



Programa de estudios de Desarrollo de Sistemas de Información

MODULO: DISEÑO DE SISTEMAS DE INFORMACION

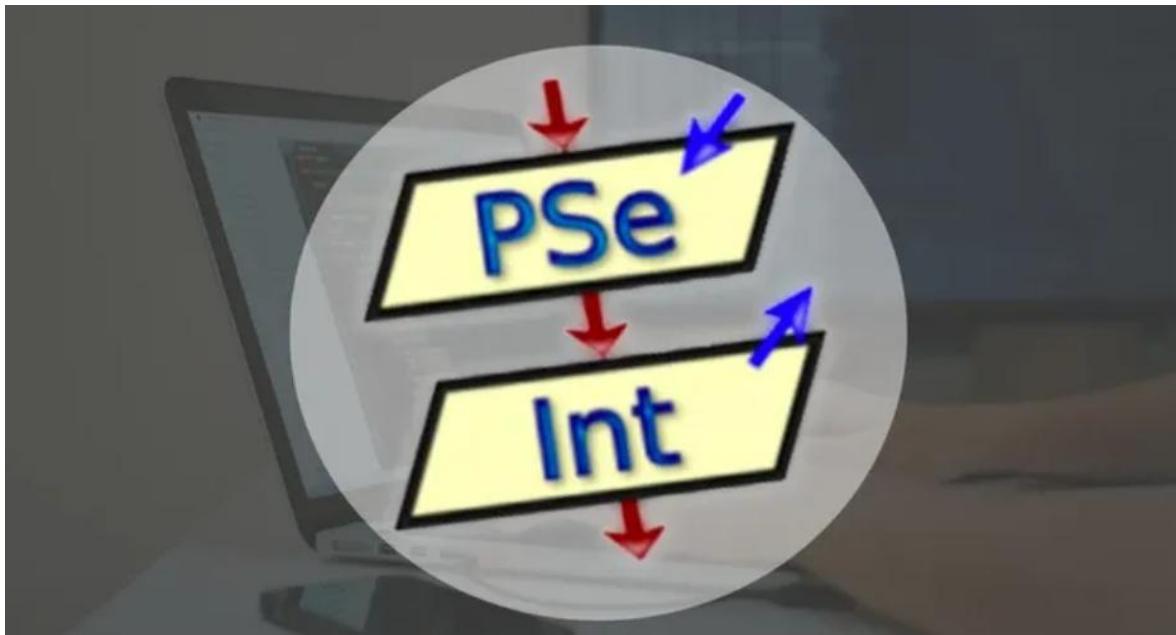
LOGICA DE PROGRAMACION

2025 -I



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA Lógica de Programación



SESION 01

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I1: Diseña algoritmos utilizando las estructuras básicas de control: secuencial, selectiva y repetitivas.



Introducción al procesamiento de datos y Diseño de Algoritmos

CONCEPTOS BASICOS:

PROGRAMA: Un programa computacional, en términos simples, consiste en una serie de pasos lógicamente escritos que permiten resolver un problema determinado, que a partir de datos de entrada obtiene información útil para la toma de decisiones. Aprender a programar consiste, por un lado, en conocer las distintas instrucciones que un determinado lenguaje de programación tiene, y más importante aún, escribir un conjunto de instrucciones en forma lógica para que la ejecución de estas instrucciones permita dar respuesta al problema planteado. No basta con conocer las instrucciones existentes, se debe adquirir la lógica que permite utilizarlas con un fin específico. La mayoría de los problemas que se resuelven en el transcurso de un curso de programación viene especificada mediante un enunciado. Los pasos necesarios para poder construir un programa que dé respuesta al enunciado lo podemos resumir de la siguiente manera:

- Entender el problema
- Buscar soluciones
- Elegir solución
- Diseñar solución
- Implementar solución
- Validar solución

Algoritmos

Los algoritmos constituyen un listado de instrucciones que indican el camino a seguir para dar solución a un problema.

Podríamos decir que un algoritmo es la suma de una parte lógica más una parte de control, en donde la parte lógica especifica el conocimiento en la solución del problema y la parte de control es la estrategia para solucionar el problema.

Características de los algoritmos

- Un algoritmo no debe de ser ambiguo.
- Debe de tener una secuencia inicial
- Cada paso deberá tener una secuencia sucesiva y única es decir que deben indicar claramente el camino a seguir en la solución del problema.



- El algoritmo debe de ser siempre eficiente y dar una solución al problema o de lo contrario deberá dar un mensaje que diga "Sin solución"

¿Qué es un lenguaje de programación?

Es un lenguaje formal que, mediante una serie de instrucciones, le permite a un programador escribir un conjunto de órdenes, acciones consecutivas, datos y algoritmos para, de esa forma, crear programas que controlen el comportamiento físico y lógico de una máquina.

Mediante este lenguaje se comunican el programador y la máquina, permitiendo especificar, de forma precisa, aspectos como:

- cuáles datos debe operar un software específico;
- cómo deben ser almacenados o transmitidos esos datos;
- las acciones que debe tomar el software dependiendo de las circunstancias variables.

Para explicarlo mejor, el lenguaje de programación es un sistema estructurado de comunicación, el cual está conformado por conjuntos de símbolos, palabras claves, reglas semánticas y sintácticas que permiten el entendimiento entre un programador y una máquina.

Es importante recalcar que existe el error común de usar como sinónimos el lenguaje de programación y el lenguaje informático, pero ¿por qué no debemos confundirlos?

Pues, es debido a que el lenguaje de programación obedece a un conjunto de reglas que permiten expresar las instrucciones que serán interpretadas por el programador. Y el lenguaje informático comprende otros lenguajes que dan formato a un texto, pero no son programación en sí mismos.

Entonces, no todos los lenguajes informáticos son de programación, pero todos los lenguajes de programación son a la vez informáticos.

¿Qué tipos de lenguaje de programación existen?

El lenguaje de programación es la base para construir todas las aplicaciones digitales que se utilizan en el día a día y se clasifican en dos tipos principales: lenguaje de bajo nivel y de alto nivel.



Lenguaje de programación de bajo nivel

Son lenguajes totalmente orientados a la máquina.

Este lenguaje sirve de interfaz y crea un vínculo inseparable entre el hardware y el software.

Además, ejerce un control directo sobre el equipo y su estructura física. Para aplicarlo adecuadamente es necesario que el programador conozca sólidamente el hardware. Éste se subdivide en dos tipos:

Lenguaje máquina

Es el más primitivo de los lenguajes y es una colección de dígitos binarios o bits (0 y 1) que la computadora lee e interpreta y son los únicos idiomas que las computadoras entienden.

Ejemplo: 10110000 01100001

No entendemos muy bien lo que dice ¿verdad? Por eso, el lenguaje ensamblador nos permite entender mejor a qué se refiere este código.

Lenguaje ensamblador

El lenguaje ensamblador es el primer intento de sustitución del lenguaje de máquina por uno más cercano al utilizado por los humanos.

Un programa escrito en este lenguaje es almacenado como texto (tal como programas de alto nivel) y consiste en una serie de instrucciones que corresponden al flujo de órdenes ejecutables por un microprocesador.

Sin embargo, dichas máquinas no comprenden el lenguaje ensamblador, por lo que se debe convertir a lenguaje máquina mediante un programa llamado Ensamblador.

Este genera códigos compactos, rápidos y eficientes creados por el programador que tiene el control total de la máquina.

Lenguaje de programación de alto nivel

Tienen como objetivo facilitar el trabajo del programador, ya que utilizan unas instrucciones más fáciles de entender.



Además, el lenguaje de alto nivel permite escribir códigos mediante idiomas que conocemos (español, inglés, etc.) y luego, para ser ejecutados, se traduce al lenguaje de máquina mediante traductores o compiladores.

Traductor

Traducen programas escritos en un lenguaje de programación al lenguaje máquina de la computadora y a medida que va siendo traducida, se ejecuta.

Compilador

Permite traducir todo un programa de una sola vez, haciendo una ejecución más rápida y puede almacenarse para usarse luego sin volver a hacer la traducción.

¿Para qué sirven los lenguajes de programación?

En general un lenguaje de programación sirve para programar. Sin embargo, cada uno tiene un alcance y forma de comunicación diferente.

En resumidas cuentas, el lenguaje de bajo nivel permite la comunicación interna de la máquina, cada instrucción tiene su código único de operación.

Y el lenguaje de alto nivel facilita la captación de instrucciones que el programador le da a la máquina, mientras que éste introduce datos en el idioma conocido la máquina lo va absorbiendo en lenguaje de máquinas mediante traductores o compiladores, permitiendo así:

- reducir el tiempo de programación;
- entender más fácilmente la tarea a realizar;
- permitir al programador desvincularse del funcionamiento interno de la máquina, entre otros.

En otras palabras, el lenguaje de bajo nivel es cercano a los idiomas de las máquinas mientras que el lenguaje de alto nivel está más cerca del entendimiento e idioma humano.

¿Qué softwares de programación existen?

Por software de programación entendemos el conjunto de todas las herramientas que le permiten al programador, crear, escribir códigos, depurar, mantener y empaquetar los proyectos.



Algunos de los distintos programas por los que pasará el proyecto para gestionarlo son:

Editores de código o texto

Al escribir los códigos se completan automáticamente marcando los errores sintácticos y la refactorización.

Compiladores

Como mencionados anteriormente, éstos traducen el código ingresado a lenguaje de máquina generando un código binario ejecutable.

Depuradores

Sirven para optimizar el tiempo de desarrollo mediante el monitoreo de la ejecución de un programa, el seguimiento a los valores de ciertas variables, las referencias a objetos en memoria y por ende, nos ayuda a corregir errores.

Enlazadores

Este programa toma objetos generados en los primeros pasos del proceso de compilación y los recursos necesarios de la biblioteca, quita aquellos procesos y datos que no necesita, y enlaza el código con dicha biblioteca para así aumentar su tamaño y extensión.

Interpretadores o traductores

El traductor (o intérprete) carga el código ingresado y traduce las instrucciones para que el programa pueda ser ejecutado.

IDE

El IDE (Integrated Development Environment) o Entorno de Desarrollo Integrado, es una aplicación informática que proporciona una serie de servicios que facilitan la programación de software, tales como:

- funciones de autocompletado;
- un editor de código fuente;
- gestión de conexiones a bases de datos;
- integración con sistemas de control de versiones;
- simuladores de dispositivos;



- un depurador para agilizar el proceso de desarrollo de software, entre otros.

En resumen

Sin el lenguaje de programación, programar sería imposible, debido a que no existirían reglas (tanto semánticas como sintácticas), expresiones (como la estructura y el significado de todos los elementos que los componen) ni una forma establecida sobre cómo deben “hablar” el programador y la máquina.

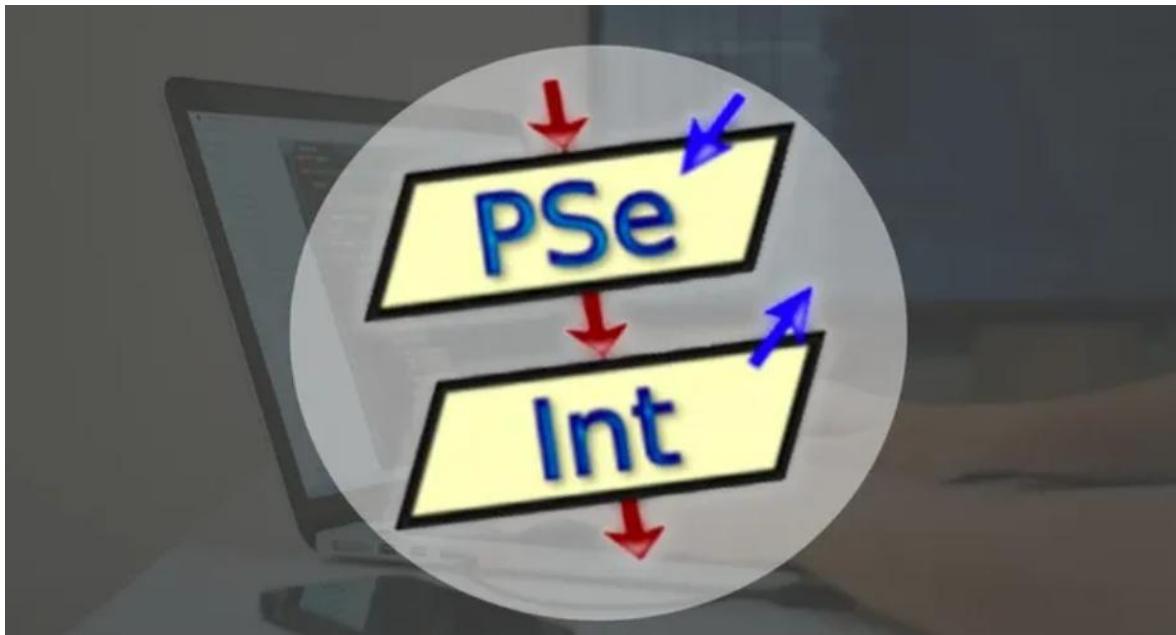
Además, algunas de las funciones que le permiten a un programador crear este lenguaje son: crear una web y hacerla funcionar o desarrollar aplicaciones para los sistemas operativos, entre muchas otras.

En la actualidad, el conocimiento y uso del mundo digital y de la informática son dos de las principales armas de cualquier empresa.



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA Lógica de Programación



SESION 02

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I1: Diseña algoritmos utilizando las estructuras básicas de control: secuencial, selectiva y repetitivas



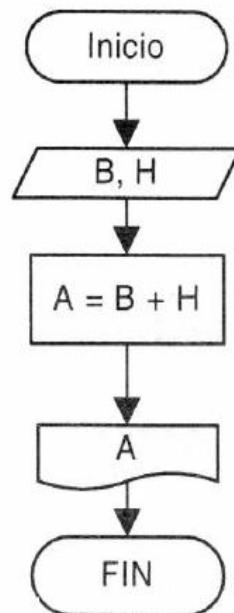
Técnicas de Diseño de Algoritmos:

Diagrama de flujo

El diagrama de flujo es la representación gráfica de dicha secuencia de instrucciones que conforman el algoritmo.

Un diagrama de flujo es un diagrama que describe un proceso, sistema o algoritmo informático. Se usan ampliamente en numerosos campos para documentar, estudiar, planificar, mejorar y comunicar procesos que suelen ser complejos en diagramas claros y fáciles de comprender. Los diagramas de flujo emplean rectángulos, óvalos, diamantes y otras numerosas figuras para definir el tipo de paso, junto con flechas conectadoras que establecen el flujo y la secuencia. Pueden variar desde diagramas simples y dibujados a mano hasta diagramas exhaustivos creados por computadora que describen múltiples pasos y rutas.

El siguiente es un ejemplo de un diagrama de flujo para sumar dos variables B y H, el resultado es almacenado en la variable A.



Los símbolos más comunes y los cuales usaremos son:



Terminal: Se usa para indicar el inicio o fin de un diagrama de flujo



Proceso: Cualquier tipo de operación que pueda originar cambio de valor, operaciones aritméticas, de transformaciones, etc.



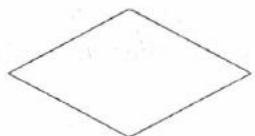
Entrada/Salida: Se usa para indicar el ingreso o salida de datos.



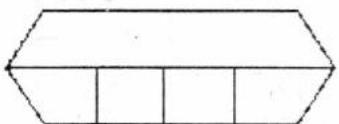
Salida: Se utiliza para mostrar datos, será el símbolo usado en este texto.



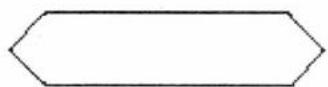
Conector: Sirve para enlazar dos partes cualesquiera de un diagrama a través de un conector en la salida y otro conector en la entrada.



Salida: Indica operaciones lógicas o de comparación entre datos y en función del resultado de la misma determina cual de los distintos caminos alternativos del programa se debe seguir.



En caso de: Usado para indicar varias acciones posibles según sea un dato de entrada al control.



Desde: Estructura repetitiva q indica un ciclo de N repeticiones de una o mas acciones.

Variables

Son los elementos que se utilizan para contener datos de distintos tipos: números, letras, cadenas de caracteres, valores lógicos, etc. El valor contenido en una variable puede cambiar a lo largo de la ejecución de un programa.

Constantes

Son elementos que contienen datos, el valor asignado a una constante es fijo y no se puede cambiar durante toda la ejecución de un programa.

Expresiones

Las expresiones son combinaciones de constantes, variables, símbolo de operación, paréntesis y nombres de funciones especiales.



Por ejemplo

$$a + (b + 3) / c$$

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones implicadas. Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican las expresiones en:

Aritméticas, Relacionales y Lógicas

Operadores

Operadores Aritméticos

Los operadores aritméticos nos permiten, básicamente, hacer cualquier operación aritmética (suma, resta, multiplicación y división). En la siguiente tabla se muestran los operadores de los que se dispone.

Operador	Java	Acción	Ejemplo	Result.
-	-	Resta	5-2	3
+	+	Suma	5+2	7
*	*	Multiplicación	5*2	10
/	/	División	5/2	2.5
^	pow	potencia	5^2 pow(5,2)	25
MOD	%	Resto de una división de dos números enteros	5 MOD 2 5 % 2	1
DIV	/	Parte entera de la división de dos números enteros	5 DIV 2 5/2	2

El operador MOD nos devuelve el residuo de una división entera, mientras que el operador DIV permite realizar una división entre dos números enteros, allí radica la diferencia con el operador

Operadores relacionales

Al igual que en matemáticas, estos operadores nos permitirán evaluar las relaciones (igualdad, mayor, menor, etc) entre un par de operandos (en principio, pensemos en números). Los operadores relationales de los que disponemos son:



Operador	Java	Acción
>	>	Mayor que
<	<	Menor que
=	==	Igual que
>=	>=	Mayor igual que
<=	<=	Menor igual que
<> o !=	!=	Diferente que

Operadores Lógicos

Los operadores lógicos producen un resultado **booleano**, y sus operandos son también valores lógicos o asimilables a ellos (los valores numéricos son asimilados a **cierto** o **falso** según su valor sea cero o distinto de cero).

Los operadores lógicos son tres; dos de ellos son binarios, el último (negación) es unario.

AND (Y)

Produce un resultado con valor de verdad (true) cuando ambos operandos tienen valor de verdad true; en cualquier otro caso el resultado tendrá un valor de verdad false.

Sintaxis: operando1 AND operando2

OR (O)

Produce un resultado con valor de falso (false) cuando ambos operadores tienen valores falsos; en cualquier otro caso el resultado tendrá un valor de verdad.

Sintaxis: operando1 OR operando2

NOT (NO)

Invierte el valor de verdad de operando.

Sintaxis: NOT operando

Prioridad de los Operadores

Los operadores deben ser evaluados según la siguiente prioridad



- 1.- ()
- 2.- ^
- 3.- *, /, Mod, Not
- 4.- +, -, And
- 5.- >, <, >=, <=, <>, =, Or

ENTRADA / SALIDA de datos

Los dispositivos de entrada/salida permiten que el usuario interactúe con la máquina. Por medio de los dispositivos de entrada el usuario ingresa los datos a procesar en el sistema y los dispositivos de salida presentan los resultados en un formato legible.

Asignaciones

Una asignación consiste en darle un determinado valor a una variable o constante, por ejemplo, en la siguiente sentencia observamos que a la variable A, se le da el valor de 5.

A = 5

De manera similar podemos tener la siguiente asignación

A = 4 + (3 * YA2)

vemos que una expresión ha sido asignada a la variable A

Algunos autores usan el símbolo \leftarrow en vez de igual (=) para una asignación.

Declaración de variables

Mediante la declaración de variables describimos las características de las mismas. La sintaxis que usaremos es la siguiente:

Nombre de_variable: Tipo

Entiéndase por tipo, al tipo de dato de la variable

Tipos de datos escalares

Son aquellos tipos de datos cuyos miembros están compuestos por un solo ítem (dato). Los tipos de datos escalares nativos son aquellos tipos de datos escalares que ya están implementados en el lenguaje junto a sus respectivas operaciones.

Tipo	Java	Tamaño
byte	byte	1 bytes (8 bits)
entero corto	short	2 bytes (16 bits)
entero	int	4 bytes (32 bits)
entero largo	long	8 bytes (64 bits)
real	float	4 bytes (32 bits)
doble	double	8 bytes (64 bits)
carácter	char	2 bytes (16 bits)



Pseudocódigo

Diremos que una notación es un pseudocódigo si mediante ella podemos describir el algoritmo, utilizando palabras y frases del lenguaje natural sujetas a unas determinadas reglas.

Todo pseudocódigo debe posibilitar la descripción de:

- Instrucciones de entrada / salida.
- Instrucciones de proceso.
- Sentencias de control del flujo de ejecución.
- Acciones compuestas, a refinar posteriormente.

Para entender mejor estos conceptos veamos algunos ejemplos.

Programación Orientada a Objetos (POO)

La programación orientada a objetos intenta tener una semejanza al modo de pensar del hombre, debido a la forma en que se manejan las abstracciones que representan las entidades del dominio del problema, y las propiedades como la jerarquía o el encapsulamiento.

A diferencia de la programación estructurada, el elemento básico es el objeto, el cual es la representación de un concepto en el programa. El objeto contiene los datos y las operaciones para manipular dichos datos.

Java incorpora el uso de la orientación a objetos como uno de los pilares básicos de su lenguaje.

Objetos

Podemos definir objeto como el "encapsulamiento de un conjunto de operaciones (métodos) que pueden ser invocados externamente, y de un estado que recuerda el efecto de los servicios

Tiempo de vida: La duración de un objeto en un programa siempre está limitada en el tiempo. La mayoría de los objetos sólo existen durante una parte de la ejecución del programa. Son creados por un constructor y finalizan mediante un destructor

Estado: Todo objeto posee un estado, definido por sus *atributos*. Con él se definen las propiedades del objeto, y el estado en que se encuentra en un momento determinado de su existencia.

Comportamiento: Todo objeto ha de presentar una interfaz, definida por sus *métodos*, para que el resto de objetos que componen los programas puedan interactuar con él.



Clases

Las clases son abstracciones que representan a un conjunto de objetos con un comportamiento e interfaz común. Es la implementación de un tipo de objeto (considerando los objetos como instancias de las clases).

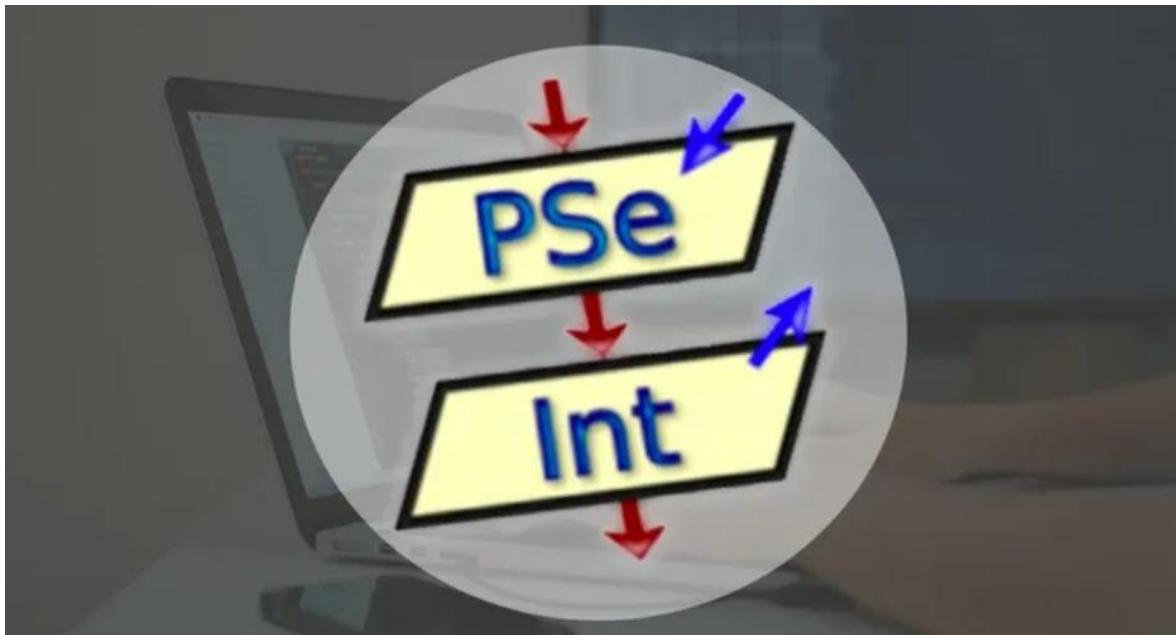
Una clase no es más que una plantilla para la creación de objetos. Cuando se crea un objeto (*instanciación*) se ha de especificar de qué clase es el objeto instanciado, para que el compilador comprenda las características del objeto.

Los *métodos* son las funciones mediante las que las clases representan el comportamiento de los objetos. En dichos métodos se modifican los valores de los atributos del objeto, y representan las capacidades del objeto (en muchos textos se les denomina *servicios*).



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA Lógica de Programación



SESION 03

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I1: Diseña algoritmos utilizando las estructuras básicas de control: secuencial, selectiva y repetitivas



SIMBOLOGIA DE LOS DIAGRAMAS DE FLUJO

¿Qué es un diagrama de flujo de datos?

Un diagrama de flujo de datos (DFD) traza el flujo de la información para cualquier proceso o sistema. Emplea símbolos definidos, como rectángulos, círculos y flechas, además de etiquetas de texto breves, para mostrar las entradas y salidas de datos, los puntos de almacenamiento y las rutas entre cada destino. Los diagramas de flujo de datos pueden variar desde simples panoramas de procesos incluso trazados a mano, hasta DFD muy detallados y con múltiples niveles que profundizan progresivamente en cómo se manejan los datos. Se pueden usar para analizar un sistema existente o para modelar uno nuevo. De forma similar a todos los mejores diagramas y gráficos, un DFD puede con frecuencia "decir" visualmente cosas que serían difíciles de explicar en palabras y funcionan para audiencias tanto técnicas como no técnicas, desde desarrolladores hasta directores. Esa es la razón por la que los DFD siguen siendo tan populares después de todos estos años. Aunque funcionan muy bien para software y sistemas de flujo de datos, en la actualidad no se aplican tanto para visualizar software o sistemas interactivos, en tiempo real u orientados a bases de datos.

DIAGRAMAS DE FLUJO: DFD

DFD es un programa de libre disposición para ayuda al diseño e implementación de algoritmos expresados en diagramas de flujo (DF). Además, incorpora opciones para el depurado de los algoritmos, lo que facilita enormemente la localización de los errores de ejecución y lógicos más habituales.

Su utilización es muy sencilla, al tratarse de una herramienta gráfica, y además incluye un menú de ayuda muy completo, por lo que en estas notas nos vamos a centrar en el uso básico de las herramientas de diseño y depuración. El resto de opciones (detalles de sintaxis más avanzados, operadores y funciones disponibles), puede consultarse directamente en la ayuda del programa.

1. INICIO DE DFD

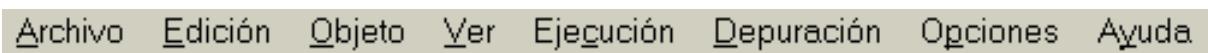
La ejecución de DFD presenta la pantalla de inicio siguiente



La ventana principal de trabajo del DFD está compuesta de los siguientes elementos:

- Barra de menús
- Botones de Archivo
- Botones de Edición
- Botones de Ejecución
- Botones de Depuración
- Botones de Zoom
- Botones de Objetos
- Botones de Subprogramas

LA BARRA DE MENUS



Aquí encontramos las siguientes opciones:

A. Archivo: Este menú se encarga del manejo del archivo que estemos trabajando.

Presenta los siguientes comandos:

- **Nuevo:** Para crear un nuevo diagrama DFD.
- **Abrir:** Para abrir un diagrama DFD guardado.



- **Guardar:** Para guardar el diagrama DFD que se está trabajando.
- **Guardar como:** Para guardar con un nombre diferente el diagrama DFD que se está utilizando.
- **Imprimir:** Para imprimir el archivo DFD abierto.
- **Salir:** Para terminar la sesión en Smart DFD 1.0.

B. Edición: Este menú es el encargado de manejar los comandos de Edición del diagrama:

- **Cortar:** Para quitar un bloque del diagrama y guardarlo en el portapapeles.
- **Copiar:** Para copiar el bloque marcado en el diagrama y guardarlo en el portapapeles.
- **Pegar:** Para pegar en el diagrama el bloque guardado en el portapapeles.
- **Eliminar:** Para borrar el bloque marcado en el diagrama.
- **Eliminar Subprograma:** Para borrar el diagrama correspondiente a un subprograma.
- **Editar Objeto:** Para cambiar las propiedades o valores del bloque marcado en el diagrama.

C. Objeto: En este menú se establecen todos los comandos para la inserción de bloques en el diagrama:

- **Cursor:** Mantiene activa la opción de puntero del mouse, permitiendo desplazarse dentro del diagrama y marcar bloques.
- **Asignación:** Permite insertar un bloque de asignación en el diagrama.
- **Ciclo Mientras:** Permite insertar una estructura repetitiva MIENTRAS en el diagrama.
- **Ciclo Para:** Permite insertar una estructura repetitiva PARA en el diagrama.
- **Decisión:** Permite insertar una estructura CONDICIONAL en el diagrama.
- **Lectura:** Permite insertar una instrucción de lectura de datos por teclado en el diagrama.
- **Llamada:** Para insertar la llamada a un subprograma o subrutina dentro del diagrama.
- **Salida:** Para insertar la presentación de datos en la pantalla.
- **Nuevo Subprograma:** Para abrir la ventana de edición para la elaboración del diagrama de un subprograma o subrutina.



D. Ver: Este menú habilita los siguientes comandos:

- **Zoom:** Para ampliar o disminuir la vista general del diagrama.
- **Anterior Subprograma:** Para ver el diagrama del anterior subprograma cuando estos existan.
- **Siguiente Subprograma:** Para ver el diagrama del siguiente subprograma cuando estos existan.
- **Depurador:** Para evaluación de expresiones y tipos de datos.

E. Ejecución: Este menú activa la ejecución del diagrama, contiene los siguientes comandos:

- **Ejecutar:** Para iniciar la ejecución del diagrama.
- **Pausar:** Para pausar la ejecución del diagrama.
- **Detener:** Para interrumpir la ejecución del diagrama.

F. Depuración: Controla las actividades de depuración del diagrama. Se compone de los siguientes comandos:

- **Paso Simple:** Evalúa cada instrucción y bloque paso a paso e indica por qué camino va el control sobre el diagrama.
- **Ejecutar Hasta:** Permite marcar un bloque dentro del diagrama y realizar la ejecución del diagrama solo hasta ese bloque.
- **Detener:** Para interrumpir la depuración.
- **Evaluar:** Permite evaluar expresiones y datos.

G. Opciones: controla las opciones de trabajar los ángulos en grados o en radianes.

H. Ayuda: Invoca el manual de ayuda del DFD.

LA BARRA DE BOTONES

DFD se caracteriza por tener botones que activan comandos. Estos botones y comandos son:



Botón	Comando	Función
	Nuevo	Crear un nuevo diagrama DFD.
	Abrir	Abrir un diagrama DFD existente.
	Guardar	Guardar el diagrama DFD que se está trabajando
	Imprimir	Imprimir el diagrama DFD que se está trabajando
	Cortar	Quitar un bloque del diagrama y guardarlo en el portapapeles
	Copiar	Copiar el bloque marcado en el diagrama y guardarlo en el portapapeles
	Pegar	Pegar en el diagrama el bloque guardado en el portapapeles
	Eliminar	Borrar el bloque marcado en el diagrama
	Ejecutar	Iniciar la ejecución del diagrama
	Detener	Interrumpir la ejecución del diagrama
	Pausar	Pausar la ejecución del diagrama
	Paso Simple	Evalúa cada instrucción y bloque paso a paso e indica por que camino va el control sobre el diagrama
	Ejecutar Hasta	Realizar la ejecución del diagrama solo hasta el bloque marcado
	Depurador	Activa el depurador del diagrama
	Alejar	Reducir la vista del diagrama
	Acercar	Ampliar la vista del diagrama
	Cursor	Activar el puntero del mouse para desplazarse dentro del diagrama y marcar bloques
	Asignación	Insertar un bloque de asignación en el diagrama
	Ciclo Mientras	Insertar una estructura repetitiva MIENTRAS en el diagrama
	Ciclo Para	Insertar una estructura repetitiva PARA en el diagrama
	Decisión	Insertar una estructura CONDICIONAL en el diagrama
	Lectura	Insertar una instrucción de lectura de datos por teclado en el diagrama
	Salida	Insertar la presentación de datos en la pantalla
	Llamada	Insertar la llamada a un subprograma o subrutina dentro del diagrama
	Anterior Subprograma	Ver el diagrama del anterior subprograma cuando estos existan.
	Siguiente Subprograma	Ver el diagrama del siguiente subprograma cuando estos existan
	Nuevo Subprograma	Abrir una nueva ventana de edición para la elaboración del diagrama de un subprograma o subrutina
	Eliminar Subprograma	Borrar el diagrama correspondiente a un subprograma



LA CREACIÓN DE DIAGRAMAS DE FLUJO EN DFD

Para crear un diagrama de flujo en Dfd deben escogerse los tipos de bloques a utilizar pulsando sobre el botón de los siguientes:



Después de seleccionado el botón se desplaza el elemento hasta el lugar en el diagrama en el que se desea insertar, para editar el objeto se debe pulsar doble clic con el botón del mouse.

a. Edición de un bloque de lectura: Al dar doble clic sobre este bloque aparece la siguiente ventana:



Dentro de la casilla leer, se deben especificar las variables que van a ser leídas separadas por coma (,) y pulsar el botón "Aceptar".

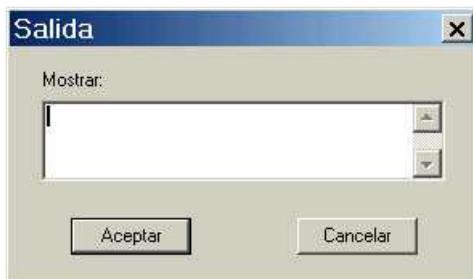
b. Edición de un bloque de asignación: Al dar doble clic sobre este bloque aparece la siguiente ventana:



En la casilla de la izquierda se coloca la variable que va a recibir la asignación y en la casilla de la derecha, el valor, la variable o la operación que se va a asignar, y después se pulsa el botón "Aceptar". En este ejemplo se tiene una asignación $A = 34 + B$.

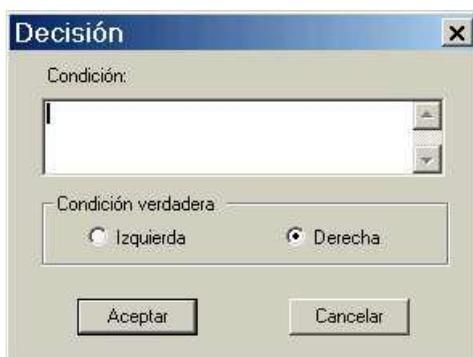


c. Edición de un bloque de salida: Al dar doble clic sobre este bloque aparece la siguiente ventana:



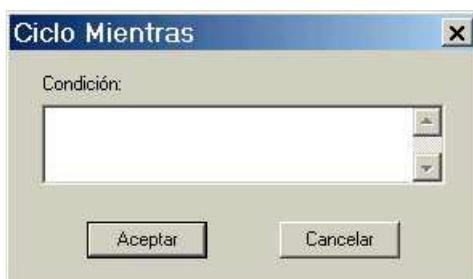
Dentro de la casilla mostrar, se deben especificar las variables que van a ser mostradas en pantalla separadas por coma (,) y pulsar el botón “Aceptar”. Si desea mostrar mensajes de texto éstos deben ser encerrados por comillas sencillas (').

d. Edición de una decisión: Al dar doble clic sobre este bloque aparece la siguiente ventana:



Dentro de la casilla Condición, se debe escribir la condición que se desea evaluar, indicar si la condición verdadera va a ser la derecha o la izquierda y pulsar el botón “Aceptar”.

e. Edición de un ciclo mientras: Al dar doble clic sobre este bloque aparece la siguiente ventana:





Dentro de la casilla Condición, se debe escribir la condición que se desea evaluar para realizar la repetición y pulsar el botón “Aceptar”.

f. Edición de un ciclo para: Al dar doble clic sobre este bloque aparece la siguiente ventana:



En la primera casilla (Cont) se debe indicar la variable que asume como contador del ciclo, en la segunda casilla (Vi) se escribe el valor inicial del contador; en la tercera casilla (Vf) se indica el valor final del contador y en la última casilla (Inrem) se escribe constante del incremento del contador; al finalizar se pulsa el botón “Aceptar”.

OPERADORES EN DFD

A. OPERADORES ARITMÉTICOS

DFD	FUNCION	SINTAXIS	RESULTADO
+	Suma	A+B	La suma de A y B.
-	Resta	A-B	La resta de A y B.
*	Multiplicación	A*B	El producto de A y B.
/	División Real	A/B	El cociente real de A entre B.
TRUNC(/)	División Entera	TRUNC(A/B)	El cociente entero de A entre B.
MOD	Residuo Entero	A MOD B	El residuo de A entre B.
^	Exponenciación	A^B	A elevado a la potencia B.

B. OPERADORES RELACIONALES

DFD	FUNCION	SINTAXIS	RESULTADO
<	Menor que	X<Y	V (Verdadero) si X menor que Y F (Falso) en caso contrario.
<=	Menor o igual que	X<=Y	V(Verdadero) si X menor o igual a Y , F (Falso) en caso contrario
=	Igual que	X=Y	V (Verdadero) si X es igual a Y y F (Falso) en caso contrario
>	Mayor que	X>Y	V (verdadero) si X mayor a Y F (falso) en caso contrario
>=	Mayor o igual que	X>=Y	V(Verdadero) si X mayor o igual a Y , F (Falso) en caso contrario
!=	Diferente de	X!=Y	V(Verdadero) si X es diferente de Y, F(Falso) en caso contrario



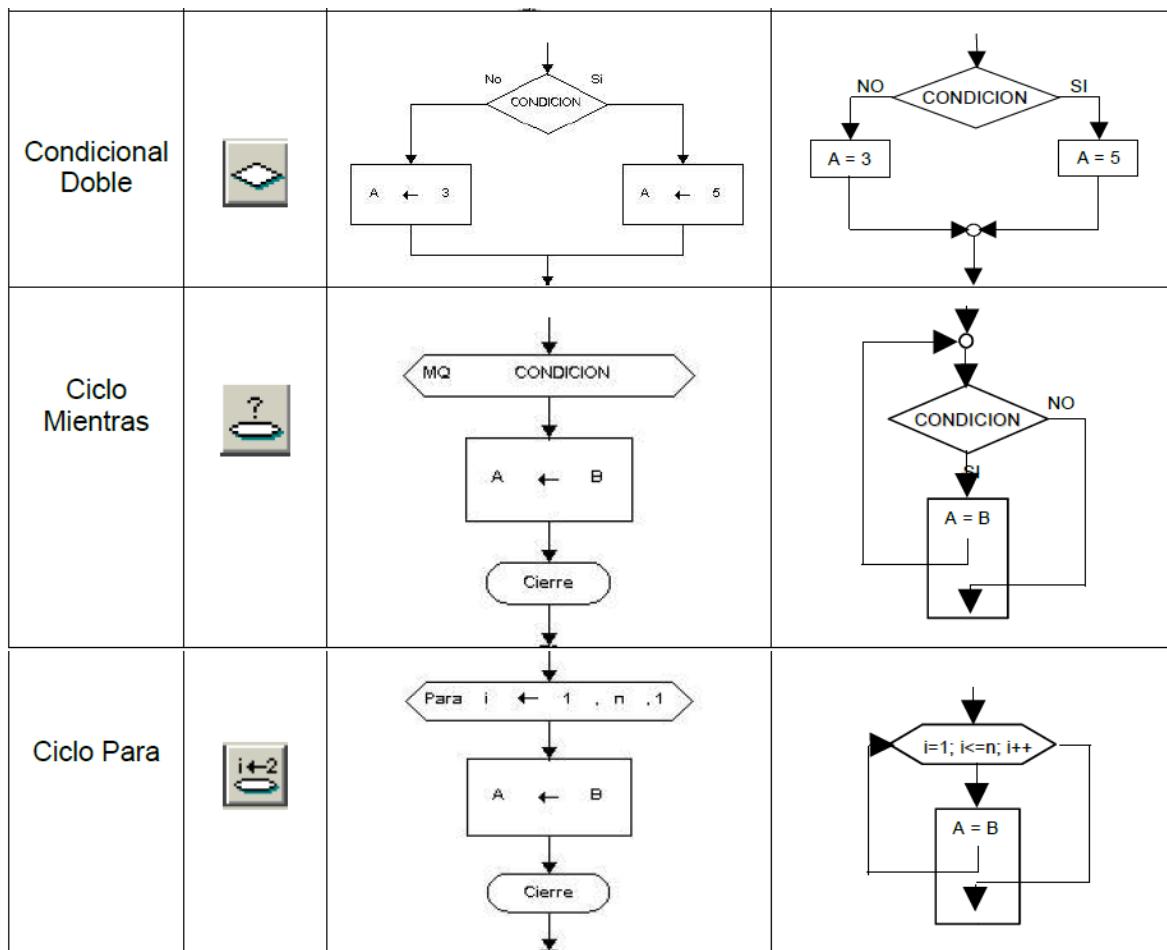
C. OPERADORES LOGICOS

DFD	FUNCION	SINTAXIS	RESULTADO
NOT	Negación Lógica	NOT A	NOT V = F NOT F = V
AND	Conjunción (Y Lógico)	A AND B	V AND V = V V AND F = F F AND V = F F AND F = F
NAND	Negación del AND	A NAND B	V AND V = F V AND F = V F AND V = V F AND F = V
OR	Disyunción (O Lógico)	A OR B	V AND V = V V AND F = V F AND V = V F AND F = F
NOR	Negación de OR	A NOR B	V AND V = F V AND F = F F AND V = F F AND F = V

FUNDAMENTOS DE PROGRAMACION

EQUIVALENCIA DE ESTRUCTURAS DFD CON LA SIMBOLOGIA ESTANDAR

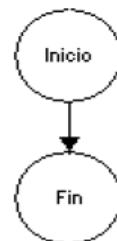
Estructura	Botón	DFD	Diagramas Estándar
Lectura			
Escritura			
Asignación			
Condicional Simple			

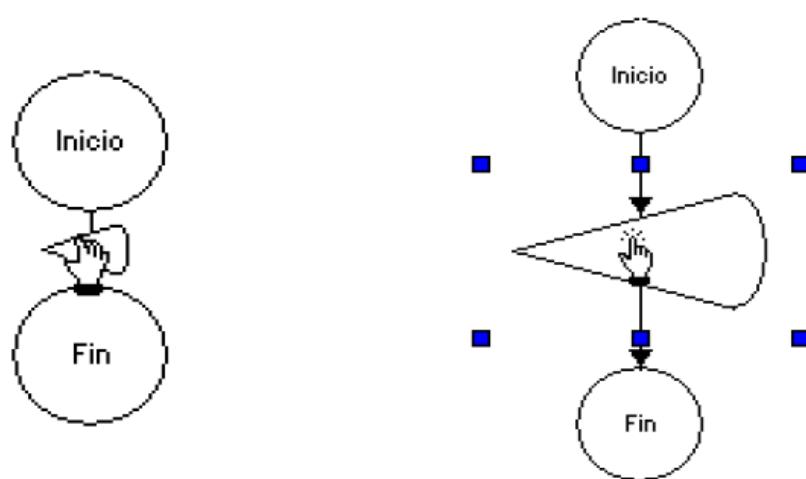


EJEMPLO 01.

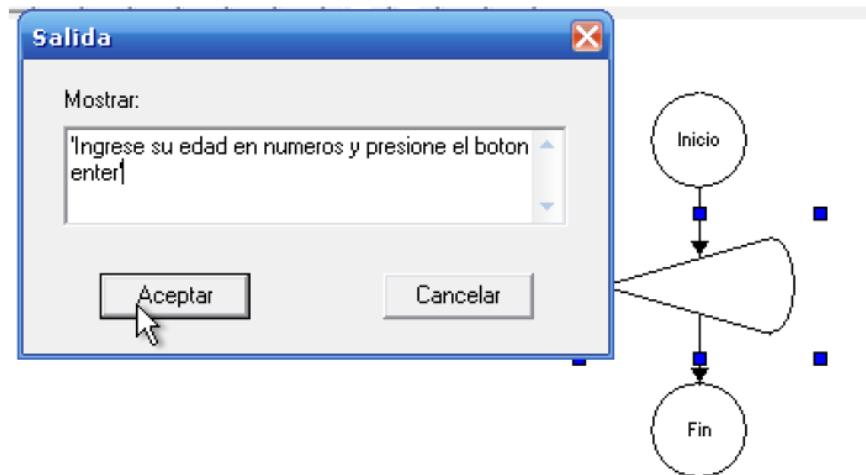
En el siguiente ejemplo consta en que el usuario ingrese su edad y él le responderá cuantos años tiene, se usaran objetos como salidas y lectura.

Abrir el programa (Dfd) y agregamos una salida.





Damos doble clic y escribimos entre **comillas simples** ('Ingrese su edad en números y presione el botón enter').

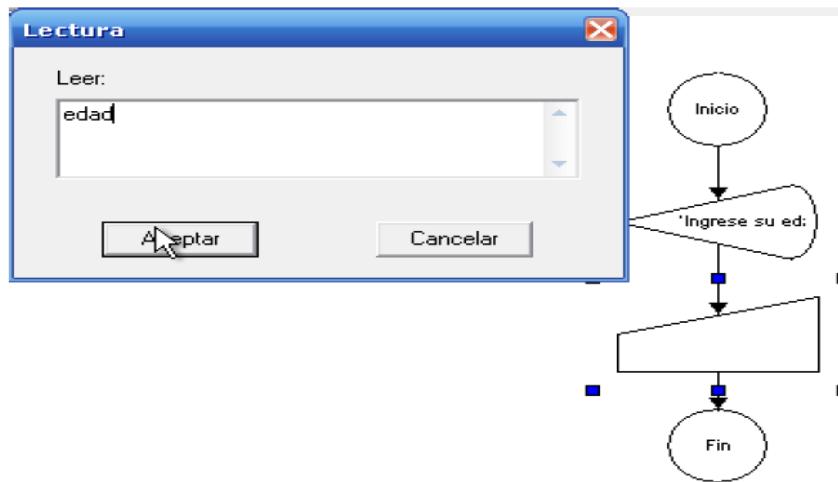


Ahora agregamos una lectura.

