UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Av. Universidad 1001, Chamilpa, Cuernavaca, Morelos, México,
C.P. 62209 Tel. (777) 329 7000 y 329 7041 Ext.3148

CURSO PROPEDÉUTICO 2017

INTRODUCCIÓN A LA PROGRAMACIÓN

CONTENIDO

Números de horas totales: 16 hrs.

MÓDULO I ASPECTOS GENERALES DE LA PROGRAMACIÓN 4
1.1 Introducción a la programación 4
1.2. Programa para el desarrollo de software 4
1.3 Clasificación de los lenguajes de programación 6
1.4 Tipos de programación 7
MÓDULO II FUNDAMENTOS DE ALGORITMOS Y PSEUDOCODIGO 8
2.1 Planteamiento del problema 9
2.2 Abstracción de datos 9
2.3 Tipos de datos 9
2.4 Identificadores, constantes y variables 12
2.5 Operaciones aritméticas 15
2.5.1 Ejercicios de Tipos de Datos en ambas herramientas 17
2.6 Operaciones de Entrada y Salida de datos 17
2.7 Operaciones relacionales 20
2.8 Operaciones lógicas 21
2.9 Algoritmo 22
2.10 Diagrama de flujo 24
2.11 Pseudocódigo 26
MÓDULO III. ELEMENTOS DEL ENTORNO DE LAS HERRAMIENTAS DFD Y PSeInt 26
3.1 Diagrama de flujo con DFD 27
3.1.1 Simbologías 27
3.2 Barra de Herramientas de DFD 28
3.3 Barra de Herramientas de PSeInt 30
MÓDULO IV. SENTENCIAS CONDICIONALES Y/O SELECTIVAS 32
5.1 Sentencias Simples 32
5.2 Estructura de decisión simple con DFD 33

5.3 Sentencias Compuestas _____ 34
5.4 SI anidados y estructuras selectivas múltiples _____ 38
5.5 Estructura de decisión Simple con PSeInt _____ 39
5.6 Selección Múltiple con PSeInt _____ 40
MÓDULO V SENTENCIAS REPETITIVAS _____ 41
6.1 Estructuras repetitivas PARA con DFD _____ 41
6.2 Ejemplo de un ciclos PARA anidados _____ 43
6.3 Estructura MIENTRAS con DFD _____ 44
6.4 Estructuras Repetitivas con PSeInt _____ 47
6.5 Estructuras repetitivas PARA con PSeInt _____ 47
6.6 Estructura MIENTRAS con PSeInt _____ 47
6.7 Estructura Repetir Hasta Que (do-while) con PSeInt _____ 48
BIBLIOGRAFIA _____ 51

MÓDULO I ASPECTOS GENERALES DE LA PROGRAMACIÓN

1.1 Introducción a la programación

La computadora realiza una serie de actividades para realizar tareas en específico, dichas actividades son operaciones que debe realizar el *hardware* y son especificadas por una serie de instrucciones llamadas *programas* o *software.*

Para crear un programa, y que la computadora lo intérprete y ejecute las instrucciones escritas en él, debe usarse un lenguaje de programación. En sus inicios las computadoras interpretaban sólo instrucciones en un lenguaje específico, del más bajo nivel, conocido como código máquina, siendo éste excesivamente complicado para programar. De hecho sólo consiste en cadenas de números 1 y 0 (sistema binario). Para facilitar el trabajo de programación, los primeros científicos que trabajaban en el área decidieron reemplazar las instrucciones, secuencias de unos y ceros, por palabras o letras provenientes del inglés; las codificaron y crearon así un lenguaje de mayor nivel, que se conoce como Assembly o lenguaje ensamblador. Por ejemplo, para sumar se usa la letra A de la palabra inglesa add (sumar). En realidad escribir en lenguaje ensamblador es básicamente lo mismo que hacerlo en lenguaje máquina, pero las letras y palabras son bastante más fáciles de recordar y entender que secuencias de números binarios. A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método sencillo para programar. Entonces, se crearon los lenguajes de alto nivel. Mientras que una tarea tan trivial como multiplicar dos números puede necesitar un conjunto de instrucciones en lenguaje ensamblador, en un lenguaje de alto nivel bastará con solo una. Una vez que se termina de escribir un programa, sea en ensamblador o en un lenguaje de alto nivel, es necesario compilarlo, es decir, traducirlo a lenguaje máquina.

Programar es explicarle al ordenador lo que tiene que hacer de la forma más precisa posible.

