saldo; las dos son características intrínsecas del objeto CajaDeAhorro, de las dos hay una que es persistente en el tiempo (número) y otra que es muy cambiante (saldo); a medida que el cliente hace operaciones, el saldo varía. A pesar de eso, los dos atributos tienen el mismo valor dentro de la aplicación.

La caja de ahorro va a tener un titular, el Cliente del Banco, dueño de esa caja de ahorro. Esto indica que

dentro del objeto CajaDeAhorro hay un objeto Cliente; la forma en que está relacionado el objeto CajaDeAhorro y el objeto Cliente es a través de esa variable "titular", o sea una referencia al objeto Cliente (no un atributo Nro. De cliente). Esta referencia al objeto cliente nos permitirá enviarle mensajes a dicho objeto.

El objeto Cliente representa a un cliente del banco, con dos simplificaciones:

- 1) Las cajas de ahorro tienen un solo titular.
- 2) Los titulares son personas.

En este análisis de la situación, podemos decir que "lo primero que se nos ocurrió es que existe un objeto CajaDeAhorro que conoce a un objeto Cliente".

Ahora supongamos que se manda al objeto CajaDeAhorro el mensaje "depositar: 100", y veamos cuál es el resultado. Lo que surge inmediatamente es:

- 1) Cuando se creó el objeto CajaDeAhorro ¿el saldo tiene que haber tenido un valor?
- 2) ¿Cómo se conoce el titular?
- 3) ¿Cómo sabemos cuál es su número?

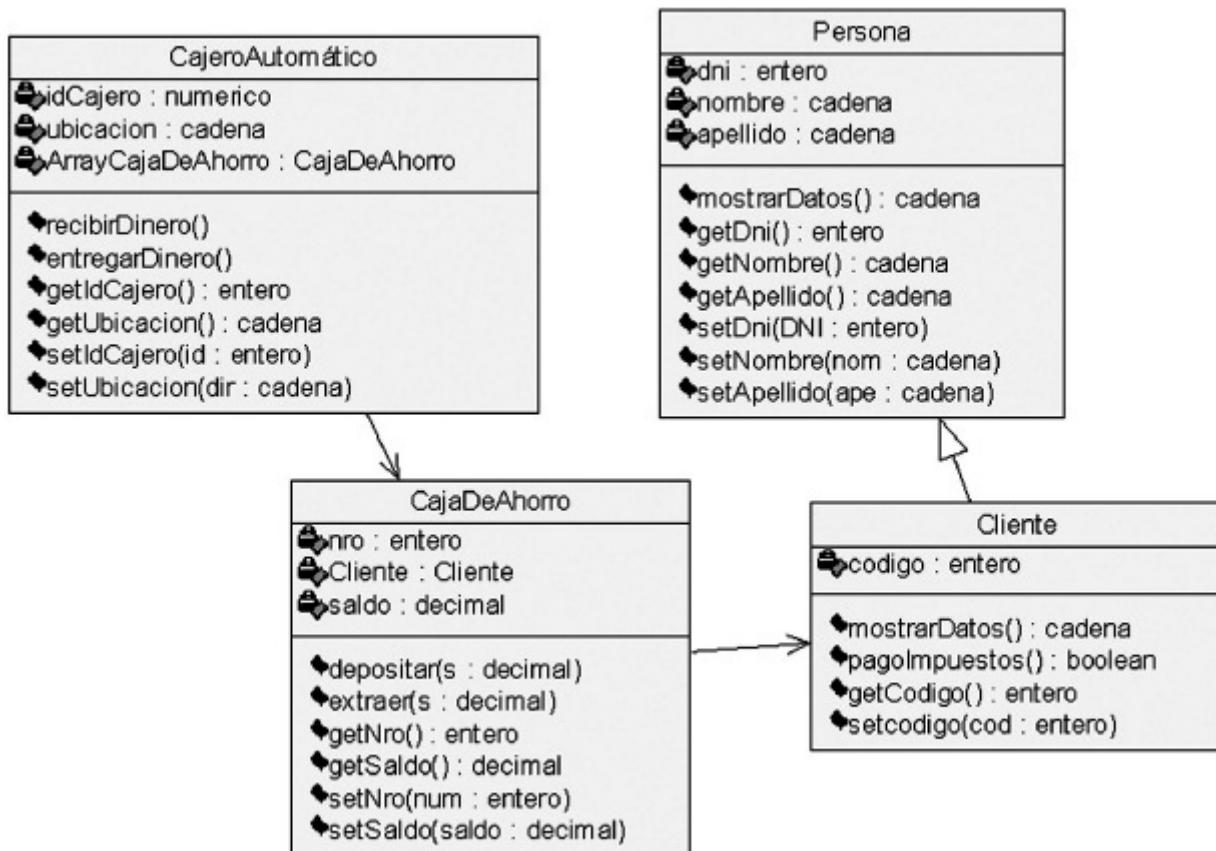
Entonces aparece la necesidad de inicializar el objeto CajaDeAhorro indicando que tiene un número, que pertenece a un titular, y cuyo saldo es cero. En la mayoría de las aplicaciones con objetos hay un mensaje de inicialización. En este caso, esa inicialización establecería el saldo inicial con valor cero, el titular y el número de la caja de ahorro.