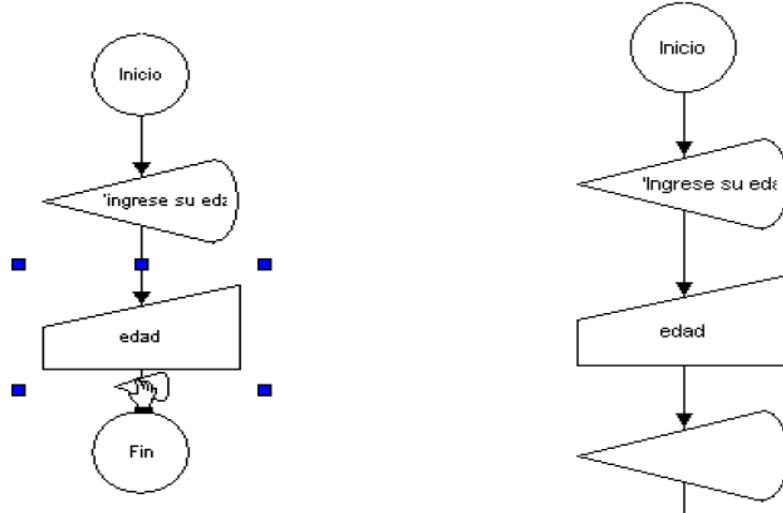




Damos doble clic y escribimos la variable edad sin comillas.

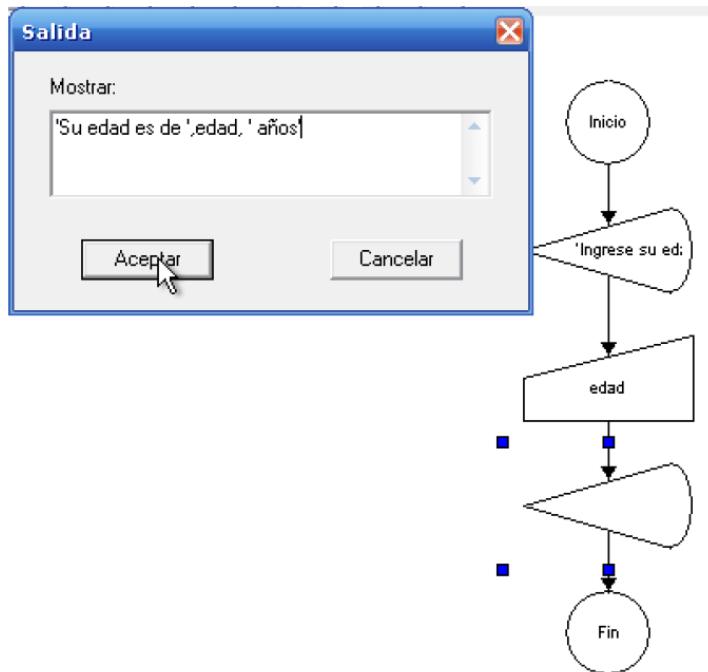


Agregamos una nueva salida.



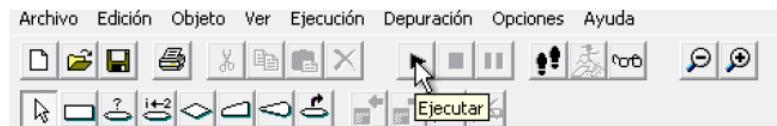


Damos doble clic y escribimos la respuesta entre comillas simples. Pero la variable (edad) debe ir fuera de las comillas y separada por comas. ('su edad es de ', edad, ' años').



Ya terminado el algoritmo tendrá la siguiente forma en la pantalla.

Damos Clic en ejecutar





El primer objeto en ejecutarse será la de SALIDA, que mostrará en pantalla el siguiente mensaje:



Seguidamente la de ENTRADA, que nos muestra un cuadro de texto donde introduciremos el valor que queramos darle a la variable edad (por ejemplo, 19):

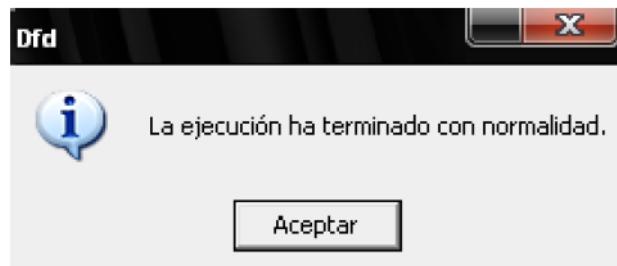


Finalmente, la última SALIDA:





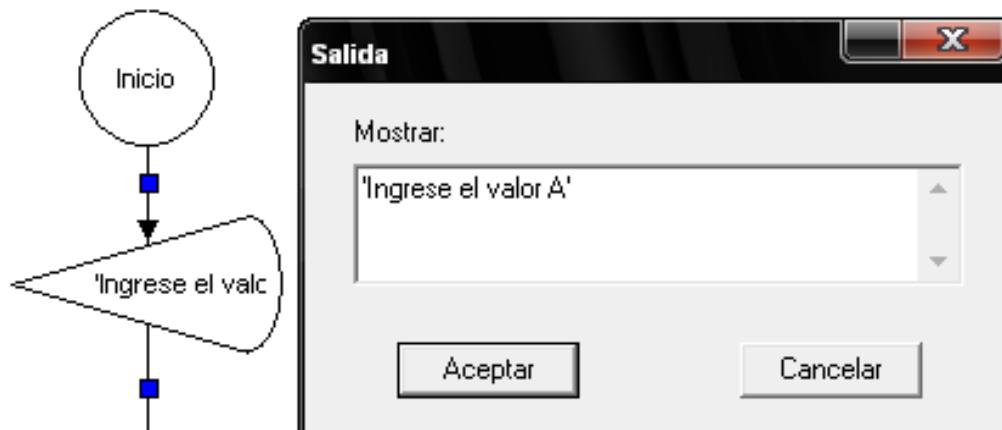
Cuando el algoritmo finaliza su ejecución y no tiene errores se muestra el siguiente mensaje:



EJEMPLO 02

Diseñaremos un algoritmo que pida tres valores A, B y C al usuario y calcule dos operaciones matemáticas como la suma, y promedio.

Ingresamos una salida y le escribimos ('Ingrese el valor A').

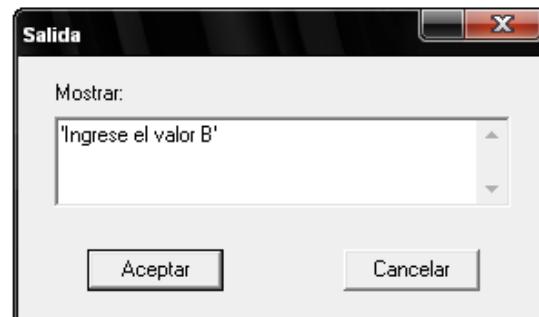
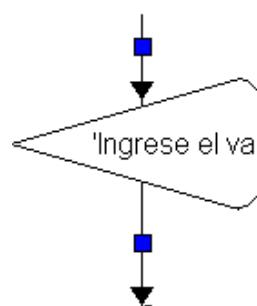


Ingresamos un objeto de lectura y le escribimos la variable (valora).

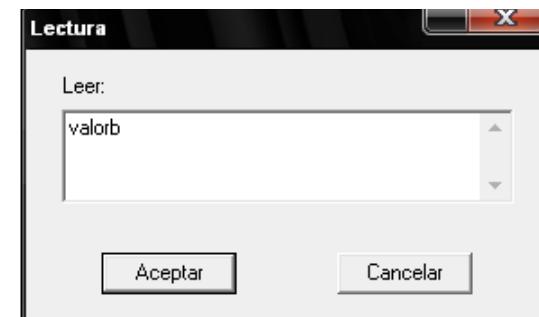
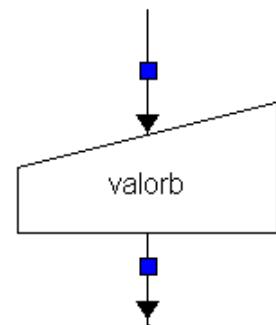




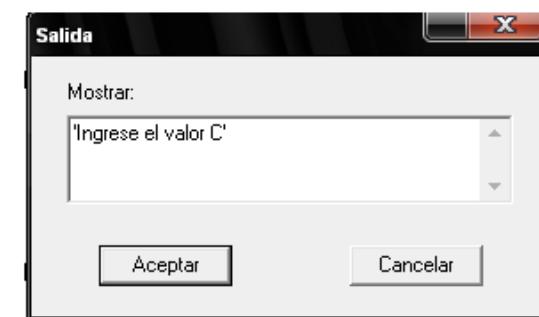
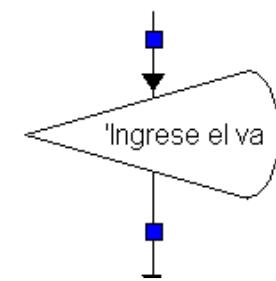
Ingrese una nueva salida para el (valor B).



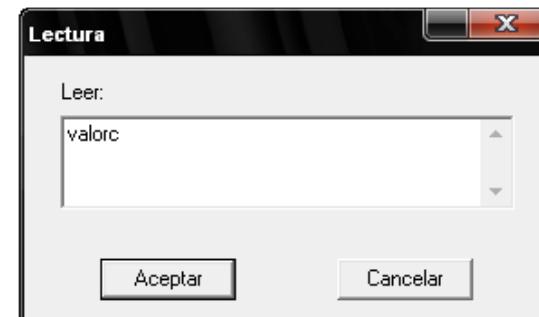
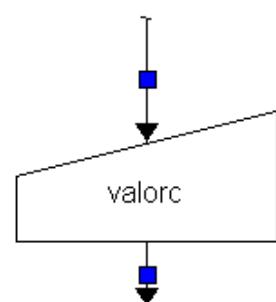
Ingrese una nueva entrada de lectura y nombrela (valorb).



Ingrese una nueva salida para el (valor C).

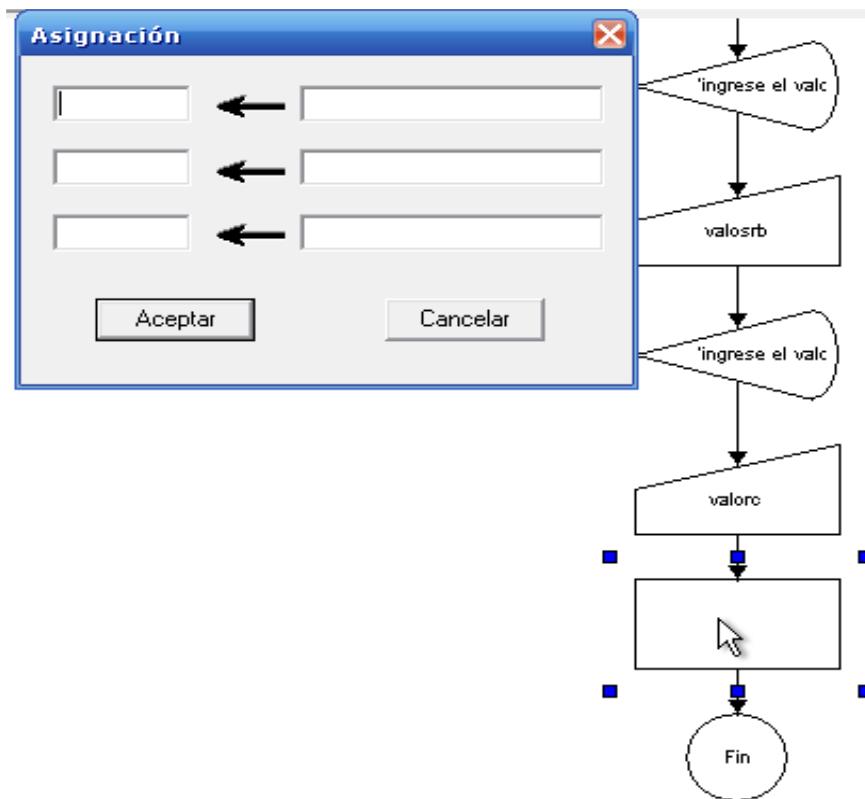
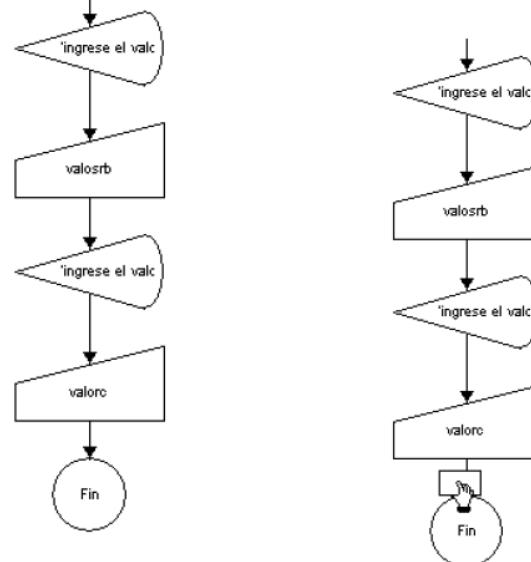


Ingrese una nueva entrada de lectura y nombrela (valorc).





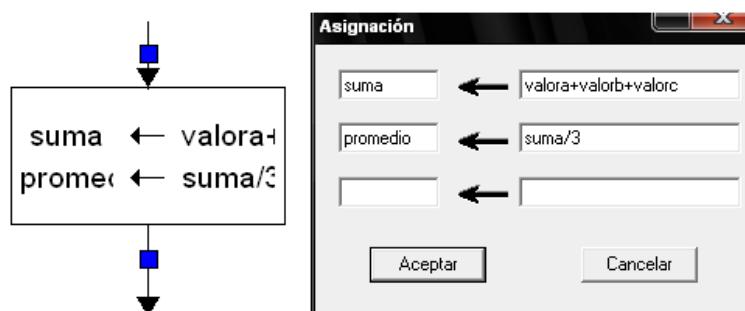
Ahora que ya hemos hecho las salidas y las entradas insertamos una asignación y le damos doble clic.



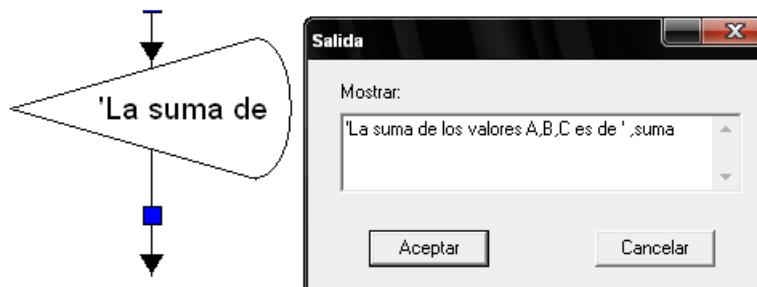


Nombramos a la primera fila de la columna izquierda (suma), la que será nuestra primera operación matemática, después en la primera fila de la columna derecha sumamos los nombres o variables que escribimos anteriormente en los tres objetos de lectura (valora+valorb+valorc).

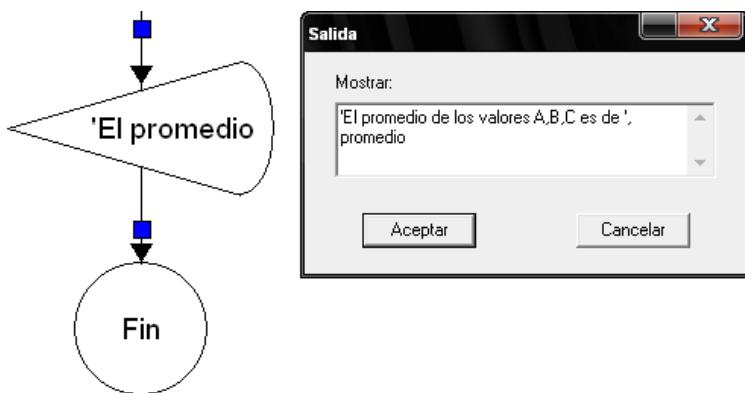
Para la segunda operación matemática de promedio simplemente llamamos a la segunda fila de la columna izquierda (promedio), y en la segunda fila de la columna derecha realizamos la operación matemática del promedio así: colocamos la asignación suma y la dividimos por el número de variables (suma/3).



Ingresamos una nueva salida para que nos muestre el resultado de la primera operación de la suma de las tres variables. ('la suma de los valores A,B,C es de', suma).



Ingresamos una nueva salida para que nos muestre el resultado de la segunda operación del promedio de las tres variables. ('El promedio de los valores A,B,C es de', promedio).





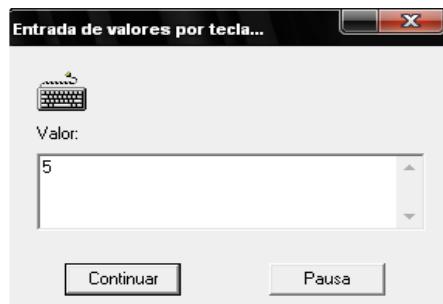
Ya terminado el algoritmo tendrá la siguiente forma en la pantalla.

Damos Clic en ejecutar

El primer objeto en ejecutarse será la de SALIDA, que mostrará en pantalla el siguiente mensaje.



Seguidamente la de ENTRADA, que nos muestra un cuadro de texto donde introduciremos el valor que queramos darle a la variable A (por ejemplo, 5):



Luego se ejecuta el segundo objeto de salida.

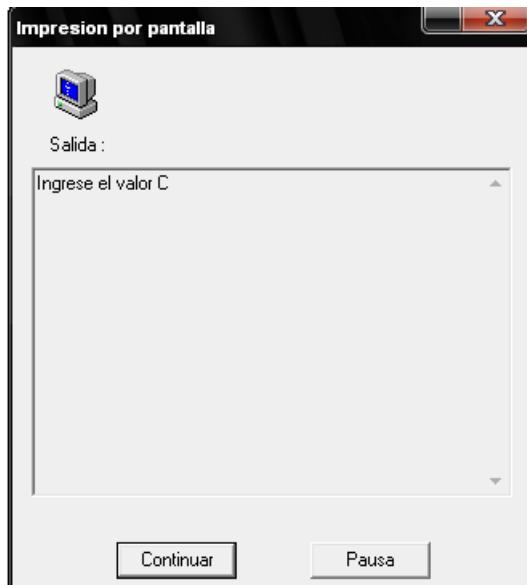




El segundo objeto de ENTRADA, que nos muestra un cuadro de texto donde introduciremos el valor que queramos darle a la variable B (por ejemplo, 4):



Luego se ejecuta el tercer objeto de salida.

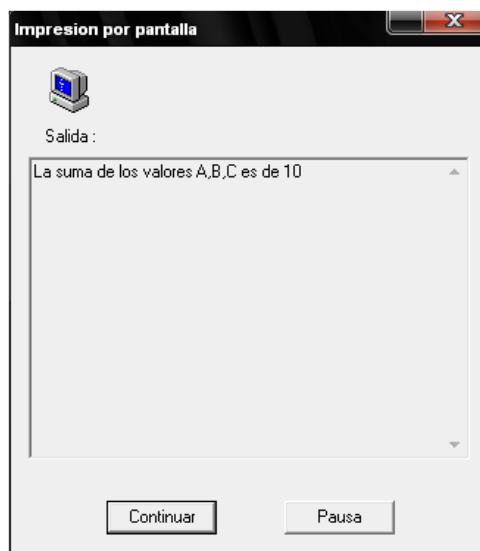


El tercer objeto de ENTRADA, que nos muestra un cuadro de texto donde introduciremos el valor que queramos darle a la variable c (por ejemplo, 1):

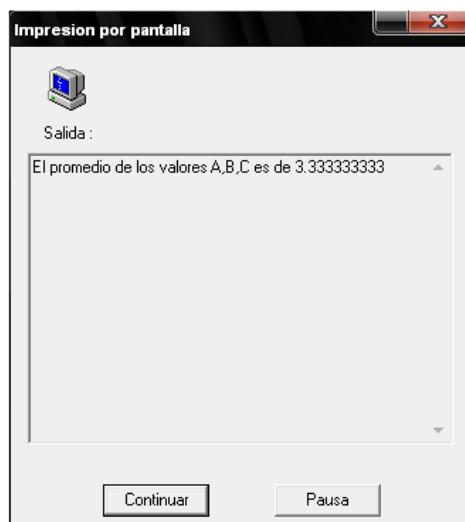




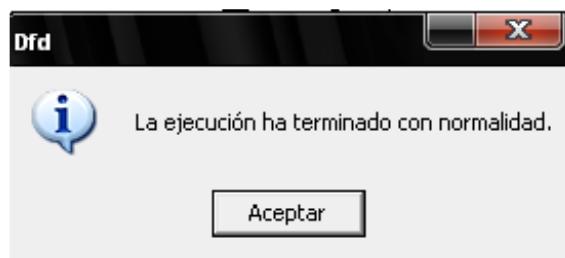
El cuarto objeto de salida con la respuesta a la primera operación matemática de sumar los valores A, B, C ($5+4+1=10$).



El quinto objeto de salida con la respuesta a la segunda operación matemática del promedio de valores A, B, C ($\text{suma}/3$).



El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.





EJERCICIOS PROPUESTOS

1. Leer un número y reportar su cuadrado.
 2. Leer un número y reportar su cubo.
 3. Leer un numero N y calcular su cuadrado y su cubo.
 4. Leer 2 números y Reportar la suma de ambos números.
 5. Leer 2 números y reportar el cuadrado del producto de ambos números.
 6. Leer dos notas y Calcular y reportar la nota promedio.



ACTIVIDAD DE ANALISIS Y DE INVESTIGACIÓN

1.- Escribir un programa que lea dos números, calcule y reporte las 4 operaciones básicas: suma, resta, multiplicación y división.

2.- Leer el lado de un cuadrado y calcular su área y su perímetro.

3.- Leer una temperatura en grados centígrados ($^{\circ}\text{C}$)y expresarla en grados Kelvin ($^{\circ}\text{K}$).

4.-Leer el radio de un círculo y calcular el área del círculo y la longitud de la circunferencia.

5.- Ingresar un número X y calcular X^2 y X^3 .

6.- Leer la base y la altura de un triángulo y calcular su área.

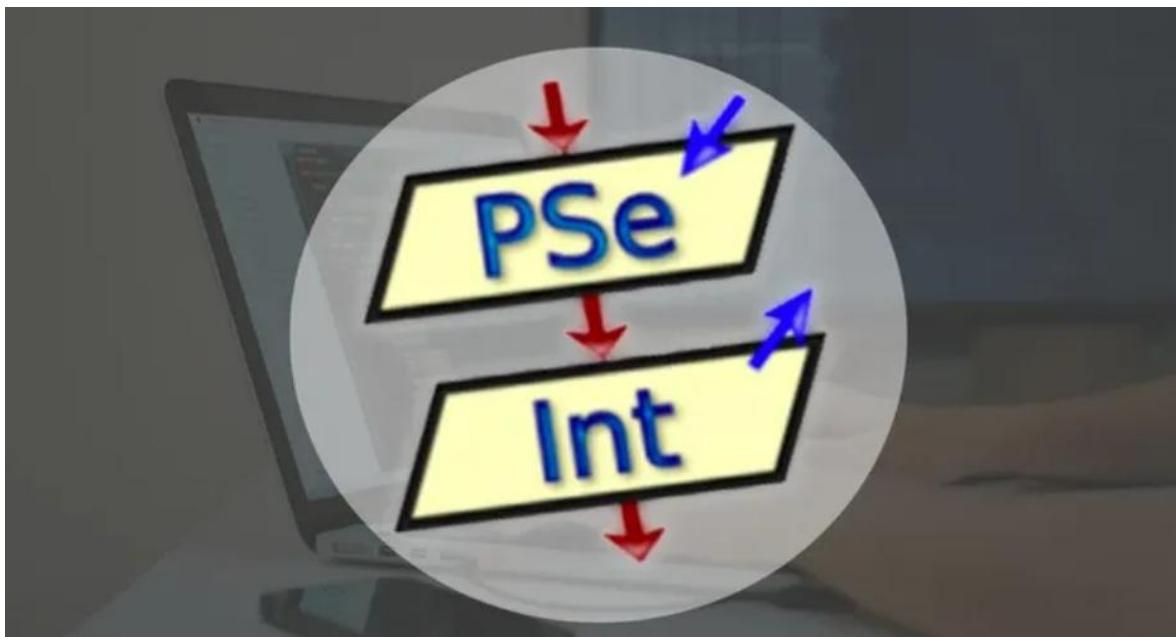
7.-Se sabe que un Megabyte equivale a 1024 Kbytes y un Gigabyte equivale a 1024 Megabytes. Si mi Computadora tiene un disco duro de X Gigabytes. Indicar a cuántos Kbytes equivale.



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA

Lógica de Programación



SESION 04

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I1: Diseña algoritmos utilizando las estructuras básicas de control: secuencial, selectiva y repetitivas



Estructura de Control Selectivas

Las estructuras lógicas selectivas se encuentran en la solución algorítmica de casi todo tipo de problemas. La utilizamos cuando en el desarrollo de la solución de un problema debemos tomar una decisión, para establecer un proceso o señalar un camino alternativo a seguir.

Esta toma de decisión se basa en la evaluación de una o más condiciones que nos señalarán como alternativa o consecuencia, la rama a seguir.

Estructura Si...Entonces (Selección simple)

La estructura selectiva *Si...Entonces* permite que el flujo del algoritmo siga por un camino específico si se cumple una condición o conjunto de condiciones. Si al evaluar la condición (o condiciones) el resultado es verdadero, entonces se ejecuta(n) cierta(s) operación(es). Luego se continúa con la secuencia normal del diagrama. La sintaxis para este tipo de estructura es la siguiente:

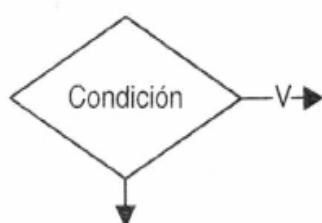
S i condición e n t o n c e s

H a c e r o p e r a c i ó n

F i n _ S i (F i n d e l c o n d i c i o n a l)

La condición es una expresión lógica. Y *operación* puede ser una operación simple o un grupo de operaciones.

El diagrama de flujo de la condicional Si simple es:

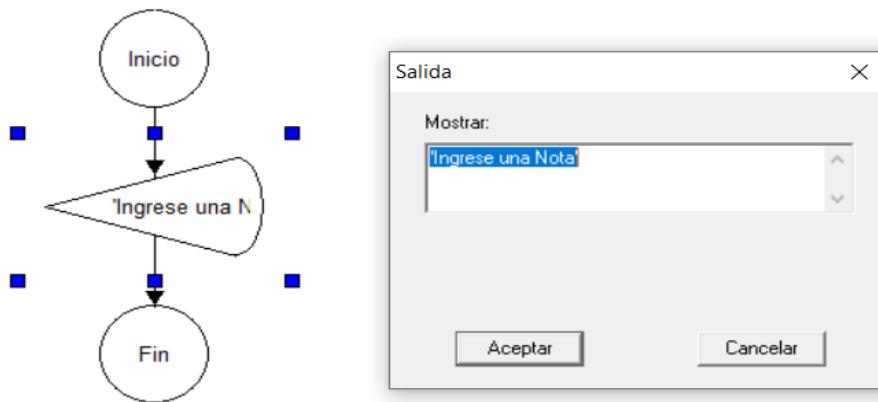


Si la condición es falsa no hace ninguna acción. Veamos unos ejemplos de una sentencia selectiva simple.

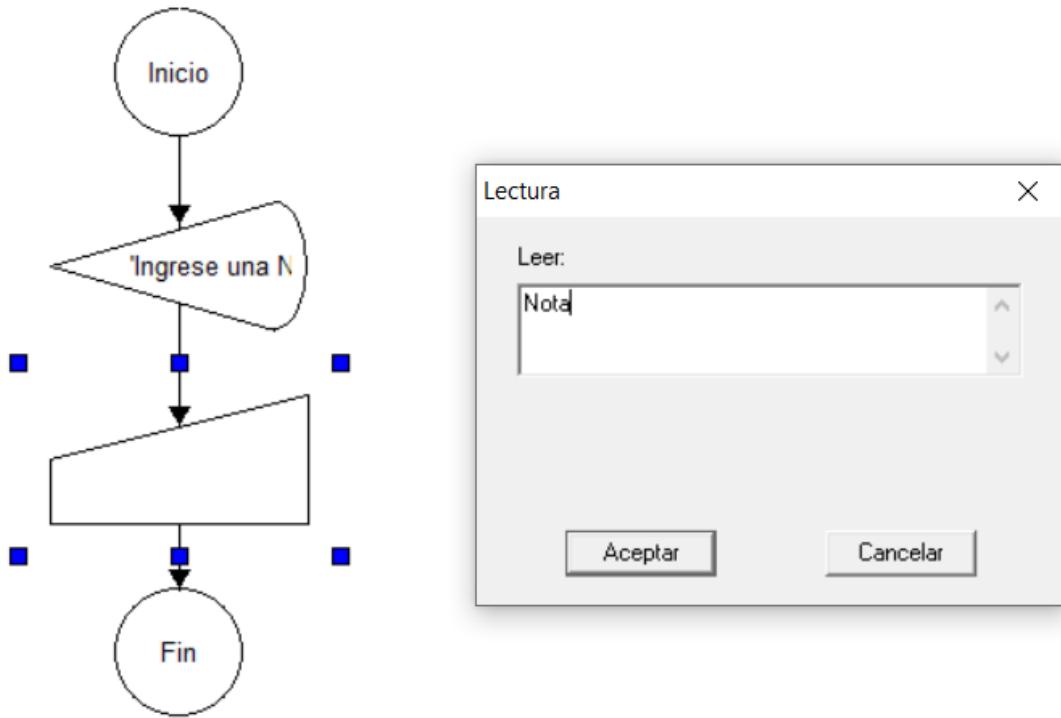


Ejemplo 01

Construya un Diagrama tal, que dado como dato la calificación de un alumno en un examen, escriba "Aprobado" en caso de que esa calificación fuese mayor que 10.5 colocamos un objeto de salida y escribimos ('Ingrese una Nota').

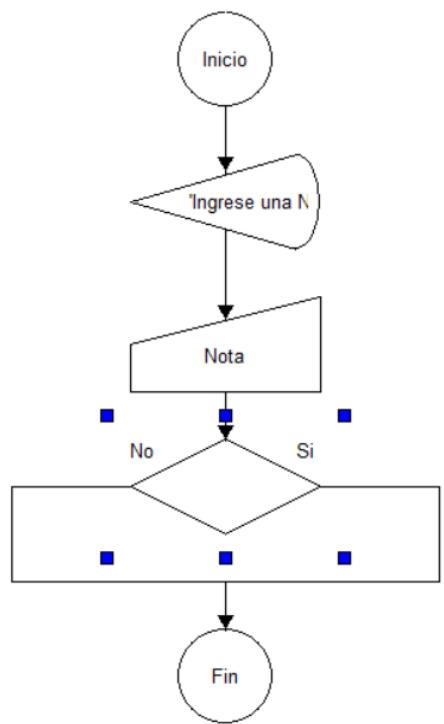


Ahora colocamos un objeto de lectura con la variable (Nota).



Ahora colocamos una decisión y le damos doble clic.

Le escribimos que la variable Nota > 10.5 y le escogemos la opción para que la condición sea verdadera (derecho).



Decisión

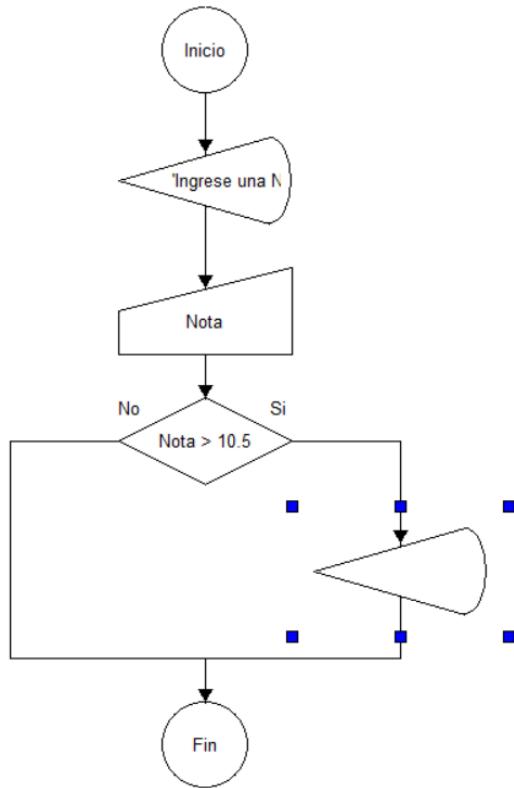
Condición:

Condición verdadera

Derecha
 Izquierda

Aceptar Cancelar

Colocamos un objeto de salida en el lado donde la condición sea verdadera (derecho) y escribimos ('APROBADO').



Salida

Mostrar:

Aceptar Cancelar



Damos Clic en ejecutar

Y lo hacemos de forma que la condición sea verdadera (Ingrese una Nota).

Primer objeto de salida.



El objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable Nota (por ejemplo, 11):

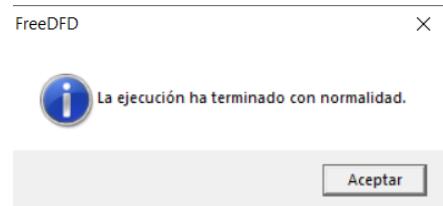


El objeto de salida que nos muestra cuando la condición es verdadera (APROBADO).



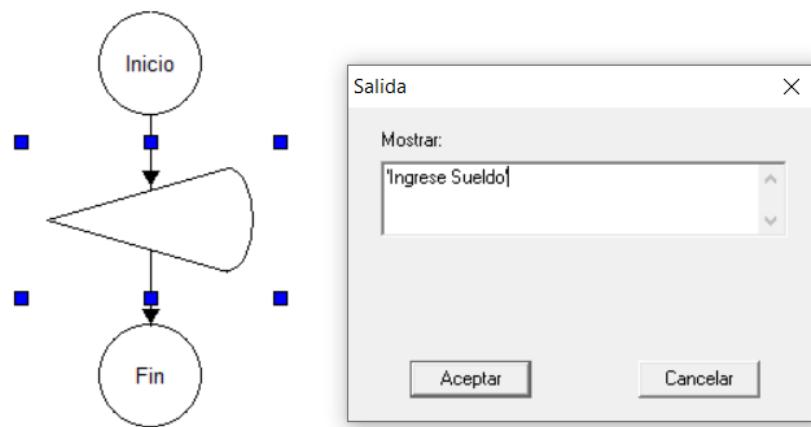


El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.

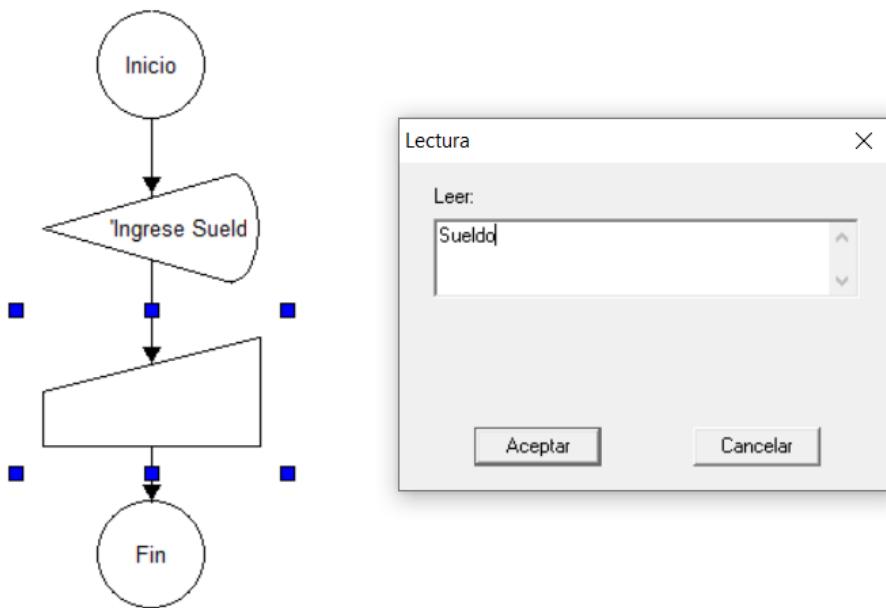


Ejemplo 02

Dado como dato el sueldo de un trabajador, aplíquele un aumento del 17% si su sueldo es inferior a S/ 1000. Imprima en este caso, el nuevo sueldo del trabajador.
colocamos un objeto de salida y escribimos ('Ingrese Sueldo').



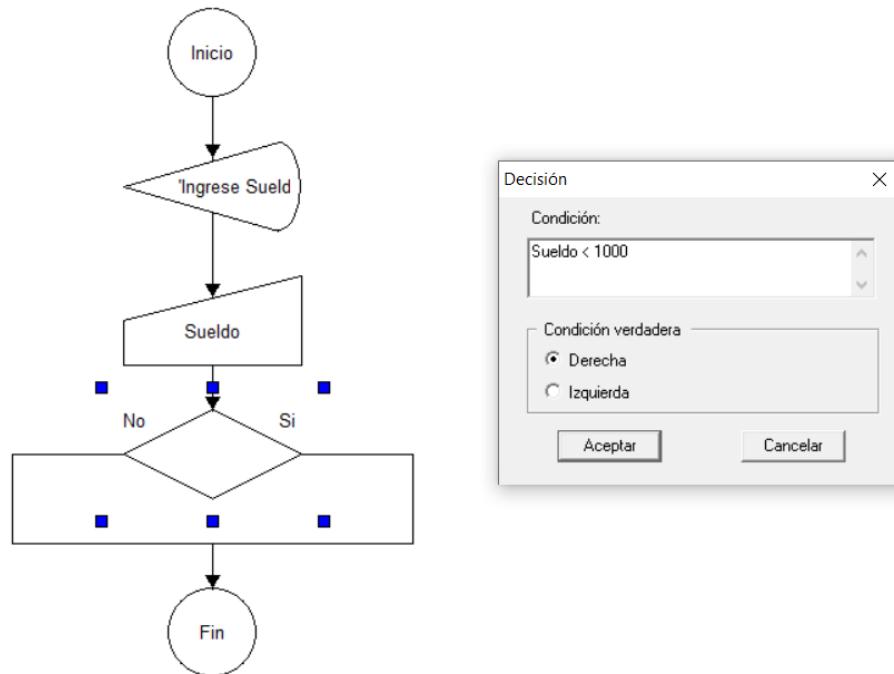
Ahora colocamos un objeto de lectura con la variable (Sueldo).



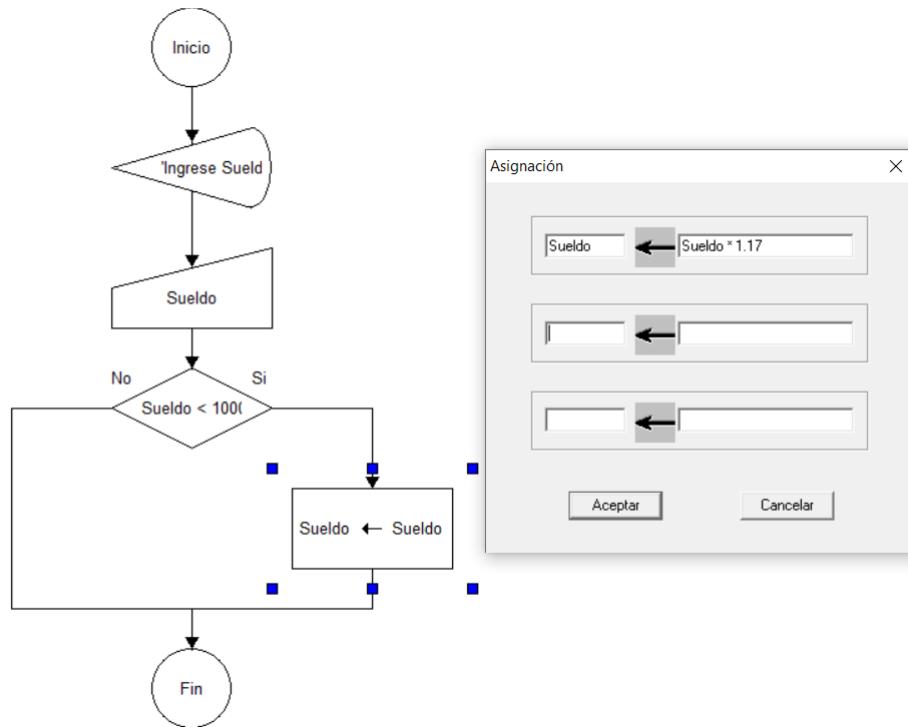


Ahora colocamos una decisión y le damos doble clic.

Le escribimos que la variable Sueldo < 1000 y le escogemos la opción para que la condición sea verdadera (derecho).

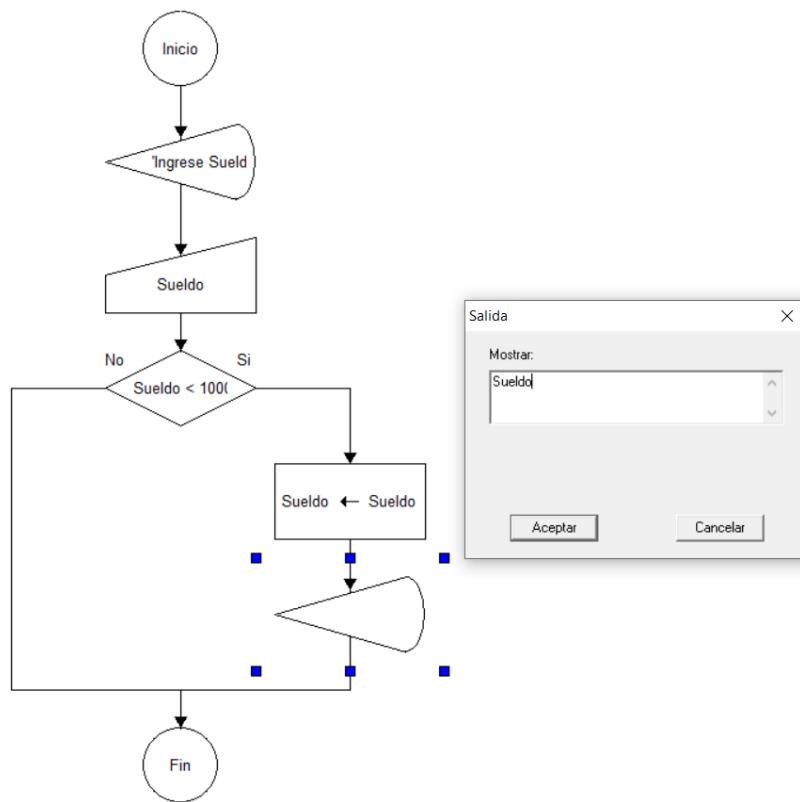


Colocamos un objeto de asignación en donde en el primer cuadro ponemos el nombre de la variable y en el segundo cuadro colocamos la condición que se nos pide en el ejercicio.





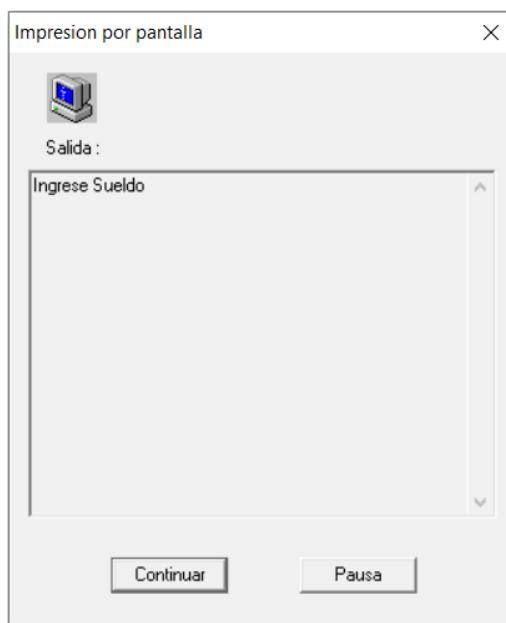
Colocamos un objeto de salida en el lado donde la condición sea verdadera (derecho) y escribimos ('Sueldo')



Damos Clic en ejecutar

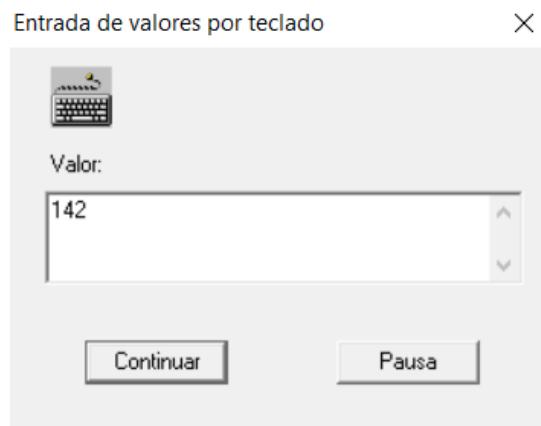
Y lo hacemos de forma que la condición sea verdadera (Ingrese Sueldo).

Primer objeto de salida.





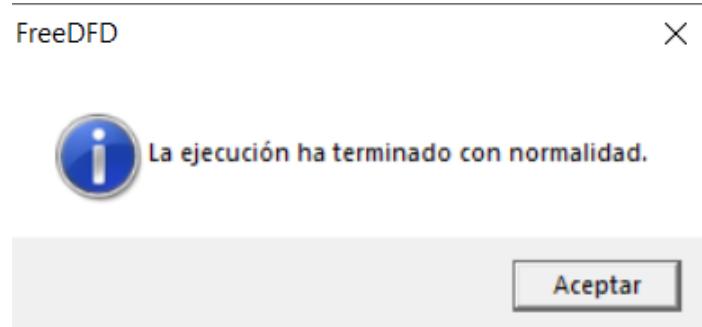
El objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable Sueldo (por ejemplo, 142):



El objeto de salida que nos muestra cuando la condición es verdadera en este caso lo que se muestra en la imagen.

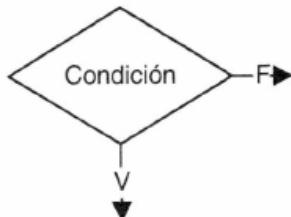


El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.





Estructura Si...Entonces...Si no (Alternativa doble)



Como se puede ver la selección simple es limitada, aunque muchas veces necesaria, por otro lado, la alternativa doble nos permite tomar decisiones en ambos sentidos, es decir cuando la sentencia de comparación sea verdadera o cuando sea falso, en otras palabras, cuando la respuesta de la comparación sea verdadera se ejecutarán una o más acciones, así mismo si la respuesta es falsa se ejecutarán acciones diferentes. Veamos el pseudocódigo el cual explica mejor el concepto de alternativa doble.

Si Condición Entonces

Acciones 1

sino

Acciones 2

Fin _ Si (Fin del condicional)

Donde:

Condición expresa la condición o conjunto de condiciones a evaluarse.

Acciones 1 expresa la operación o conjunto de operaciones que se van a realizar si la condición resulta verdadera.