Donald Knuth, “The Art of Computer Programming”

1.2. Programa para el desarrollo de software

Es el conjunto de herramientas que permiten al programador desarrollar programas informáticos, usando diferentes alternativas y lenguajes de programación, de una manera práctica. Incluye entre otros:

- Editores de texto
- Compiladores
- Intérpretes
- Enlazadores
- Depuradores

Entornos Integrados de Desarrollo (EID): Agrupan las anteriores herramientas, usualmente en un entorno visual, de forma tal que el programador no necesite introducir múltiples comandos para compilar, interpretar, depurar, etc. Habitualmente cuentan con una avanzada interfaz gráfica de usuario (GUI).

1.2.1 Traductores

Los traductores son programas que traducen los programas en código fuente, escritos en lenguajes de alto nivel, a programas escritos en lenguaje máquina. Los traductores pueden ser de dos tipos: compiladores e intérpretes

1.2.2 Interpretes

Traduce a lenguaje máquina, cada línea del programa fuente y la ejecuta de inmediato.

Traducen en Lenguaje de Alto Nivel a Lenguaje Maquina, se encargan de traducir cada instrucción, una por una (o cada línea de instrucciones) contenida en un programa escrito en cualquier lenguaje de alto nivel a instrucciones en código binario, comprensible por las computadoras.

Los intérpretes realizan la traducción y ejecución de forma simultánea, es decir, un intérprete lee el código fuente y lo va ejecutando al mismo tiempo.

Figura 1 Intérprete

1.2.3 Compiladores

Un compilador es un programa que traduce los programas fuente escritos en lenguaje de alto nivel a lenguaje máquina. La traducción del programa completo se realiza en una sola operación denominada compilación del programa; es decir, se traducen todas las instrucciones del programa en un solo bloque. El programa compilado y depurado (eliminados los errores del código fuente)
se denomina programa ejecutable porque ya se puede ejecutar directamente y cuantas veces desee; sólo deberá volver a compilarse de nuevo en el caso de que se modifique alguna instrucción del programa.

Compilación

La compilación de un programa es el paso mediante el cual traducimos dicho programa al lenguaje máquina entendible por la computadora (código binario), para después ejecutarlo en la memoria.

El compilador o traductor analiza todo el programa fuente (conjunto de instrucciones escritas en texto simple) y detecta limitaciones de sintaxis ocasionadas por fallas en la codificación o la transcripción (los errores de lógica que pueda tener nuestro programa fuente no son detectadas por el compilador). Cuando no hay fallas graves en la compilación, el compilador traduce cada orden del programa fuente a instrucciones propias de la máquina, creando el programa objeto.

Figura 2 La compilación de programas

1.3 Clasificación de los lenguajes de programación

Los lenguajes de programación son lenguajes creado por el ser humano para poder comunicarse con las computadoras. Estos son un conjunto de símbolos y palabras que permiten al usuario de una computadora darle instrucciones y órdenes para que esta las pueda realizar.

Existen diferentes clases o tipos de lenguajes de programación:

1.3.1 Código de Maquina

Es el lenguaje de programación que entiende directamente la computadora o máquina. Este lenguaje de programación utiliza el alfabeto binario, es decir, el 0 y el 1. Con estos dos únicos dígitos, conocidos como bits, forma las cadenas binarias (combinaciones de ceros y unos) son con las que se escriben las instrucciones que el microprocesador de la computadora entiende nuestra peticiones. El lenguaje máquina fue el primer lenguaje de programación. Dejo de usarse por su gran dificultad y por la facilidad para cometer errores.

1.3.2 Bajo nivel

Son mucho más fáciles de utilizar que el lenguaje máquina, pero dependen mucho de la computadora como sucedía con el lenguaje máquina. El lenguaje ensamblador fue el primer lenguaje de programación de bajo nivel que trato de sustituir el lenguaje máquina por otro mucho más parecido al de los seres humanos. El programa fuente es un conjunto de instrucciones escrito en lenguaje ensamblador, y cuyo objeto es la traducción a lenguaje máquina del programa fuente. Los lenguajes de este tipo son agiles, difíciles de usar, específicos de cada procesador, si nos llevamos el programa a otro computador será preciso reescribir el programa desde el comienzo.

Ejemplo.

Un programa escrito en lenguaje ensamblador consiste en una serie de instrucciones que corresponden al flujo de órdenes ejecutables por un microprocesador.

Por ejemplo, en el lenguaje ensamblador para un procesador x86: La sentencia
MOV AL, 61h

Asigna el valor hexadecimal 61 (97 decimal) al registro "AL".