Si observa la resolución gráfica que está más abajo, podrá apreciar que un cliente puede tener más de una caja de ahorro, pero que los objetos CajaDeAhorro conozcan a los objetos Cliente no impide que los objetos Cliente conozcan a los objetos CajaDeAhorro.

Recordemos que la regla de negocio por respetar dice que cuando un cliente quiere sacar de la caja de ahorro más de \$1500 es necesario chequear que esté al día impositivamente. Cuando se da el mensaje "extraer: 1500" lo primero que hace el objeto CajaDeAhorro es mandarle un mensaje al objeto Cliente, para saber si el titular está al día con los impuestos; pero también puede ocurrir que un cliente quiera ver cuáles son las cajas de ahorro que tiene, para elegir de cual extraer; entonces el mensaje va del objeto Cliente a los objetos CajaDeAhorro que tienen como titular a ese objeto Cliente.

La conclusión es que entre dos objetos las relaciones de conocimiento pueden ir en ambos sentidos.

Veamos la resolución gráfica:



Hemos utilizado en nuestro diseño de clases:

- Herencia: entre Persona y Cliente
- Encapsulamiento: los atributos son privados y los métodos `getxx()` y `setxx()` son públicos
- Polimorfismo: en el método `mostrarDatos()` y los métodos constructores de las clases Persona y Cliente
- Conocimiento: entre los objetos CajeroAutomático, CajaDeAhorro y Cliente
- Estado: dado por los atributos de cada objeto
- Comportamiento: dado por los métodos de cada objeto
- This y super

El pseudocódigo es:

PSEUDO: Pseudocódigo del ejercicio del Cajero Automático

```

Clase Persona
    'Declaración de variables
    variable Privada dni tipo numerica
    variable Privada nombre tipo alfanumerica
    variable Privada apellido tipo alfanumerica
    'Constructor por defecto
    Procedimiento Persona()
    Fin Procedimiento
    'Constructor con paso de parámetros
    Procedimiento Persona(variable DNI tipo numerica,variable nom tipo alfanumerica,_

```

```

        _ variable ape tipo alfanumerica)
dni = DNI
nombre = nom
apellido = ape
Fin Procedimiento

Procedimiento mostrarDatos()
    'Muestra el valor de los atributos de la clase Persona
    Escribir ("DNI: " + dni)
    Escribir ("Apellido: " + apellido)
    Escribir ("Nombre: " + nombre)
Fin Procedimiento
Procedimiento getDni() tipo numerico
    Devuelve dni
Fin Procedimiento

Procedimiento setDni(variable DNI tipo numerico)
    this.dni = DNI
Fin Procedimiento
Procedimiento getNombre() tipo alfanumerica
    Devuelve nombre
Fin Procedimiento

Procedimiento setNombre(variable nom tipo alfanumerica)
    this.nombre = nom
Fin Procedimiento
Procedimiento getApellido() tipo alfanumerico
    Devuelve apellido
Fin Procedimiento
Procedimiento setApellido(variable ape tipo alfanumerico)
    this.apellido = ape
Fin Procedimiento

Fin Clase
Clase Cliente Hereda de Persona
    variable Privada codigo tipo numerica
    'Constructor por defecto
    Procedimiento Cliente()

    Fin
    'Constructor con paso de parámetros.
    Procedimiento Cliente(variable DNI tipo alfanumerica, variable nom tipo alfanumerica,_
                           _ variable ape tipo alfanumerica variable cod tipo numerica)
        super(DNI, cadena, ape)
        Codigo = cod
    Fin Procedimiento
    Procedimiento mostrarDatos()
        'Muestra el valor de los atributos de la clase Persona
        super.mostrarDatos()
        'Muestra el valor de los atributos de la clase Cliente
        Escribir ("Código: " + this.getCodigo())
    Fin Procedimiento

    Procedimiento getCodigo() tipo numerico
        Devuelve codigo