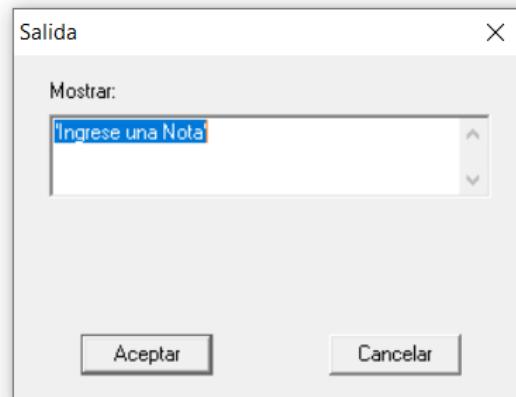
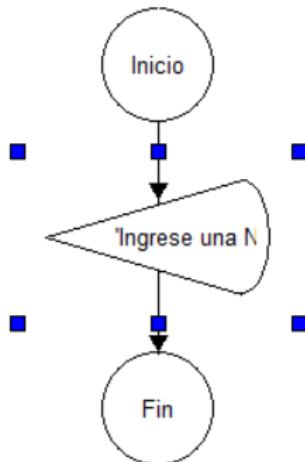
Acciones 2 expresa la operación o conjunto de operaciones que se van a realizar si la condición resulta falsa.

Ejemplo 01

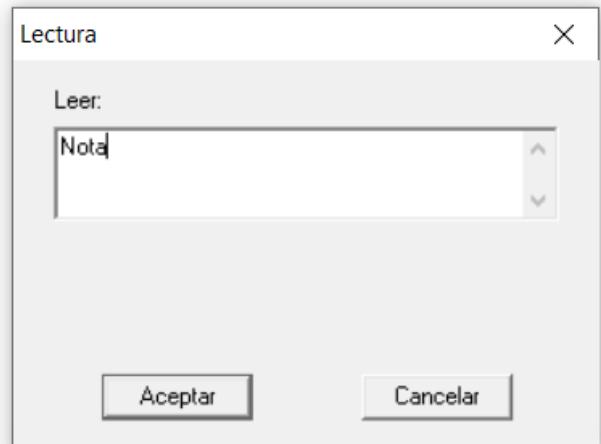
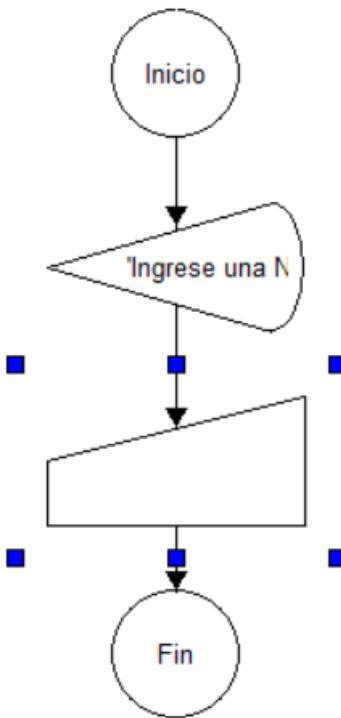
Construya un algoritmo, que dado como dato la calificación de un alumno en un examen, escriba "Aprobado" si su calificación es mayor que 10.5 y "Reprobado" en caso contrario.



colocamos un objeto de salida y escribimos ('Ingresé una Nota').

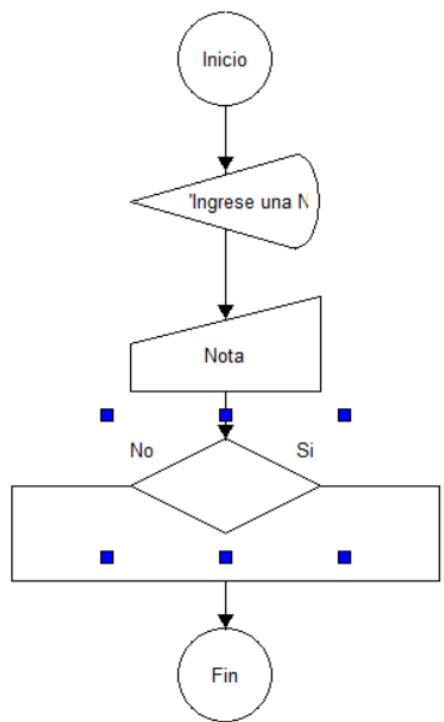


Ahora colocamos un objeto de lectura con la variable (Nota).



Ahora colocamos una decisión y le damos doble clic.

Le escribimos que la variable Nota > 10.5 y le escogemos la opción para que la condición sea verdadera (derecho).



Decisión

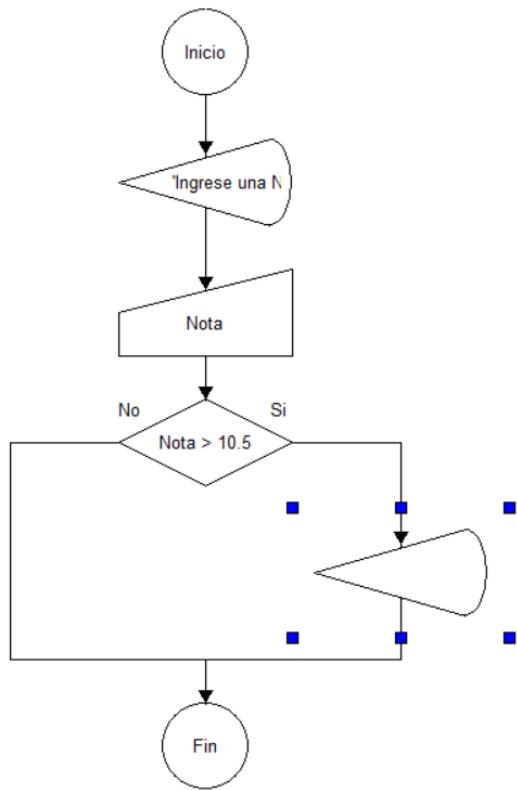
Condición:

Condición verdadera

Derecha
 Izquierda

Aceptar Cancelar

Colocamos un objeto de salida en el lado donde la condición sea verdadera (derecho) y escribimos ('APROBADO').



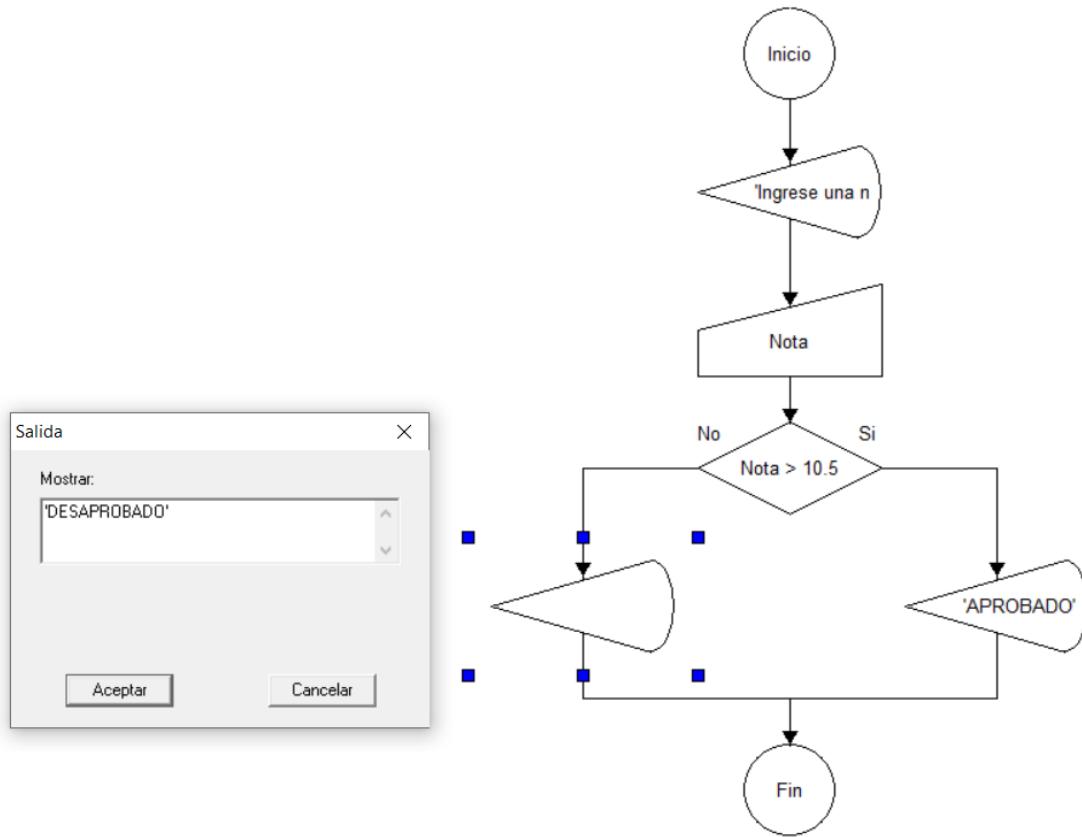
Salida

Mostrar:

Aceptar Cancelar



Colocamos un objeto de salida en el lado donde la condición sea falso (izquierda) y escribimos ('DESAPROBADO').



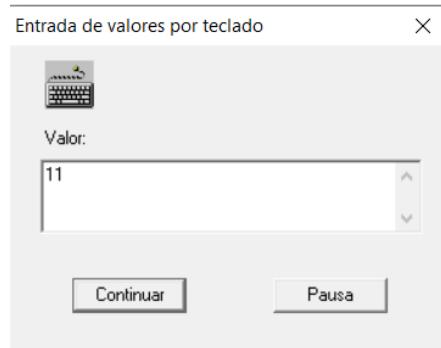
Damos Clic en ejecutar

Y lo hacemos de forma que la condición sea verdadera (Ingresar una Nota).

Primer objeto de salida.



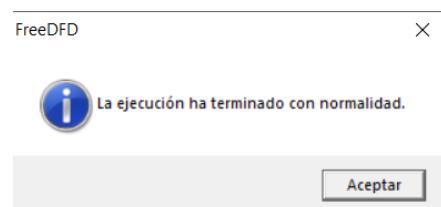
El objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable Nota (por ejemplo, 11):



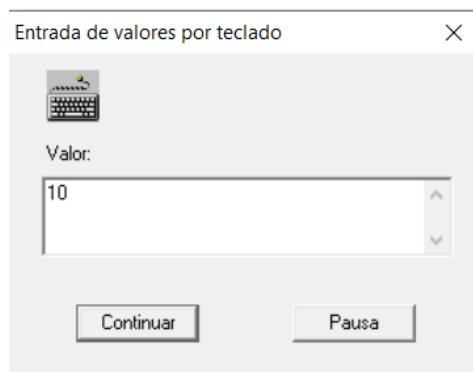
El objeto de salida que nos muestra cuando la condición es verdadera (APROBADO).



El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.



Ahora ingresamos una Nota menor a la condición en este caso pondremos 10 como se muestra en la imagen.

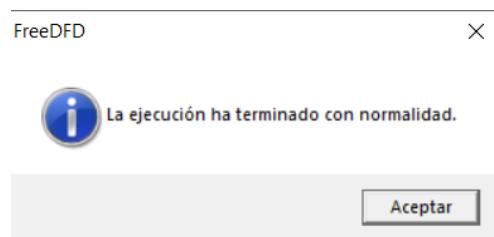




El objeto de salida que nos muestra cuando la condición es falsa (DESAPROBADO).



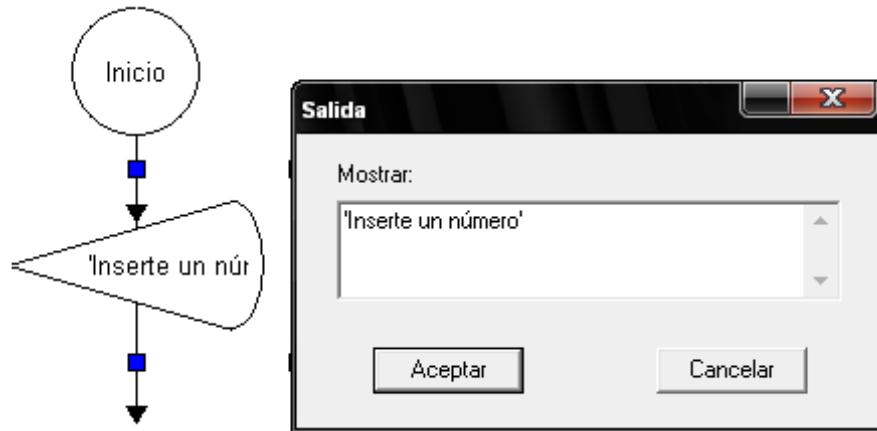
El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.



EJEMPLO 02

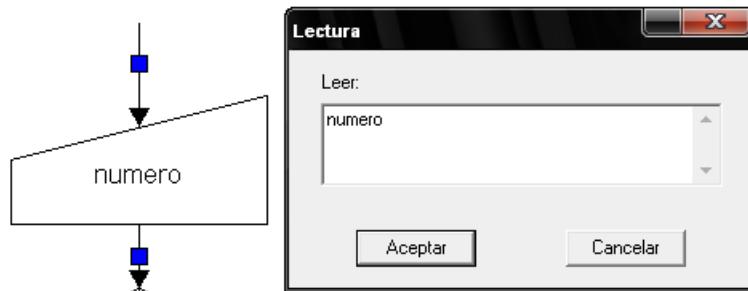
Diseñar y ejecutar un algoritmo que indique si un número **a** pedido por teclado es positivo o negativo.

colocamos un objeto de salida y escribimos ('inserte un número').

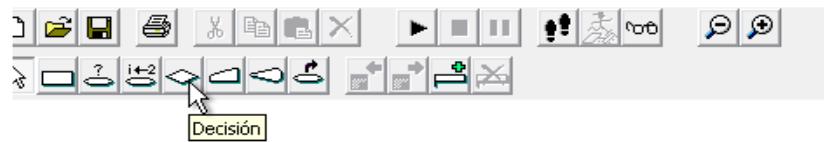




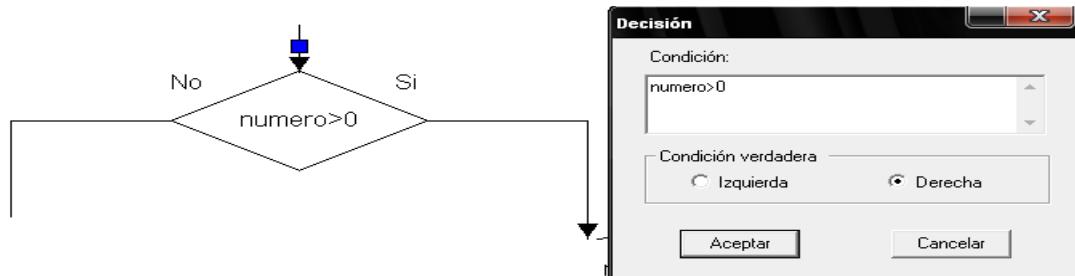
Ahora colocamos un objeto de lectura con la variable (número).



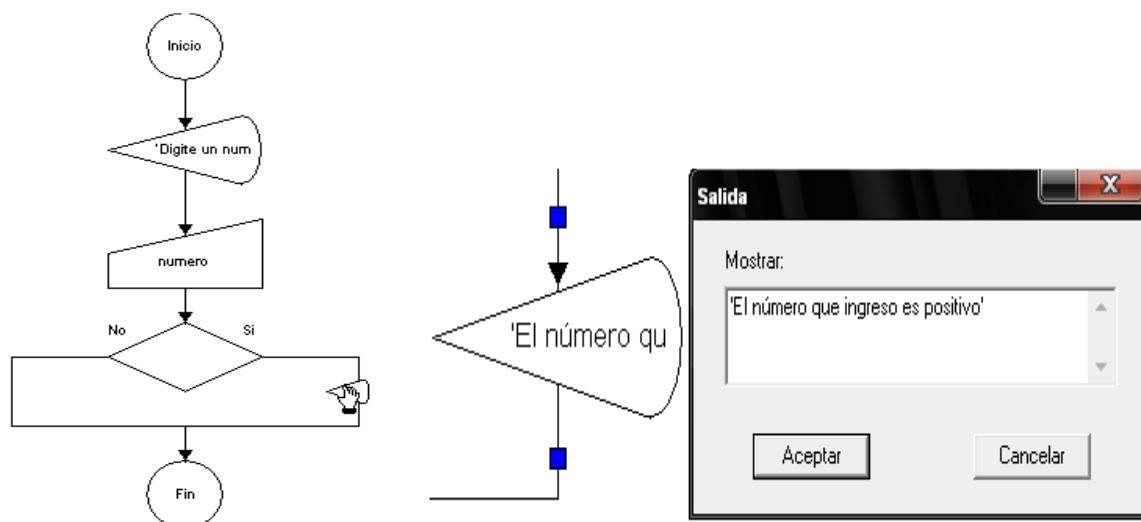
Ahora colocamos una decisión y le damos doble clic.



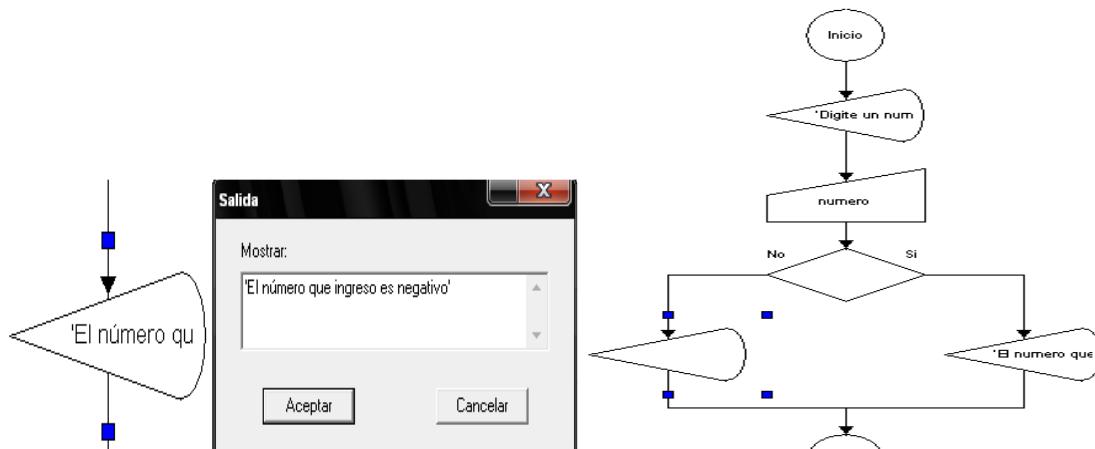
Le escribimos que la variable número > 0 y le escogemos el lado para que la condición sea verdadera (derecho).



Colocamos un objeto de salida en lado donde la condición sea verdadera (derecho) y escribimos ('El número que ingreso es positivo').



También colocamos otro objeto de salida en el lado donde la condición sea falsa (izquierdo) y escribimos ('El número que ingreso es negativo').



Damos Clic en ejecutar

Y lo hacemos de forma que la condición sea verdadera (número positivo).

Primer objeto de salida.

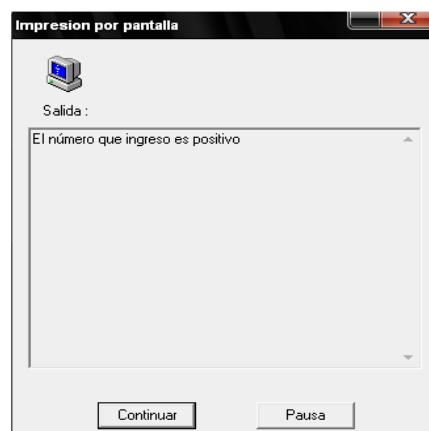




El objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable número (por ejemplo, 7):



El objeto de salida que nos muestra cuando la condición es verdadera (positiva).



El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.



Ahora lo hacemos de tal forma para que la condición sea falsa (número negativo).

En el objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable número (por ejemplo, -3):

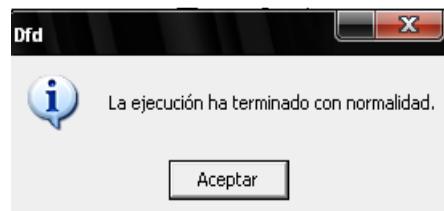




Ahora nos muestra el objeto de salida para cuando la condición es falsa (negativa).



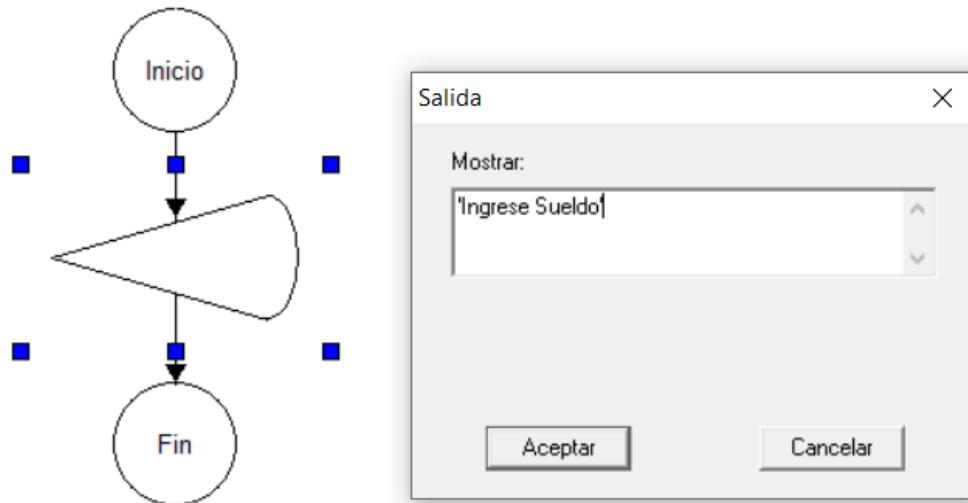
El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.



Ejemplo 03

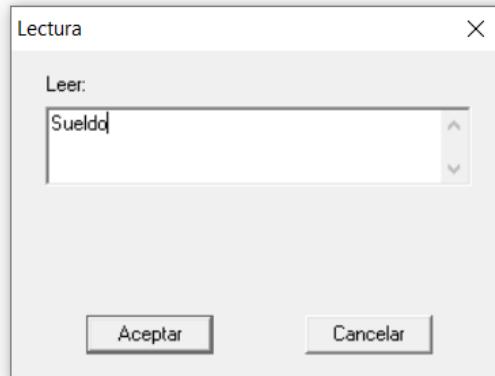
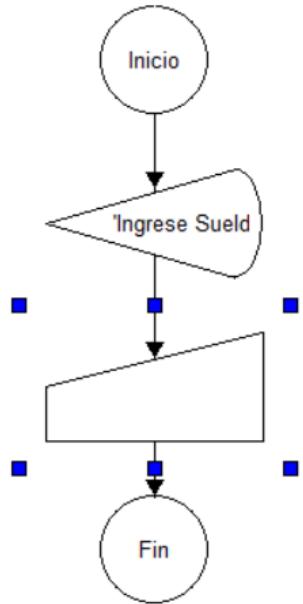
Dado como dato el sueldo de un trabajador, aplíquele un aumento del 17% si su sueldo es inferior a S/ 1000 y 12% caso contrario. Imprima en este caso, el nuevo sueldo del trabajador.

colocamos un objeto de salida y escribimos ('Ingrese Sueldo').



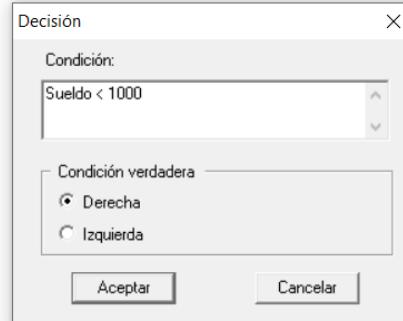
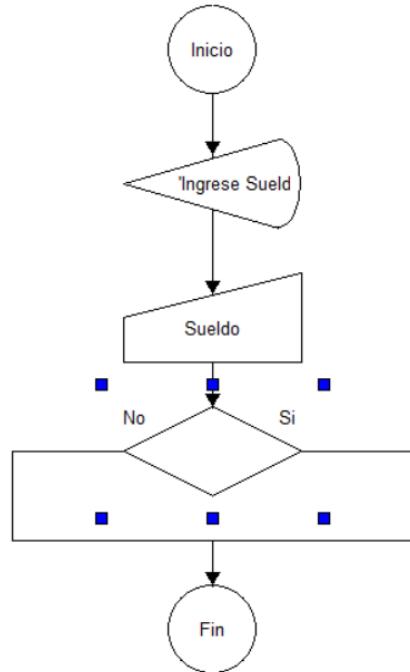


Ahora colocamos un objeto de lectura con la variable (Sueldo).

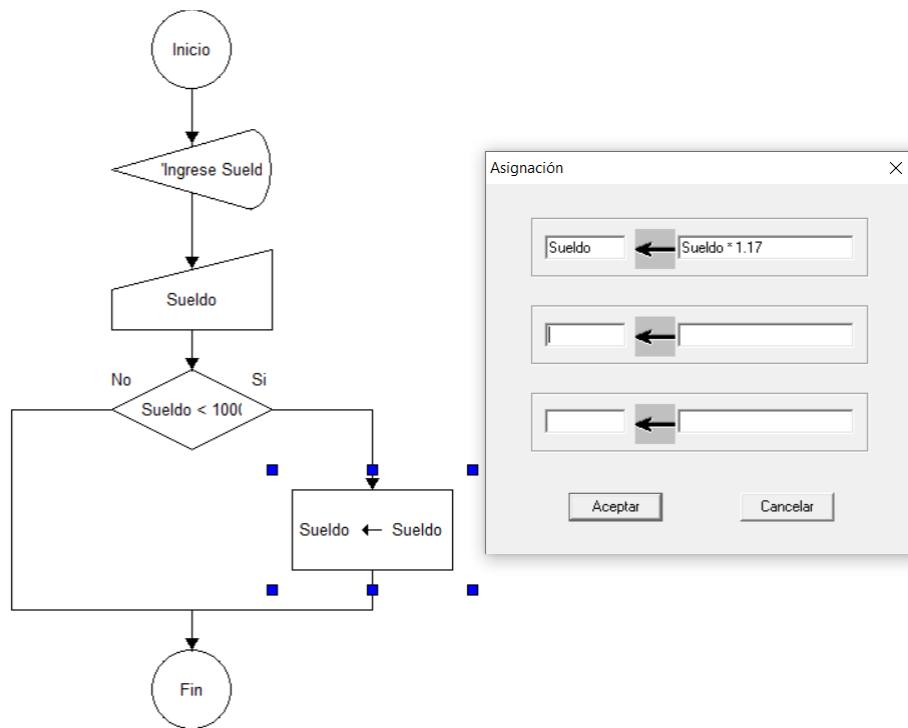


Ahora colocamos una decisión y le damos doble clic.

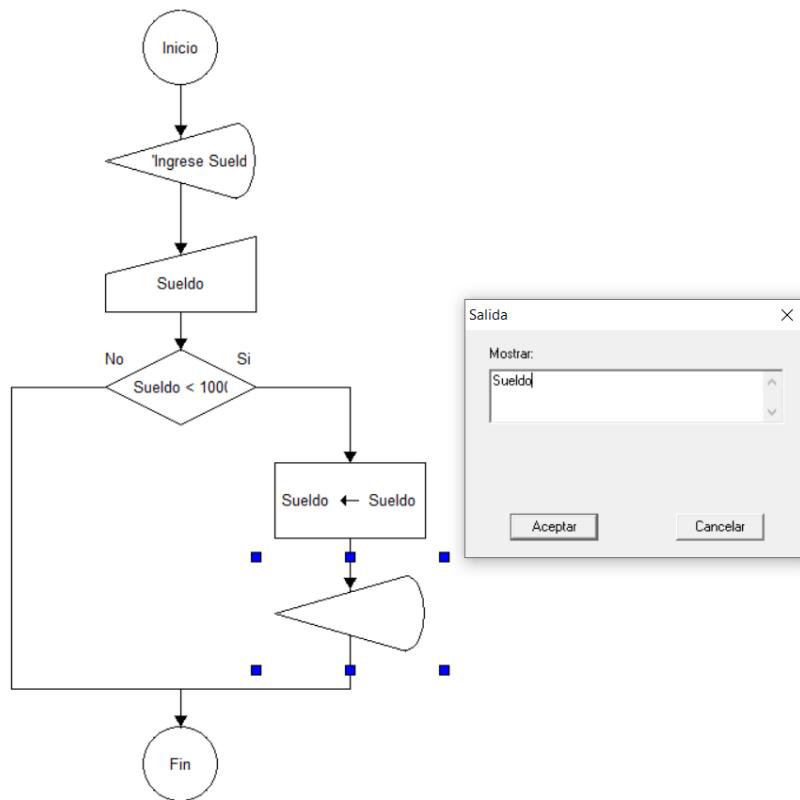
Le escribimos que la variable $Sueldo < 1000$ y le escogemos la opción para que la condición sea verdadera (derecho).



Colocamos un objeto de asignación en donde en el primer cuadro ponemos el nombre de la variable y en el segundo cuadro colocamos la condición que se nos pide en el ejercicio.

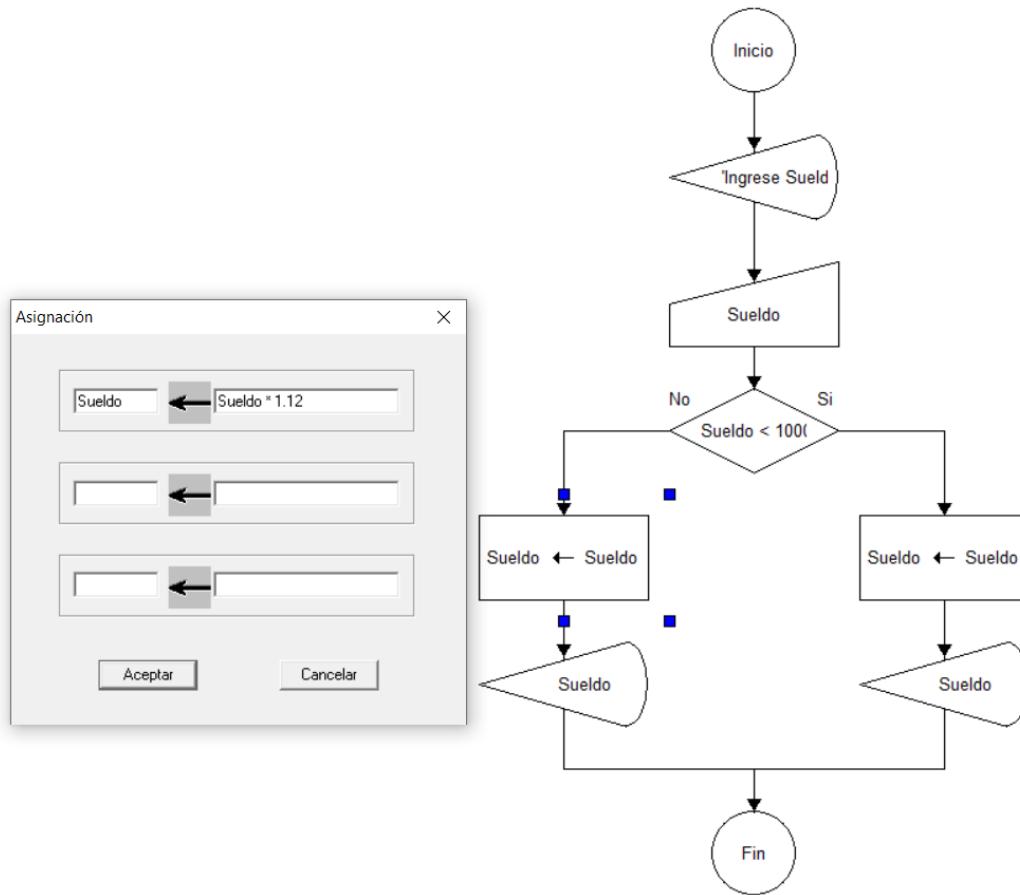


Colocamos un objeto de salida en el lado donde la condición sea verdadera (derecho) y escribimos ('Sueldo')





Colocamos un objeto de asignación en donde en el primer cuadro ponemos el nombre de la variable y en el segundo cuadro colocamos la condición que se nos pide en el ejercicio



Damos Clic en ejecutar

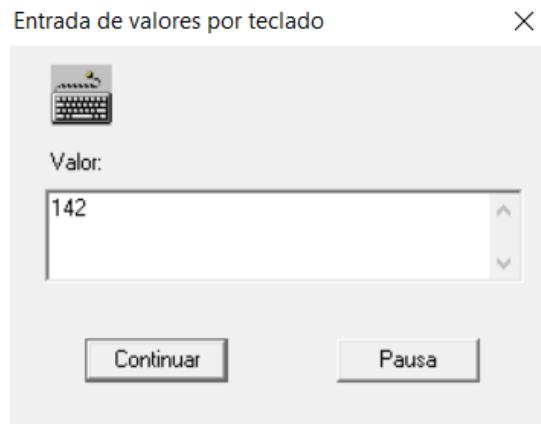
Y lo hacemos de forma que la condición sea verdadera (Ingrese Sueldo).

Primer objeto de salida.





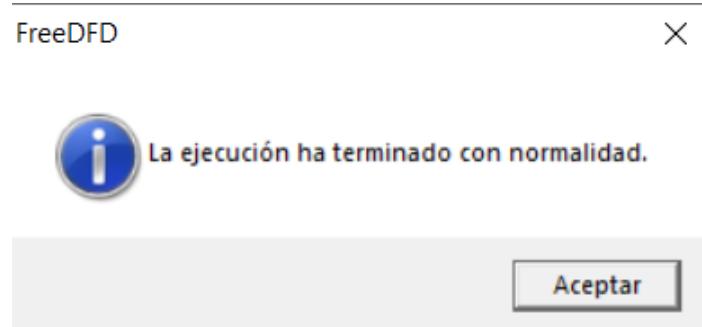
El objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable Sueldo (por ejemplo, 142):



El objeto de salida que nos muestra cuando la condición es verdadera en este caso lo que se muestra en la imagen.



El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.

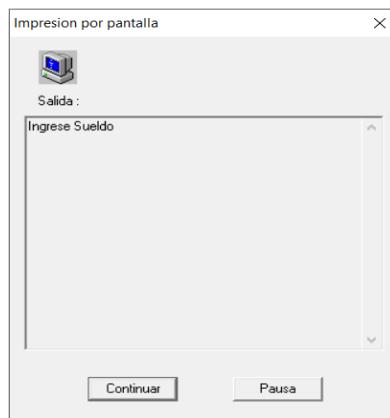




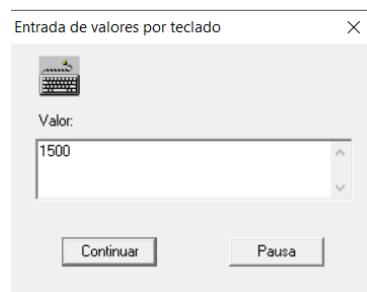
Damos Clic en ejecutar

Y lo hacemos de forma que la condición sea verdadera (Ingrese Sueldo).

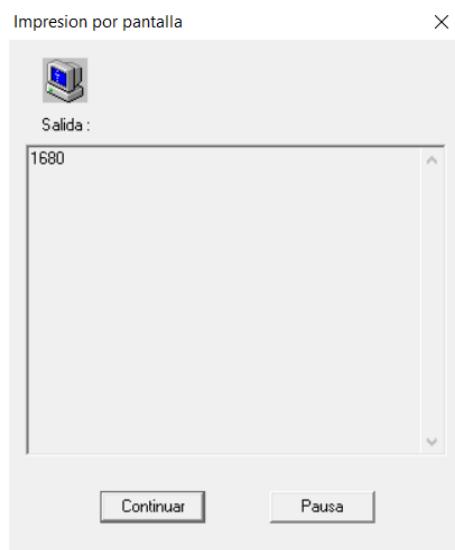
Segundo objeto de salida.



El objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable Sueldo (por ejemplo, 1500):

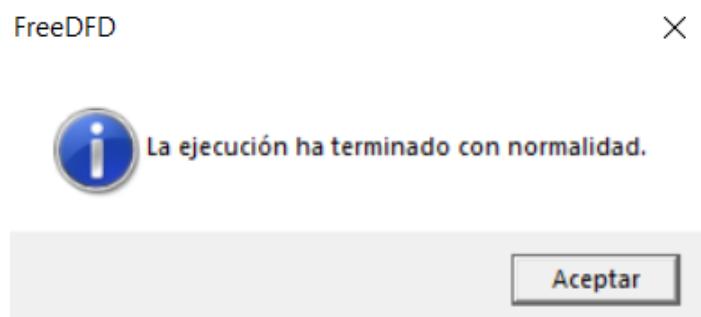


El objeto de salida que nos muestra cuando la condición es falsa, en este caso lo que se muestra en la imagen.





El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.



EJERCICOS PROPUESTOS

1. Leer la edad de un alumno e indicar si es mayor o menor de edad.

2. Leer un número entero positivo e indicar si es par o impar.

3. Leer el sexo de un alumno e indicar si es Masculino o Femenino.

4. Leer 2 números enteros positivos y reportarlos en orden ascendente.

5. Leer un número N, si es positivo calcular su cuadrado sino calcular su cubo.



ACTIVIDAD DE REPASO Y DE INVESTIGACIÓN

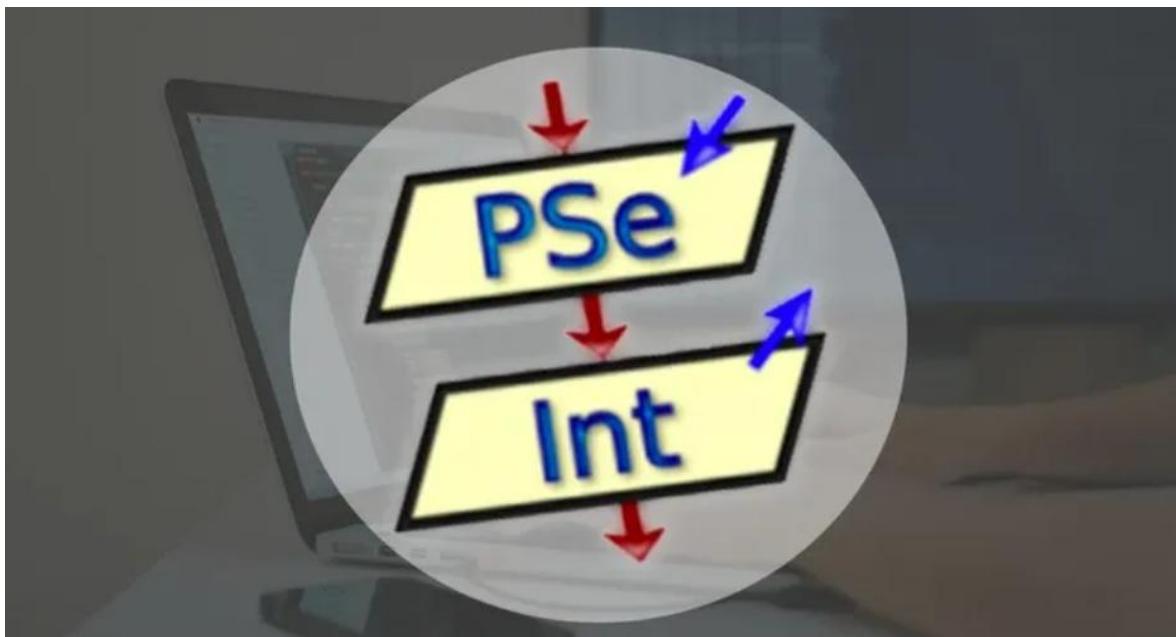
1. Determinar si un alumno aprueba o repreuba la asignatura de Lógica de programación, sabiendo que aprobará si su promedio de 3 notas es mayor o igual a 13; reprobara en caso contrario.
2. En una tienda X se hace un 25% de descuento a los clientes cuya compra supere los S/ 500 ¿Cuál será la cantidad que pagará la persona por su compra?
3. Diseñe un algoritmo para calcular y mostrar la suma de dos números cualquiera, solo si estos son distintos.
4. Ingresar dos números y Determinar cuál de los dos números es el mayor.
5. Crear un programa que se digite el número 10 y escriba número correcto, sino escribir número no existe.
6. Calcule el total a pagar por la compra de camisas. Si se compran cinco camisas o más, se aplica un descuento del 25% sobre el total de la compra, de lo contrario tendrá un descuento del 10%



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA

Lógica de Programación



SESION 05

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I1: Diseña algoritmos utilizando las estructuras básicas de control: secuencial, selectiva y repetitivas



Estructura de Control Selectivas Múltiples:

La estructura de selección múltiple permite que el flujo del programa se divida por varias ramas en el punto de la toma de decisión(es), esto en función del valor que tome el selector. Así, si el selector toma el valor 1 se ejecutará la acción 1, si toma el valor 2 se ejecutará la acción 2, si toma el valor N se realizará la acción N, y si toma un valor distinto de los valores comprendidos entre 1 y N, se continuará con el flujo normal del diagrama realizándose la acción N + 1. A continuación, presentamos el pseudocódigo que ilustra esta estructura selectiva.

En caso de Selector

Caso Valor 1: Hacer Acción 1

Caso Valor 2: Hacer Acción 2

.

Caso Valor N: Hacer Acción N

En otro caso: Acción N + 1

Fin_Caso {Fin del condicional}

Donde:

Selector es la variable o expresión, a evaluarse, según la cual se tomará una de las "múltiples" decisiones o alternativas.

Acción1 expresa la operación o conjunto de operaciones que se van a realizar si el selector toma el valor1.

Acción2 expresa la operación o conjunto de operaciones que se van a realizar si el selector toma el valor 2.

Acción N expresa la operación o conjunto de operaciones que se van a realizar si el selector toma el valor N.

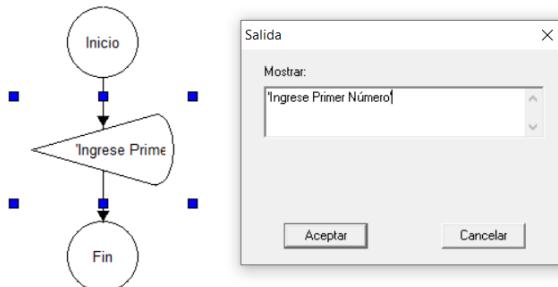
La estructura selectiva *En caso de* es muy flexible, lo que permite aplicarla de diferentes formas.



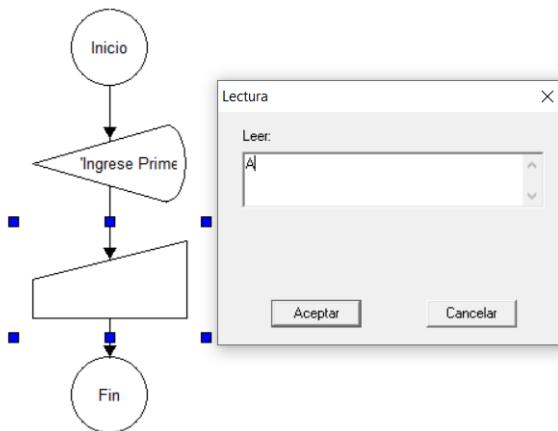
Ejemplo 1

Escribir un programa que lea tres números enteros por teclado y muestre por pantalla el mayor de los tres.

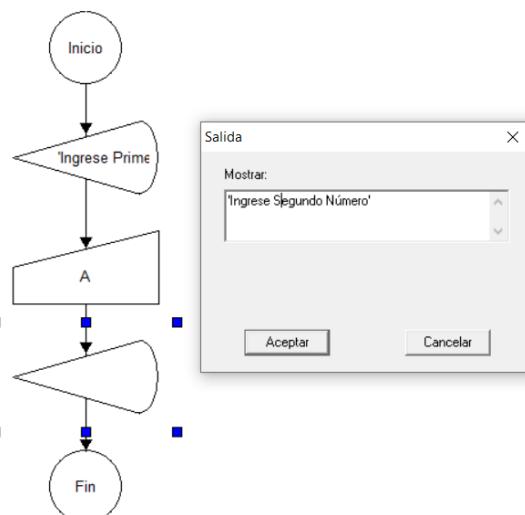
colocamos un objeto de salida y escribimos ('Ingrese Primer Número').



Ahora colocamos un objeto de lectura con la variable (A).

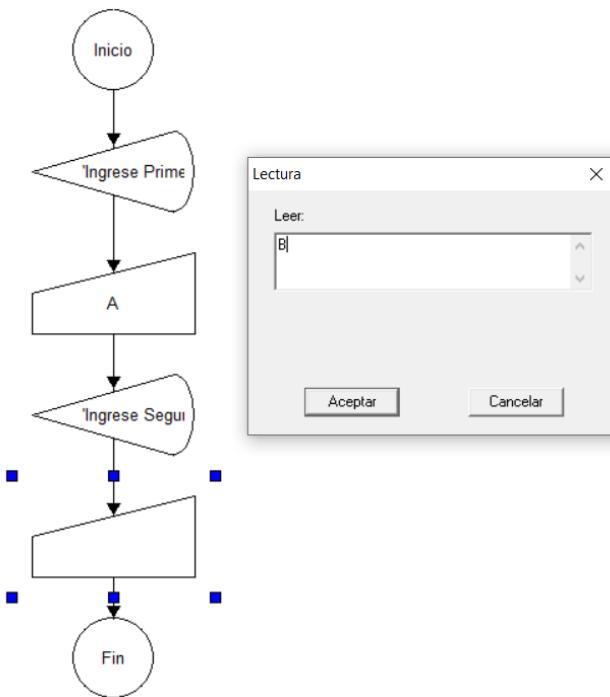


colocamos un objeto de salida y escribimos ('Ingresar segundo Número').

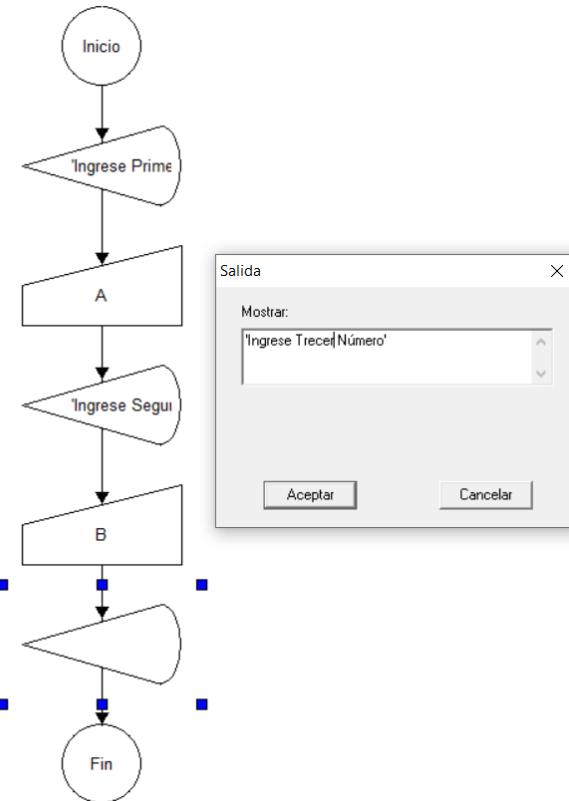




Ahora colocamos un objeto de lectura con la variable (B).