El programa ensamblador lee la sentencia de arriba y produce su equivalente binario en lenguaje de máquina

Binario: 10110000 01100001 (hexadecimal: B61)

1.3.3 Alto Nivel

Este lenguaje es independiente de la máquina, lo podemos usar en cualquier computador con muy pocas modificaciones o sin ellas, son muy similares al lenguaje humano. Necesitan un programa intérprete o compilador que lo traduzca uno de bajo nivel, como el lenguaje de máquina para que la computadora pueda entenderlo. Este tipo de lenguaje es más fáciles de aprender porque se usan palabras o comandos del lenguaje natural, como por ejemplo: palabras en inglés. Este es el caso del BASIC, el lenguaje de programación más conocido.

Ejemplos

VB.NET	Java	PL/1
Ada	Lisp	PL/SQL
ALGOL	Modula-2	Python
BASIC	Pascal	Ruby
C Sharp	Perl	Matlab
FORTRAN	PHP	

1.4 Tipos de programación

Existen varias clases de programación, dependiendo de los métodos utilizados y las técnicas empleadas.

1.4.1 Programación modular

En la programación modular consta de varias secciones dividas de forma que interactúan a través de llamadas a procedimientos, que integran el programa en su totalidad. El programa principal coordina las llamadas a los módulos secundarios y pasa los datos necesarios en forma de parámetros.

En este tipo de programación el programa es dividido en módulos, cada uno de las cuales realiza una tarea específica, codificándose independientemente de otros módulos. Cada uno de éstos son analizados, codificados y puestos a punto por separado.

Los programas contienen un módulo denominado módulo principal, el cual supervisa todo lo que sucede, transfiriendo el control a submódulos (los que son denominados subprogramas), para que puedan realizar sus funciones. Sin embargo, cada submódulo devolverá el control al módulo principal una vez completada su tarea. Si las tareas asignadas a cada submódulo son demasiado complejas, se procederá a una nueva subdivisión en otros módulos más pequeños aún.

Este procedimiento se realiza hasta que cada uno de los módulos realicen tareas específicas. Estas pueden ser entrada, salida, manipulación de datos, control de otros módulos o alguna combinación de éstos. Puede ser que un módulo derive el control a otro mediante un proceso denominado bifurcación, pero se debe tomar en cuenta que esta derivación deberá ser devuelta a su módulo original.

1.4.2 Programación estructurada

Cuando hablamos de Programación Estructurada, nos referimos a un conjunto de técnicas que con el transcurrir del tiempo han evolucionado. Gracias a éstas, la productividad de un programa se ve incrementada de forma considerable y se reduce el tiempo de escritura, de depuración y mantenimiento de los programas. Aquí se hace un número limitado de estructuras de control, se reduce la complejidad de los problemas y se minimiza los errores.

Gracias a la programación estructurada, es más fácil la escritura de los programas, también lo es su verificación, su lectura y mantenimiento. Esta programación es un conjunto de técnicas que incorpora:

- diseño descendente (top-down)
- recursos abstractos
- estructuras básicas

Para separar un programa en términos de recursos abstractos debemos descomponer acciones complejas en acciones más simples, las que son capaces de ejecutar o constituyen instrucciones de computadora disponible.

Diseño descendente (Top-Down)

Este es un proceso en el cual el problema se descompone en una serie de niveles o pasos sucesivos (stepwise). Esta metodología consiste en crear una relación entre las etapas de estructuración, las que son sucesivas, de tal forma que se interrelacionen mediante entradas y salidas de información. Considerando los problemas desde dos puntos de vista: ¿que hace? Y ¿cómo lo hace?

1.4.3 Programación Orientada a Objetos

La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento.

La programación Orientada a objetos (POO) es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación.

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo vamos a pensar en un coche para tratar de modelizarlo en un esquema de POO.

Diríamos que el coche es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca. Además tiene una serie de funcionalidades asociadas, como pueden ser ponerse en marcha, parar o aparcar.

Pues en un esquema POO el coche sería el objeto, las propiedades serían las características como el color o el modelo y los métodos serían las funcionalidades asociadas como ponerse en marcha o parar.

MÓDULO II FUNDAMENTOS DE ALGORITMOS Y PSEUDOCODIGO

2.1 Planteamiento del problema

Albert Einstein una vez dijo que si tenía una hora para salvar el mundo iba a utilizar cincuenta y cinco minutos definiendo el problema y sólo cinco minutos para encontrando la solución.