```

```

Fin Procedimiento
Procedimiento setCodigo(variable cod tipo numerica)
    this.codigo = cod
Fin Procedimiento
Procedimiento pagoImpuestos() tipo logico
    Si pago = VERDADERO
        Devuelve VERDADERO
    Si No
        Devuelve FALSO
    Fin Si
Fin Procedimiento
Fin Clase
Clase CajaDeAhorro
    variable Privada nro tipo numerica
    Cliente c
    variable Privada saldo tipo numerica

'Constructor por defecto
Procedimiento CajaDeAhorro()
    'Inicializa la variable saldo con 0.
    Saldo = 0
Fin Procedimiento
Procedimiento getNro() tipo numerico
    Devuelve nro
Fin Procedimiento
Procedimiento setNro(variable num tipo numerica)
    this.nro = num
Fin Procedimiento
Procedimiento tipo decimal getSaldo()
    Devuelve saldo
Fin Procedimiento
Procedimiento setSaldo(variable sal tipo numerico)
    this.saldo = sal
Fin Procedimiento
Procedimiento depositar(variable s tipo numerico)
    'Incrementa el saldo
    saldo = saldo + s
Fin Procedimiento
Procedimiento extraer(variable s tipo numerica)
    Si s <= saldo
        Si s < 1500
            'Disminuye el saldo
            Saldo = saldo - s
        Si No
            variable res = c.pagoImpuestos()
            'Verifica el pago de impuestos
            Si res = VERDADERO
                saldo = saldo - s
                'Disminuye el saldo
            Si No
                Mensaje "no pago sus impuestos"
            Fin Si
        Fin Si
    Si No
        Mensaje "no tiene saldo suficiente"

```

Fin Si
Fin Procedimiento
Fin Clase

SP7 / Ejercicio por resolver

Diseñe un modelo OO y su pseudocódigo que resuelva la siguiente situación:

Una institución educativa necesita plantear un modelo de objetos que permita a los alumnos inscribirse en las materias de una carrera. La regla para tener en cuenta es que para la inscripción el alumno debe tener abonada la matrícula.

Utilice en la resolución todos los conceptos adquiridos que responden al POO.

SP7 / Evaluación de paso



¿Te animás a medir cuánto aprendiste?

1. Indique la opción correcta

Los principios de la POO son la Abstracción, la Herencia, los objetos y las clases.

- Verdadero
- Falso

2. Indique la opción correcta

Cuando se declara una clase se pueden declarar numerosos constructores diferenciados por la cantidad de parámetros que reciben.

- Verdadero
- Falso

3. Indique la opción correcta

Para instanciar un objeto de la clase persona, lo hacemos de la siguiente manera "Persona A = new Persona()"

- Verdadero
- Falso

4. Indique la opción correcta

¿Cuál de los siguientes datos no es un atributo de la clase Persona?:

- Persona.
- Nombre.
- Domicilio.

5. Indique la opción correcta

El estado de un objeto se determina por sus:

- Por sus procedimientos.
- Por sus atributos.
- Por sus constructores.

Respuestas de la Autoevaluación

1. Indique la opción correcta

Los principios de la POO son la Abstracción, la Herencia, los objetos y las clases.

Verdadero

Falso

2. Indique la opción correcta

Cuando se declara una clase se pueden declarar numerosos constructores diferenciados por la cantidad de parámetros que reciben.

Verdadero

Falso

3. Indique la opción correcta

Para instanciar un objeto de la clase persona, lo hacemos de la siguiente manera "Persona A = new Persona()"

Verdadero

Falso

4. Indique la opción correcta

¿Cuál de los siguientes datos no es un atributo de la clase Persona?:

Persona.

Nombre.

Domicilio.

5. Indique la opción correcta

El estado de un objeto se determina por sus:

Por sus procedimientos.

Por sus atributos.

Por sus constructores.

Cierre

¡Muy bien, lo lograste!, todo lo dicho en la introducción a esta altura debe ser un hecho.

Ahora cuando sienta hablar de variables, constantes, vectores, matrices no te estarán hablando de nada que no pueda entender.

Ya ha aprendido a realizar programas con interfaz gráfica y con una lógica computacional compuesta por reglas de estructuras y de control de flujo (secuenciales, alternativas, repetitivas). También, estará en condiciones de organizar las funcionalidades de sus programas en procedimientos que permitan una mejor lectura y mantenimiento del código desarrollado.

Para completar esta formación, solo deberá conocer las instrucciones de cualquier lenguaje de programación (C/C++, .Net, JAVA, etc.) y convertir el pseudocódigo aprendido en código fuente que darán origen a sus aplicaciones y soluciones informáticas.

Pero a no confiarse y relajarse. Solo acaba de dar un paso en este largo sendero que es nuestra profesión. Tendrá muchas materias que completarán su formación como profesional informático, y una continua formación una vez recibido, para estar actualizado en las nuevas tecnologías.

¡Felicitaciones por lo que logró hasta el momento y suerte para lo que viene!

El autor

Bibliografía

- FARRELL, Joyce (2001). "Introducción a la Programación Lógica y Diseño" Editorial THOMSON INTE RNATIONAL.
- JOYANES AGUILAR, L. (1999). "Fundamentos de Programación - Algoritmos y Estructuras de Datos".
- RODRIGUEZ ALMEIDA, Miguel Ángel (1991). "Metodología de la Programación. A través de pseudocódigo". M^c Graw-Hill.
- RUBBLE (1998). "Análisis y Diseño de Sistemas Orientado a Objetos con GUI".
- WIRTH, N. "Algoritmos + Estructuras de Datos = Programas". Ed. del Castillo.