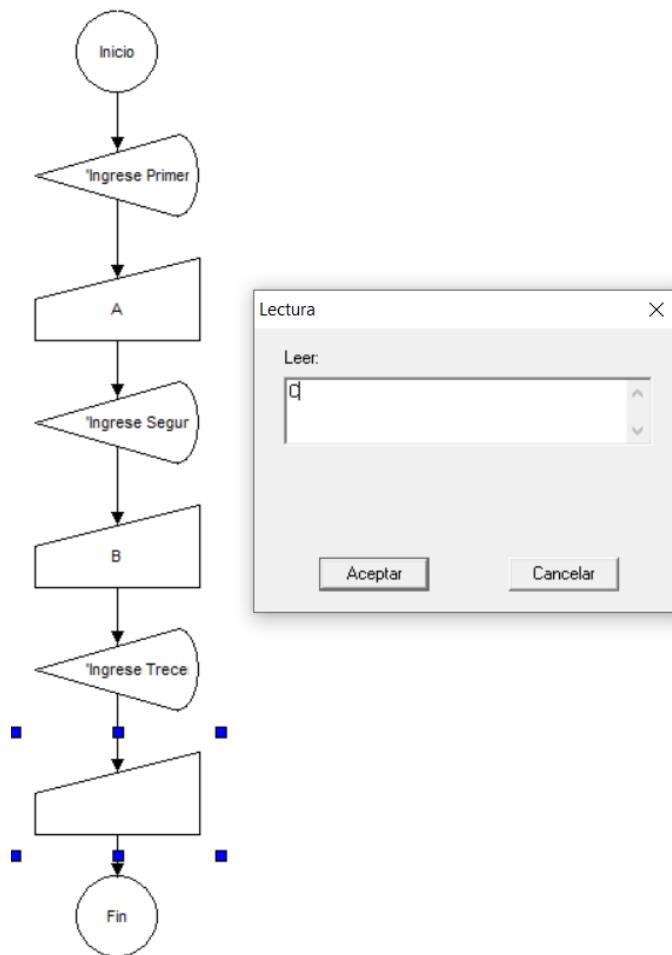


colocamos un objeto de salida y escribimos ('Ingrrese Tercer Número').

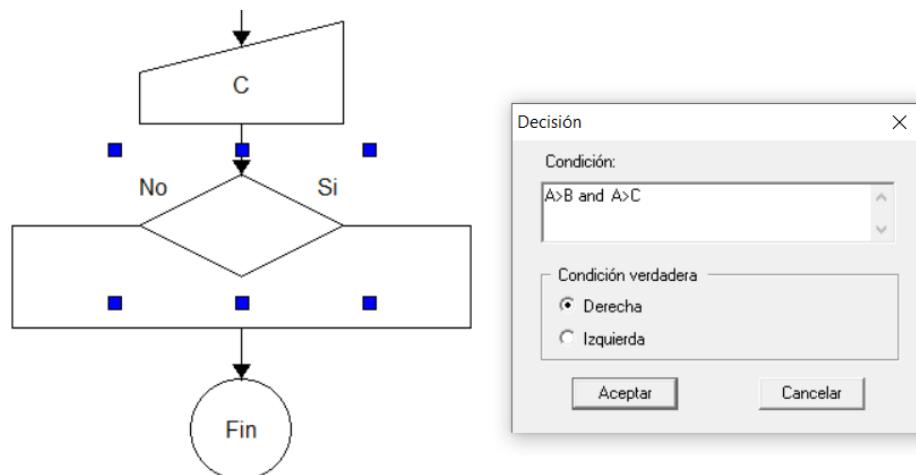




Ahora colocamos un objeto de lectura con la variable (C).

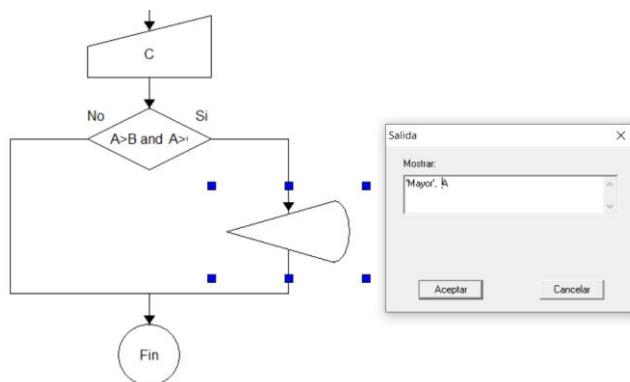


Colocamos un objeto de decisión y escribimos que las variables $A>B$ y $A>C$ y escogemos la opción para que la condición sea verdadera (derecho).

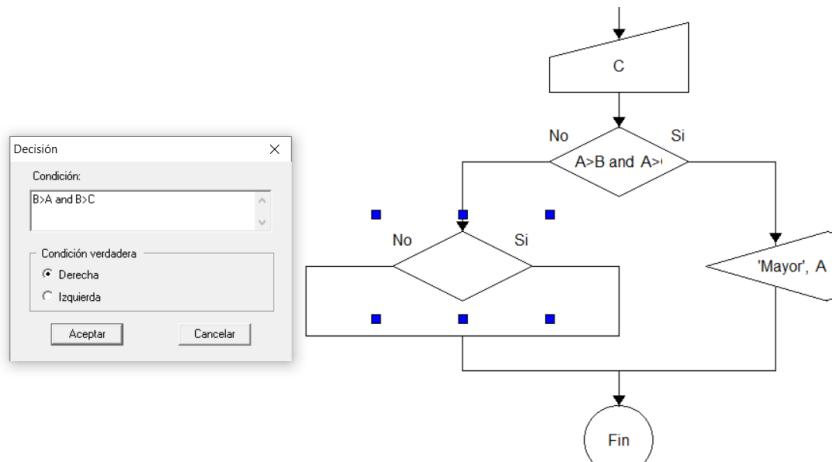




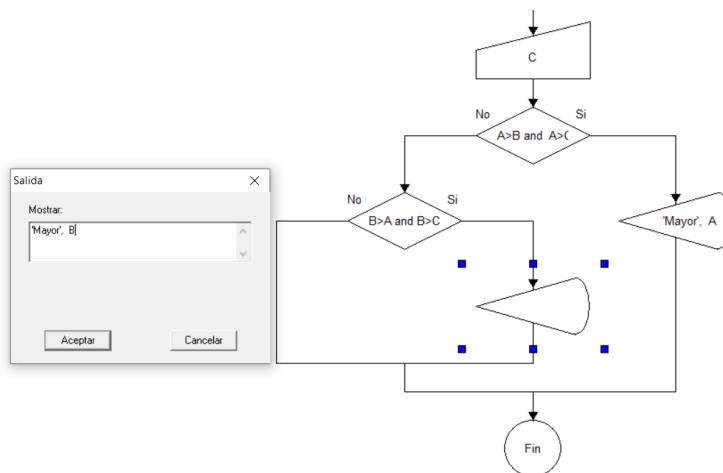
Colocamos un objeto de salida en donde de acuerdo a la condición nos dice que el número Mayor es A



Colocamos un objeto de decisión y escribimos que las variables $B>A$ y $B>C$ y escogemos la opción para que la condición sea verdadera (derecho).

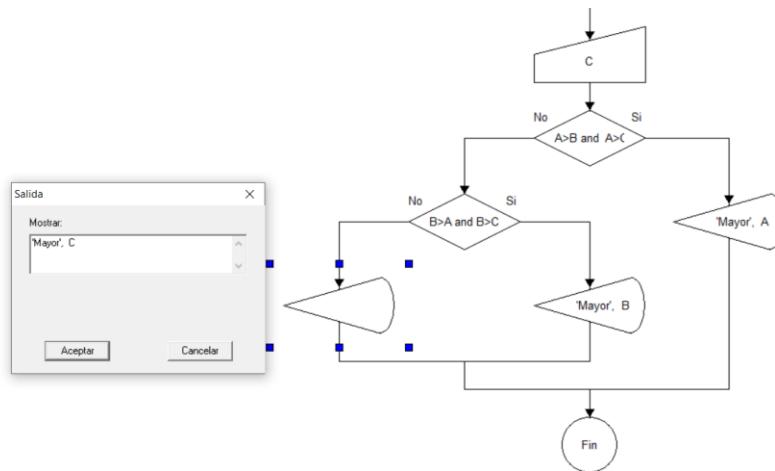


Colocamos un objeto de salida en donde de acuerdo a la condición nos dice que el número Mayor es B en la condición verdadera (Derecha)





Colocamos un objeto de salida en donde de acuerdo a la condición nos dice que el número Mayor es C en la condición Falsa (Izquierda)



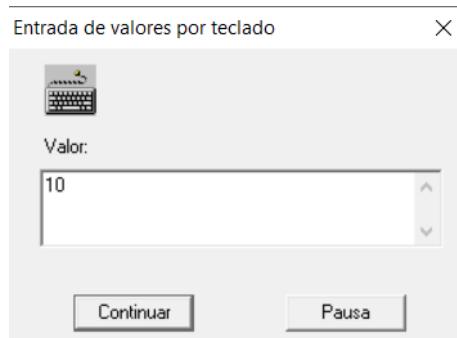
Damos Clic en ejecutar

Y lo hacemos de forma que la condición sea verdadera (Ingrese Primer Numero).

Primer objeto de salida.



El objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable A (por ejemplo, 10):





Segundo objeto de salida.

Impresion por pantalla

Continuar Pausa

El objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable B(por ejemplo, 8):

Entrada de valores por teclado

Continuar Pausa

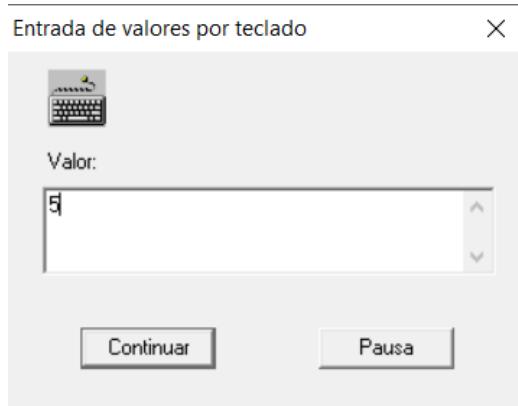
Tercer objeto de salida.

Impresion por pantalla

Continuar Pausa



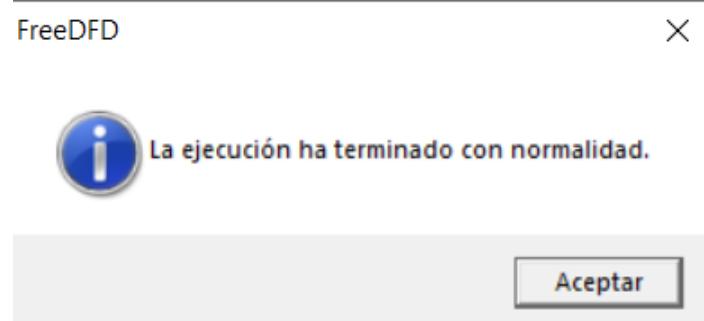
El objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable C(por ejemplo, 5):



Ahora nos mostrara el resultado de cual es el mayor de los números ingresados como se muestra en la imagen:



El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.

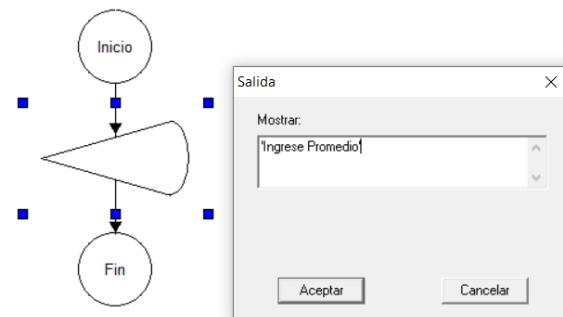




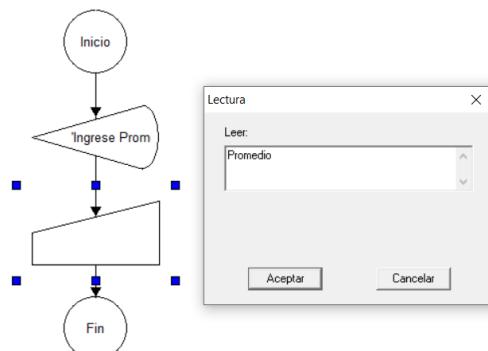
Ejemplo 2

Escribir un programa que lea el promedio de un alumno del I ciclo de la UD Lógica de Programación, si el promedio es mayor o igual a 13(APROBADO), si el promedio es mayor o igual a 10 y menor que 13(RECUPERACION) y si el promedio es menor a 10 (REPROBADO)

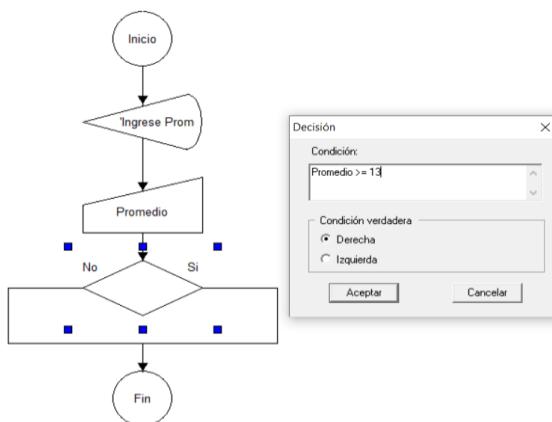
colocamos un objeto de salida y escribimos ('Ingrese Promedio').



Ahora colocamos un objeto de lectura con la variable (Promedio).

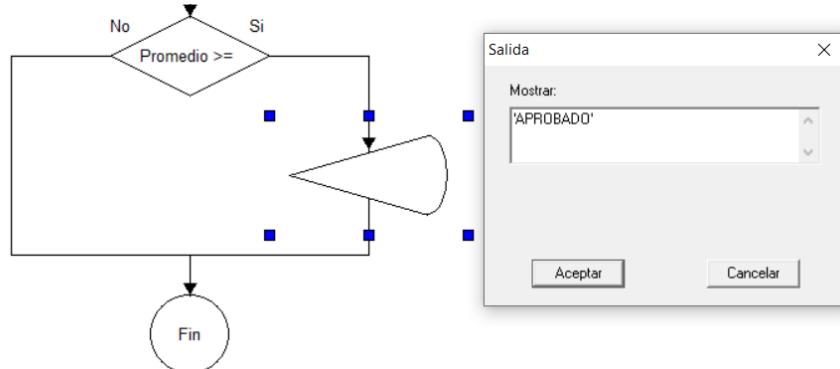


Colocamos un objeto de decisión y escribimos que las variables $Promedio \geq 13$ y escogemos la opción para que la condición sea verdadera (derecho).

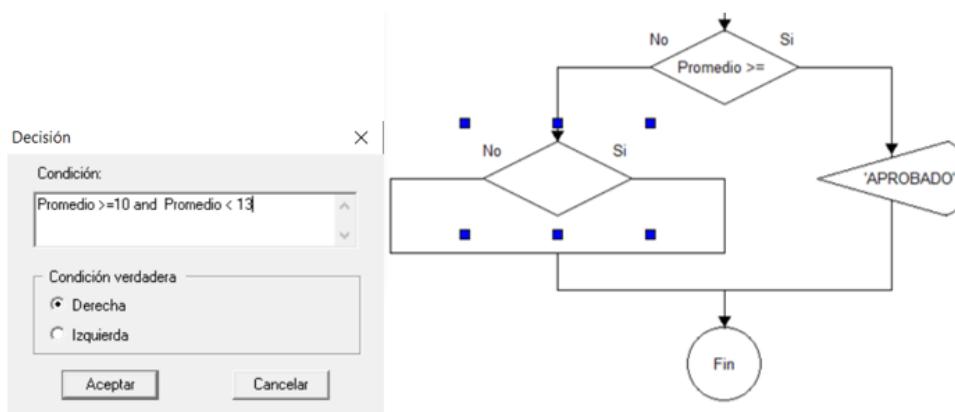




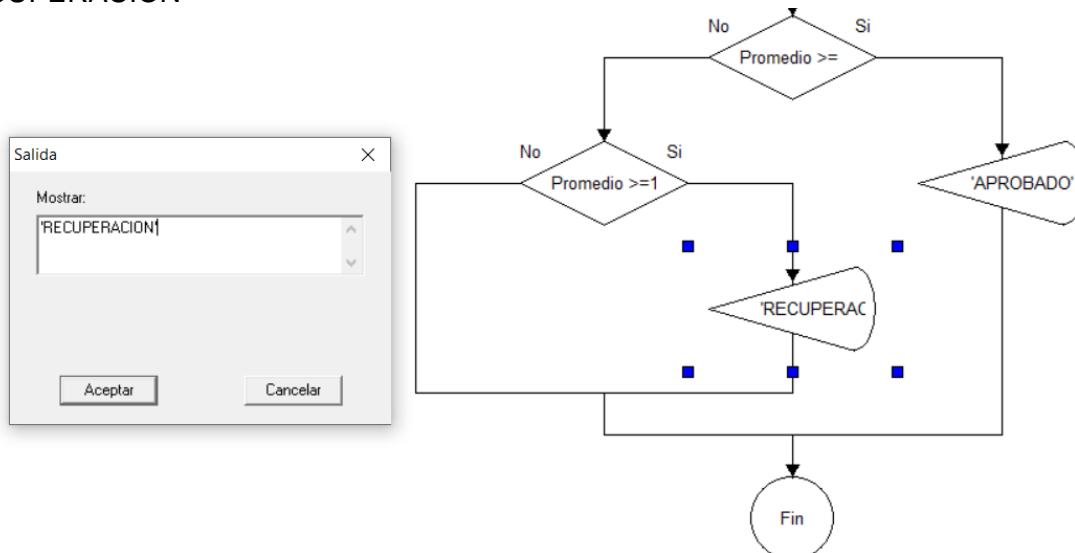
Colocamos un objeto de salida en donde de acuerdo a la condición nos dice APROBADO



Colocamos un objeto de decisión y escribimos que la variable Promedio ≥ 10 y Promedio < 13 , y escogemos la opción para que la condición sea verdadera (derecho).

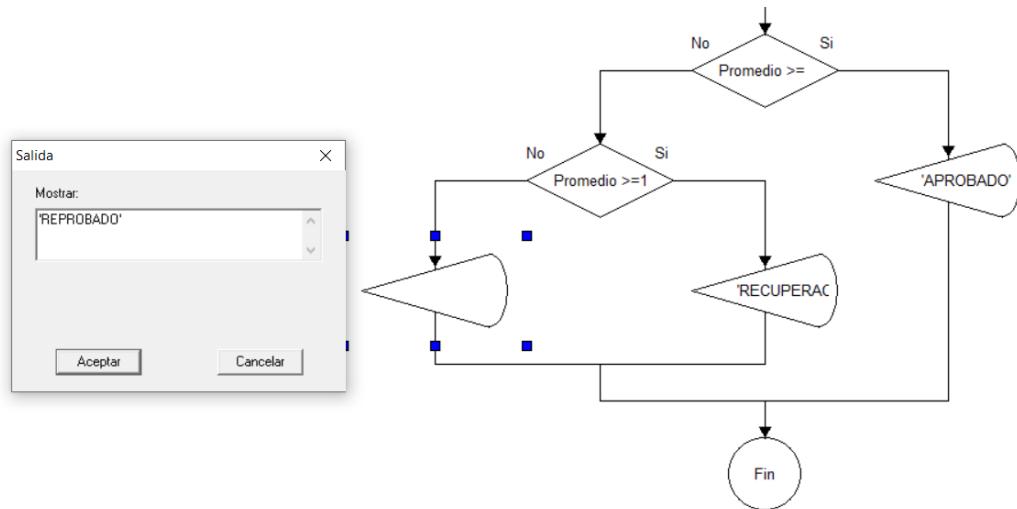


Colocamos un objeto de salida en donde de acuerdo a la condición nos dice RECUPERACION





Colocamos un objeto de salida en donde de acuerdo a la condición nos dice REPROBADA en la condición Falsa (Izquierda)



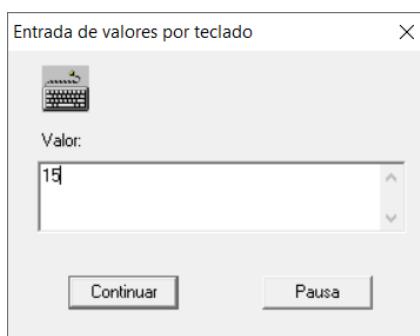
Damos Clic en ejecutar

Y lo hacemos de forma que la condición sea verdadera (Ingrese Promedio).

Primer objeto de salida.



El objeto de entrada, nos muestra un cuadro de texto en el que introducimos el valor que queramos darle a la variable Promedio (por ejemplo, 15):

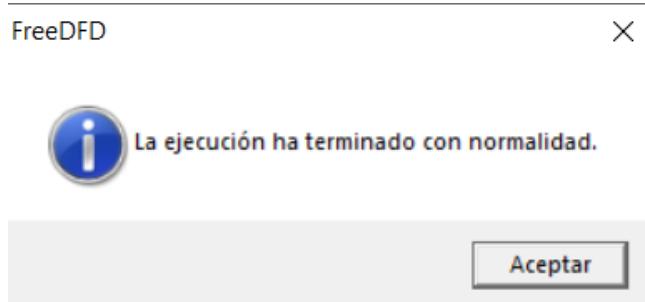




Ahora nos mostrara el resultado de acuerdo a la condición que se a establecido como se muestra en la imagen:



El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.



EJERCICOS PROPUESTOS

1. Escribir un programa que lea dos números enteros por teclado y determine cuál es el mayor y cuál es el menor. También deberá considerar el caso en el que los dos números sean iguales.

2. Escribir un programa que lea tres números enteros por teclado y emita un mensaje indicando si están o no ordenados crecientemente.

3. Construya un pseudocódigo tal, que dados 3 calificaciones de un alumno; imprima el promedio y la palabra "Aprobado" si el alumno tiene un promedio mayor o igual que 12.5, y la palabra "No aprobado" en caso contrario



ACTIVIDAD DE REPASO Y DE INVESTIGACIÓN

1. Escribir un programa que lea dos números n1 y n2. Si $n1 \leq n2$ que calcule y reporte la suma y resta en caso contrario debe calcular la multiplicación y división.
2. Leer el lado de un cuadrado. Si lado > 0 debe calcular su área y su perímetro. En caso contrario debe imprimir el mensaje “Valor Incorrecto”.
3. Leer el radio R de un círculo. Si R es > 0 calcular el área del círculo En caso contrario debe imprimir el mensaje “No se Puede Calcular”.
4. Ingresar un numero X. Si X es ≥ 100 calcular X^2 en caso contrario calcular X^3
5. Leer la base y altura de un triángulo. Si ambos valores son positivos calcular su área. Caso contrario no hacer nada.



https://www.youtube.com/watch?v=iZ2ZpdQGcLs&ab_channel=LuzArelyMonroyGonzález

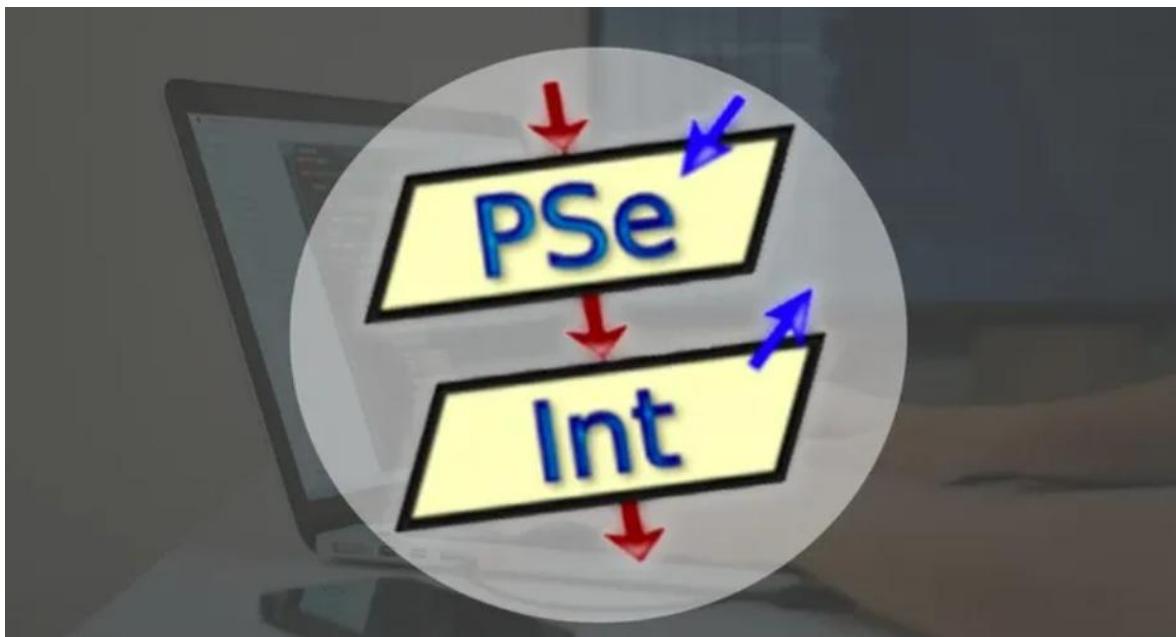
https://www.youtube.com/watch?v=CU5SEtEp8c8&ab_channel=ElProfeDami%C3%A1n



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA

Lógica de Programación



SESION 06

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I1: Diseña algoritmos utilizando las estructuras básicas de control: secuencial, selectiva y repetitivas



La Estructura de Control Repetitivas

Introducción

Es muy común encontrar en los algoritmos operaciones que se deben ejecutar un número repetido de veces. Si bien las instrucciones son las mismas, los datos sobre los que se opera varían. El conjunto de instrucciones que se ejecuta repetidamente se llama ciclo.

Todo ciclo debe terminar de ejecutar luego de un número finito de veces, por lo que es necesario en cada iteración del mismo, evaluar las condiciones necesarias para decidir si debe seguir ejecutándose o debe detenerse. En todo ciclo, siempre debe existir una condición de parada o fin de ciclo.

Estructura Hacer Mientras (do while)

La estructura *Hacer Mientras*, es la estructura algorítmica adecuada para utilizar en un ciclo que se ejecutará un número definido de veces. Por ejemplo, cuando necesitamos calcular la nómina total de la empresa, tenemos que sumar los sueldos de los N empleados de la misma. Cuando necesitamos obtener el promedio de calificaciones de un curso, debemos sumar las N calificaciones de los alumnos y dividir esa suma entre N. Es decir, sabemos de antemano cuántas veces tenemos que repetir una determinada operación, acción o tarea. El número de repeticiones no depende de las proposiciones dentro del ciclo.

La sintaxis para esta estructura es la siguiente:

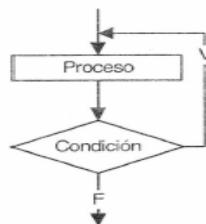
Hacer

{Proceso}

Mientras (Condición)

{Fin del ciclo}

Proceso, es cualquier operación o conjunto de operaciones que deban ejecutarse mientras se repita el ciclo. El ciclo se repetirá siempre que la Condición sea verdadera, es decir se cumplirá hasta que la Condición sea falsa.





Esta estructura garantiza por lo menos una iteración del bloque de proceso, debido a que primero realiza el proceso y al final evalúa la condición. Muchas veces usaremos esta estructura con el uso de un contador, y la condición estará ligada a dicho contador, obteniendo así la siguiente sintaxis:

V = Vi

Hacer

{proceso}

Calcular V = V + INC

Mientras que (Condición)

(Fin del ciclo)

Donde

V: es una variable de control

Vi: es el valor inicial,

INC: es el incremento

V (contador del ciclo, generalmente representado por las letras I, J, K, V) toma un valor inicial (Vi) y se compara con el valor esperado en la condición. El ciclo se ejecuta hasta que la condición sea verdadera. El valor de V se incrementa en cada iteración.

El incremento de INC por lo general es en un unidad, pero eso depende mucho de la naturaleza del problemas, veremos casos en que el incremento será de dos unidades o de tres unidades, etc , en general podremos tener un incremento en X unidades, donde X es un número entero.

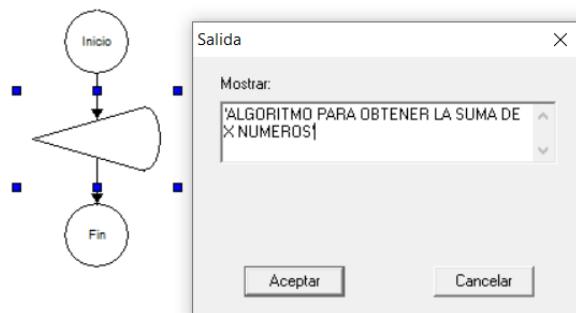
Las variables de control son enteras o son reales o cualquier otro tipo, pero no pueden ser una mezcla de ambas. Podemos tener tanto incrementos o decrementos, eso dependiendo de la naturaleza del programa.

Veamos a continuación algunos ejemplos de esta estructura.

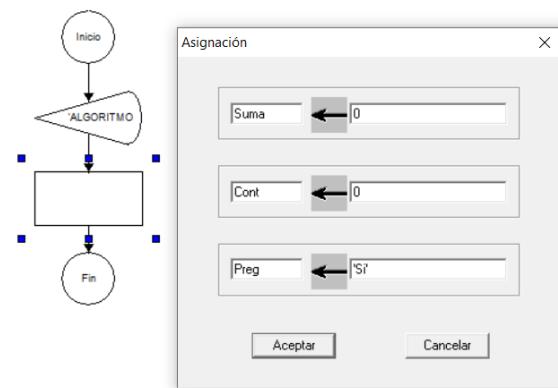


Ejemplo 01:

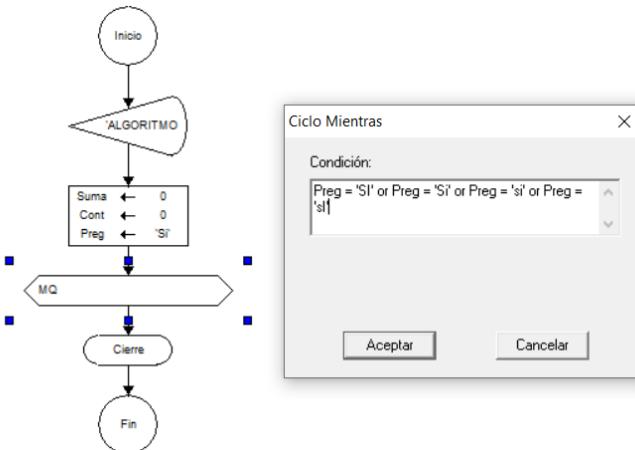
Realizar un algoritmo que permita obtener la suma de X numeros enteros. Mostrar la suma total de los numeros ingresados.



Agregamos nuestras variables y las inicializamos en 0

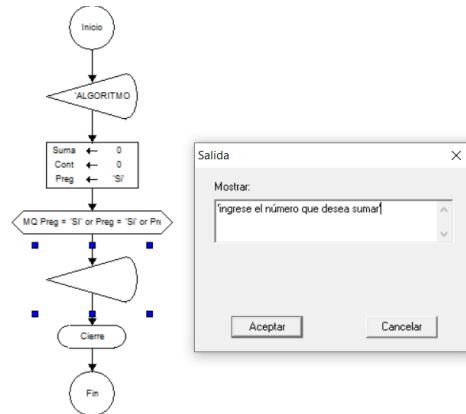


Ahora agregamos el elemento **Mientras** para poder agregar la condición, la condición será preguntarle al usuario si quiere agregar un número más para realizar una suma

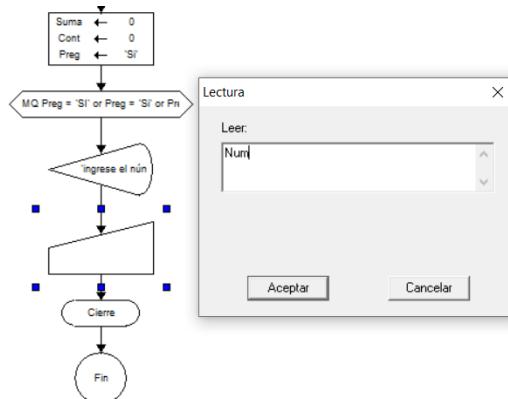




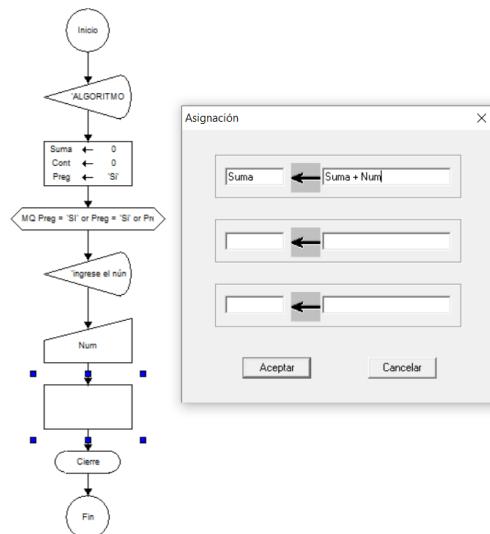
Agregamos un mensaje diciendo al usuario que ingrese el número que desea sumar



Ese número lo ingresaremos en la variable Num que declaramos al inicio

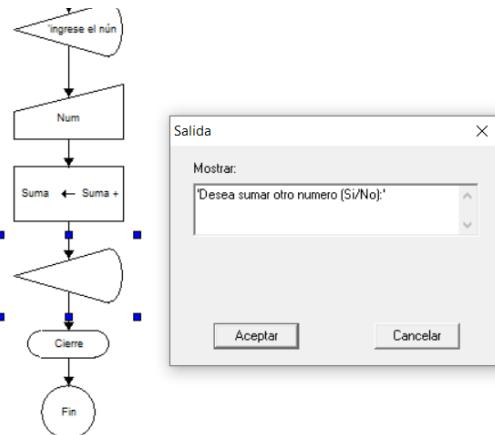


Ahora solo nos toca poder realizar los cálculos para lo cual agregamos un elemento asignación como se muestra en la imagen.

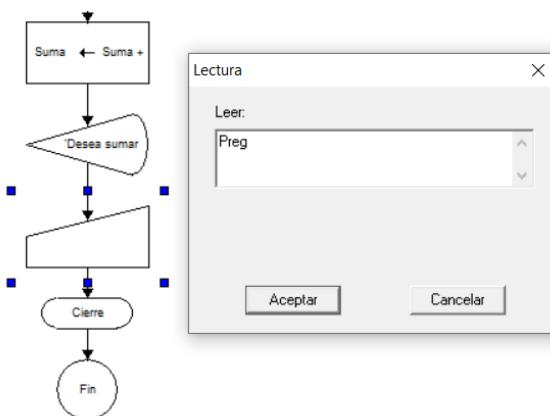




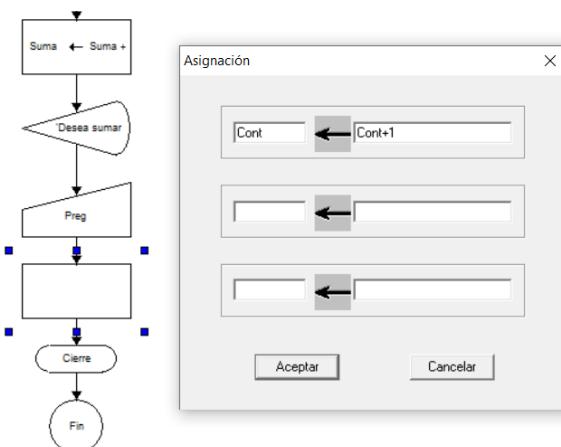
Y preguntamos al usuario si desea sumar otro numero o ya no desea sumar



Esa respuesta que me da el usuario tengo que guardarla nuevamente en una variable que lo habíamos definido al inicio del ejercicio la cual era Preg.

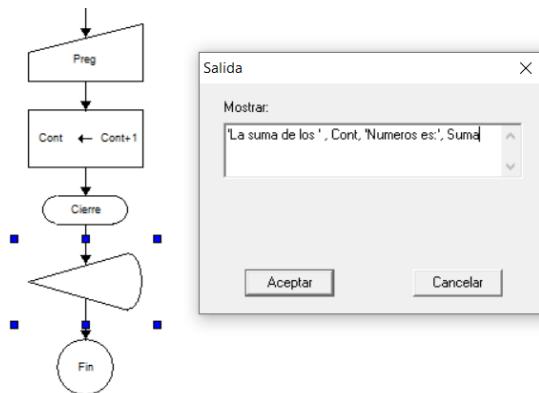


Ahora debemos utilizar la variable contador que la declaramos como Cont al inicio también





Finalmente agregamos un elemento salida para que nos muestre el mensaje de la operación realizada.



Ahora si probamos si el programa funciona, ejecutamos y nos muestra el primer mensaje:

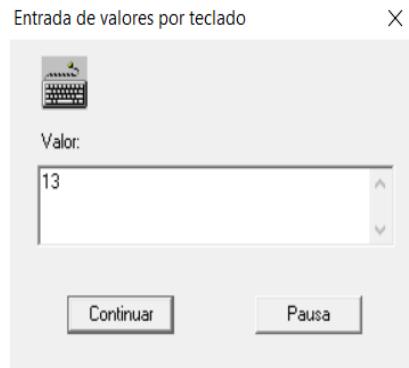


Nos va a pedir ingresar un número cualquiera





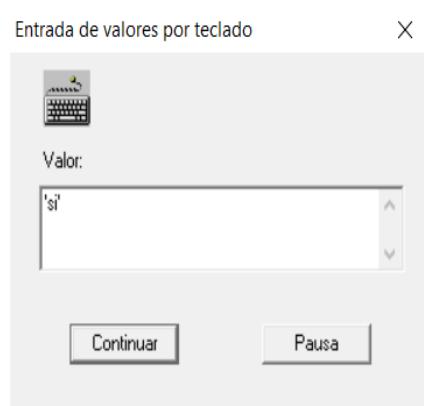
Para este ejemplo ingresaremos el número 13



Nos aparece un mensaje que nos indica si queremos sumar otro numero



Ingresamos en este caso que 'Sí', recordar que debemos ingresar la palabra si entre comillas simples como se muestra en la imagen y presionamos en continuar

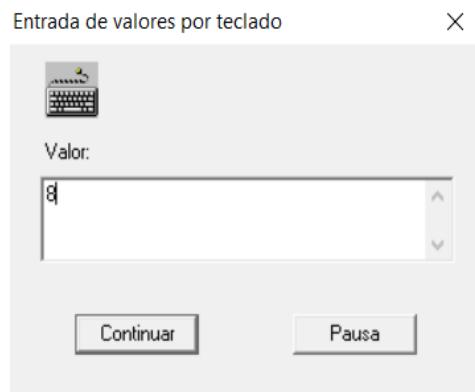




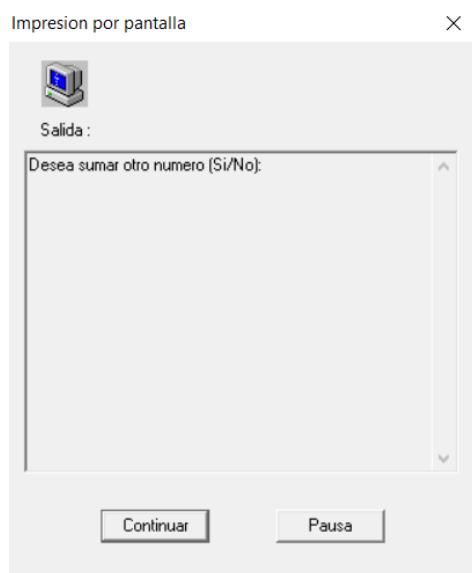
Nos aparece el siguiente mensaje para ingresar el número que deseamos sumar



Ingresamos otro número en este caso ingresaremos el número 8

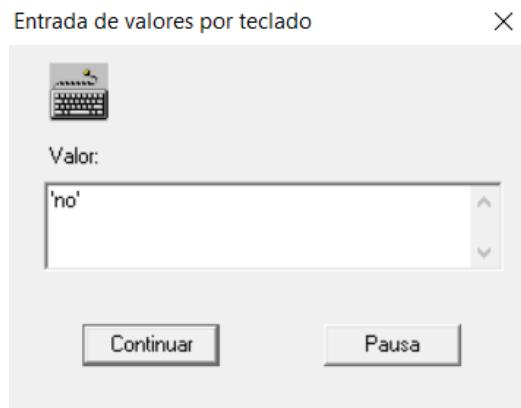


Ahora nos pregunta si deseamos ingresar otro número le damos click en continuar





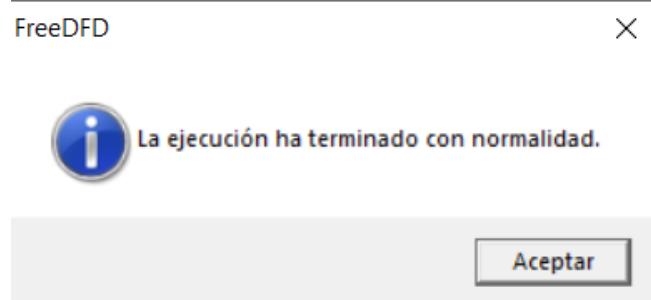
En el cuadro vamos a poner que 'No' como se muestra en la imagen:



Ahora nos aparece el resultado como se muestra en la imagen:



El cuadro de diálogo que nos muestra que el algoritmo no tiene errores.

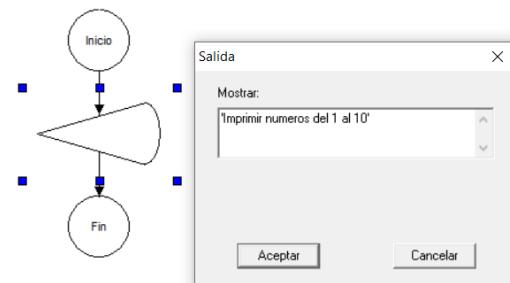




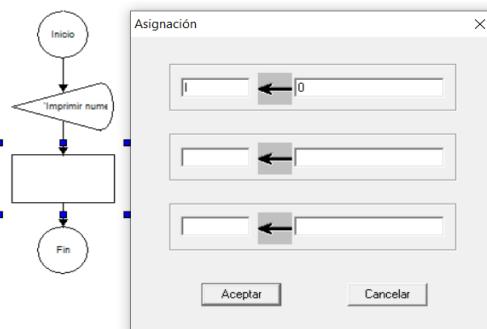
Ejemplo 02:

Realizar un algoritmo que permita Imprimir números del 1 al 10

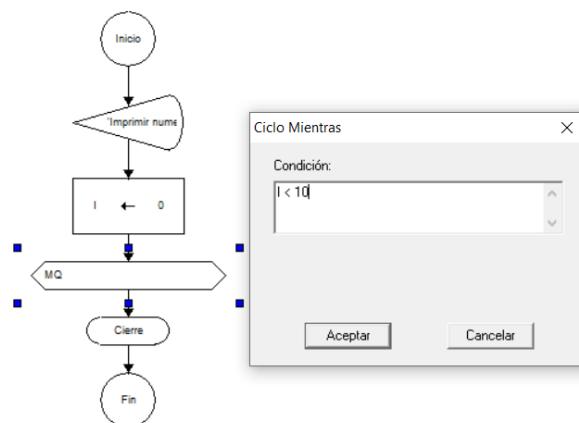
Empezamos agregando un elemento Salida para poder mostrar un mensaje que indique lo que vamos a realizar



Ahora vamos a asignar una variable para inicializar los números que necesitamos mostrar en este caso se hará agregamos un elemento asignación como se muestra en la imagen:

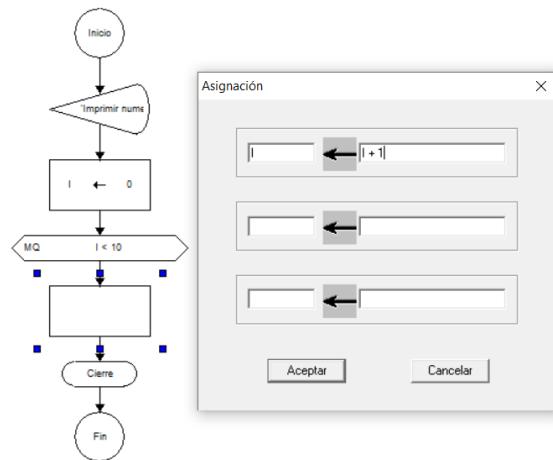


Ahora ingresamos el elemento para crear el ciclo “mientras” e ingresamos la condición que vamos a emplear

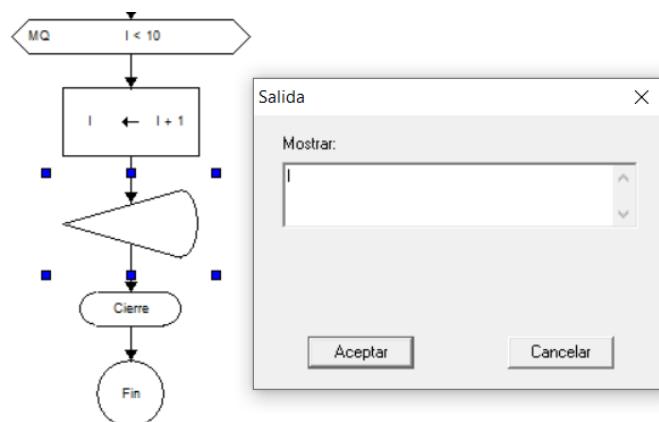




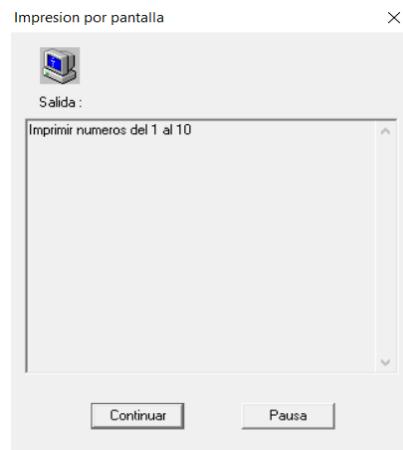
Ahora vamos a ingresar un contador para que pueda llegar hasta el numero solicitado en la condición para lo cual agregamos un elemento asignación



Mandamos a imprimir I como mostramos en la imagen insertando un elemento salida:

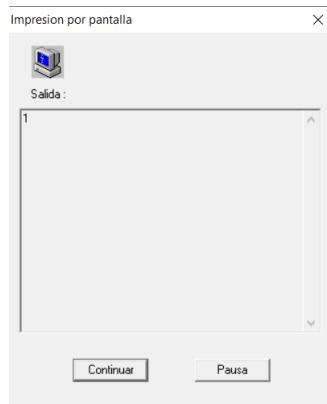


Ejecutamos el programa y nos aparece de la siguiente manera





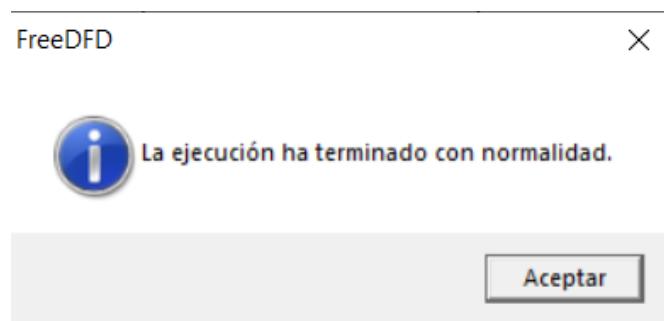
El primer numero en aparecer será el numero 1, damos click en continuar y nos seguirá apareciendo los números correspondientes.