El planteamiento del problema es una declaración clara y concisa que describe los síntomas del problema a abordar.
Los pasos que se siguen generalmente a la hora de desarrollar un programa son los siguientes:
• Análisis de requerimientos: Se define el problema a resolver y todos los objetivos que se pretenden, pero sin indicar la forma en la que se resuelve.
• Especificación: Se determina la forma en la que se resolverá el problema, pero sin entrar aún en su implementación informática. Se determina asimismo la interfaz con el usuario.
• Diseño del programa: Se divide el problema en módulos, se especifica lo que hace cada módulo, así como las interfaces de cada uno de ellos.
• Diseño detallado de los módulos: Para cada módulo se diseñan detalladamente las estructuras de datos y los algoritmos a emplear, normalmente descritos mediante pseudocódigo.
• Codificación: Se escribe el programa en el lenguaje de programación elegido.
• Pruebas de módulos: Se prueban los módulos del programa aisladamente y se corrigen los fallas hasta conseguir un funcionamiento correcto.
• Integración y Prueba de sistema: Se unen todos los módulos, y se prueba el funcionamiento del programa completo

2.2 Abstracción de datos

La abstracción es un proceso mental que tiene dos aspectos complementarios:

• El aspecto de destacar los detalles relevantes del objeto en estudio.

• El aspecto de ignorar los detalles irrelevantes del objeto. Se entiende que son irrelevantes en ese nivel de abstracción. Si descendemos de nivel, es probable que algunos de estos detalles pasen a ser relevantes.

La abstracción funcional, es decir, la idea de crear procedimientos y funciones e invocarlos mediante un nombre, se desarrolló muy temprano. Los primeros lenguajes de alto nivel (Fortran, Cobol), ya tenían este mecanismo. En este tipo de abstracción lo que se destaca es qué hace la función y lo que se ignora es cómo lo hace, es decir, el algoritmo concreto y las variables auxiliares necesarias para conseguir el efecto pretendido. El usuario del procedimiento sólo necesita conocer la especificación de la abstracción (el qué) y puede ignorar el resto de los detalles (el cómo). Diremos que la abstracción produce un ocultamiento de información

2.3 Tipos de datos

Un dato se define como la expresión general que describe los objetos con los cuales opera una computadora. Los datos de entrada se transforman por el programa, después de las etapas intermedias, en datos de salida.

Los datos se clasifican en diversas categorías, según el tipo de máquina o del lenguaje en uso. Generalmente podemos encontrar las siguientes categorías:

- Numéricos
- Lógicos
- Cadenas

Datos numéricos

Son aquellos que representan una cantidad o valor determinado. Su representación se lleva a cabo en los formatos ya conocidos (enteros, punto y fracciones decimales si estas existen).

Estos pueden representarse en dos formas distintas:

- Tipo Numérico Entero (integer).
- Tipo Numérico Real (real).

Enteros

Es un conjunto finito de los números enteros. Los enteros son números completos, no tienen componentes fraccionarios o decimales y pueden ser negativos y positivos.

Algunos ejemplos son:

37,-109, 50

Reales

Consiste en un subconjunto de los números reales. Estos números siempre tienen un punto decimal y pueden ser positivos o negativos. Un número real consiste de un número entero y una parte decimal. Algunos ejemplos son:

0.52, 664.32, 6.579, 8.0, -9.3, -47.23

Cadenas

Son los datos que representan información textual (palabras, frases, símbolos, etc). No representan valor alguno para efectos numéricos. Pueden distinguirse porque son delimitados por apóstrofes o comillas.

Se clasifica en dos categorías:

- Datos tipo carácter (char)
- Datos tipo Cadena (string)

Datos tipo carácter

Es un conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato de este tipo contiene solo un carácter.

Reconoce los siguientes caracteres:

Caracteres Alfabéticos (A,B,C,…Z,a,b,c…z) Caracteres Numéricos (0,1,2,…9)
Caracteres Especiales (+, -, *, /, ^, . , ;, <, >, $, …….)

Datos tipo cadena

Es una sucesión de caracteres que se encuentran delimitados por una comilla (apóstrofe) o dobles comillas, según el tipo de lenguaje de programación. La longitud de una cadena de caracteres es el número de ellos comprendidos entre los separadores o delimitadores.

Ejemplos:

'Hola mundo'

'12 de octubre de 1496'

'Enunciado cualquiera'

Lógicos

También se le denomina Booleano, es aquél dato que solo puede tomar uno de dos valores: Falso y verdadero.

Se utiliza para representar las alternativas (si/no) a determinadas condiciones. Por ejemplo, cuando se pide si un valor entero sea primo, la respuesta será verdadera o falsa, según sea.

Diagramas de Flujo de Datos (DFD)

Es un editor e intérprete de diagramas de flujo. Permite editar, ejecutar y depurar algoritmos representados como diagramas de flujo.

Este programa esta liberado bajo la licencia GPL, por lo tanto es software libre y no tiene restricciones para su uso bajo ninguna circunstancia, aunque se creó específicamente para el uso educativo.

Tipos de Datos en DFD

Real: Valores numéricos que van desde –1*10 ^ 2000 hasta 1*10 ^ 2000 . Los valores más cercanos a 0 que se pueden manejar son 1*10 ^ -2000 y –1*10 ^ -2000.

Ejemplo: 1998, 1.0007, 0, 328721, -3242781

Cadena de Caracteres: Secuencia de caracteres encerrada entre comillas simples.

Ejemplo: 'Diagramar es fácil' , 'París' , '1955'

Lógico: La letra V ó F encerrada entre puntos, para indicar verdadero ó falso respectivamente.

Ejemplo: .V. , .F. , .v. , .f.

PSeInt

PSeInt es principalmente un intérprete de pseudocódigo. El proyecto nació como trabajo final para la cátedra de Programación I de la carrera Ingeniería en Informática de la Universidad nacional del Litoral, Argentina.

Actualmente incluye otras funcionalidades como editor y ayuda integrada, generación de diagramas de flujo dado un algoritmo en pseudocódigo o exportación a código C++.
El proyecto se distribuye como software libre bajo licencia GPL y para descargarlo o conseguir actualizaciones se hacen desde esta página http://pseint.sourceforge.net

PSeInt, es una herramienta de software libre. El pseudocódigo se utiliza para introducir conceptos básicos como el uso de estructuras de control, expresiones, variables, etc, sin tener que lidiar con la sintaxis de un lenguaje real.
PSeInt facilita la escritura de algoritmos en pseudo lenguaje presentando un conjunto de ayudas y asistencias, y brindarle además algunas herramientas adicionales que ayudan a encontrar errores y comprender la lógica de los algoritmos.
El conjunto ordenado de pasos seguidos con el fin de resolver un problema o lograr un objetivo es conocido como algoritmo.

Tipos de Datos en PSeInt

Solo 3 tipos de datos básicos:

- **numérico,**
- **carácter/cadenas de caracteres y**
- **lógico (verdadero/falso).**

Tipos Simples en PSeInt:

Las dos acciones que pueden crear una variable son la lectura(LEER) y la asignación(<-). Por ejemplo, la asignación "A<-0;" está indicando implícitamente que la variable A será una variable numérica.

Existen tres tipos de datos básicos:

Numérico: números, tanto enteros como reales. Para separar decimales se utiliza el punto.
Ejemplos: 12 23 0 -2.3 3.14

- **Lógico**: Lógico o Booleano solo puede tomar dos valores: VERDADERO o FALSO.
- **Caracter:** caracteres o cadenas de caracteres encerrados entre comillas (pueden ser dobles o simples).
  Ejemplos 'hola' "hola mundo" '123' 'FALSO' 'etc'

2.4 Identificadores, constantes y variables

Identificadores

En la mayoría de los programas de computadora, es necesario manejar datos de entrada o de salida, los cuales necesitan almacenarse en la memoria principal del computador en el tiempo de ejecución. Para poder manipular dichos datos, necesitamos tener acceso a las localidades de memoria donde se encuentran almacenados; esto se logra por medio de los nombres de los datos o IDENTIFICADORES.

Los identificadores también se utilizan para los nombres de los programas, los nombres de los procedimientos y los nombres de las funciones, así como para las etiquetas, constantes y variables.

Hay reglas básicas para los identificadores que siempre debes seguir:

- No se puede usar una palabra reservada como idenficador
- Dos elementos distintos no pueden tener el mismo identificador
- Un identificador siempre comienza con una letra
- Un identificador no puede tener espacios en blanco
    - Escribe todo junto o usa guiones bajos _ en vez de los espacios
- Un identificador sólo puede tener:
    - letras
    - dígitos
    - guión bajo_

Constante

Una constante es un valor que no puede ser alterado durante la ejecución de un programa.

Una constante corresponde a una longitud fija de un área reservada en la memoria principal del ordenador, donde el programa almacena valores fijos.

Por ejemplo:

El valor de pi = 3.1416

Constantes en DFD

Con su nombre muestran su valor y éste no se puede cambiar.

Ejemplo: 1996 , 'Los algoritmos son útiles' , .V.

Constantes en PSeInt

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

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

Las constantes lógicas son Verdadero y Falso.