Y así seguimos viendo que van apareciendo todos los números que consignamos en la condición hasta llegar al último número que fue 10 como se muestra en la imagen



Siendo 10 el ultimo numero la condición termina y nos mostrara el siguiente mensaje de finalización

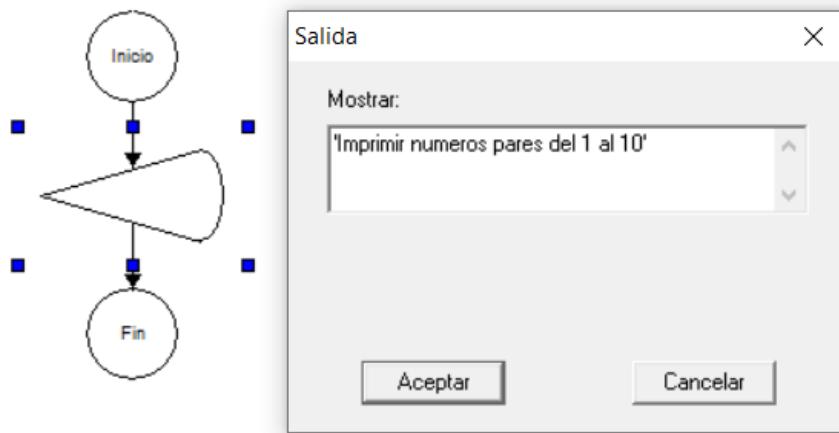




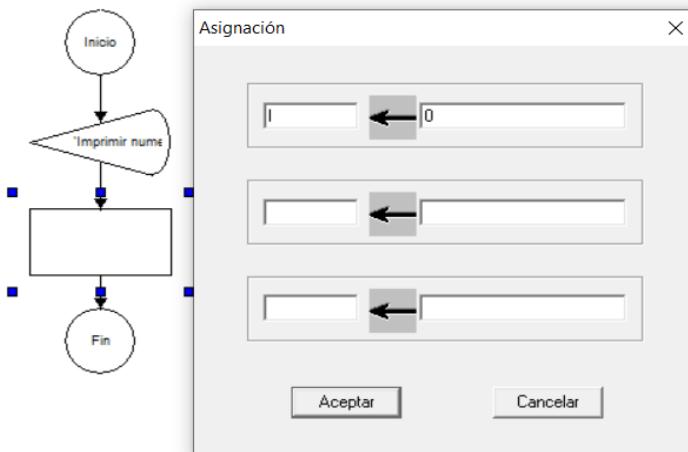
Ejemplo 03

Realizar un algoritmo que permita Imprimir números pares del 1 al 10

Empezamos agregando un elemento Salida para poder mostrar un mensaje que indique lo que vamos a realizar

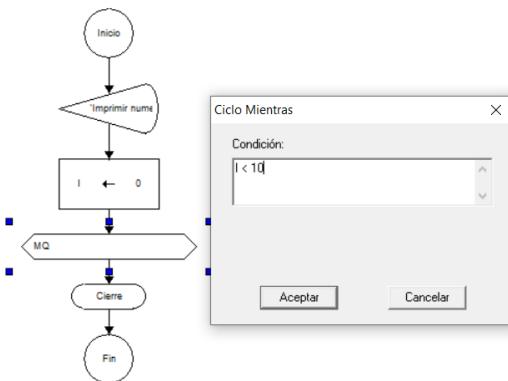


Ahora vamos a asignar una variable para inicializar los números que necesitamos mostrar en este caso se hará agregamos un elemento asignación como se muestra en la imagen:

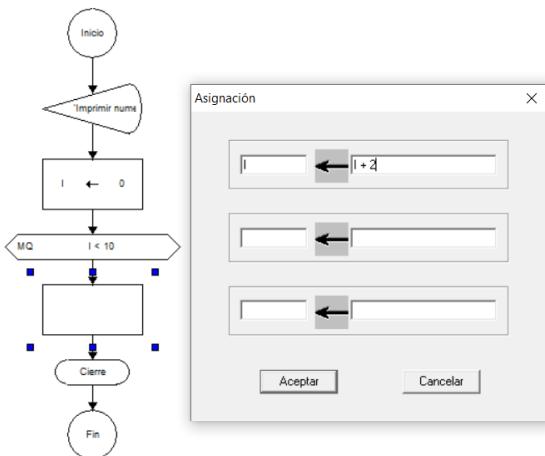




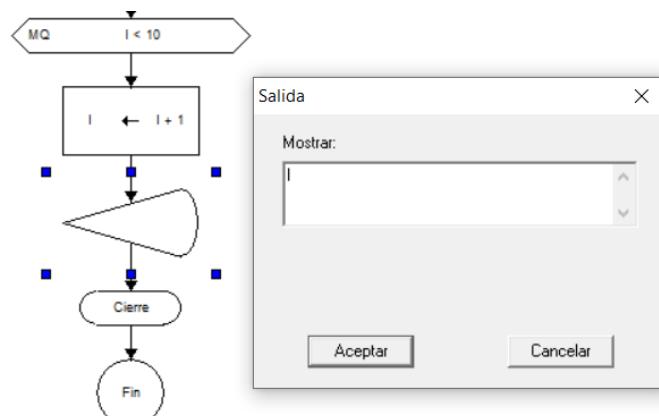
Ahora ingresamos el elemento para crear el ciclo “mientras” e ingresamos la condición que vamos a emplear



Ahora vamos a ingresar un contador para que pueda llegar hasta el numero solicitado en la condición para lo cual agregamos un elemento asignación



Mandamos a imprimir I como mostramos en la imagen insertando un elemento salida:

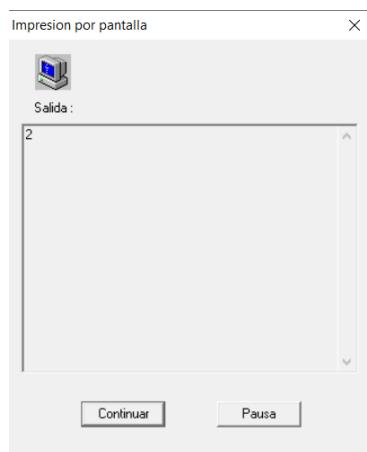




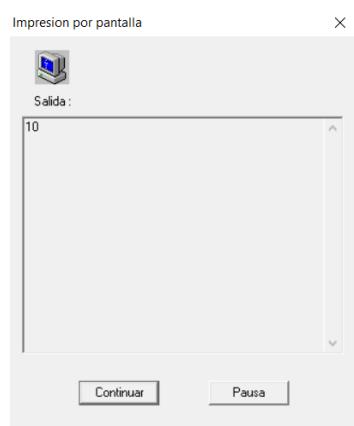
Ejecutamos el programa y nos aparece de la siguiente manera



El primer número en aparecer será el número 2, damos click en continuar y nos seguirá apareciendo los números correspondientes.

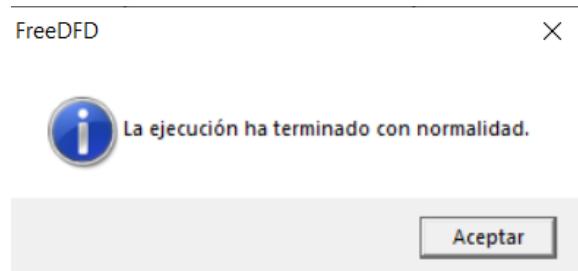


Y así seguimos viendo que van apareciendo todos los números que consignamos en la condición hasta llegar al último número que fue 10 como se muestra en la imagen





Siendo 10 el ultimo numero la condición termina y nos mostrara el siguiente mensaje de finalización



EJERCICOS PROPUESTOS

1. Realizar un algoritmo que permita Imprimir números del 10 al 1

2. Realizar un algoritmo que permita sumar los números comprendidos entre el 1 y el 5

3. Realizar un algoritmo que permita Ingresar 3 números y sumarlos

4. Realizar un algoritmo que permita Ingresar 4 números y contar los negativos



ACTIVIDAD DE REPASO Y DE INVESTIGACIÓN

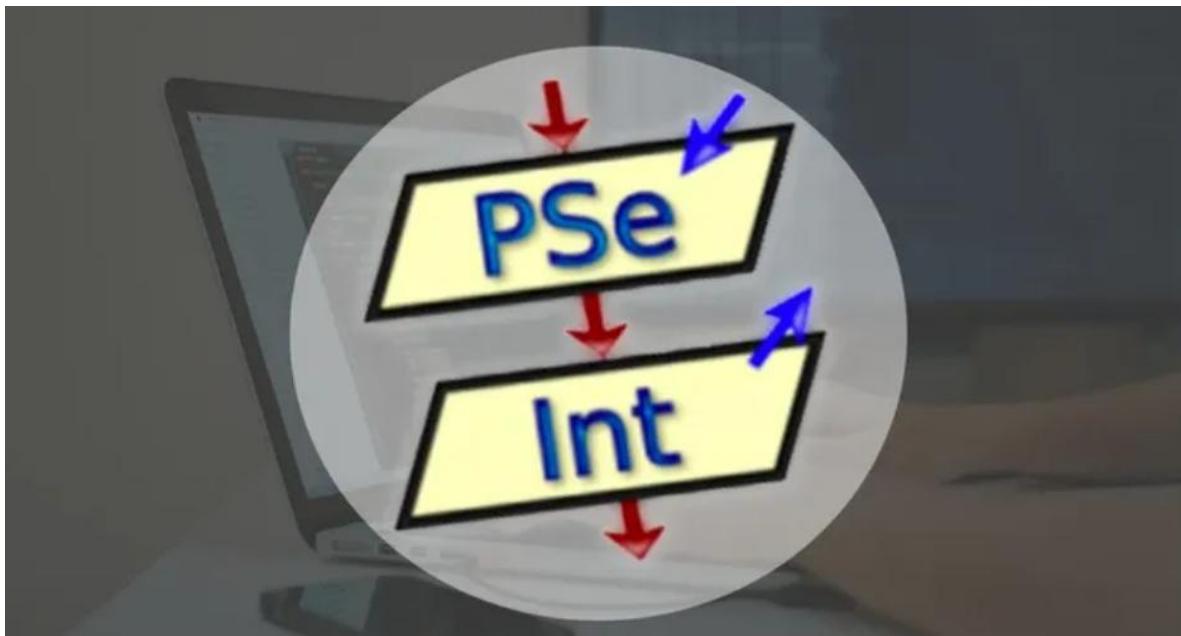
1. Imprimir en pantalla la palabra “Hola”, mientras se presione el número 1.
 2. Leer la nota de N alumnos y reportar la cantidad de aprobados (cap) y la cantidad de desaprobados (cde).
 3. Leer la edad de N alumnos y reportar la cantidad de mayores de edad (cma)y la cantidad de menores de edad (cme).
 4. Leer el sexo de N alumnos y reportar la cantidad de hombres (ch) y la cantidad de mujeres (cm).
 5. Leer N números enteros positivos y reportar la cantidad de pares (cp) y la cantidad de impares. (ci).



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA

Lógica de Programación



SESION 07

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I1: Diseña algoritmos utilizando las estructuras básicas de control: secuencial, selectiva y repetitivas



La Estructura de Control Repetitivas

Estructura Para (FOR)

Una estructura repetitiva o bucle se utiliza cuando se quiere repetir un conjunto de sentencias un número determinado de veces o mientras se mantenga el cumplimiento de una condición. El bucle FOR utiliza la primera opción usando una variable numérica capaz de controlar el número de iteraciones.

En general, la estructura FOR se usa en aquellas situaciones en las cuales CONOCEMOS la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Ejemplo: cargar 10 números, ingresar 5 notas de alumnos, etc. Conocemos de antemano la cantidad de veces que queremos que el bloque se repita.

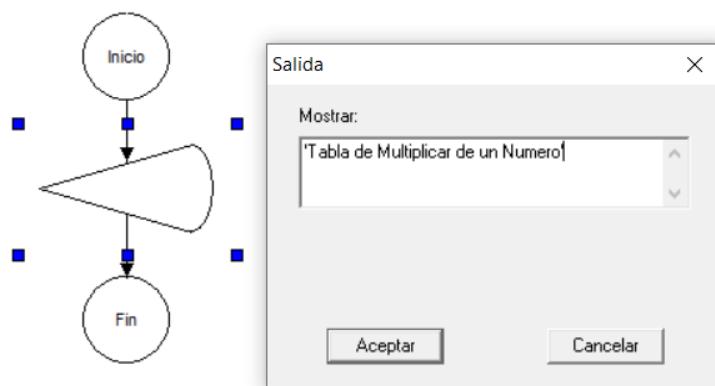
Este símbolo, es parecido al anterior (ciclo: mientras que..), solo que a éste le podemos agregar un inicio, un fin y un incremento. **Para** la variable con la que trabajemos. Pero sus usos son muy parecidos. Entonces vamos a ver un ejercicio para dejar más claro cómo funciona el ciclo: «para..»

Ejemplo 01:

Hacer un algoritmo que muestre la tabla de multiplicar de un número ingresado por el usuario. Y que la muestre con el formato: A x B = C

Bien, como dice el ejercicio simplemente hay que realizar una tabla de multiplicar de algún número que ingrese el usuario. Recordando que la tabla de multiplicar lleva un formato general del 1 al 12. Comencemos:

- Abrimos DFD
- Colocamos el detalle del ejercicio en un elemento salida como se muestra en la imagen siguiente:



Declaramos las variables que vamos a usar, en éste caso 3 variables:

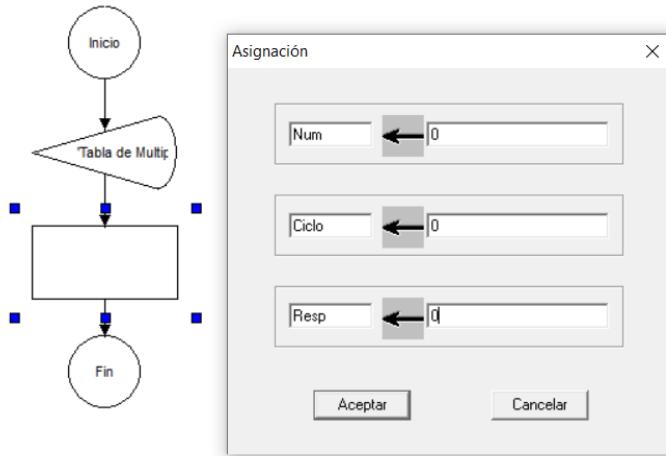
Num = cantidad que va a ingresar el usuario, del cual va a ser la tabla. Por ejemplo si ingresa el 4, se va a generar la tabla de multiplicar del 4.

Ciclo = controlador que vamos a usar, como ya les dije en la introducción el ciclo **para** sirve parecido al ciclo: «mientras que..» pero a diferencia que lleva un inicio, un fin, y un incremento. Entonces por eso siempre debemos crear una variable para hacer las condiciones. En este caso la variable se va a llamar Ciclo.

Resp = respuesta, es en donde vamos a almacenar el valor de la multiplicación, osea el resultado. Por ejemplo:

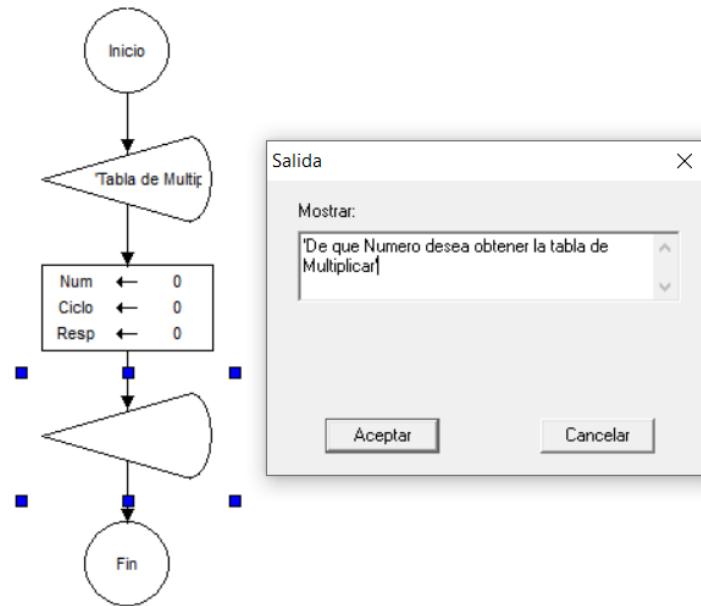
$$A \times B = C$$

El valor de «C» es el que se almacenará en Resp. Y es el que mostraremos.

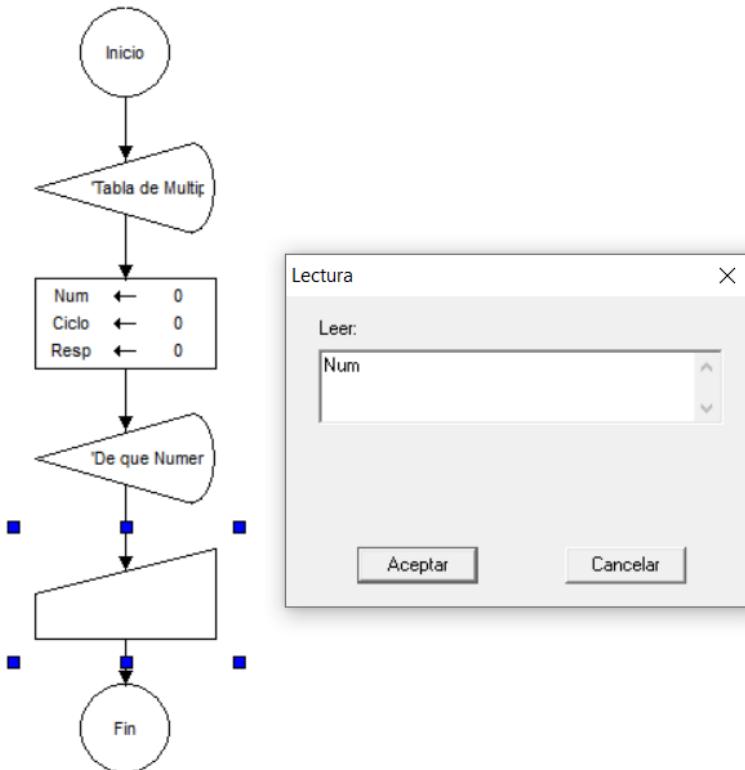




Ahora pedimos el dato, osea de que numero desea generar la tabla

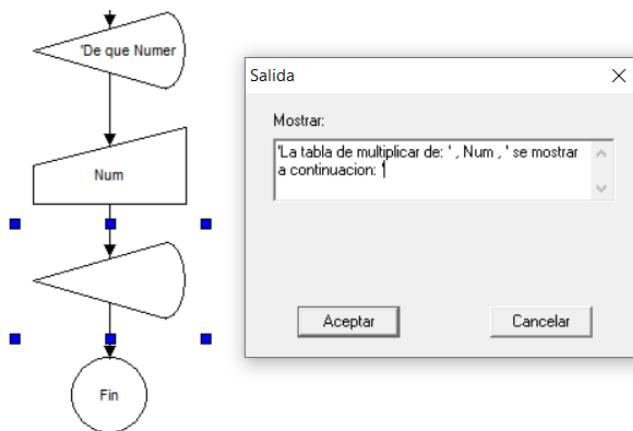


Ingresamos el numero correspondiente para este ejemplo vamos a ingresar el numero 4 para lo cual declaramos una variable Num

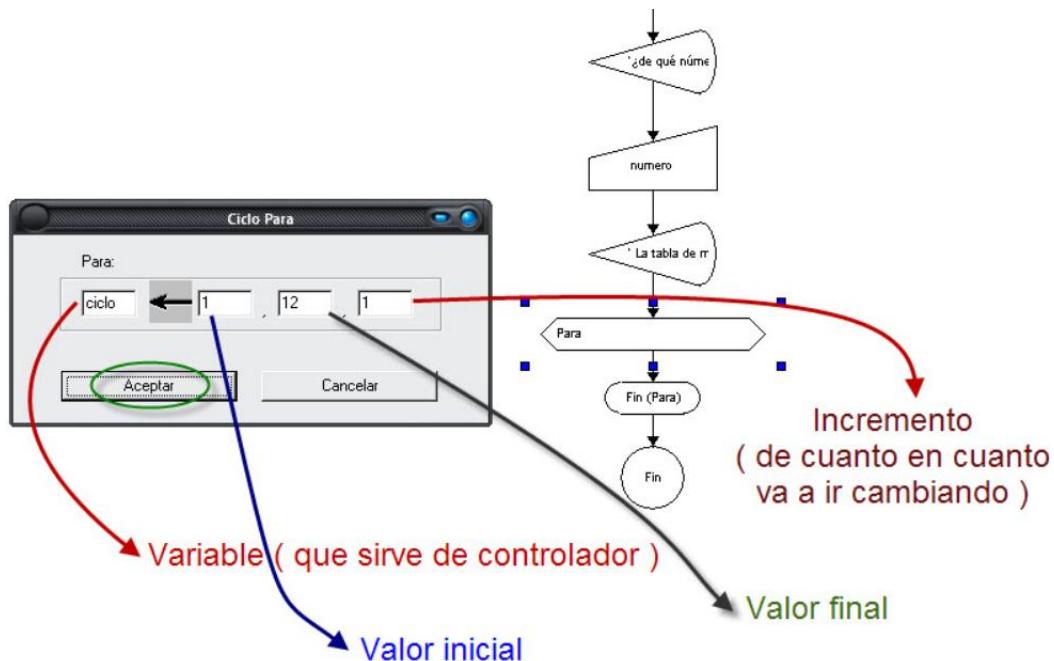




Mostramos un mensaje de comprobación, para que al usuario observe que de ese número que ingreso, se mostrará la tabla de multiplicación



Ahora vamos a abrir el «ciclo para..» porque el usuario/a, ya nos ha dado el numero para generar la tabla de multiplicar. Entonces decimos que:



EXPLICACION: el símbolo «para..» tiene la siguiente estructura: De izquierda a derecha,

Primer recuadro: ahí se coloca la variable controladora, es decir la variable que vamos a usar para el ciclo. En nuestro caso, la variable se llama: **ciclo**



Segundo recuadro: es el valor inicial (comienzo / partida) que le daremos a la variable.

Es decir que nuestra variable **ciclo** va a valer 1 al comienzo, por lo tanto, la tabla de multiplicación la comenzaremos con 1, lo cual está correcto porque las tablas de multiplicar llevan un formato:

$$2 \times 1 = 2$$

$$2 \times 2 = 4 \dots$$

Tercer recuadro: es el valor final (tope / meta) que le daremos a la variable. Es decir que nuestra variable **ciclo** va a llegar a valer un máximo de 12 al final, por lo tanto, la tabla de multiplicación la terminaremos con 12

$$2 \times 11 = 22$$

$$2 \times 12 = 24$$

Cuarto recuadro: es el incremento, es decir de cuanto en cuanto va a ir aumentando el ciclo. En este caso de uno en uno. Porque la tabla de multiplicar va de uno en uno, demostración:

$$2 \times 5 = 10$$

$$2 \times 6 = 12$$

$$2 \times 7 = 14$$

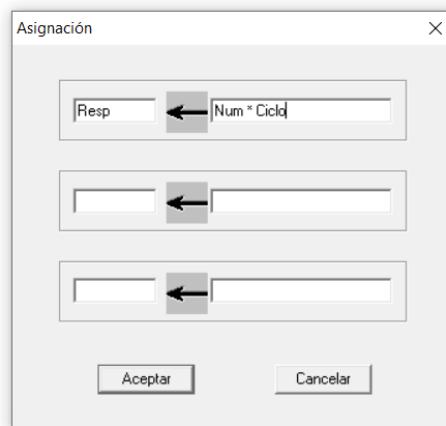
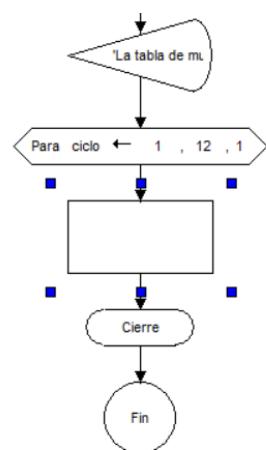
$$2 \times 8 = 16 \dots$$

Ven que la tabla va incrementando, en 1. Porque 5,6,7,8 es el orden con que va avanzando la tabla, y así seguirá hasta el tope, en este caso el tope es 12

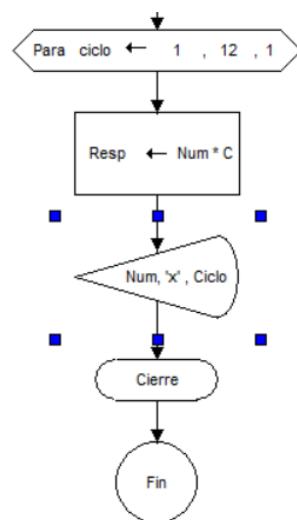
Ahora queda calcular la respuesta, ya tenemos expresado el ciclo que vamos a usar, osea desde 1 hasta 12 de uno en uno (nuestra tabla de multiplicar). Entonces solo nos queda expresar la respuesta, que es la multiplicación de:

- a) el número ingresado** (que no va a cambiar), por ejemplo, arriba en la tabla del 2, el dos (2) no cambia en toda la tabla porque de él se trata.
- b) el incremento** (de cuanto en cuanto va subiendo), en éste caso de uno en uno, y como vimos arriba en el ejemplo de la tabla del 2, el incremento si va cambiando Valia: 5, después 6, después 7, etc..

Vamos a realizar la operación, y como ya sabemos. Para realizar operaciones se usa el símbolo de «**asignación**» porque le vamos a asignar a Resp un valor (la multiplicación del número y el incremento). Así:



Por último, solo nos queda mostrar la tabla con el formato: A x B = C

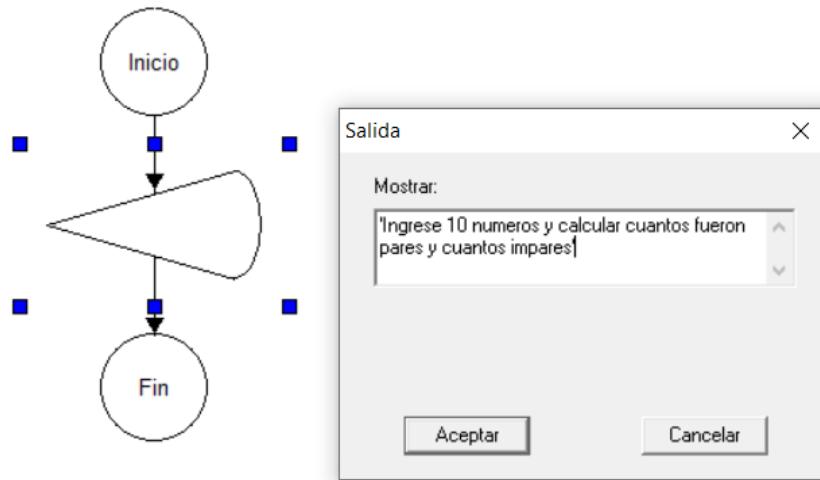




Ejemplo 02:

Hacer un algoritmo que pida 10 números y luego indique cuantos fueron pares y cuantos impares.

Abrimos DFD y Mostramos el detalle del ejercicio



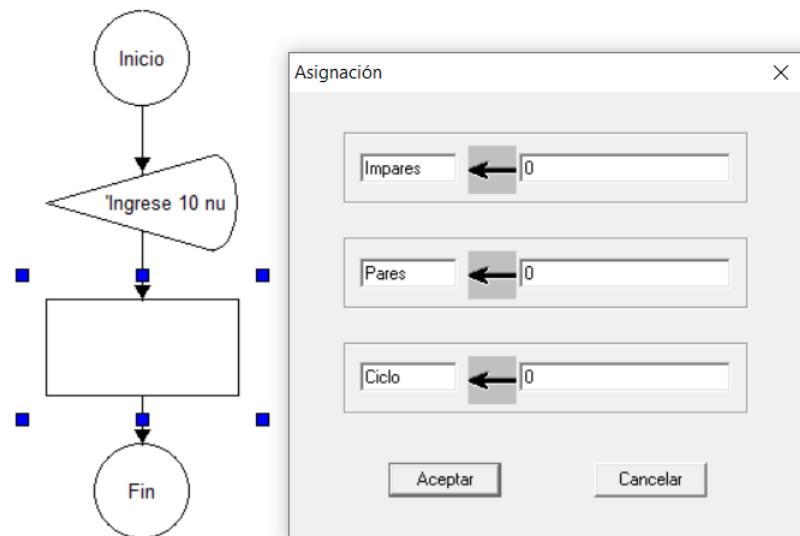
Declaramos las variables que usaremos, en este caso 4 variables:

Pares = la cantidad de números pares que se ingresaron

Impares = la cantidad de números impares que se ingresaron

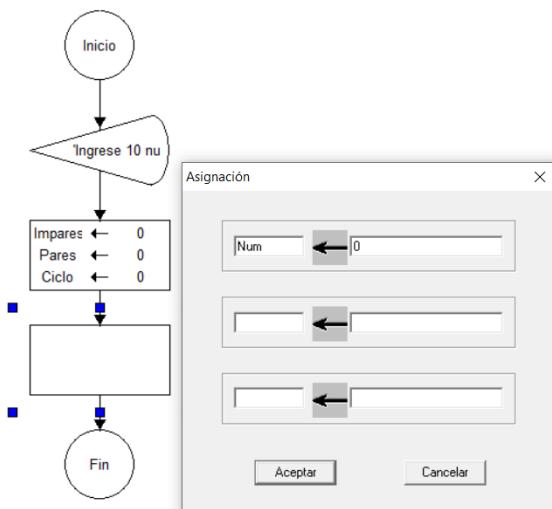
Ciclo = el nombre de la variable controladora que usaremos en el ciclo

Num = número que vamos a ingresar (10 en total)

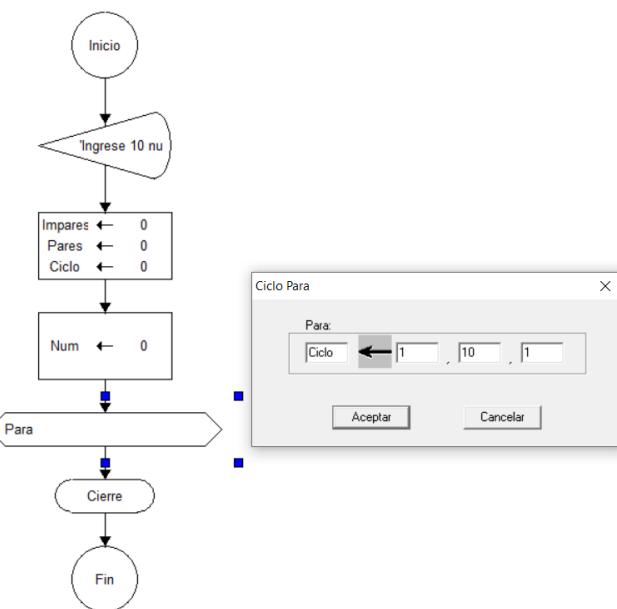




Ahora agregamos otro elemento asignación con la variable Num



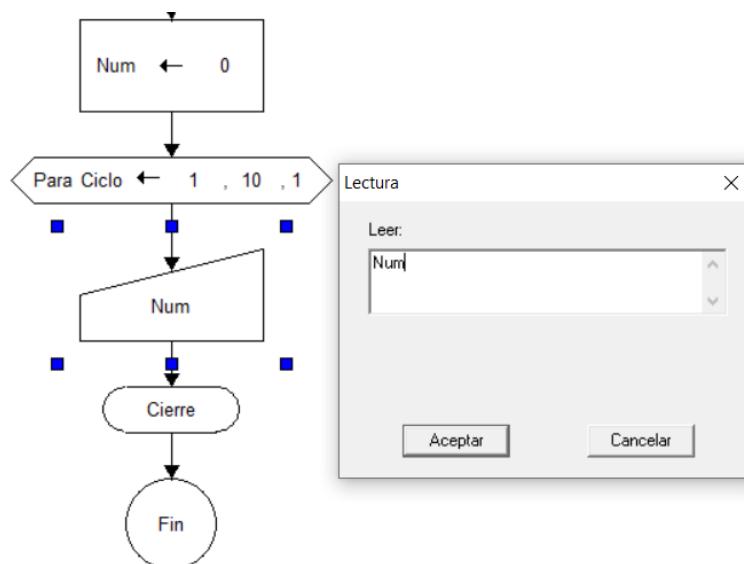
Comenzamos con el ciclo para, para pedir 10 números.



EXPLICACION: la variable «**ciclo**» nos va a servir de controlador, osea que va a controlar que se pida 10 veces el número, porque según la estructura del símbolo **para** hemos colocado que comience en 1 hasta 10, osea que va a pedir el dato desde **la primera vez** hasta **la décima vez**. y lo va a ir pidiendo de uno en uno, porque el incremento le hemos colocado **1**.



Bien, ahora colocamos el símbolo de **lectura** para que el usuario pueda ingresar el dato, y se almacene en la variable **Num**

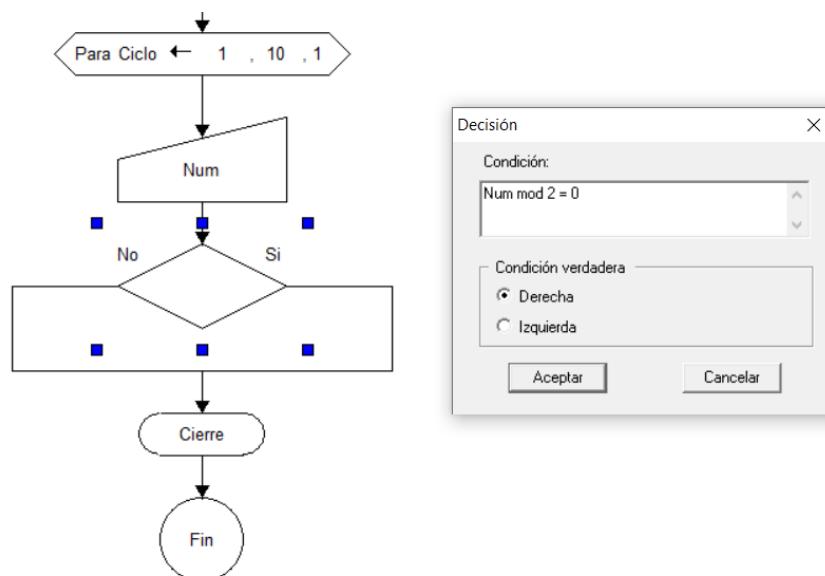


Ahora calcularemos si el número que ingresó es par o impar para poder contarlos y almacenarlos en las variables respectivas:

pares (si al dividirlo para 2 su residuo es cero)

impares (si al dividirlo para 2 su residuo es diferente que cero)

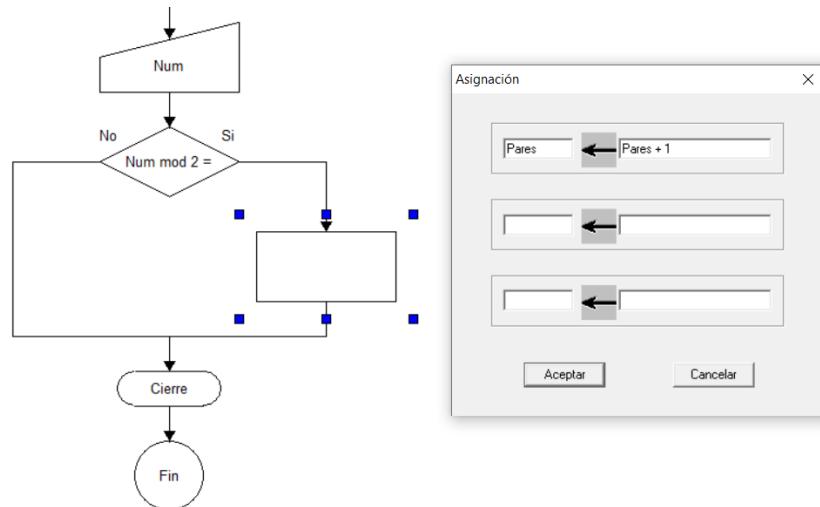
En la siguiente imagen decimos que si el residuo (mod) el dividir el número (Num) ingresado para 2, es cero. Entonces seguimos en el lado verdadero, caso contrario es falso.





Y esto lo analizará el programa en cada reiteración, es decir en cada ocasión que pida el número, va a calcular si es par o impar. Porque la operación de calcularlo está dentro del ciclo para. **Todo** lo que se encuentre dentro de algún ciclo forma parte de él.

En caso de que el residuo sea cero (0), es decir el lado verdadero:



EXPLICACION: eso que ven arriba es un **contador**, esa es la típica estructura que lleva un contador, pero cabe recalcar que:

¿Qué es un contador y para qué sirve?

Pues, un contador es una manera de ir sumando o restando valores a una variable, en nuestro caso a la variable **Pares**

Para dejarlo más claro, veamos por partes:

pares = 0 (cuando declaramos las variables)

entonces eso quiere decir que nuestra variable «pares» vale cero.

¿Y qué queríamos almacenar en ella?

queríamos almacenar una cantidad, es decir un numero por ejemplo **1,6,8** porque queremos que ahí se almacene cuantos números pares ingresó el usuario. Pero como vale cero por defecto, quedamos con algo así en el contador:

pares = pares + 1

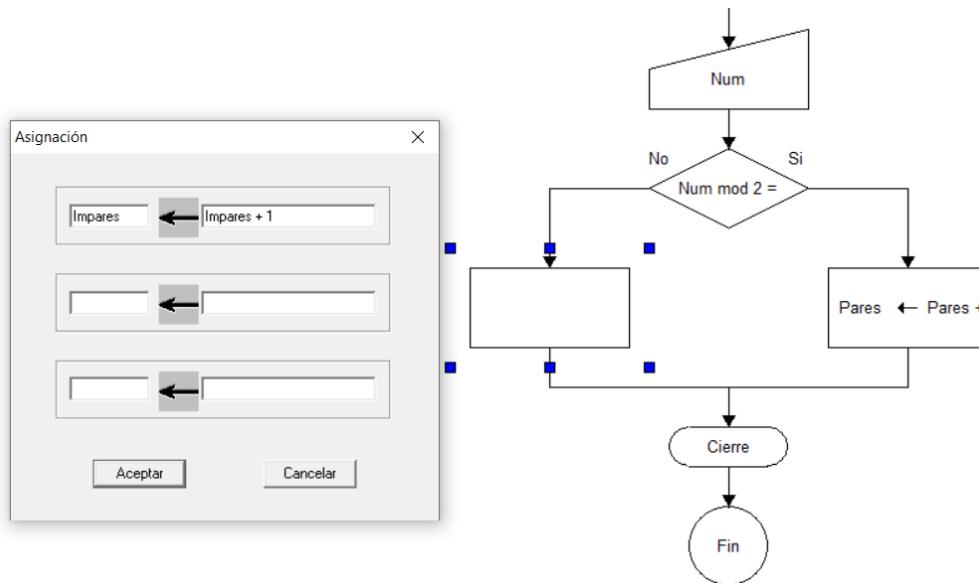
Equivale a decir esto en este momento:

$$0 = 0 + 1$$



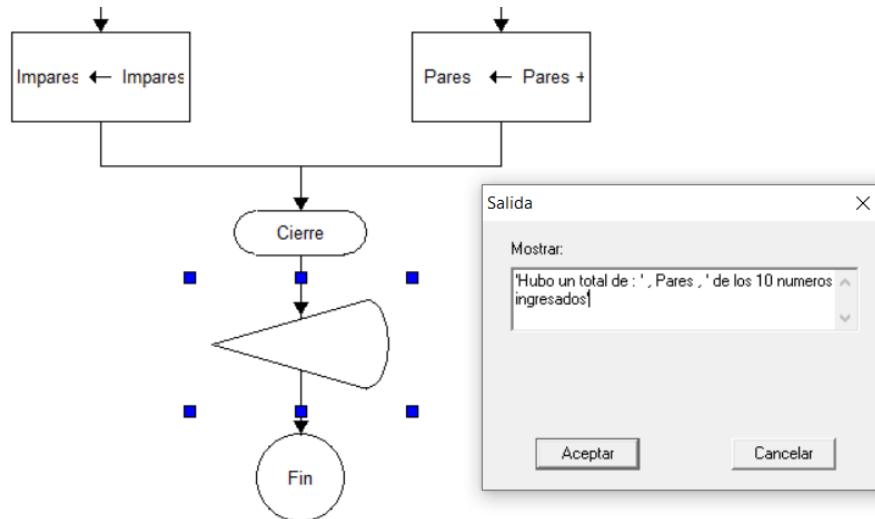
eso quiere decir que a la variable **pares** se le asignará el valor de la misma variable **pares** (que va a ir cambiando en cada reiteración) + 1.

Bien, el mismo funcionamiento va con el lado contrario solo que en cambio acá es con la variable **impares**.



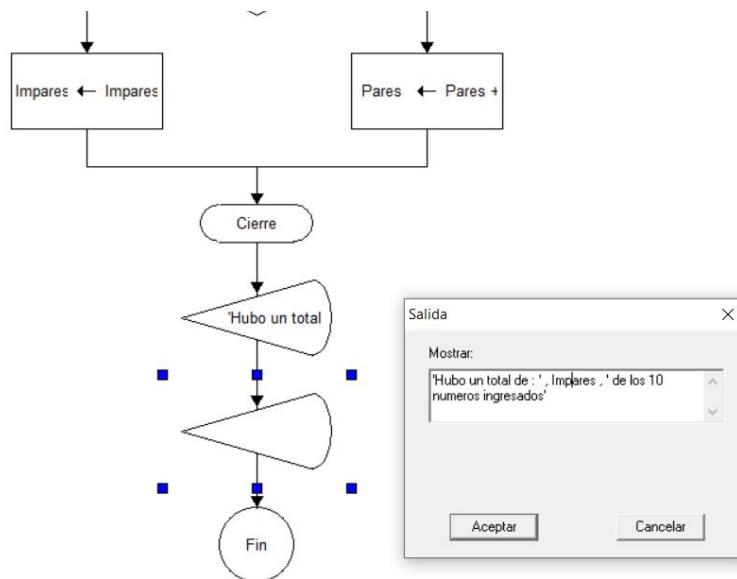
Ahora ya tenemos los 10 números, tenemos calculados y almacenados cuantos fueron pares y cuantos fueron impares, solo nos resta mostrar ese dato. Vamos a ello:

Pero debemos mostrar el resultado después del ciclo, porque si no nos mostraría el resultado cada reiteración. Es decir, nos mostraría 10 veces, una vez por número ingresado. Por eso debemos mostrarlo al final del ciclo. Así:





Y también hacemos lo mismo con los números Impares

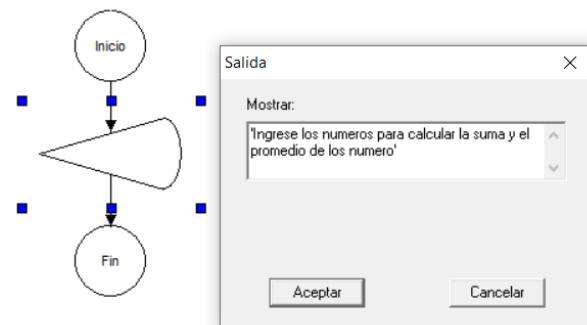


Ejemplo 03:

Hacer un algoritmo que permita ingresar N números y que luego calcule la suma y el promedio de los números ingresados.