Variable

Una variable está formada por un espacio en el sistema de almacenaje (memoria principal de un ordenador) y un nombre simbólico (un identificador) que está asociado a dicho espacio. Ese

espacio contiene una cantidad o información conocida o desconocida, es decir un valor. El nombre de la variable es la forma usual de referirse al valor almacenado: esta separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa. El identificador, en el código fuente de la computadora puede estar ligado a un valor durante el tiempo de ejecución y el valor de la variable puede por lo tanto cambiar durante el curso de la ejecución del programa. En computación una variable puede ser utilizada en un proceso repetitivo: puede asignársele un valor en un sitio, ser luego utilizada en otro, más adelante reasignársele un nuevo valor para más tarde utilizarla de la misma manera. Procedimientos de este tipo son conocidos con el nombre de iteración. En programación de computadoras, a las variables, frecuentemente se le asignan nombres largos para hacerlos relativamente descriptivas para su uso, mientras que las variables en matemáticas a menudo tienen nombres escuetos, formados por uno o dos caracteres para hacer breve en su transcripción y manipulación.

Variables con DFD: Es posible modificar su valor. El nombre de una variable debe comenzar por una letra seguida de letras, números o el caracter (_).

Ejemplo: Valor , Contador , año , Valor_1

No se tiene en cuenta la diferencia entre mayúsculas y minúsculas para el nombre de una variable; es decir, CASA equivale a casa. Cuando una variable recibe un valor por primera vez, el tipo de dato de ésta será igual al tipo de dato del valor.

Variables con PSeInt:

En la mayoría de los lenguajes reales los nombres de variables no pueden contener acentos, ni diéresis, ni eñes(ñ), en PSeInt, esto se permite si se activa la Sintaxis Flexible (ver Opciones del PSeudocódigo). En algunos lenguajes se puede guardar cualquier información en cualquier variable, mientras en otros las variables solo pueden guardar cierto tipo de información. En PSeInt las variables tienen un tipo de dato asociado, por lo que durante la ejecución del algoritmo una variable deberá guardar datos siempre del mismo tipo.
Por ejemplo, si una variable se utiliza para guardar números, no puede utilizarse después para guardar texto. Este tipo se puede declarar explícitamente con la palabra clave Definir, o se puede dejar que el intérprete intente deducirlo a partir de los datos que se guardan en la misma y la forma en que se la utiliza en el algoritmo. Si utiliza el perfil de lenguaje por defecto (Flexible), la definición explicita es opcional, pero se puede configurar el lenguaje para que la misma sea obligatoria.
Hay dos formas de crear una variable y/o asignarle un valor:
La lectura y la asignación. Si se lee o asigna un valor en una variable que no existe, esta se crea.
Si la variable ya existía, esta toma el nuevo valor, perdiendo el viejo.
Por esto se dice que la asignación y la lectura son acciones destructivas (aunque se debe notar que en la asignación pueden intervenir más de una variable, y solo se destruye el contenido previo de la que se encuentra a la izquierda del signo de asignación).
Una vez inicializada, la variable puede utilizarse en cualquier expresión (para realizar un cálculo en una asignación, para mostrar en pantalla, como condición en una estructura de control, etc.)
Los tipos de datos son determinados automáticamente cuando se crean las variables o se les asigna un valor. Este tipo de dato deberá permanecer constante durante todo el proceso, si no es así el proceso será interrumpido.

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

<variable>=<expresión> ;

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

Ejemplo num=1

2.5 Operaciones aritméticas

Las operaciones aritméticas son las que operan sobre valores numéricos y entregan otro valor numérico como resultado. Los valores numéricos son los que tienen tipo entero, real o complejo.

Operadores de Asociatividad con DFD

( ) : Los paréntesis modifican la secuencia de evaluación de una expresión.

Ejemplo :
3 * 2 + 5 da como resultado 11.
3 * (2+5) da como resultado 21.

Operadores de Cadenas de Caracteres con DFD

El operador (+) concatena dos cadenas de caracteres.

Ejemplo :

'Diagramar' + ' es fácil' da como resultado 'Diagramar es fácil'

Operadores matemáticos con DFD

Suma

Sintaxis X+Y
Entrada X, Y Valores de tipo Real.
Resultado La suma de X e Y.

Resta

Sintaxis X-Y
Entrada X, Y Valores de tipo Real.
Resultado La resta de X e Y.

Exponenciación

Sintaxis X^Y

Entrada X,Y Valores de tipo Real.
Resultado Valor de X elevado a la potencia Y.