Abrimos DFD

Colocamos el detalle del ejercicio



Declaramos las variables, en este caso 5 que son:

N = número que nos dirá cuántos números desea ingresar el usuario

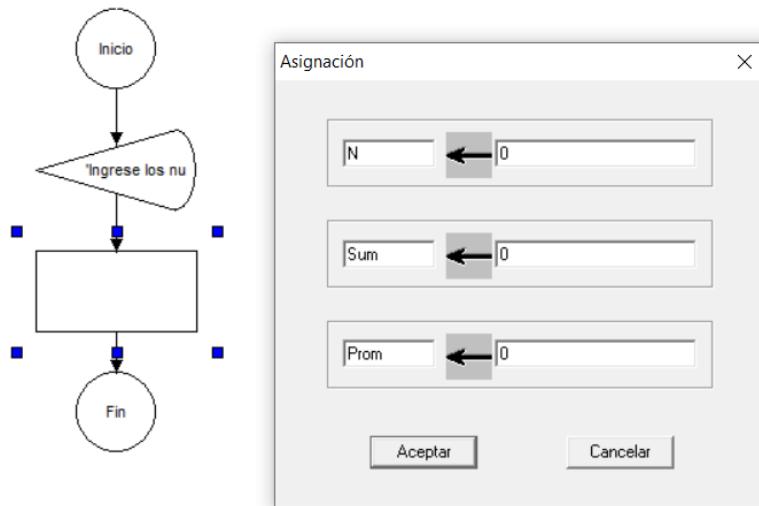
Sum = suma de los números ingresados

Prom = el promedio de los números ingresados

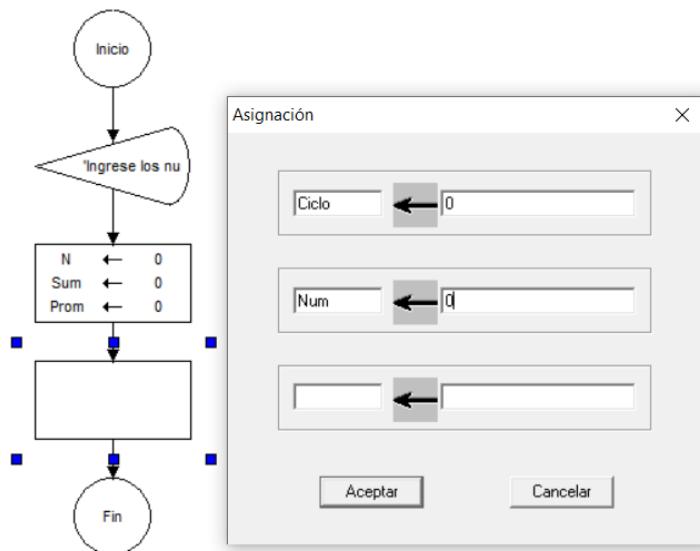
Ciclo = la variable controladora para el ciclo: para



Num = variable donde se almacenará el valor en cada vez que pida el numero el algoritmo

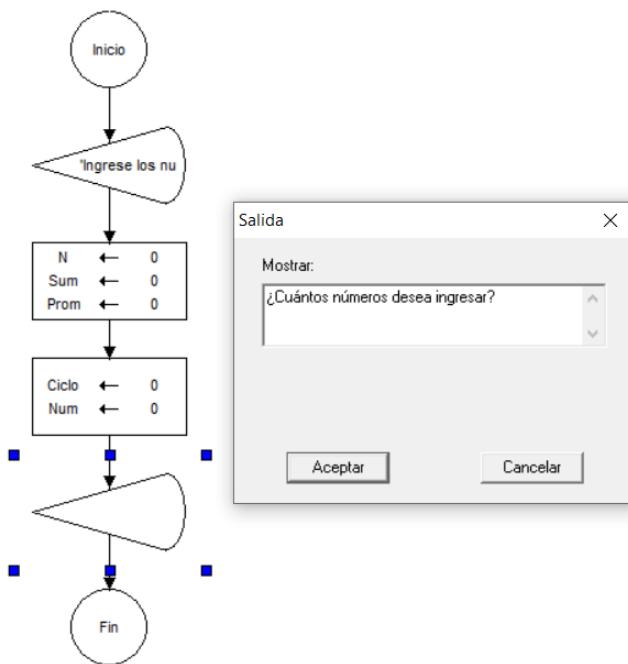


Agregamos otro elemento asignacion para ingresar las variables restantes



Ahora agregamos el ciclo **para** pero ésta vez no nos dice el problema cuantos números debemos pedir, si no que pidamos los números dependiendo de cuantos números desea ingresar el usuario, por lo tanto debemos preguntarle al usuario:

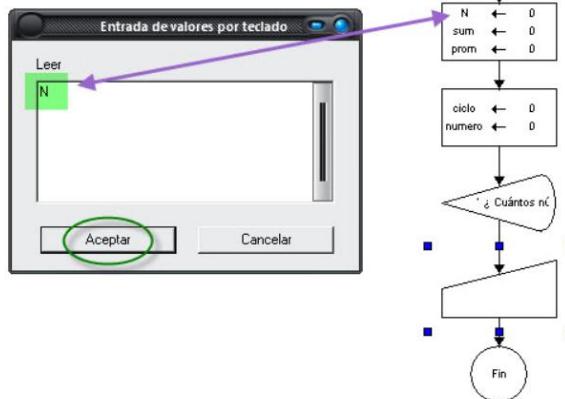
¿Cuántos números desea ingresar?, para que con ese valor pueda realizar el ciclo



Ahora ingresamos un elemento lectura para ingresar la cantidad de numeros que deseamos

Si fué declarada con mayúscula,
pues debe ser pedida con mayúscula.

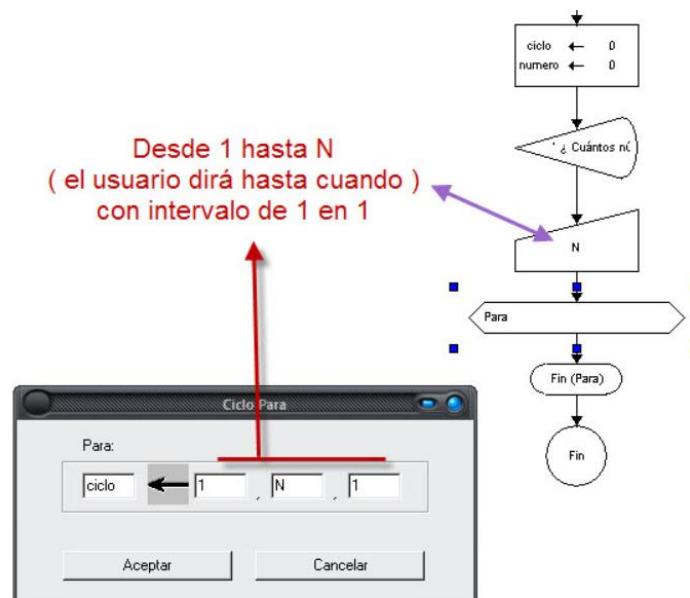
- regla para la declaración de variables -



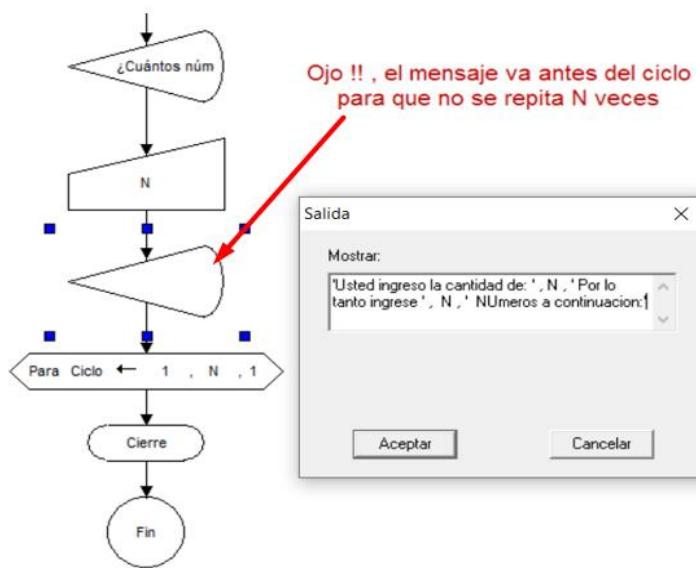
NOTA: como N fue declarada con mayúscula, pues debemos pedirla así mismo y si vamos a hacer una operación pues así mismo en mayúscula, porque se respeta de como fue declarada ([reglas para la declaración de variables](#))



Bien, ahora que nos ha dado el valor de N, por ejemplo que haya sido 5 pues nosotros pediremos 5 números, pero cada vez que ejecuten el programa cambiara de valor ya no será 5 si no 9 , etc.. por lo tanto usaremos la variable N para asignarle un tope a nuestro ciclo «para..» diremos que vaya desde 1 hasta N con un intervalo de 1 en 1 , así el algoritmo pedirá números una vez hasta el número de veces que haya querido el usuario, ya que N tiene en su interior un numero (el que pedimos al comienzo) y ese será nuestro **valor final o tope**

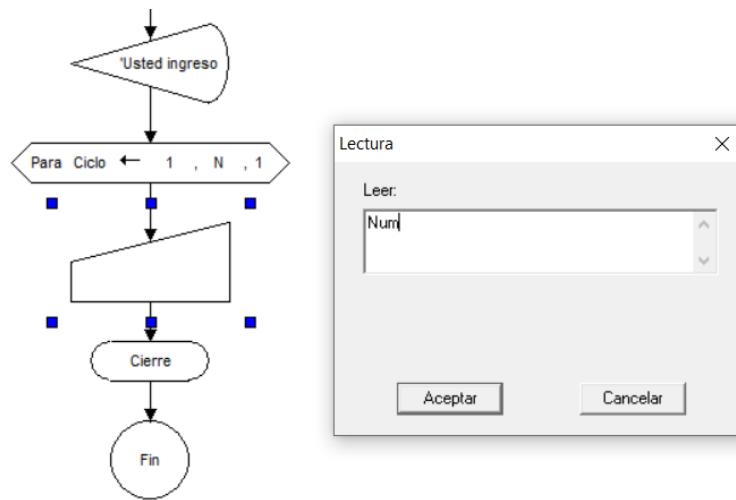


Luego vamos a insertar lo que va dentro del ciclo **para**, es decir lo que se va a ejecutar N número de veces, dependiendo de cuanto fue el número que ingresó el usuario. Aquí como ya estamos dentro del ciclo, vamos a pedir el número, para eso declaramos nuestra variable **Num**, entonces vamos a hacerle una especie de confirmación de cuanto fue lo que ingresó y por eso se van a pedir tantos números



NOTA: así como escribí en la captura, el mensaje debe ir antes del ciclo porque si lo ponemos dentro se va a repetir por cada vez que pida un número, y eso sería molesto para el usuario/a que va a ingresar las cantidades

Bien ya hemos colocado el mensaje de confirmación de cuantos números se van a pedir, ahora solo nos falta pedirlos. Para ello usamos el signo de **lectura** que ya lo conocemos al revés y al derecho



Ahora en cada reiteración, es decir las N veces que se ingresarán los números van a ir ingresando números distintos y el ejercicio pide que calculemos la suma y el promedio de



los números. Por lo tanto, en cada reiteración debemos ir sumando los números ingresados, para ello usaremos la variable **Sum**, con el siguiente formato:

$$\text{Sum} = \text{Sum} + \text{Num}$$

Lo que quiere decir, que en cada reiteración Sum va a ser igual a el mismo valor que tiene **Sum** (al principio cero) más el número ingresado. Veamos un ejemplo:

Primera vez,

(el usuario ingresó el valor de 15):

$$\text{Sum} = \text{Sum} + \text{Num} \longrightarrow 0 = 0 + 15 \longrightarrow 15$$

Segunda vez,

(el usuario/a ingresó el valor de 22):

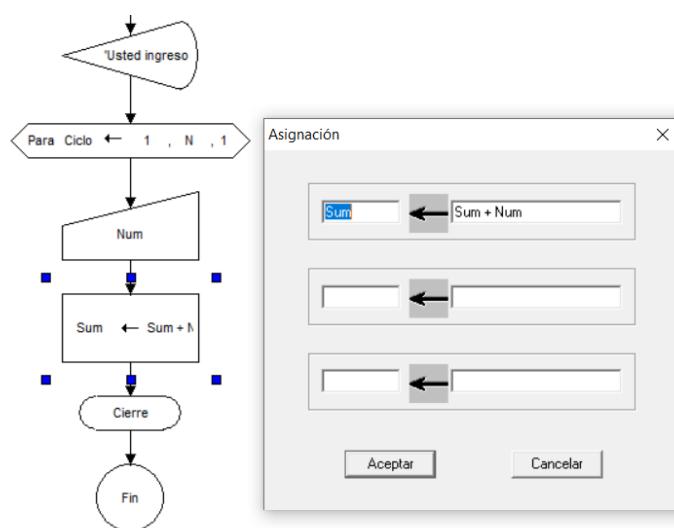
$$\text{Sum} = \text{Sum} + \text{Num} \longrightarrow 15 = 15 + 22 \longrightarrow 37$$

Tercera vez,

(el usuario/a ingresó el valor de 100):

$$\text{Sum} = \text{Sum} + \text{Num} \longrightarrow 37 = 37 + 100 \longrightarrow 137$$

Y así sucesivamente, hasta llegar a las N veces.



Y así como lo explique arriba en cada reintento para ingresar un número, se va a ir sumando los números ingresados, en el ejemplo se ingresaron las cantidades de: 15,22,100

Lo cual nos dice que $15 + 22 + 100 = 137$

Ya ingresados los N números y calculada la suma de los N números ingresados, nos falta calcular el promedio y luego mostrar los resultados. Como sabemos.



El promedio de algunas cantidades es igual a la suma de sus cantidades dividido para el número total de las cantidades sumadas.

Entonces, por ejemplo, si yo tengo 7 valores (cantidades), el promedio sería la suma de los 7 valores dividido entre el número de total de los valores sumados, osea 7. Así:

Tenemos 7 cantidades: 1,2,3,4,5,6,7

Queremos calcular el promedio, para ello vamos a la regla que dice:

es igual a la suma de sus cantidades

entonces: **1+2+3+4+5+6+7 = 28**

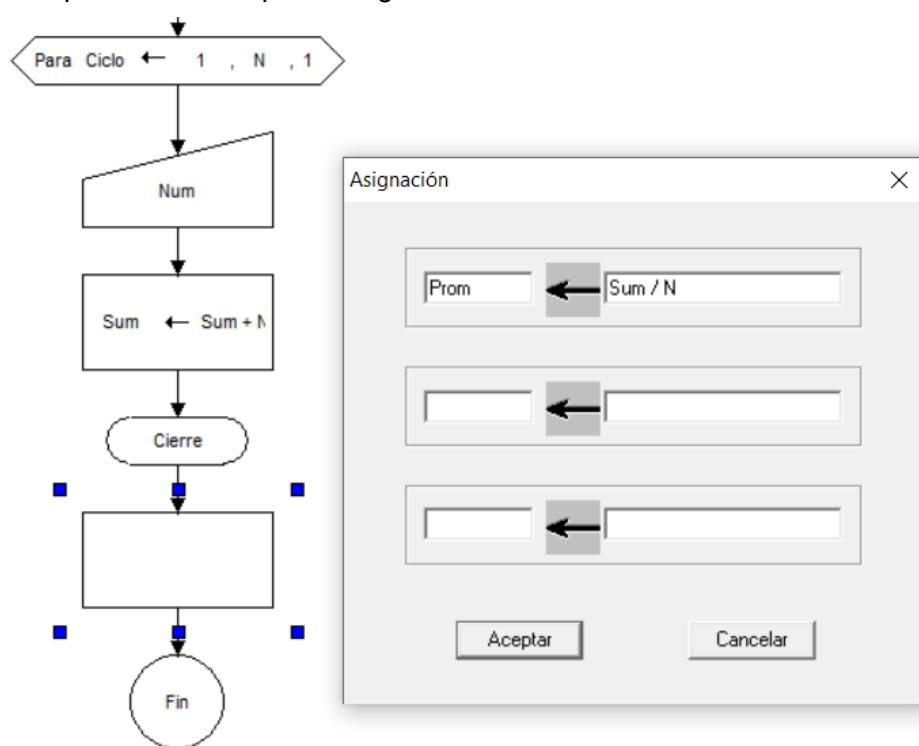
Seguimos con la regla que dice:

dividido para el número total de las cantidades sumadas

¿cuántos números totalmente sumé?, 7 por lo tanto:

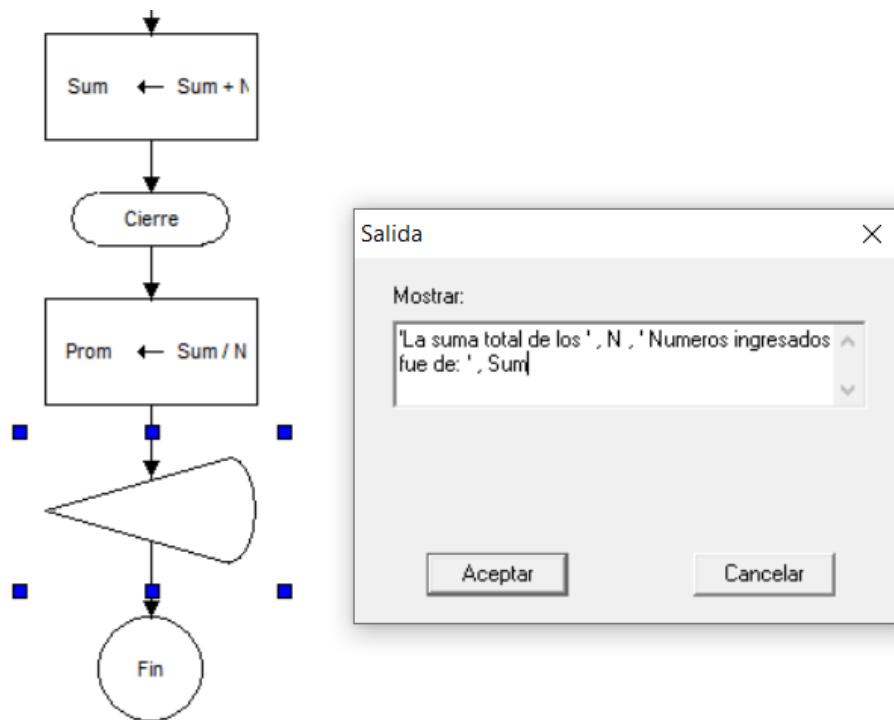
28 / 7 = 4

Donde 4 es el promedio de aquellos dígitos

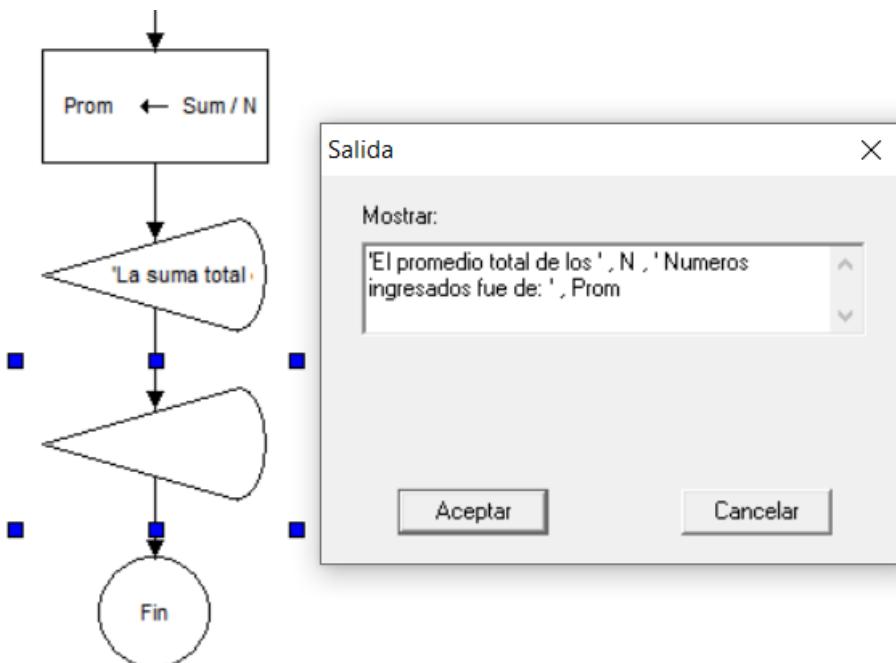


Bien, ya tenemos calculado la suma y el promedio. Solo nos resta mostrar los resultados obtenidos tal y como lo dice el enunciado del problema.

Para la suma:



Para el promedio:





EJERCICOS PROPUESTOS

1. Realizar un algoritmo que lea 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores.

2. Realizar un algoritmo que lea n números enteros y calcule la cantidad de valores mayores o iguales a 1000.

3. Realizar un algoritmo que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados.

4. Realizar un algoritmo que lea la edad de N alumnos y reportar la cantidad de mayores de edad y la cantidad de menores de edad teniendo en cuenta que un mayor de edad es cuando su edad es mayor o igual a 18 años

5. Realizar un algoritmo que lea la nota de N alumnos y reportar la suma de las notas de los aprobados y la suma de las notas de los desaprobados.

6. Realizar un algoritmo que lea el sexo de N alumnos y reportar la cantidad de hombres y la cantidad de mujeres.



ACTIVIDAD DE REPASO Y DE INVESTIGACIÓN

1. Leer un número N entero positivo y reportar si es primo o No.
2. Leer un número entero positivo N y reportar el factorial del número.
3. Escribir un programa que lea el nombre de N alumnos con su respectiva nota y al final reporte la nota promedio.
4. Escribir un programa que lea Nombre y precio de N productos y reporte cuanto se debe pagar en total.
5. Programa que lea N números enteros y para cada uno de ellos muestre su correspondiente cuadrado.
6. Programa que lea edad y sexo(M,F) de N alumnos y reporta:
 - a) Cantidad de hombres mayores de edad.
 - b) Cantidad de mujeres menores de edad.
 - c) Edad promedio de hombres
 - d) Edad promedio de mujeres
 - e) Edad promedio total.

<https://eperdomo89.wordpress.com/category/programando/dfd/>

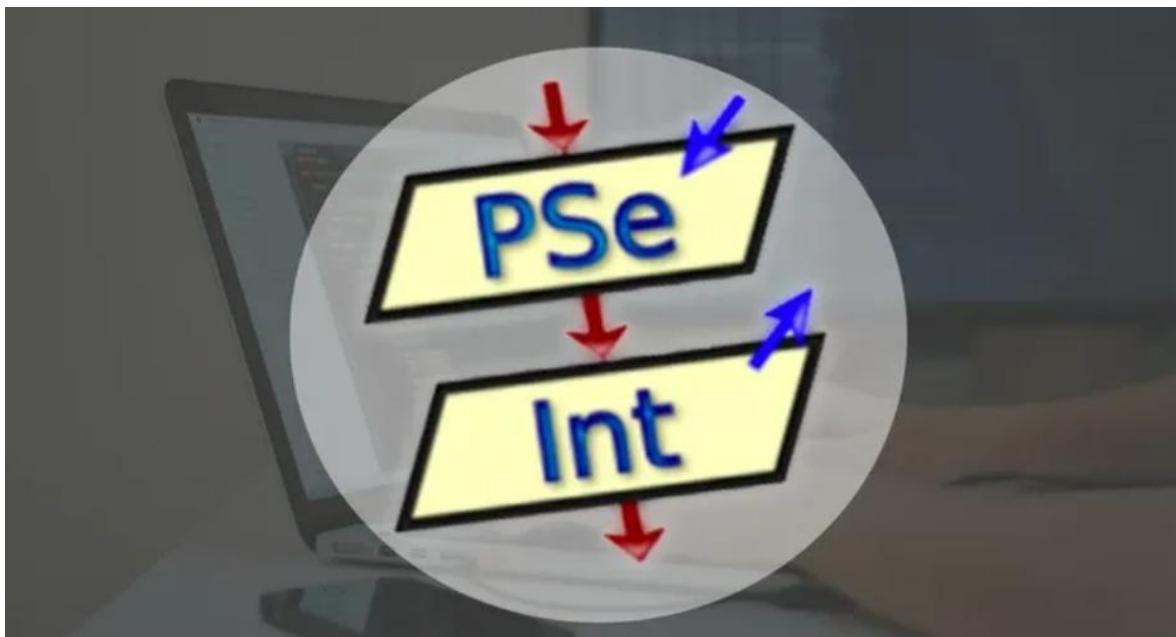
<https://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?codigo=84>



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA

Lógica de Programación



SESION 08

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I1: Diseña algoritmos utilizando las estructuras básicas de control: secuencial, selectiva y repetitivas



PROYECTO

Un proyecto es esencialmente un conjunto de actividades interrelacionadas, con un inicio y una finalización definida, que utiliza recursos limitados para lograr un objetivo deseado. Los dos elementos básicos que incluye esta definición son: las actividades y los recursos.

LAS ACTIVIDADES son las tareas que deben ejecutarse para llegar en conjunto a un fin preestablecido (objetivo deseado); por ejemplo: recopilar información; realizar diagnósticos; confeccionar un diseño global de un procedimiento, programar, escribir manuales de procedimiento, etc.

Un aspecto fundamental en todo proyecto es el orden en el cual se realizan las actividades. Y para determinar la secuencia lógica de las actividades se debe establecer el método, el tiempo y el costo de cada operación.

LOS RECURSOS son los elementos utilizados para poder realizar la ejecución de cada una de las tareas; como, por ejemplo: hardware, programas de base (sistemas operativos), programas de aplicación, discos de almacenamiento, energía, servicios, inversiones de capital, personal, información, dinero y tiempo.

Entonces: El fin primario de desarrollar un proyecto debe ser producir un programa calendario en el cual los recursos, siempre limitados, se asignen a cada una de las actividades en forma económico-óptima.

Estas limitaciones en cuyo contexto se resuelve planear un proyecto pueden ser internas, por ejemplo: computadoras disponibles, capacidad del personal, disposiciones presupuestarias, o bien externas, como ser: fechas de entrega de cualquier tipo de recursos, factores climáticos, aprobaciones de organismos oficiales. En ambos casos las limitaciones deben tenerse particularmente en cuenta al estimar los tiempos de cada actividad.

En cuanto al objetivo del proyecto, este puede ser sencillo y no demandar ni muchas tareas ni demasiados recursos; o, por el contrario, puede ser complejo y exigir múltiples actividades y una gran cantidad de recursos para poder alcanzarlo.



Pero independientemente de su complejidad, característicamente todo proyecto reúne la mayoría de los siguientes criterios:

1. Tener un principio y un fin
2. Tener un calendario definido de ejecución
3. Plantearse de una sola vez
4. Constar de una sucesión de actividades o de fases
5. Agrupar personas en función de las necesidades específicas de cada actividad
6. Contar con los recursos necesarios para desenvolver las actividades

Ahora piense por un instante en cada uno de los proyectos que se desarrollan en las organizaciones, y verá que todos ellos tienen cometidos que deben cumplirse en un cierto plazo de tiempo y que además requieren de la concurrencia de otras personas.

Y es aquí donde empieza a tener relevancia la figura del administrador, en los proyectos a realizarse en las organizaciones; incluidos los proyectos informáticos.

Algo importante a tener siempre presente es que: si el administrador realiza un buen trabajo en la gestión del proyecto, su éxito podrá ser visto y verificado por los demás; en caso contrario, naturalmente, el fracaso también estará a la vista de todo el mundo. La responsabilidad es muy alta: alcanzar el objetivo o no. Pero la oportunidad de "demostrar la capacidad profesional", es de las que no pueden dejarse pasa por alto.

Los administradores eficaces de proyectos, son los que logran que el trabajo se ejecute a tiempo, dentro del presupuesto, y conforme a las normas de calidad especificadas.

QUÉ ES UN PROYECTO INFORMÁTICO

De la definición de proyectos, vista en el punto anterior, podemos aplicarla a los proyectos informáticos; y decir que: un proyecto informático es un sistema de cursos de acción simultáneos y/o secuenciales que incluye personas, equipamientos de hardware, software y comunicaciones, enfocados en obtener uno o más resultados deseables sobre un sistema de información.



Presentaremos cuáles son las tareas y en qué secuencia deben realizarse para alcanzar el objetivo. Ahora vamos a describir los distintos objetivos que caracterizan a un proyecto informático.

El inicio de un proyecto informático generalmente está dado en la solicitud de requerimientos de los usuarios, y siendo que los diferentes sistemas de Información abordan los diferentes tipos de problemas organizacionales; podemos clasificar a los Sistemas de Información según sean las aplicaciones que necesite cada usuario en: Sistemas de Transacciones, Sistemas de Soporte para la toma de decisiones, y Sistemas Expertos.

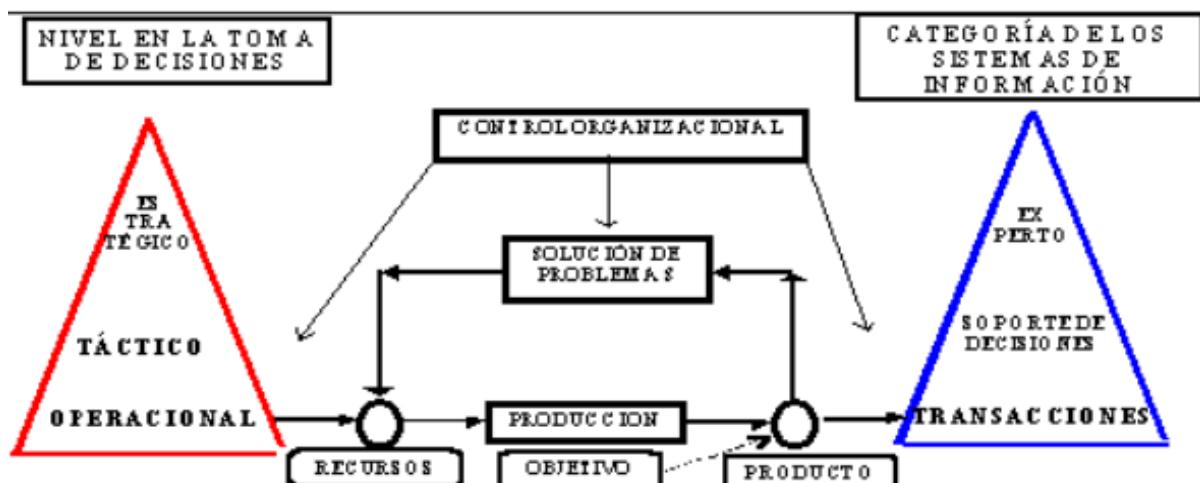


FIGURA1 Clasificación de los sistemas de información

INICIO DE UN PROYECTO INFORMÁTICO

Ya vimos una clasificación, que nos permite clarificar el origen de un proyecto informático, pero ¿cómo podremos determinar la magnitud de un Proyecto informático?

En un entorno informático estable, la decisión de iniciar un proyecto viene dada por las necesidades de: **mantenimiento, modificación, mejoramiento, reemplazo o capacidad**; encuadrándose así, el proyecto informático, dentro de una categoría de complejidad.

El Mantenimiento del programa; es una consecuencia de una omisión realizada en la etapa del diseño del sistema e involucra solucionar fallas menores del sistema, que



obligará a la realización de cambios en el programa; como por ejemplo el descuido de no considerar que puedan ocurrir en el sistema, ciertas condiciones extraordinarias; como sería el caso de un aumento no previsto del 60 %, en la emisión de órdenes de compra. Las fallas también pueden provenir de otros factores, como ser en el caso de que existan cambios en las expectativas de los usuarios.

La Modificación del programa; involucra algo más que un simple cambio en el programa; involucra un cambio estructural de una entidad, Por ejemplo, un cambio en el número de dígitos del código postal, o en el código de zona telefónica. La diferencia con el Mantenimiento es el grado de importancia.

El Mejoramiento del sistema; es el agregado de capacidades que no formaron parte del sistema de información original; por ejemplo, cuando en una división se implementó un sistema de inventarios, este sistema no incluía un módulo para calcular la futura demanda de bienes y partes. La inclusión de este sofisticado módulo de cálculo es considerada un mejoramiento del sistema.

El Reemplazo del sistema; ocurre cuando los sistemas de información se tornan físicamente, tecnológicamente o competitivamente obsoletos. Como es el caso de la utilización del láser, en el reconocimiento óptico de caracteres para la lectura del código de barras, remplazando a la entrada por teclado.

La Nueva Capacidad del sistema; son sistemas de información para los cuales no es necesario el uso de la automatización. Están dados por la capacidad de poder modelizar la aplicabilidad de nuevos sistemas. Un ejemplo de ello, es la aplicación de los sistemas expertos.





FUNCIONES

Se considera que una función es una parte de código capaz de realizar una tarea y/o de transformar valores para obtener otro valor. Una función se define por

El nombre. Este no debe ser ambiguo; según el lenguaje, el nombre debe ser único con respecto a las variables globales y a otras funciones.

El tipo de valor que la función regresa.

El número fijo (o variable) de parámetros y la lista ordenada de tipo aceptable por los parámetros

El código. Este es único para cada función.

Cercanas a la noción de función, se encuentran las nociones de procedimiento, rutina o subrutina, las cuales significan una parte del programa encargada de realizar una tarea sin regresar expresamente un valor.

Los valores calculados o transformados por el código se regresan en la lista de los parámetros. Si el lenguaje permite la redefinición de las funciones, por ejemplo, los lenguajes orientados a objetos, como C++ o Java, o que las funciones tengan varias listas de parámetros, el código de una función no es único.

En el primer caso se toma en cuenta la última definición de la función, mientras que en el segundo se hace la correspondencia entre la lista de parámetros actuales y las listas de parámetros. Una vez que la función está definida (y si no tiene restricciones de acceso; por ejemplo, no es privada, como en el caso del lenguaje Java, o no es una función interna de otra función, como en el lenguaje C), en todo el código se pueden hacer una o varias llamadas a la función, con la única restricción de que la lista de los parámetros reales (especificados en la llamada) correspondan en nombre y tipo con los parámetros formales (que aparecen en la definición de la función).

La llamada de la función se hace especificando:

- El nombre de la función
- La lista de los parámetros reales, los cuales pueden ser expresiones que se evalúan o variables.



Por su parte, la sintaxis de una llamada de función es casi la misma para prácticamente todos los lenguajes (a excepción de algunos lenguajes funcionales, como LISP o SCHEME) y es inspirada en la notación matemática:

Nombre_función(parametro1, parametro2, ...)

La sintaxis de la definición de una función varía considerablemente de un lenguaje a otro, al igual que la semántica de la definición y el modo de ejecución de las llamadas. En el caso de las llamadas de funciones, estas tienen varias semánticas, según el paradigma del lenguaje de programación. Por el paradigma imperativo y por las funciones que regresan valores, las llamadas se comportan como expresiones del tipo regresado.

De acuerdo con el lenguaje de programación, los parámetros pueden modificarse o no en el código de la función, donde el valor del parámetro a la salida de la función es cambiado. O se indica expresamente si los parámetros son de entrada (cuando sus valores no cambian) o de salida (por ejemplo, el lenguaje PL/SQL) si los valores van a cambiar.

Un parámetro real que no es de salida puede ser cualquier expresión posible del parámetro formal. En el lenguaje C solo existe la noción de parámetro de entrada y de salida, lo cual depende de la forma en que se transmite: el valor indicado por una variable o una expresión; o un apuntador al contenido de una variable. Solo un apuntador transmitido como parámetro puede cambiar el contenido de la memoria. Hablamos de parámetros transmitidos por valor o por referencia.

En todos los lenguajes, más que las funciones definidas por el usuario, se utilizan funciones que provienen de librerías externas, estándares (anexadas al compilador o al intérprete) o funciones adicionales. La función citada, que borra la ventana de trabajo, proviene de una librería estándar para los dos lenguajes.



RECUSIVIDAD

La recursividad es un concepto fundamental en matemáticas y en computación.

- Es una alternativa diferente para implementar estructuras de repetición (ciclos). Los módulos se hacen llamadas recursivas.
- Se puede usar en toda situación en la cual la solución pueda ser expresada como una secuencia de movimientos, pasos o transformaciones gobernadas por un conjunto de reglas no ambiguas.

Las funciones recursivas se componen de:

- Caso base: una solución simple para un caso particular (puede haber más de un caso base).

Caso recursivo: una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base. Los pasos que sigue el caso recursivo son los siguientes:

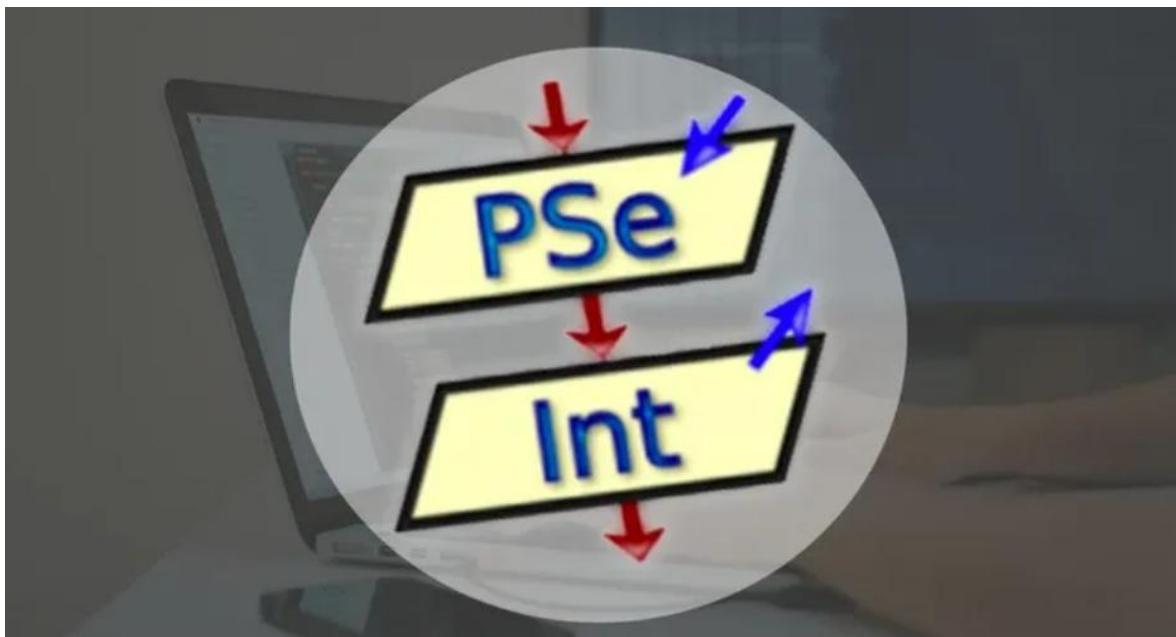
1. El procedimiento se llama a sí mismo
2. El problema se resuelve, tratando el mismo problema, pero de tamaño menor
3. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA

Lógica de Programación



SESION 09

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I2: Diagrama clases de objetos aplicando los elementos de las técnicas de programación orientadas a objetos.



Introducción al Lenguaje PSeInt

El programa conocido como PSeInt, debe su nombre a la contracción Pseudocode Interpreter, (Intérprete de Pseudocódigo), y en su primera versión, fue el resultado del trabajo de proyecto final de la asignatura Programación 1, del estudiante Pablo Novara de la carrera Ingeniería en Informática de la Facultad de Ingeniería y Ciencias Hídricas de la Universidad Nacional del Litoral.

Según su autor, el principal propósito de la aplicación es ayudar al aprendizaje de diseño de algoritmos. Con los apoyos que brinda la herramienta, así, el aprendiz podrá poner sus esfuerzos intelectuales principalmente en los aspectos lógicos de la solución, en las acciones, la secuenciación de ellas como procesos, y en los aspectos referidos a la organización de estas como estructura de programación. En su versión actual, es un software de licencia GLP (General Public License). Dentro de sus principales características podemos destacar:

- Es un software libre de licencia GPL y gratuito.
- Multiplataforma, existen versiones para instalación en sistemas operativos Microsoft Windows de 32 y 64 bits, Mac OS i686 y Mac PowerPC, GNU/Linux de 32 y 64 bits
- Permite la definición de perfiles para adaptar los lenguajes de pseudocódigos y de diagrama de flujos diferentes estilos y sintaxis.
- La escritura de los algoritmos se puede hacer directamente como sentencias en pseudocódigo, o utilizando elementos de la ventana de comandos que representan las sentencias usando símbolos del lenguaje de diagramas de flujo.
- Para la escritura de los pseudocódigos se dispone de facilidades de: autocompletado; ayudas emergentes; plantillas de comandos; coloreado de sintaxis; resaltado de bloques lógicos y listado de funciones, operadores y variables de uso común en programación.
- Permite generar y editar el diagrama de flujo a partir del pseudocódigo, con la opción de grabarlo como archivo de imagen.
- Permite la edición simultánea de múltiples algoritmos.
- Puede interpretar (ejecutar) los algoritmos escritos, contando con facilidades para: modificar el algoritmo y ver los cambios en la ejecución inmediatamente (sin reingresar los datos); modificar uno o más datos seleccionados de una ejecución ya finalizada para observar cómo varían los resultados; deshacer una ejecución para reiniciarla o repetirla



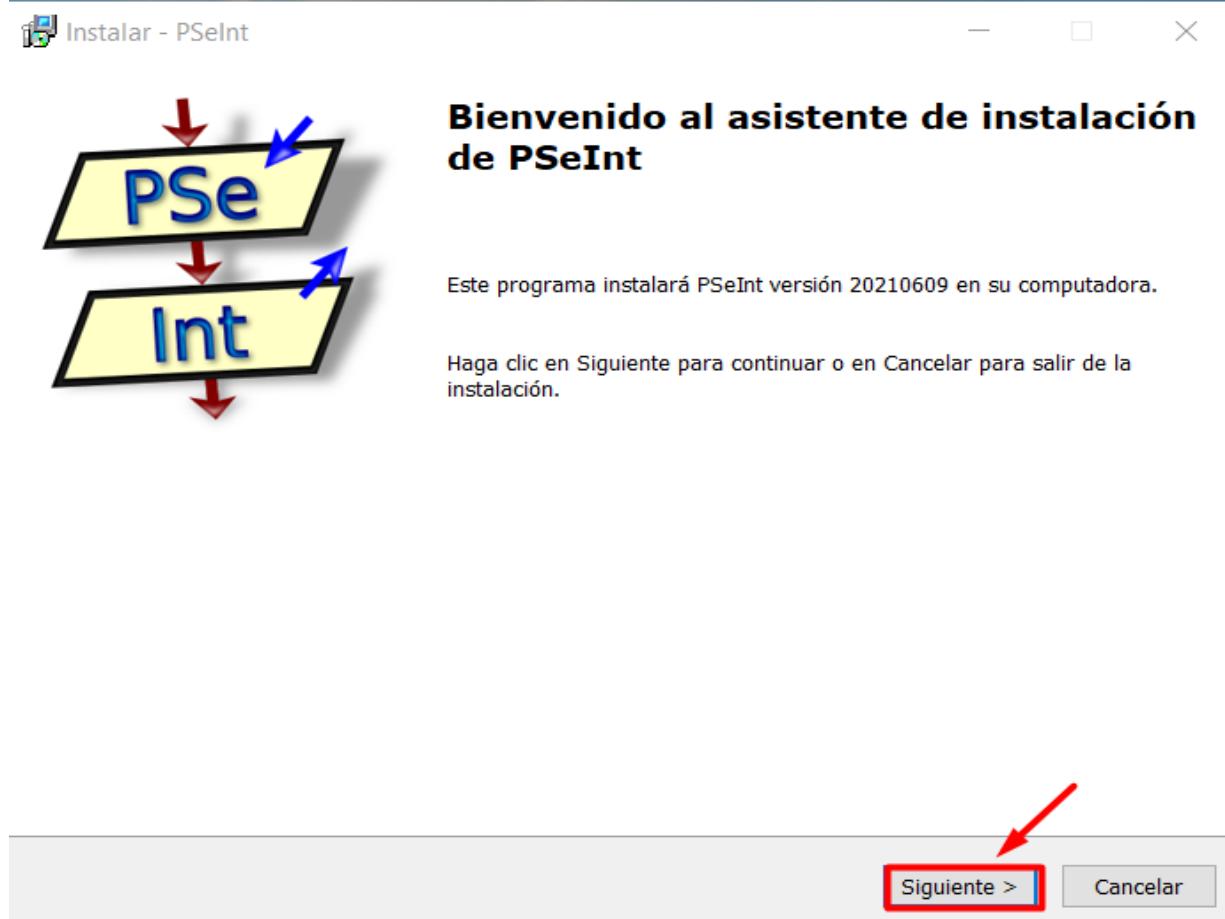
desde un punto arbitrario; ejecutar el algoritmo paso a paso controlando la velocidad e inspeccionando variables y expresiones; confeccionar automáticamente una tabla de prueba de escritorio; y un modo especial en el que describe las acciones realizadas en cada paso.

- Determina y marca claramente los errores señalando: errores de sintaxis en tiempo real (mientras escribe); errores en tiempo de ejecución; y ofrece descripciones detalladas de cada error, con sus causas y soluciones más frecuentes.
- Permite convertir el algoritmo de pseudocódigo a código en variados lenguajes de programación tales como: C, C++, C#, Java, JavaScript, MatLab, Pascal, PHP, Python 2, Python 3, QBasic Visual Basic

Instalación

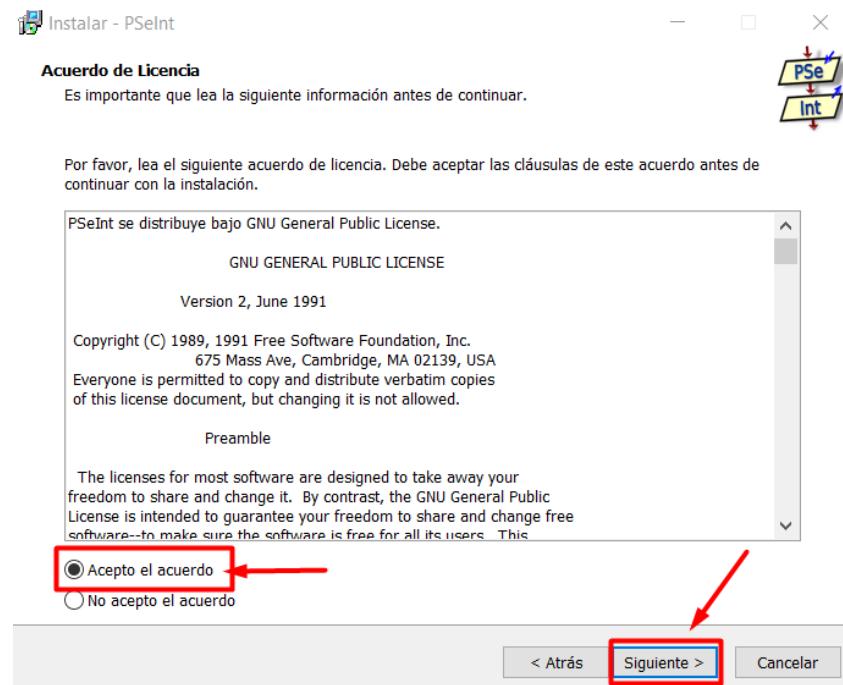
El software puede ser descargado desde <http://pseint.sourceforge.net/>

Luego de descargar y ejecutar el instalador, se abrirá la siguiente ventana:

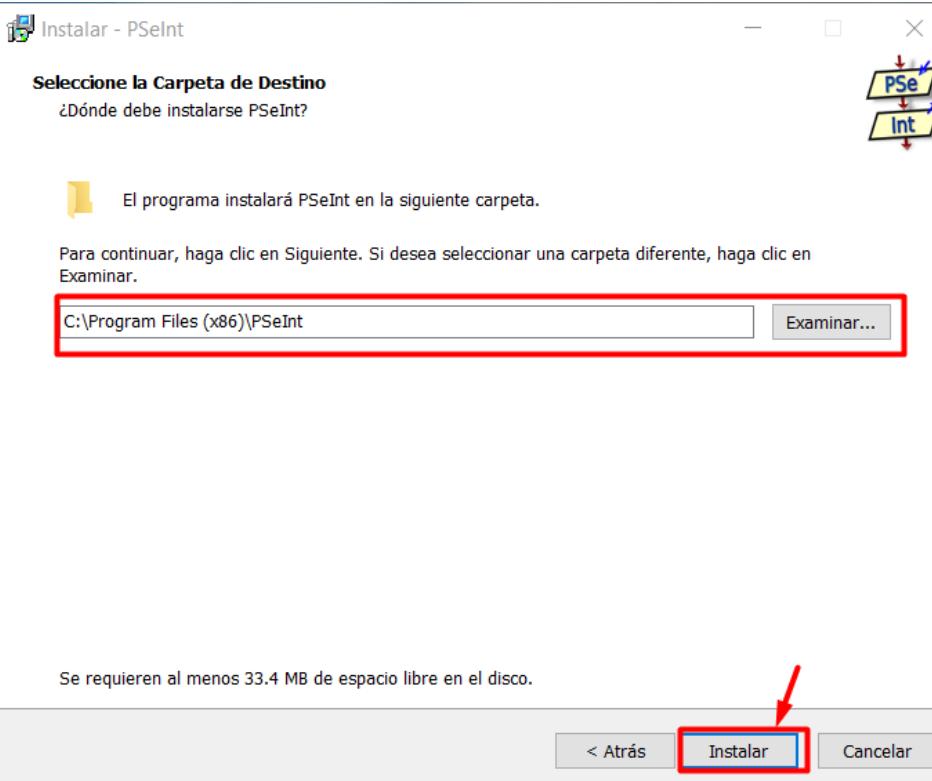




Aceptamos la licencia de contrato y damos click en siguiente



Indicamos la ruta de instalación y presionamos el botón siguiente

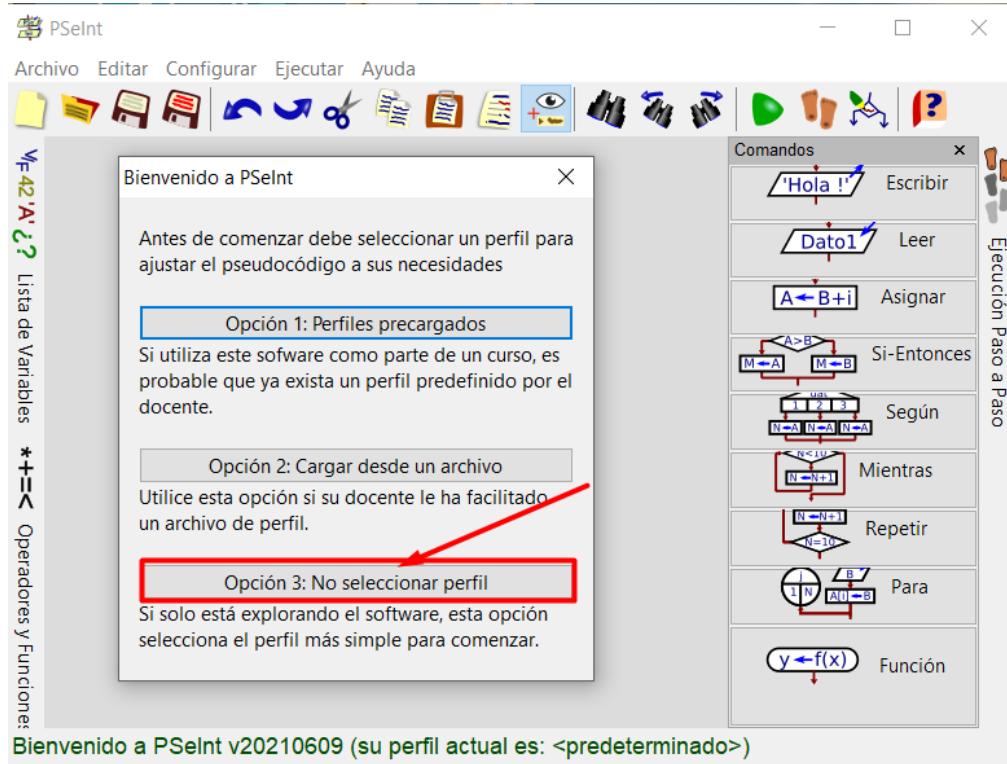




Completamos la instalación y presionamos el botón siguiente como se muestra en la imagen



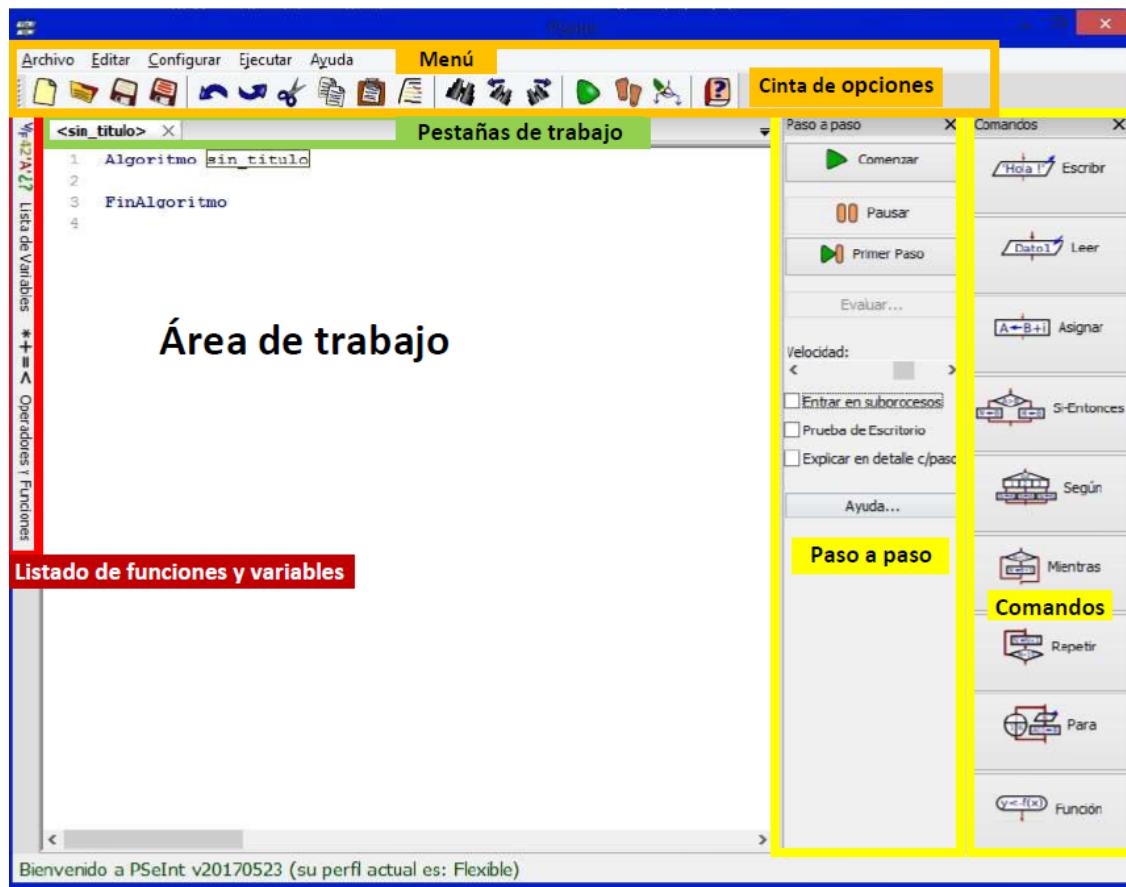
Al iniciar el programa nos solicitará si queremos seleccionar un perfil elegimos la opción 3



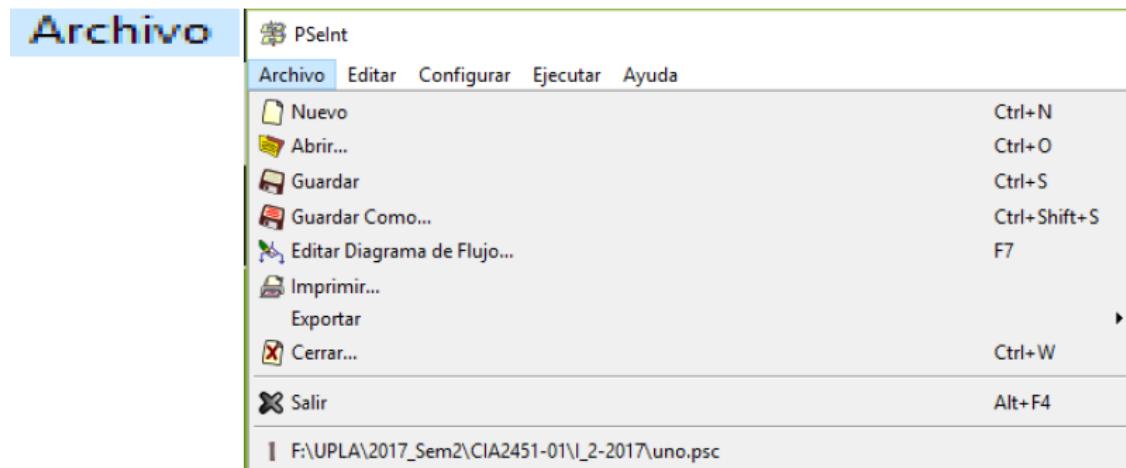


Entorno de PSeInt.

En la ventana de la aplicación se pueden identificar los siguientes elementos:



Contiene las opciones de Menú para acceder Menú a las siguientes acciones:





Editar

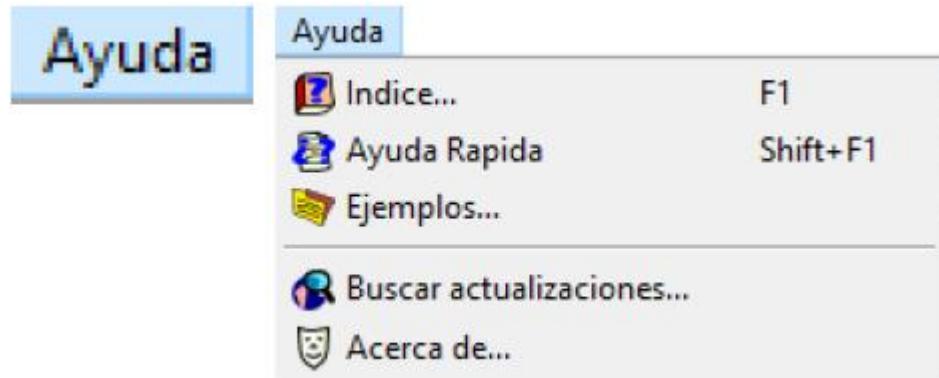
Editar	
	Deshacer Ctrl+Z
	Rehacer Ctrl+Shift+Z
	Cortar Ctrl+X
	Copiar Ctrl+C
	Pegar Ctrl+V
	Mover Hacia Arriba Ctrl+T
	Mover Hacia Abajo Ctrl+Shift+T
	Buscar... Ctrl+F
	Buscar Anterior Shift+F3
	Buscar Siguiente F3
	Reemplazar... Ctrl+R
	Seleccionar Todo Ctrl+A
	Duplicar Lineas Ctrl+L
	Eliminar Lineas Ctrl+Shift+L
	Comentar Lineas Ctrl+D
	Descomentar Lineas Ctrl+Shift+D
	Corregir Indentado Ctrl+I

Ejecutar

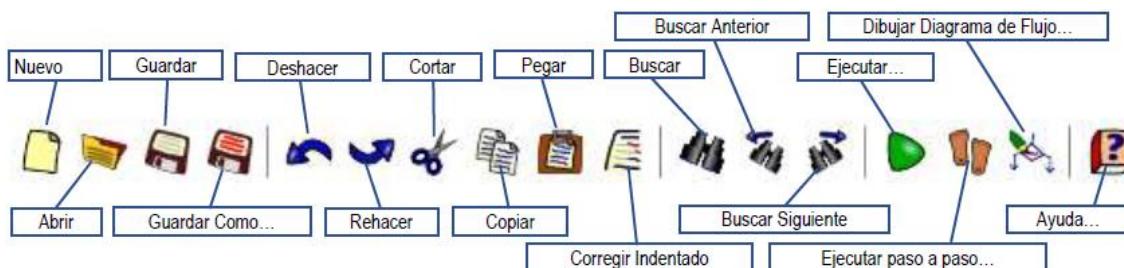
Ejecutar	
	Ejecutar F9
	Ejecutar Paso a Paso F5
	Ejecución Explicada
	Verificar Sintaxis Shift+F9
	Predefinir Entrada... Ctrl+F9

Configurar

Configurar	
	Asistencias ▶
	Presentación ▶
	Opciones del Lenguaje (perfiles)...
	Utilizar diagramas Nassi-Shneiderman



En tanto desde la **Cinta de opciones** se tiene acceso de forma directa a las siguientes acciones:

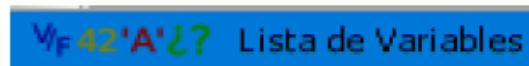


Área de trabajo: Corresponde a la zona dónde se escribe el **Pseudocódigo** del algoritmo. El **Pseudocódigo** se puede escribir directamente, o escogiendo el comando correspondiente desde la ventana de comandos.

A la izquierda de cada línea de pseudocódigo se muestra un número entero positivo que es el identificador de la línea, cuya referencia se utiliza cuando se indican los errores.

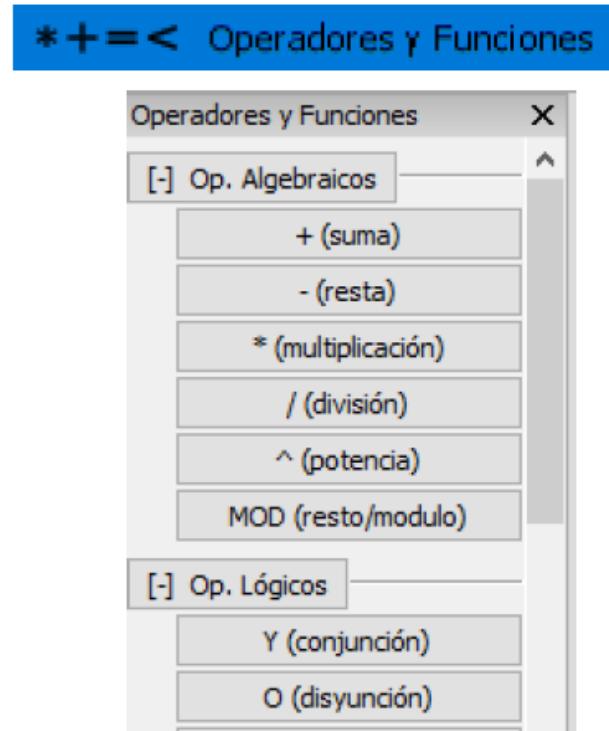
Pestañas de trabajo: Cada pseudocódigo nuevo o abierto se muestra en una pestaña diferente. Si se asigna un nombre al algoritmo, este se mostrará en la pestaña correspondiente, en caso <sin_título> contrario se muestra con el nombre **Listado de funciones y variables** Corresponde a la zona vertical que se muestra en el borde izquierdo del **Área de trabajo**. Esta zona está compuesta por dos pestañas que mostramos en posición horizontal en las siguientes imágenes:

Al activar la pestaña superior **Lista de Variables**, se abre una ventana que muestra un listado de las variables, según el orden en que aparecen en el **Pseudocódigo**.

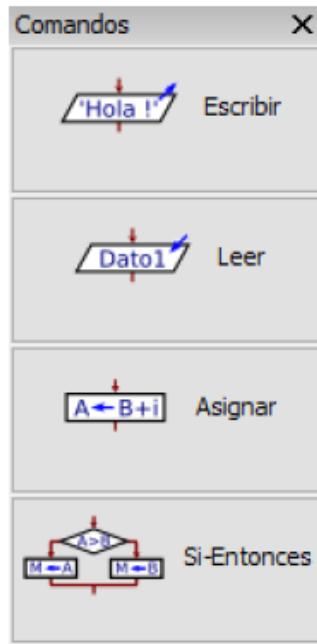




Al activar la pestaña de **Operadores y Funciones**, se abre una ventana que muestra la lista ordenada por categorías y dentro de cada categoría, , los operadores y funciones que se pueden utilizar como parte del pseudocódigo.

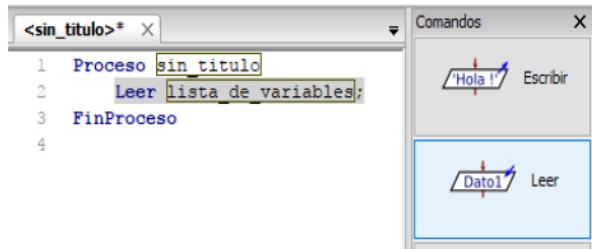


Comandos: En esta ventana están disponibles como botones, los distintos comandos de sentencias y estructuras de control, expresados en lenguaje de **Diagrama de flujo (DF)**.





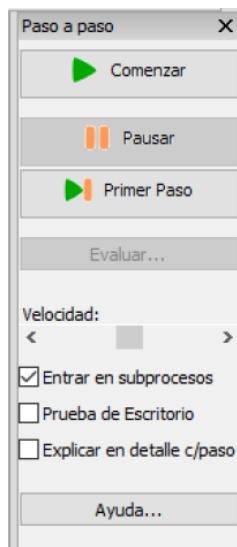
Al hacer doble clic sobre uno de los botones de la ventana de **Comandos**, se agrega en lenguaje de **Pseudocódigo**, la sentencia o estructura de control correspondiente, en la línea de edición que se encuentre activada **el área de trabajo**



Si la ventana de **Comandos** no está visible, puede hacer clic sobre la pestaña superior que se encuentra a la derecha del **Área de trabajo**



Paso a Paso: En esta venta se tiene acceso a las opciones para el control de la ejecución
Paso a paso.





Si la ventana **Paso a paso** no está visible, puede hacer clic sobre la pestaña superior que se encuentra a la derecha del **Área de trabajo**.



Modo de uso de los Comandos básicos para escribir instrucciones y estructuras de control.

Teniendo en consideración que el propósito fundamental de este manual, es ser un apoyo para el aprendizaje del diseño de algoritmos de los estudiantes de un primer curso de programación, sumado a ello la creencia de que el uso de esta aplicación les será de utilidad para el aprendizaje. A continuación, se muestran los aspectos generales de la escritura y significancia de los comandos de instrucciones y estructuras de control.



Este comando refiere a la sentencia para escribir. La acción de escribir puede hacerse respecto de un contenido textual, o una variable, o una expresión, o combinaciones de estos elementos.

Al hacer doble clic sobre este comando, en el área de trabajo se mostrará el equivalente en pseudocódigo de esta sentencia, la que corresponde a:

Escribir lista de expresiones; donde lista de expresiones; puede ser alguna de las siguientes:

- Un contenido textual, para lo cual el contenido debe ponerse entre comilla quedando como: **Escribir "Contenido Textual";**
- El valor de una variable y/o expresión como: **Escribir a,a+b;** En este ejemplo, se quiere escribir el valor de una cierta variable a y el resultado de sumar el valor de la misma variable a más otra cierta variable b. Notar que cada elemento a escribir se separa por una coma (,)



- En el siguiente ejemplo se quiere escribir un contenido textual y luego el valor de una cierta variable a **Escribir "El resultado es",a;**

Leer Este comando refiere a la sentencia para leer. Las acciones de lectura se realizarán para ingresar valores de variables (datos) durante la ejecución del algoritmo. Al hacer doble clic sobre este comando, en el área de trabajo se mostrará el equivalente en pseudocódigo de esta sentencia, la que corresponde a:
Leer lista de variables;

lista de variables puede contener una o más variables separadas por coma (,)

Leer a,b;

Asignar Mediante este comando se declara la acción de Asignar a una variable (la que se encuentra al lado izquierdo de \leftarrow) el contenido de otra variable o el resultado de una expresión (que se encuentra a lado derecho de \leftarrow)

Al hacer doble clic sobre este comando, en el área de trabajo se mostrará el equivalente en pseudocódigo de esta sentencia, la que corresponde a:

variable \leftarrow expresión;

Ejemplos de la forma que puede esta sentencia, son: **p \leftarrow b;** **c \leftarrow a+b;**



Este comando permite incorporar en el algoritmo a una estructura de control mediante la cual, se define que sentencia o conjunto de sentencias se ejecutarán si una cierta expresión_lógica es VERDADERA y que sentencias o conjunto de ellas se ejecutarán si la misma expresión_lógica es FALSA.

Al hacer doble clic sobre este comando, en el área de trabajo se mostrará el equivalente en pseudocódigo de esta estructura de control, la que corresponde a:

```
Si expresion_logica Entonces
    acciones_por_verdadero
    SiNo
        acciones_por_falso
    Fin Si
```



Un ejemplo de uso de esta estructura podría ser:

```
Si a>=b Entonces
    Escribir "a es mayor o igual que b";
SiNo
    Escribir "b es mayor que a";
Fin Si
```

De ser necesario, esta estructura también puede ser usada en su versión simplificada como:

```
Si a>b Entonces
    Escribir "a es mayor que b";
Fin Si
```

para lo cual se deben eliminar las líneas correspondientes al **SiNo** y las siguientes, hasta la anterior al **Fin Si**

En **expresión lógica** la pueden usarse operadores lógicos tales como En & ó Y (Conjunción), | ó O (Disyunción), ~ ó NO (Negación).

Ejemplo:

```
Si a>b O a=b Entonces
    Escribir "a es mayor o igual que b";
SiNo
    Escribir "b es mayor que a";
Fin Si
```



Este comando incorpora en el algoritmo, la estructura de control Mientras para repetir una sentencia o conjuntos de sentencias.

Al hacer doble clic sobre el comando, a partir de la línea correspondiente del pseudocódigo se agregará la estructura:

```
Mientras expresión_lógica Hacer
    secuencia_de_acciones
Fin Mientras
```

Al igual que en el caso de la estructura de control Si-Entonces anterior, la **expresión_lógica** puede contener una comparación simple de la forma a<7, o usar los operadores lógicos & ó Y, | ó O, ~ ó NO.



Ejemplo:

```
Mientras i<5 O i=5 Hacer
    i<-i+1;
    Escribir "Sigo sumando 1 a i";
Fin Mientras
```



Repetir

Este comando incorpora en el algoritmo, la estructura de control Repetir la que corresponde a otra estructura para repetir una sentencia o conjuntos de sentencias.

Al hacer doble clic sobre el comando, a partir de la línea correspondiente del pseudocódigo se agregará la estructura:

```
Repetir
    secuencia de acciones
Hasta Que expresion logica
```

Al igual que en los casos `expresion logica` anteriores, la puede utilizar una expresión lógica simple u operadores lógicos



Para

Este es el comando para la tercera forma de incorpora una estructura para repetir una sentencia o conjuntos de sentencias, dentro de un algoritmo.

Al hacer doble clic sobre el comando, a partir de la línea correspondiente del pseudocódigo se agregará la estructura:

```
Hasta Que expresion logica
Para variable numerica<-valor inicial Hasta valor_final Con Paso paso Hacer
    secuencia de acciones
Fin Para
```



IDENTIFICADORES

Un identificador es un nombre que define el programador, que sirve para denotar ciertos elementos de un programa. Estos elementos pueden ser las denominadas variables, constantes y funciones. Cuando se ejecuta el programa, el sistema relaciona estos nombres con alguna dirección de memoria. De este modo, a la hora de programar, ya no se requiere recordar posiciones de memoria sino los nombres dados a estas posiciones de memoria.

Para asignar nombres válidos en un algoritmo a los elementos mencionados la diapositiva anterior, existen una serie de reglas que facilitan su escritura. Es importante mencionar también que, existen reglas y recomendaciones propias, dependiendo del lenguaje de programación en el que se vaya a codificar el algoritmo; de esta manera, la forma de asignar identificadores puede ser ligeramente diferente de un lenguaje a otro.

RECOMENDACIONES PARA ESCRIBIR IDENTIFICADORES

- 1.Definir identificadores nemotécnicos, es decir, alusivos o relacionados con la función del elemento que se está nombrando.
- 2.El primer carácter del identificador debe ser una letra.
- 3.No utilizar caracteres especiales dentro de los identificadores como vocales tildadas, la letra ñ, o símbolos como: \$, #, !, ?, entre otros.
- 4.No se deben dejar espacios en blanco dentro del nombre de un identificador.
- 5.No utilizar palabras propias del lenguaje algorítmico o de programación que se está utilizando (Palabras reservadas)
- 6.En un identificador se pueden utilizar varias palabras, preferiblemente unidas. También se puede usar un guion bajo entre cada una de ellas.
- 7.Evite el uso de artículos y proposiciones, tales como: el, los, la, un, unos, a, para, de, entre otros.
- 8.Los identificadores suelen tener reglas dependiendo del lenguaje, en general, se escriben en minúscula. Cuando el identificador se componga de dos o más palabras, la primera letra a partir de la segunda deberá escribirse en mayúsculas.
- 9.El identificador para el nombre del algoritmo, comienza en mayúscula.
- 10.Si el identificador corresponde al nombre de una constante, su primera letra debe escribirse en mayúscula.



Variable1	variable2	dato#5
dospalabras	Dospalabras	dos_palabras
constante1	Constante2	3cuadrados
variable 3	variable?	dato#5
los_datos	cabaña 2	variable_5

VARIABLES

Para almacenar los datos en un dispositivo de memoria en el cual se almacena un dato. Su valor puede cambiar en cualquier momento de la ejecución del algoritmo, precisamente por eso recibe el nombre de variable.

Para almacenar los datos en un dispositivo de procesamiento de datos o computador, se utiliza la memoria de este, la cual se puede comparar con un conjunto de cuadritos que guardan valores.

Cada “cuadrito” o celda representa una dirección física dentro de la memoria de la máquina a la cual se le puede asignar un nombre mediante un identificador.

DECLARACIÓN DE VARIABLES

Cuando en un algoritmo se requiera utilizar una variable, esta debe ser declarada. Declarar una variable quiere decir que se va a reservar un espacio de memoria, el cual tendrá un nombre y un tipo de dato.

La forma general para declarar variables es la siguiente:

tipo variable

Ejemplo:

entero numeroManzanas
real temperatura
booleano estado_interruptor
cadena nombre_persona
caracter primeraLetraPersona



ALMACENAMIENTO DE UN DATO EN UNA VARIABLE

Cuando nos referimos a variables nos referimos a lugares de memoria en los cuales se almacena algún tipo de información, por ejemplo el número de gallinas, la altura, la edad, el nombre y el peso. Existen diferentes tipos de datos tal y como se muestra en la siguiente tabla:

Tipo de dato	Descripción	Ejemplo
entero	Tipo de dato asociado a cantidades enteras. No poseen parte decimal. Ejemplo: 5, 6, -15, 199,...	Numero de vacas, edad.
real	Tipo de dato asociado a cantidades con parte decimal. Por ejemplo: 0.06, -3.4, 2.16, 1000.345,...	Estatura, peso, volumen.
lógicos	Se refiere a aquellos datos que pueden tomar solo dos posibles valores falso (F) o verdadero (T)	
alfanuméricos	Asociado a aquellos datos que contienen caracteres alfanuméricos (letras, número, signos de puntuación, etc).	Nombre, cedula, telefono

Cuando se trabaja con variables un aspecto de vital importancia (además del tipo) es el nombre que estas van a tener, se recomiendan nombres relacionados con la información que van a manejar, por ejemplo, si se va a manejar un **salario**, un nombre apropiado para una variable puede ser **sal**.

Instrucción de asignación: Escribe sobre una variable el valor de una expresión. Así:

variable = expresión

Donde, una expresión es una combinación de valores, variables y operadores, los siguientes son algunos ejemplos de expresiones:

a = 5

b = c * d + (c - f) * m

z = (x + y) / (w + s)

s = (a/b)^3



CONSTANTES

Su valor no podrá ser cambiado durante la ejecución del algoritmo.

constante Tipo IDENTIFICADOR = valor

Ejemplo:

constante:

ConstanteReal VALOR_PI = 3.1415926

ConstanteReal DESCUENTO = 0.10

ConstanteEntero MAXIMO = 10

ConstanteReal SALARIOMINIMO = 781000.0

ConstanteCaracter CATEGORIAPORDEFECTO = 'B'

OPERADORES Y EXPRESIONES

Un operador es un símbolo que permite realizar una operación con números o con datos que se encuentran almacenados en las variables y constantes. En lógica de programación, existen 3 tipos de operadores:

Aritméticos, Relacionales y Lógicos.

Una expresión es una instrucción que puede estar compuesta por operadores, variables, constantes y números, que generalmente produce un resultado, ya sea numérico o lógico. Las expresiones para ser usadas dentro de un algoritmo, deben escribirse en notación algorítmica (en una sola línea), para ello se seguirá usando la siguiente forma general:

Operando1 Operador Operando2

Operando1 y Operando2 representan los datos que intervienen en la expresión y, Operador es un símbolo que indica el tipo de operación que se va a realiza entre los operandos.

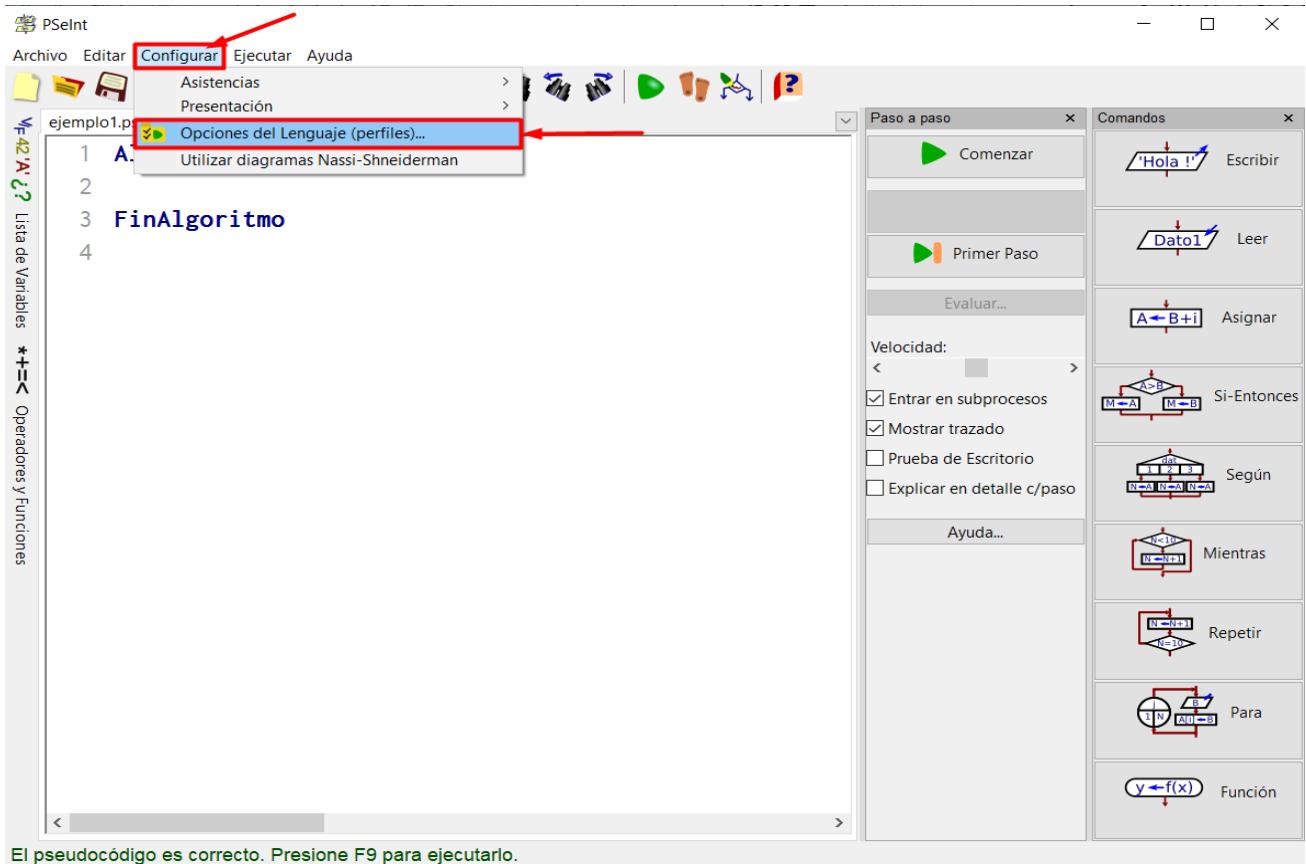
Codificación en PSEINT

Ya se tiene el pseudocódigo del programa codificado en nuestro pseudocódigo por convención sin embargo la idea es probarlo para ver cómo funciona. Existe una herramienta llamada PSeint el cual es un pseudo interprete, es decir entiende Pseudocódigo, sin embargo, en el momento de codificar el anterior programa en **PSeInt** debemos tener en cuenta que el pseudocódigo manejado en le **PSeInt** es un poco diferente, la siguiente tabla muestra esto en detalle:



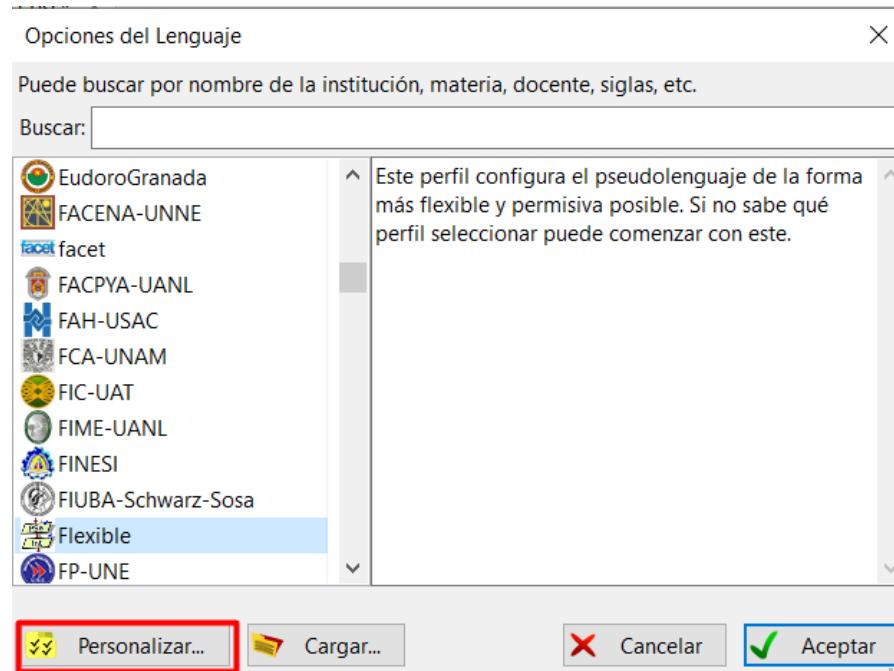
Instrucción	Pseudocódigo propio	Pseudocódigo PSeint	Observaciones
Asignación	$c = 2*a*(b + c)$	$c <- 2*a*(b + c);$	La asignación en PSeint no es con igual (=) sino con flecha (<-) y al final va punto y coma (;).
Entrada	LEA(a,b,c)	Leer a,b,c;	La instrucción de entrada en PSeint se llama <i>Leer</i> no hace uso de paréntesis y termina con signo de punto y coma (;).
Salida	ESCRIBA("Hola ",nombre)	Escribir "Hola",nombre;	La instrucción de salida en PSeint se llama <i>Escribir</i> no hace uso de paréntesis y termina con signo de punto y coma (;). Al igual que en el ESCRIBA usado por convención la parte del mensaje que no cambia (que es constante) va entre comillas (""), y la parte variable va sin comillas ("").

Antes de empezar el programa vamos haciendo algunos ajustes para poder realizar y utilizar sin ningún problema todas las bondades que nos ofrece PSeint.

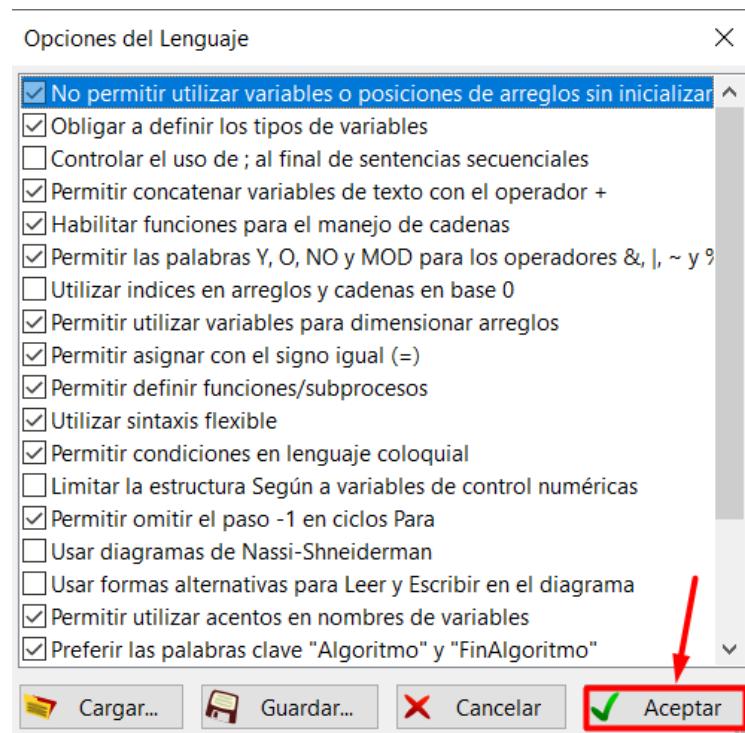




Luego de haber presionado en la opción seleccionada nos aparece la siguiente ventana



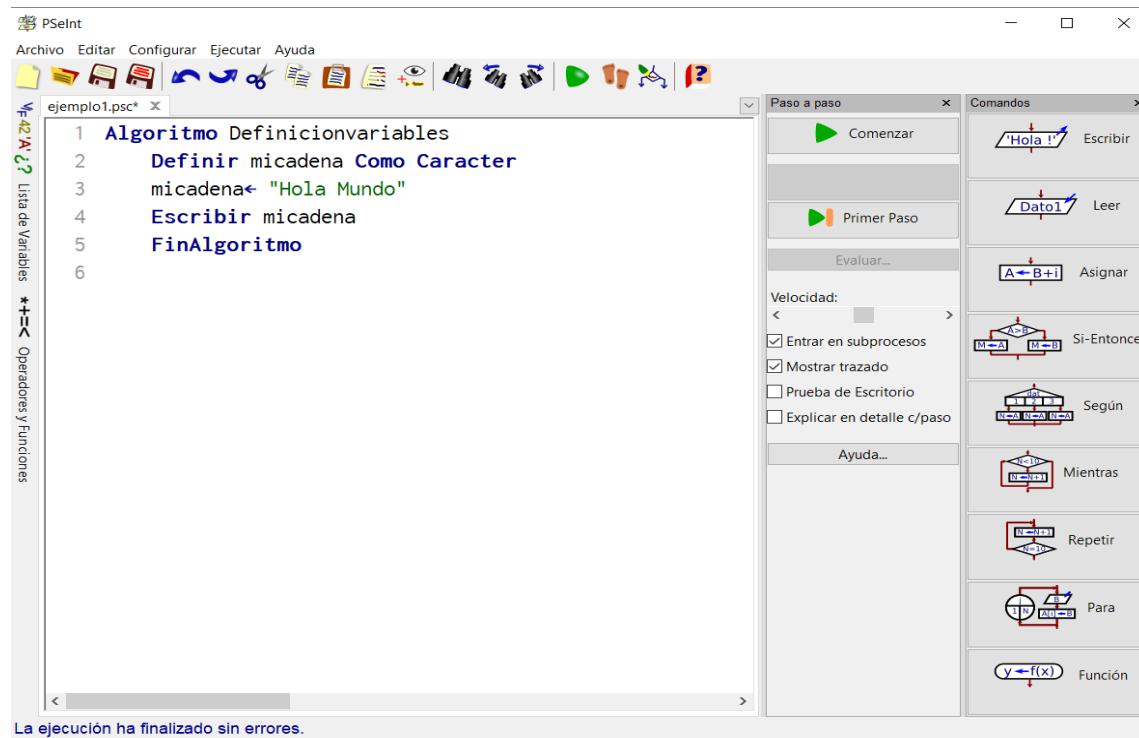
Seleccionamos en personalizar y seleccionamos las opciones y damos click en aceptar como se muestra en la imagen a continuacion y ahora si a empezar a trabajar:



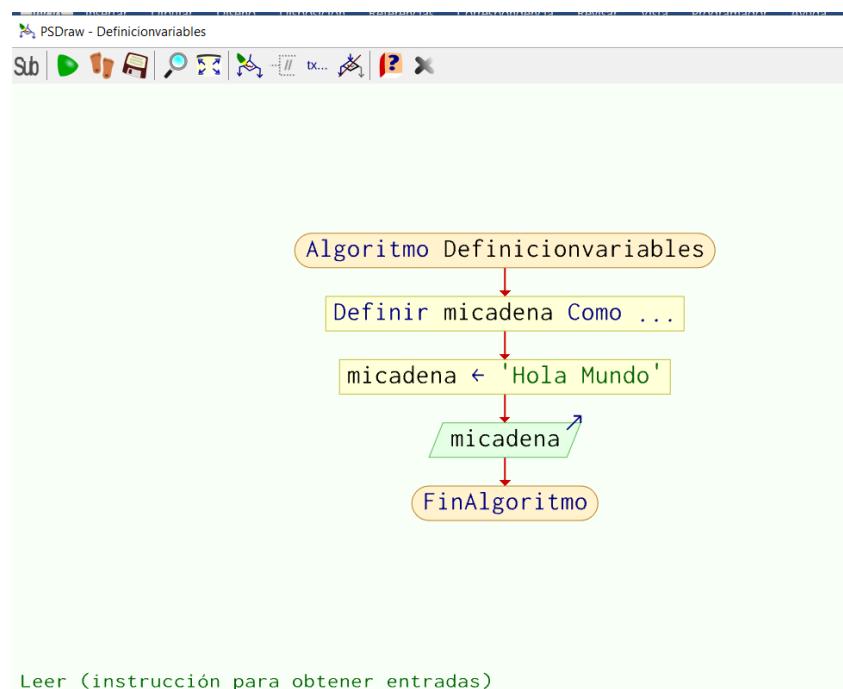


Ejemplo 01

Imprimir en pantalla un “Hola Mundo” siguiendo los pasos y orden vistos en Pselnt



Ahora Nuestro algoritmo quedaria de esta manera si lo trasladamos a diagrama de flujo





Ejemplo 02

Hallar la suma de tres números.

Archivo Editar Configurar Ejecutar Ayuda



<sin_titulo>* X

1 //Hallar la suma de tres números.
2 **Proceso Ejer1**
3 **Definir** n1, n2, n3, suma **Como Real**
4 **Escribir** "Escriba tres numeros"
5 **Leer** n1, n2, n3
6 Suma \leftarrow n1+n2+n3
7 **Escribir** "La suma de tres numeros es:", suma
8 **FinProceso**
9

PSelnt - Ejecutando proceso EJER1
*** Ejecución Iniciada. ***
Escriba tres numeros
> 5
> 7
> 4
La suma de tres numeros es:16
*** Ejecución Finalizada. ***

Ejemplo 03

Hallar el área de un cuadrado.

Archivo Editar Configurar Ejecutar Ayuda



<sin_titulo>* <sin_titulo>* X

1 //Hallar el área de un cuadrado.
2 **Proceso Ejer2**
3 **Definir** n1, potenciacion **Como Real**
4 **Escribir** "Ingrese el numero del lado"
5 **Leer** n1
6 Potenciacion \leftarrow n1 2
7 **Escribir** "El area de un cuadrado es:", Potenciacion
8 **FinProceso**
9

PSelnt - Ejecutando proceso EJER2
*** Ejecución Iniciada. ***
Ingrese el numero del lado
> 5
El area de un cuadrado es:25
*** Ejecución Finalizada. ***



Ejemplo 04

Hallar el área del triángulo.

The screenshot shows the PSeInt Integrated Development Environment. The menu bar includes Archivo, Editar, Configurar, Ejecutar, and Ayuda. The toolbar contains various icons for file operations like new, open, save, print, and cut/paste. A status bar at the bottom right shows the message "PSeInt - Ejecutando proceso EJER". The code editor window displays the following pseudocode:

```
1 //Hallar el área del triángulo.
2 Proceso Ejer
3     Definir n1, n2, multiplicacion, division Como Real
4     Escribir "Ingrese la base y altura"
5     Leer n1, n2
6     Multiplicacion_division=n1*n2/2
7     Escribir "El area de un triangulo es:", Multiplicacion_division
8 FinProceso
9
```

To the right of the code, the output window shows the execution process:

- PSeInt - Ejecutando proceso EJER
- *** Ejecución Iniciada. ***
- Ingrese la base y altura
- > 12
- > 3
- El area de un triangulo es:18
- *** Ejecución Finalizada. ***

EJERCICOS PROPUESTOS

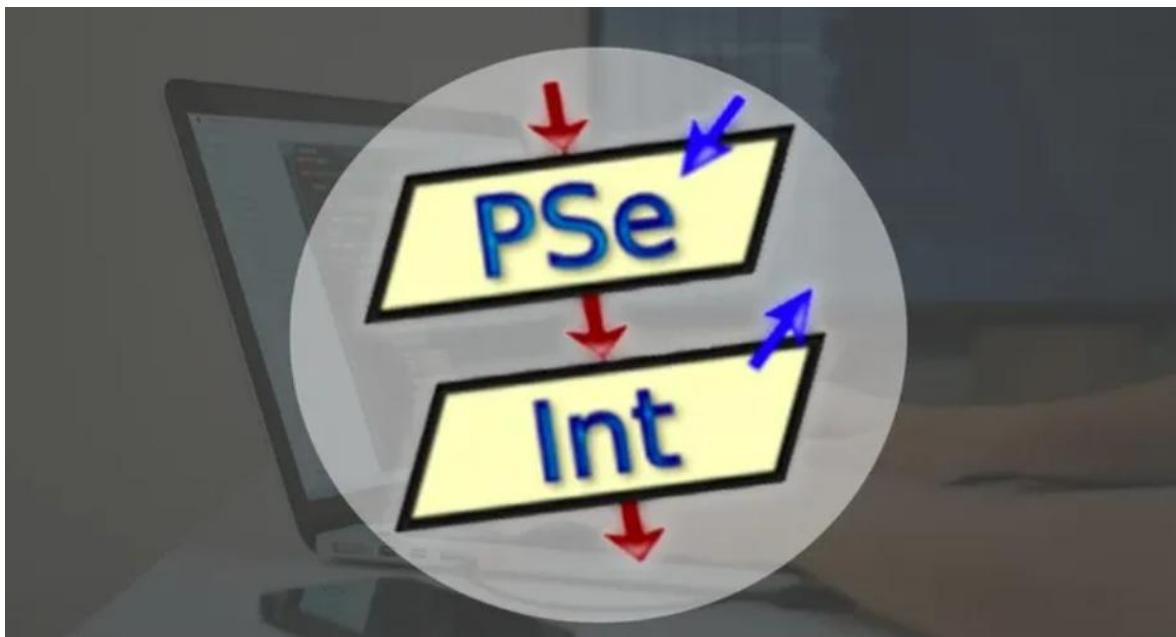
1. Calcular el 25% de descuento del precio de un artefacto.
2. Cuántos meses debe ahorrar para comprar un celular del precio de S/. 500 si recibe mensualmente x soles.
3. Hallar el área de un círculo.
4. Se debe permitir ingresar dos números, luego mostrar la suma y el producto de ambos.
5. Se debe permitir ingresar un número entero, luego mostrar el 20% de este número.
6. Se debe permitir ingresar un valor en metros, luego mostrar su valor en centímetros y en milímetros.



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA

Lógica de Programación



SESION 10

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I2: Diagrama clases de objetos aplicando los elementos de las técnicas de programación orientadas a objetos.



Practica trabajando con variables

Ejemplos:

1. Ingresando la base y altura de un triángulo, luego mostrar su área.

The screenshot shows the PSeint IDE interface. The menu bar includes Archivo, Editar, Configurar, Ejecutar, and Ayuda. The toolbar contains various icons for file operations like New, Open, Save, Print, and Help. On the left, there's a vertical toolbar with icons for file operations and a 'Lista de Variables' button. The code editor window has tabs labeled <sin_titulo>*. The main code is:

```
1 //Ingresando la base y altura de un triángulo, luego mostrar su área.
2 Proceso Ejer4
3     Definir b, h, area Como Real
4     Escribir "Ingrese la base y altura:"
5     Leer b, h
6     area←(b*h)/2
7     Escribir "El area de un triangulo es: ", area
8
9 FinProceso
10
```

To the right, the output window shows the execution results:

```
PSeint - Ejecutando proceso EJER4
*** Ejecución Iniciada. ***
Ingrese la base y altura:
> 3
> 7.5
El area de un triangulo es: 11.25
*** Ejecución Finalizada. ***
```

2. Ingresar un valor en dólares, luego mostrar su equivalente en soles.

The screenshot shows the PSeint IDE interface. The menu bar includes Archivo, Editar, Configurar, Ejecutar, and Ayuda. The toolbar contains various icons for file operations like New, Open, Save, Print, and Help. On the left, there's a vertical toolbar with icons for file operations and a 'Lista de Variables' button. The code editor window has tabs labeled <sin_titulo>*. The main code is:

```
1 //Ingresar un valor en dólares, luego mostrar su equivalente en soles.
2 Proceso Ejer5
3     Definir n1, soles Como Real
4     Escribir "Ingrese el valor en dolares:"
5     Leer n1
6     soles←n1*3.38
7     Escribir "El valor en soles es: ", soles
8 FinProceso
9
```

To the right, the output window shows the execution results:

```
PSeint - Ejecutando proceso EJER5
*** Ejecución Iniciada. ***
Ingrese el valor en dolares:
> 5
El valor en soles es: 16.9
*** Ejecución Finalizada. ***
```



3. Leer una medida en pulgadas e imprimir su equivalente en milímetros.

PSelnt

Archivo Editar Configurar Ejecutar Ayuda

Lista de Variables Operadores

```
<sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* X
```

1 //Leer una medida en pulgadas e imprimir su equivalente en milímetros.
2 Proceso Ejer6
3 Definir p, mm Como Real
4 Escribir "Ingrese la medida en pulgadas:"
5 Leer p
6 mm=p*25.4
7 Escribir "El equivalente en mm es: ", mm
8 FinProceso
9

PSelnt - Ejecutando proceso EJER6
*** Ejecución Iniciada. ***
Ingrese la medida en pulgadas:
> 5.7
El equivalente en mm es: 144.78
*** Ejecución Finalizada. ***

4. Ingresar un número de 3 cifras y luego mostrar la suma de sus cifras elevada al cuadrado.

PSelnt

Archivo Editar Configurar Ejecutar Ayuda

Lista de Variables Operadores

```
<sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* X
```

1 //Ingresar un número de 3 cifras y luego mostrar la suma de sus cifras elevada al cuadrado.
2 Proceso Ejer7
3 Definir n1, n2, n3, suma, pot Como Real
4 Escribir "Ingrese las tres cifras del numero:"
5 Leer n1, n2, n3
6 suma=n1+n2+n3
7 pot=suma^2
8 Escribir "La suma de las cifras elevada al cuadrado es: ", pot
9
10 FinProceso
11

PSelnt - Ejecutando proceso EJER7
*** Ejecución Iniciada. ***
Ingrese las tres cifras del numero:
> 4
> 5
> 1
La suma de las cifras elevada al cuadrado es: 100
*** Ejecución Finalizada. ***



EJERCICOS PROPUESTOS

1. Se debe ingresar un monto de dinero, luego mostrar cuanto le toca a cada socio según la siguiente tabla:

Socio	Porcentaje
A	30%
B	20%
C	50%

2. Desarrollar un algoritmo que pida el ingreso del precio base de un producto y se debe calcular el IGV (19% del precio base), luego mostrar el valor del IGV y el precio final del producto (precio base más el IGV).
3. Una empresa realiza anualmente un evento para fines solidarios con las comunidades nativas del Perú, para ello se requiere un algoritmo que permita ingresar la cantidad de entradas vendidas de los tres tipos detallados en el siguiente tarifario:

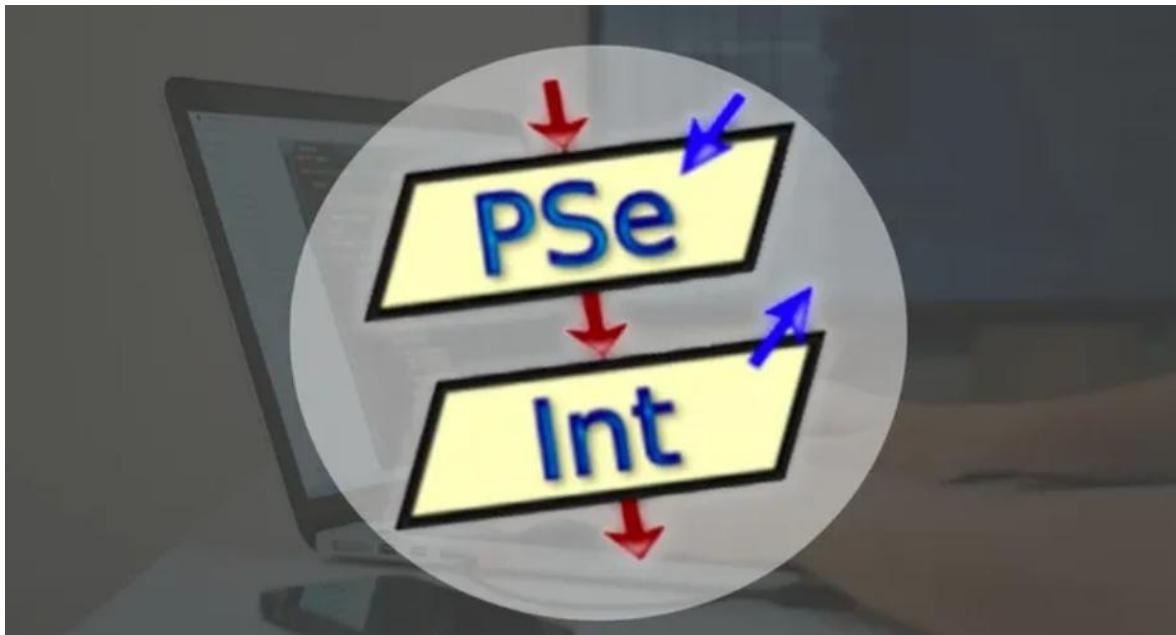
Concepto	Precio
Entrada general	S/ 150.00
Entrada para mayores de 65	S/ 50.00
Entrada para menores de edad	S/ 80.00

Luego mostrar la cantidad de dinero recaudado por cada tipo de tarifa y el monto recaudado en total.



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA Lógica de Programación



SESION 11

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I2: Diagrama clases de objetos aplicando los elementos de las técnicas de programación orientadas a objetos.



EJERCICOS PROPUESTOS

1. Algoritmo que pida el nombre y edad de una persona, si es mayor de edad muestra un mensaje diciendo "Usted es mayor de edad" de lo contrario uno que diga "Usted es menor de edad". Tomar en cuenta que un mayor de edad cuando tienen una edad mayor o igual a 18 años.

2. Algoritmo que me permita ingresar un número, si el número es par me muestra un mensaje diciéndome que es un numero par, de lo contrario uno que me diga que el número es impar.

3. En una llantería se ha establecido una promoción que consiste en lo siguiente:
 - Si se compran menos de cinco llantas el precio es de S/. 300 cada una, de S/. 250 si se compran de cinco a 10 y de S/. 200 si se compran más de 10.
 - Obtener la cantidad de dinero que una persona tiene que pagar por cada una de las llantas que compra y la que tiene que pagar por el total de la compra

4. Una frutería ofrece las manzanas con descuento según la siguiente tabla:

NUM. DE KILOS COMPRADOS	% DESCUENTO
0 – 2	0%
2 – 5	10%
6 – 9	15%
10 en adelante	20%

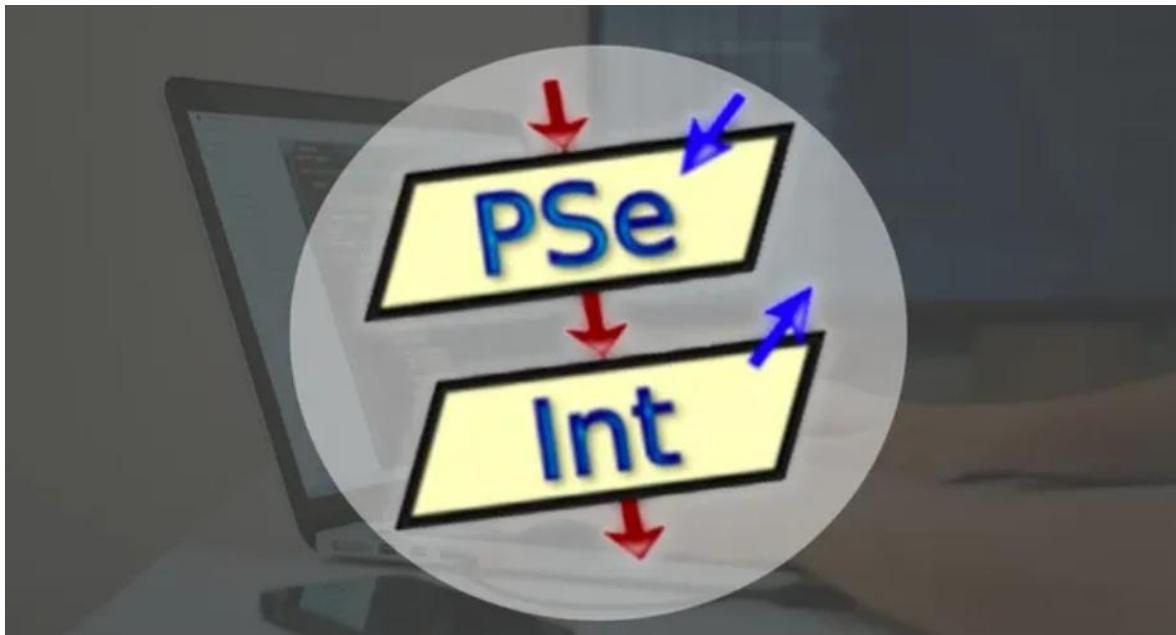
Determinar cuánto pagara una persona que compre manzanas en esa frutería.

5. El promedio de prácticas de un curso se calcula con base en cuatro prácticas calificadas de las cuales se elimina la nota menor y se promedian las tres notas más altas. Diseñe un algoritmo que determine la nota eliminada y el promedio de prácticas de un estudiante.



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA Lógica de Programación



SESION 12

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I2: Diagrama clases de objetos aplicando los elementos de las técnicas de programación orientadas a objetos.



OPERADORES LÓGICOS: CONJUNCIÓN, NEGACIÓN, DISYUNCIÓN

Conjunción:

Una conjunción lógica (comúnmente simbolizada como Y o \wedge) es, en lógica y matemáticas, un operador lógico que resulta en verdadero si los dos operadores son verdaderos.

En lógica y matemáticas una conjunción es un «enunciado con dos o más elementos simultáneos»

Una lámpara eléctrica se enciende si hay corriente eléctrica, el interruptor está conectado, el fusible está bien y la lámpara no está fundida, en cualquier otro caso la lámpara no se encenderá.

Disyunción:

Una disyunción lógica, comúnmente conocida como O, es un operador lógico que resulta verdadero si cualquiera de los operadores es también verídico.

Ejemplo 01

Escribir un programa que indique cuál es el mayor de cuatro números enteros

```
<sin_titulo>* x |
1 Algoritmo sin_titulo
2   Definir a,b,c,d Como Entero
3   Escribir "Programa para indicar el mayor de los numeros"
4   Escribir "Ingrese los numeros"
5   Leer a,b,c,d
6   si a>b y a>c y a>d Entonces
7     Escribir "Mayor: " , a
8   FinSi
9   si b>a y b>c y b>d Entonces
10    Escribir "Mayor: " , b
11  FinSi
12  si c>a y c>b y c>d Entonces
13    Escribir "Mayor: " , c
14  FinSi
15  si d>a y d>b y d>c Entonces
16    Escribir "Mayor: " , d
17  FinSi
18 FinAlgoritmo
--
```



Ejemplo 02

Crearemos un algoritmo que pida ingresar usuario y contraseña, donde el usuario será ERICO y la contraseña será 1705, de forma que si alguno de ellos no coincide nos debe mostrar el mensaje de Acceso Negado, pero si las credenciales fueran válidas nos debe mostrar Acceso concedido.

```
<sin_titulo>* <sin_titulo>
1 Algoritmo sin_titulo
2 Definir usu, clave Como Caracter
3 Escribir "Ingrese el usuario"
4 Leer usu
5 Escribir "Ingrese la clave"
6 Leer clave
7 si usu="ERICO" y clave="1705" Entonces
8   Escribir "Acceso concedido"
9 sino
10  Escribir "Acceso negado"
11 FinSi
12 FinAlgoritmo
13
```

Ejemplo 03

Dado un monto calcular el descuento: 25% si el monto es mayor o igual que 300. 20% si el monto es mayor o igual que 150 y menor que 300. no hay descuento si el monto es menor que 150.

```
<sin_titulo>* <sin_titulo>
1 Algoritmo sin_titulo
2 definir monto, desc Como Real
3 Escribir "Ingrese monto"
4 leer monto
5 si monto ≥ 300 Entonces
6   desc = monto*0.25
7   escribir "El descuento es: ", desc
8 SiNo
9   si monto ≥ 150 y monto < 300 Entonces
10    desc=monto*0.20
11    escribir "El descuento es: " desc
12  SiNo
13    escribir "No hay descuento"
14  FinSi
15 FinSi
16 FinAlgoritmo
```



EJERCICOS PROPUESTOS

1. Programa que indique el promedio de tres notas y además le diga si ha reprobado o no. Sabiendo que para aprobar la nota debe ser mayor o igual a 13

2. Diseñe un algoritmo que lea tres números y los imprima de mayor a menor y de menor a mayor.

3. Verificar que el número ingresado por teclado se encuentra dentro de la primera docena de números naturales, es decir entre el 1 y el 12.

4. Realizar un algoritmo que permita al usuario ingresar su fecha de nacimiento y que le diga si es mayor o menor de edad.

5. De un operario se conoce su sueldo y los años de antigüedad. Se pide confeccionar un programa que lea los datos de entrada e informe:
 - a) Si el sueldo es inferior a 500 y su antigüedad es igual o superior a 10 años, otorgarle un aumento del 20 %, mostrar el sueldo a pagar.

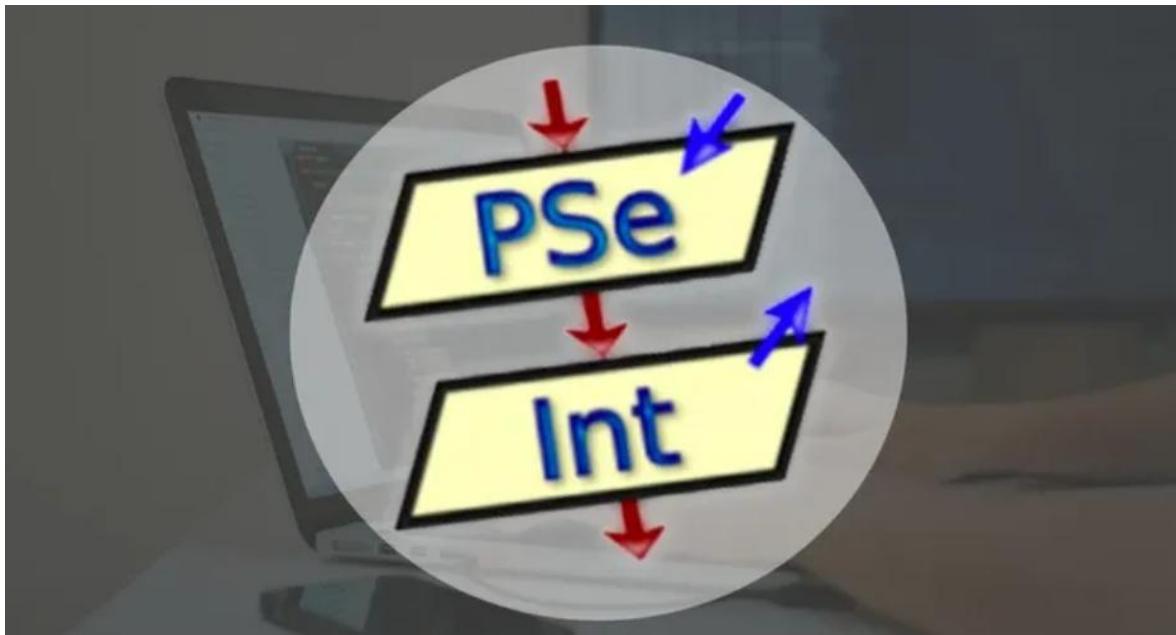
 - b) Si el sueldo es inferior a 500 pero su antigüedad es menor a 10 años, otorgarle un aumento de 5 %.

 - c) Si el sueldo es mayor o igual a 500 mostrar el sueldo en pantalla sin cambios.



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA Lógica de Programación



SESION 13 Y 14

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I2: Diagrama clases de objetos aplicando los elementos de las técnicas de programación orientadas a objetos.



ESTRUCTURAS REPETITIVAS

Las estructuras repetitivas son aquellas que permiten ejecutar repetidamente un conjunto de instrucciones, bien un número predeterminado de veces, o bien hasta que se verifique una determinada condición.

En términos de un lenguaje de programación, que se verifique o no una condición se traduce en que una (adecuada) expresión lógica tome el valor VERDADERO (TRUE) o tome el valor FALSO (FALSE). En los casos más sencillos y habituales la condición suele ser una comparación entre dos datos, como por ejemplo:

si $a < b$ hacer una cosa y en caso contrario hacer otra distinta.

Estructura Mientras:

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

Mientras <condición> Hacer

<instrucciones>

FinMientras

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

Note que las instrucciones del cuerpo del ciclo pueden no ejecutarse nunca, si al evaluar por primera vez la condición resulta ser falsa.

Si la condición siempre es verdadera, al ejecutar esta instrucción se produce un ciclo infinito. A fin de evitarlo, las instrucciones del cuerpo del ciclo deben contener alguna instrucción que modifique la o las variables involucradas en la condición, de modo que ésta sea falsificada en algún momento y así finalice la ejecución del ciclo.



Ejemplo 1:

Diseñar un algoritmo que permita ingresar n cantidad de notas validas, sabiendo que el rango de nota valida es de 1 a 10 pts.

```
<sin_titulo>* 
1  Algoritmo sin_titulo
2      definir n, cont, suma Como Entero
3      Definir nota Como Real
4      Escribir "¿Cuantas notas desea ingresar?"
5      Leer n
6      cont = 1
7      Mientras cont ≤ n Hacer
8          Escribir "Ingrese una nota"
9          Leer nota
10         si nota ≥ 1 y nota ≤ 10 Entonces
11             cont = cont + 1
12         SiNo
13             Escribir "Ingresa nota valida"
14             FinSi
15         FinMientras
16     FinAlgoritmo
```

Estructura Repetir - Hasta:

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

Repetir

<instrucciones>

Hasta Que <condición>

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

Esto se repite hasta que la condición sea verdadera.

Note que, dado que la condición se evalúa al final, las instrucciones del cuerpo del ciclo serán ejecutadas al menos una vez. Además, a fin de evitar ciclos infinitos, el cuerpo del ciclo debe contener alguna instrucción que modifique la o las variables involucradas en la condición de modo que en algún momento la condición sea verdadera y se finalice la ejecución del ciclo.



Ejemplo 2:

Diseñar un algoritmo que permita ingresar n cantidad de notas validas, sabiendo que el rango de nota valida es de 1 a 10 pts.

```
<sin_titulo>* 
1 Algoritmo sin_titulo
2     definir n, cont, suma Como Entero
3     Definir nota Como Real
4     Escribir "¿Cuantas notas desea ingresar?"
5     Leer n
6     cont = 0
7     Repetir
8         Escribir "Ingrese una nota"
9         Leer nota
10        si nota ≥ 1 y nota ≤10 Entonces
11            cont = cont + 1
12        SiNo
13            Escribir "Ingresa nota valida"
14            Escribir ""
15        FinSi
16        Hasta Que cont ≥n
17 FinAlgoritmo
```

Teniendo en cuenta que el ejemplo es el mismo utilizado en la estructura MIENTRAS-HACER pero adaptándolo a esta nueva estructura, apreciamos la primera y al ingresar la cantidad de notas igual a cero sencillamente no se ejecuta el ciclo ya que el evalúa la condición en el inicio, pero ingresamos cero también en el segundo y el ciclo se ejecuta una vez para luego detenerse, esto sucede porque en esta otra se evalúa al final, por lo tanto REPITA-HASTA se debe utilizar solo en problemas que necesiten por lo menos una vez su ejecución.

Estructura Para:

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

Para <variable> <- <inicial> Hasta <final> **Con Paso** <paso> **Hacer**

<instrucciones>

FinPara



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

Ejemplo 3:

Diseñar un algoritmo que permita repetir n cantidad de veces un mensaje cualquiera:

```
<sin_titulo>* <sin_titulo>* x
1  Algoritmo sin_titulo
2      definir n, i Como Entero
3      Escribir "Cantidad de repeticiones"
4      Leer n
5      Para i=1 hasta n Con Paso 1 Hacer
6          Escribir "Repeticion numero " , i
7      FinPara
8  FinAlgoritmo
9 |
```

En esta estructura la instrucción Para es un poco más compleja que las otras dos, pero simplifica la ejecución de los ciclos.

En la instrucción comenzamos con una variable de repetición que se inicializa en la misma instrucción y no afuera como en las anteriores, luego se delimita, en el caso del ejemplo de n y por último se indica el valor del incremento de la variable, en el ejemplo se incrementa de 1 en 1, pero ese valor puede variar y además si el incremento es igual a 1 este se puede omitir, por ejemplo: (Para i=1 Hasta n Hacer)



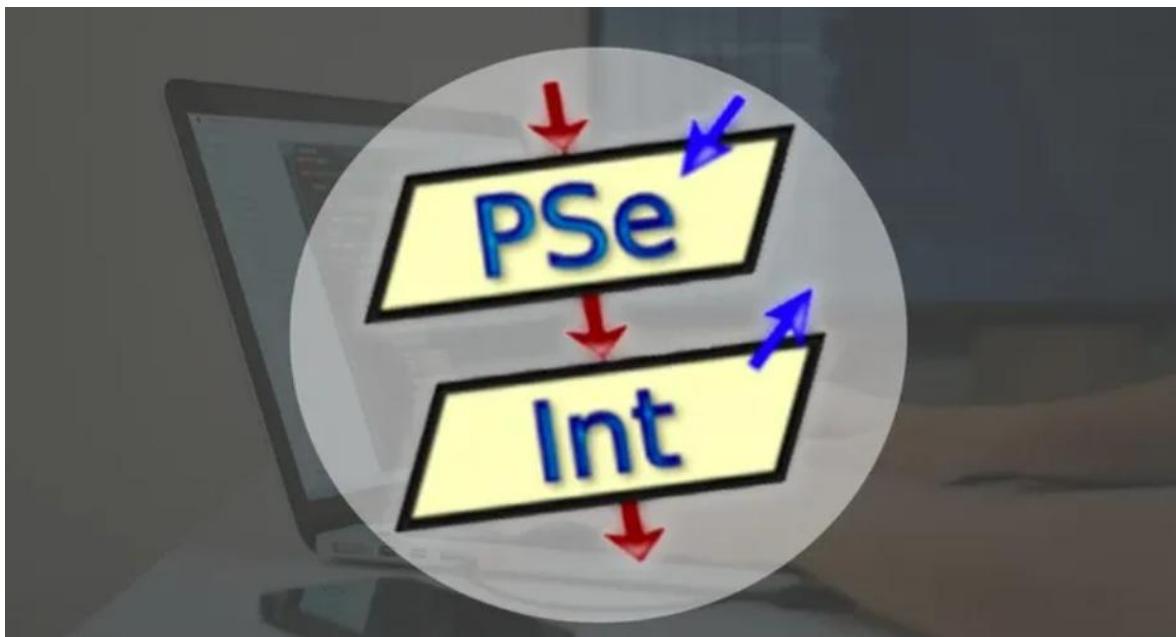
EJERCICOS PROPUESTOS

1. Hacer un algoritmo en Pseint para calcular la suma de los primeros cien números con un ciclo repetir.
2. Hacer un algoritmo en Pseint para calcular la suma de los primeros cien números con un ciclo mientras.
3. Hacer un algoritmo en Pseint para calcular la suma de los primeros cien números con un ciclo para.
4. Calcular el salario de N trabajadores y la suma total de los mismos.
5. Hacer un algoritmo que permita calcular la suma de n números.
6. Hacer un programa que cuente del 20 a 1 en forma descendente
7. Hacer un programa que cuente del uno al 10



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA Lógica de Programación



SESION 15

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I2: Diagrama clases de objetos aplicando los elementos de las técnicas de programación orientadas a objetos.



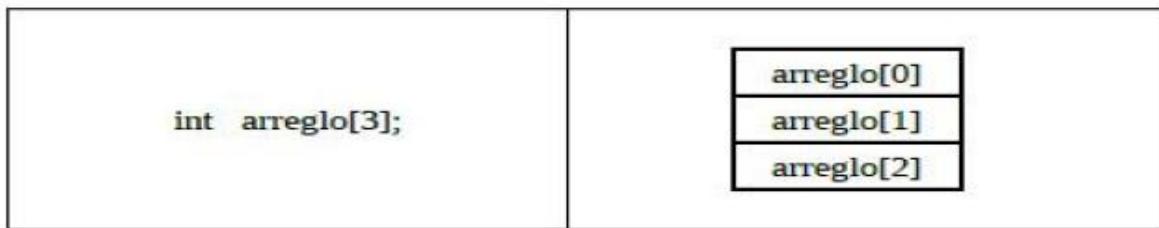
ARREGLOS UNIDIMENSIONALES

Un arreglo unidimensional es un tipo de datos estructurado que está formado por una colección finita y ordenada de datos del mismo tipo. Es la estructura natural para modelar listas de elementos iguales. Los datos que se guarden en los arreglos todos deben ser del mismo tipo.

El tipo de acceso a los arreglos unidimensionales es el acceso directo, es decir, podemos acceder a cualquier elemento del arreglo sin tener que consultar a elementos anteriores o posteriores, esto mediante el uso de un índice para cada elemento del arreglo que nos da su posición relativa.

Para implementar arreglos unidimensionales se debe reservar espacio en memoria.

Los arreglos nos permiten hacer un conjunto de operaciones para manipular los datos guardados en ellos, estas operaciones son: ordenar, buscar, insertar, eliminar, modificar entre otras.



Declaración: ***tipo nombre_arreglo[tamaño];***

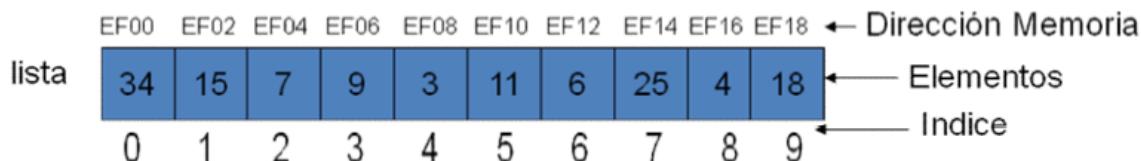
Donde - ***tipo***, declara el tipo base del arreglo, que es el tipo de cada elemento del mismo

nombre_arreglo, es el nombre del arreglo.

tamaño, es el nro. de elementos del arreglo.

Ejemplo:

`int lista[10]; //declara un arreglo lista con 10 elementos de tipo int.`



Los elementos del arreglo lista del ejemplo serian:

lista[0] = 34 primer elemento del arreglo

lista[1] = 15 segundo elemento del arreglo

.....

lista[9] = 18 último elemento del arreglo

ARREGLOS UNIDIMENSIONALES EN PSEINT

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

Para crear un arreglo en PSeInt se utiliza la palabra clave **Dimension**, seguido del nombre del arreglo (identificador) y su tamaño entre corchetes [].

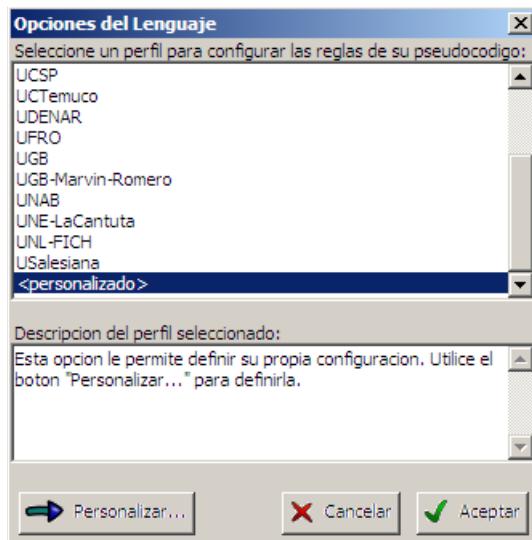
Sintaxis:

Dimension identificador [tamaño];

En PSeInt los arreglos pueden comenzar desde cero o uno. Depende de cómo se configure el programa.

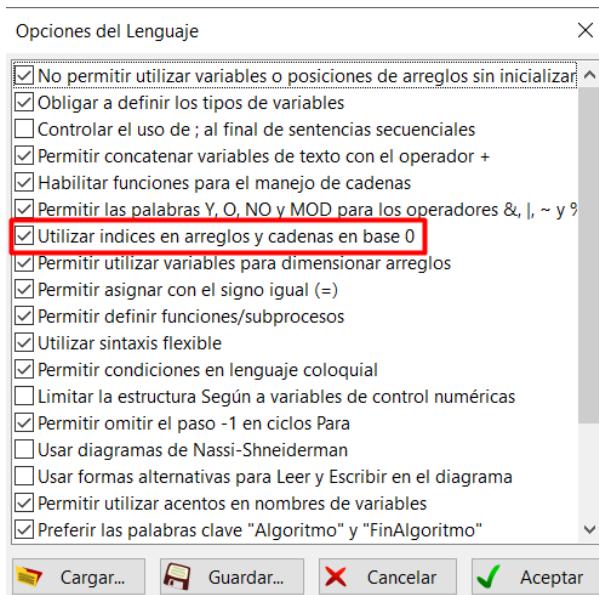
Arreglos en base cero

Para empezar a programar nuestros algoritmos con arreglos en base cero, vamos al menú **configurar** y escogemos **Opciones del lenguaje**, nos saldrá un cuadro como este:



Seleccionamos el perfil **<personalizado>** y hacemos clic en el botón **Personalizar...**

Saldrá el siguiente cuadro:



Marcamos la casilla: *Utilizar arreglos en base cero* y presionamos **aceptar**. De esta manera ya queda configurado el programa para utilizar arreglos en base cero.



Ejemplo 1:

Crear un arreglo llamado **num** que almacene los siguientes datos: 20, 14, 8, 0, 5, 19 y 24.

```
1 Algoritmo sin_titulo
2     Definir num Como Entero
3     Dimension num[7]
4     num[0]=20
5     num[1]=14
6     num[2]=8
7     num[3]=0
8     num[4]=5
9     num[5]=19
10    num[6]=24
11    Escribir "El dato en la posicion 0 es: " , num[0]
12    Escribir "El dato en la posicion 1 es: " , num[1]
13    Escribir "El dato en la posicion 2 es: " , num[2]
14    Escribir "El dato en la posicion 3 es: " , num[3]
15    Escribir "El dato en la posicion 4 es: " , num[4]
16    Escribir "El dato en la posicion 5 es: " , num[5]
17    Escribir "El dato en la posicion 6 es: " , num[6]
18
19 FinAlgoritmo
20
```

PSelnt - Ejecutando proceso SIN_TITULO

```
El dato en la posicion 1 es: 14
El dato en la posicion 2 es: 8
El dato en la posicion 3 es: 0
El dato en la posicion 4 es: 5
El dato en la posicion 5 es: 19
El dato en la posicion 6 es: 24
*** Ejecución Finalizada. ***
```

No cerrar esta ventana Siempre visible Reiniciar



Ejemplo 2:

Haremos el mismo ejercicio con el ciclo para para poder aprender la forma de trabajar con las estructuras.

```
<sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* X
1 Algoritmo sin_titulo
2   Definir num,i Como Entero
3   Dimension num[7]
4   num[0]=20
5   num[1]=14
6   num[2]=8
7   num[3]=0
8   num[4]=5
9   num[5]=19
10  num[6]=24
11  para i=0 hasta 7-1 Con Paso 1 Hacer
12    |  Escribir "El dato en la posicion: " , i , " es: " , num[i]
13  FinPara
14 FinAlgoritmo
```

El ciclo **Para** nos ahorra la tarea de escribir los siete mensajes que muestran los siete datos pedidos inicialmente.

Podemos ver que la salida es la misma:

```
PSelint - Ejecutando proceso SIN_TITULO
*** Ejecución Iniciada. ***
El dato en la posicion: 0 es: 20
El dato en la posicion: 1 es: 14
El dato en la posicion: 2 es: 8
El dato en la posicion: 3 es: 0
El dato en la posicion: 4 es: 5
El dato en la posicion: 5 es: 19
El dato en la posicion: 6 es: 24
*** Ejecución Finalizada. ***
```

No cerrar esta ventana Siempre visible Reiniciar



Pero no solo podemos imprimir los datos del arreglo con un ciclo, también podemos llenar con datos los arreglos con el ciclo **Para**.

Ejemplo 3:

Crear un arreglo de 5 posiciones y llénelo con los números que el usuario desee.

```
<sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* <sin_titulo>* ×
1 Algoritmo sin_titulo
2     Definir num,i, numero Como Entero
3     //Creamos el arreglo le damos un nombre y un tamaño de 5 posiciones
4     Dimension num[5]
5     //A cada posición le damos un dato con el ciclo para
6     para i=0 hasta 4 Con Paso 1 Hacer
7         //Pedimos los datos
8         escribir "Digite un numero para la posicion: " , i
9         Ler numero // capturo los numeros ingresados en la variable numero
10        num[i]=numero //Al arreglo se le pasa todos los numeros ingresados
11    FinPara
12    //Imprimir los datos ingresados con el ciclo para
13    para i=0 hasta 4 Con Paso 1 Hacer
14        escribir "El dato en la posicion: " , i , " es: " , num[i]
15
16    FinPara
17 FinAlgoritmo
```

Tenemos como resultado lo siguiente:

```
► PSeInt - Ejecutando proceso SIN_TITULO
*** Ejecución Iniciada. ***
Digite un numero para la posicion: 0
> 10
Digite un numero para la posicion: 1
> 20
Digite un numero para la posicion: 2
> 30
Digite un numero para la posicion: 3
> 40
Digite un numero para la posicion: 4
> 50
El dato en la posicion: 0 es: 10
El dato en la posicion: 1 es: 20
El dato en la posicion: 2 es: 30
El dato en la posicion: 3 es: 40
El dato en la posicion: 4 es: 50
*** Ejecución Finalizada. ***

 No cerrar esta ventana  Siempre visible Reiniciar ▾
```

Como se puede apreciar en la salida, los números ingresados por el usuario son: **10, 20, 30, 40, 50.**

Hemos visto arreglos con datos numéricos, pero también se le pueden llenar con datos de tipo **cadenas de texto**.

Ejemplo 4:

Crear un arreglo de n posiciones y llenarlo con nombres de personas.

En este ejemplo el usuario eligió 3 posiciones, llenando el arreglo con los siguientes nombres: **Alexander, Jose y Erico**.



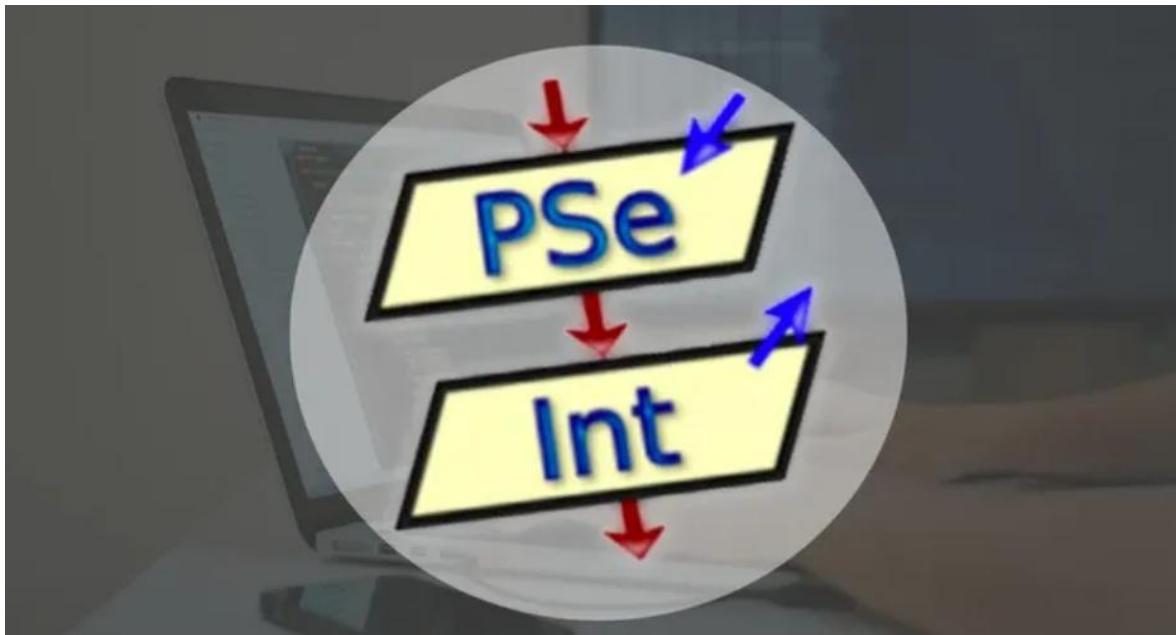
EJERCICIOS PROPUESTOS

1. Crear un arreglo de n posiciones y llenarlo con los números que el usuario desee.
 2. Crear dos arreglos uno que almacene 2 nombres y otro que almacene 3 números.
 3. Sumar todos los elementos de un arreglo de tamaño n.
 4. Crear un arreglo con n números, ingresados por teclado y mostrar sus valores elevados al cuadrado.
 5. Llenar un arreglo con números enteros (números positivos ó negativos). Mostrar la cantidad de números positivos que hay en dicho arreglo.



Módulo Guía de Aprendizaje

UNIDAD DIDÁCTICA Lógica de Programación



SESION 16

Capacidad Asociada a la UD:

UC4.C1: Elaborar especificaciones del sistema a desarrollar de acuerdo a los requerimientos de la organización, haciendo uso de los estándares de gestión del ciclo de vida de desarrollo de TI

Indicador de Logro:

C1.I2: Diagrama clases de objetos aplicando los elementos de las técnicas de programación orientadas a objetos.



ARREGLOS BIDIMENSIONALES

Se considera arreglo dimensional a todo aquel que posee más de una dimensión, normalmente se distinguen los arreglos de dos dimensiones como “**bidimensionales**” y arreglos de más de dos dimensiones como “**multidimensionales**”.

A los arreglos de dos dimensiones (**bidimensionales**) se les conoce como matrices y son estructuras de datos que organizan su información en forma de tablas, es decir, los elementos que la conforman están dispuestos bajo dos conceptos de clasificación (fila y columna). Para poder indicar el lugar donde se encuentra un determinado elemento, es necesario utilizar dos índices: uno para indicar el renglón o fila y otro para indicar la columna.

Puede mirarse una matriz como un vector de vectores; por lo tanto, es un conjunto de componentes en el que se necesitan dos subíndices para identificar un elemento que pertenezca al arreglo.

Por ejemplo, si tenemos un arreglo llamado matriz de dos dimensiones como se muestra a continuación:

Dimensión matriz[3, 6]

Gráficamente podemos representar al arreglo bidimensional como se muestra a continuación:

0	1	2	3	4	5
1					
2					

Un arreglo bidimensional **N * M** tiene N filas y M columnas; por lo tanto, tiene **N * M** elementos dispuestos interiormente en memoria en forma sucesiva.



Ejemplo 1

El presente ejemplo muestra como almacenar elementos dentro de las posiciones de un arreglo de dos dimensiones.

```
1 Algoritmo sin_titulo
2     Definir fila, columna, matriz Como Entero
3     Dimension matriz[3, 4]; //arreglo de dos dimensiones (12 elementos)
4
5     Para fila < 0 Hasta 3-1 Con Paso 1 Hacer
6         Para columna < 0 Hasta 4-1 Con Paso 1 Hacer
7             Escribir "Ingrese el elemento [", fila, ", ", columna, "] "
8             Leer matriz[fila, columna]; //se almacena el elemento
9         FinPara
10    FinPara
11
12 FinAlgoritmo
13
```

Si nos imaginamos el cómo se vería el arreglo **matriz** gráficamente, seria por ejemplo:

Columns			
1	1,1	1,2	1,3
2	2,1	2,2	2,3
3	3,1	3,2	3,3
			3,4

Filas { Columns }

Último elemento →

Para poder almacenar un valor en el elemento correspondiente a la fila 2 columna 3, realizamos la siguiente sentencia:

matriz[2,3] <- Valor //Donde valor corresponde al elemento a almacenar.

Por tal motivo para poder almacenar un elemento dentro de un arreglo dimensional necesitamos dos índices, el primer índice para la primera dimensión (fila) y el segundo índice para segunda dimensión (columna). Por ello en el ejemplo anterior tenemos la sentencia:



Leer matriz[fila, columna]

Esta sentencia permite almacenar un elemento dentro del arreglo **matriz** en la posición dada por los valores de los índices **fila** y **columna**.

Ejemplo 2

Esta muestra como acceder para extracción al contenido de un arreglo de dos dimensiones.

```
1 Algoritmo sin_titulo
2     definir matriz, fila, columna Como Entero
3
4     Dimension matriz[3, 4]; //arreglo de dos dimensiones (12 elementos)
5
6     Para fila = 0 Hasta 3-1 Con Paso 1 Hacer
7         Para columna = 0 Hasta 4-1 Con Paso 1 Hacer
8             Escribir "Ingrese el elemento [", fila, ", ", columna, "] "
9             Leer matriz[fila, columna]; //se almacena el elemento
10            FinPara
11        FinPara
12
13        Para fila = 0 Hasta 3-1 Con Paso 1 Hacer
14            Para columna = 1 Hasta 4-1 Con Paso 1 Hacer
15                Escribir "El elemento [", fila, ", ", columna, "] = ", matriz[fila, columna]
16            FinPara
17        FinPara
18
19 FinAlgoritmo
20
```

Ejemplo 3

Hacer un algoritmo que encuentre e imprima la matriz transpuesta de una matriz **MAT**.

Datos de entrada

- Número de filas de la matriz.
- Número de columnas de la matriz
- Elementos del arreglo.

Datos de salida

- Los elementos de la matriz transpuesta.



Definición de variables

- **nFilas**, número de filas de la matriz.
- **nColumnas**, número de columnas de la matriz.
- **MAT**, nombre de la matriz.
- **fila, columna**, variables de control de ciclos e índices de la matriz.
- **TMAT**, matriz transpuesta de **MAT**.

Proceso

La matriz transpuesta de la matriz **MAT** se encuentra intercambiando las filas por las columnas y las columnas por las filas. Si **TMAT** es la matriz transpuesta de **MAT**, implica entonces que **TMAT [columna, fila]** es igual a **MAT [fila, columna]**. Si el contenido de **MAT** es:

5	8	1	9
15	11	9	4
16	4	3	0

5	15	16
8	11	4
1	9	3
9	4	0

Como se puede ver, se invierte el orden de la matriz; es decir, el número de filas de **MAT** es igual al número de columnas de **TMAT** y el número de columnas se invierte por el número de filas de **TMAT**.



```
1 Algoritmo sin_titulo
2   definir MAT, TMAT, nFilas, nColumnas, fila, columna Como Entero
3   Escribir "Número de filas de la matriz "
4   Leer nFilas;
5
6   Escribir "Número de columnas de la matriz "
7   Leer nColumnas
8   //Definir MAT, TMAT, nFilas, nColumnas como entero;
9   Dimension MAT[nFilas, nColumnas]; //Matriz
10  Dimension TMAT[nColumnas, nFilas]; //Almacenara la matriz transpuesta
11
12  //Se piden los datos de la matriz
13  Para fila = 0 Hasta nFilas-1 Con Paso 1 Hacer
14    Para columna = 0 Hasta nColumnas-1 Con Paso 1 Hacer
15      Escribir "Ingrese el elemento [", fila, ", ", columna, "] "
16      Leer MAT[fila, columna]; //se almacena el elemento
17      FinPara
18  FinPara
19  Para fila = 0 Hasta nFilas-1 Con Paso 1 Hacer
20    Para columna = 0 Hasta nColumnas-1 Con Paso 1 Hacer
21      TMAT[columna, fila] ← MAT[fila, columna]; //Se copia el elemento
22      FinPara
23  FinPara
24
25  Para fila = 0 Hasta nColumnas-1 Con Paso 1 Hacer
26    Para columna = 0 Hasta nFilas-1 Con Paso 1 Hacer
27      Escribir "El elemento [", fila , ", ", columna, "] = ", TMAT[fila, columna]
28      FinPara
29  FinPara
30
31 FinAlgoritmo
```



EJERCICOS PROPUESTOS

- Realice un algoritmo que pida un arreglo de N elementos y luego que presente dicho arreglo en orden inverso.

Por ejemplo, si tenemos un arreglo llamado vector como se presenta a continuación:

2	4	5	7	1	9
---	---	---	---	---	---

Este se muestra en pantalla en orden inverso, es decir en el orden: 9,1,7,5,4,2

- Crea un arreglo o array multidimensional que contenga 3 columnas y las filas que tu quieras, las dos primeras columnas tendrán números y en la 3 columna sera el resultado de sumar el número de la primera y segunda columna. Muestra el array o arreglo de la siguiente forma:

$$\begin{aligned} 3 + 5 &= 8 \\ 4 + 4 &= 7 \end{aligned}$$

- Crea un arreglo o array multidimensional con un tamaño que definiremos nosotros por teclado, contendrá números aleatorios usando la función anterior y crearemos un array o arreglo unidimensional donde se copiaran los números que contiene el array multidimensional. Piensa que tamaño debe tener el array o arreglo unidimensional