Multiplicación

Sintaxis X*Y
Entrada X, Y Valores de tipo Real.
Resultado X Multiplicado por Y.

División

Sintaxis X/Y
Entrada X,Y Valores de tipo Real.
Resultado X dividido entre Y.

Módulo

Sintaxis X MOD Y
Entrada X, Y Valores de tipo Real.
Resultado El residuo de dividir X entre Y, definido como un valor

R, tal que : X= Y*K+R, donde K es un entero y ABS(R) < ABS(Y).

Operadores con PSeInt: Este pseudolenguaje dispone de un conjunto básico de operadores que pueden ser utilizados para la construcción de expresiones más o menos complejas. Las siguientes tablas exhiben la totalidad de los operadores de este lenguaje reducido:

Operaciones aritméticas con PSeInt:

La jerarquía de los operadores matemáticos es igual a la del álgebra, aunque puede alterarse mediante el uso de paréntesis.
Para el caso de los operadores & y |, la evaluación se realiza en cortocircuito. Esto significa que si dos expresiones están unidas por el operador ? y la primera se evalúa como Falso, o están unidas por el operador | y la primera se evalúa como Verdadero, la segunda no se evalúa ya que no altera el resultado.

Operador	Significado	Ejemplo
Algebraicos		
+	Suma	total <- cant1 + cant2
-	Resta	stock <- disp - venta
*	Multiplicación	area <- base * altura
/	División	porc <- 100 * parte / total
^	Potenciación	sup <- 3.41 * radio ^ 2
% ó MOD	Módulo (resto de la división entera)	resto <- num MOD div

Este es el ejemplo más simple. Muestra cómo cargar dos números de dos variables, calcular la suma de los mismos y mostrarla en pantalla.

2.5.1 Ejercicios de Tipos de Datos en ambas herramientas

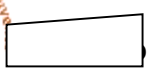
EJECICIO 1. Temas: Variables y expresiones aritméticas. Escribir el resultado final de las siguientes expresiones

Expresión	**resultado**	Expresión	**resultado**	Expresión	**Resultado**	Expresión	**Resultado**
Round(5/2)		5/2.0		10.0/3.0		15.0/2	
5 mod 2		10 mod 2		2 mod 4		14 mod 0	

Expresión	**Resultado**	Expresión	**Resultado**
2+3*4		2*3+4	
(2+3)*4		2*(3+4)	
6+1/2*4-1*10		1*3/3+8%8-2	
1.1+2+3/3		**4.1+2/3+3**	
2. 8*8+5/2+1*34		**5.17+9/2.0**	
3. 2*2*(43+7)%2-1		**6.4%23.123**	

2.6 Operaciones de Entrada y Salida de datos

En **DFD** para leer y escribir datos se usan los siguientes objetos:



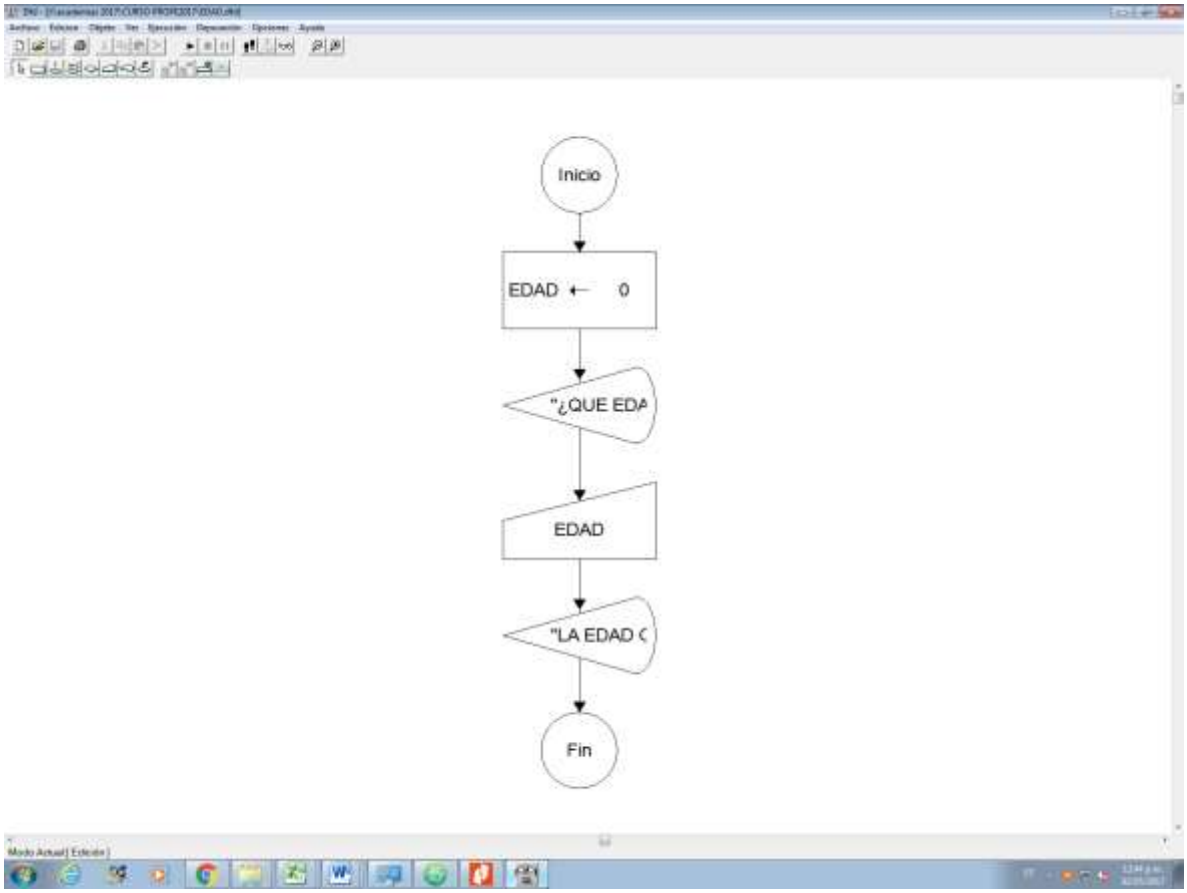
Lectura

El objeto Lectura permite la entrada de valores constantes desde el teclado y se los asigna a campos variables . Podrá ser leída cualquier cantidad de variables utilizando un objeto Lectura. Al ejecutarse, el objeto despliega un cuadro de diálogo por cada variable presente en la lista, este cuadro de diálogo espera que el usuario introduzca un valor constante que será asignado a la respectiva variable.
El cuadro de diálogo para la edición del objeto contiene un espacio para ingresar una lista de variables separadas por comas. Debe existir por lo menos una variable.

Objeto Salida

El objeto Salida muestra valores por pantalla. Puede ser visualizada cualquier cantidad de valores utilizando un objeto Salida. Al ejecutarse, este objeto evalúa cada una de las expresiones que contiene y despliega un cuadro de diálogo que muestra el valor obtenido en cada una de las expresiones en su respectivo orden.
El cuadro de diálogo para la edición del objeto contiene un espacio para ingresar una lista de expresiones separadas por comas. Debe existir por lo menos una expression.

Ejemplo con DFD


Inicio
EDAD ← 0
"¿QUE EDA
EDAD
"LA EDAD (
Fin

Para la lectura y escritura de datos en PSeInt se usa lo siguiente:

La instrucción **Leer** permite ingresar información desde el ambiente.

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

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

La instrucción **Escribir** permite mostrar valores al ambiente.

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

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

Ejemplo con PSeInt

```
Escribir "Que edad tienes "
Leer edad
Escribir “La edad que tienes es: ”, edad
```

2.7 Operaciones relacionales

Las operaciones relacionales sirven para comparar valores. Sus operandos son cualquier cosa que pueda ser comparada, y sus resultados siempre son valores lógicos.

Sirven para realizar comparaciones. El resultado de estos operadores es verdadero o falso (uno o cero).

Operaciones relacionales con DFD

Mayor que

Sintaxis X>Y
Entrada X,Y Valores del mismo tipo de dato y que no sean de tipo Lógico.
Resultado .V. (verdadero) si X>Y y .F. (falso) en caso contrario.

Menor que

Sintaxis X<Y
Entrada X,Y Valores del mismo tipo de dato y que no sean de tipo Lógico.
Resultado .V. (Verdadero) si X menor que Y .F. (Falso) en caso contrario.

Menor o Igual a

Sintaxis X<=Y
Entrada X, Y Valores del mismo tipo de dato y que no sean de tipo Lógico.

Resultado .V. (Verdadero) si X menor igual a Y y .F. (Falso) en caso contrario.

Mayor o Igual a

SintaxisX>=Y

Entrada X,Y Valores del mismo tipo de dato y que no sean de tipo Lógico.
Resultado .V. (Verdadero) si X es mayor o igual a Y y .F. (Falso) en caso contrario.

Igual a

Sintaxis X =Y
Entrada X, Y Valores del mismo tipo de dato.
Resultado .V. (Verdadero) si X es igual a Y y .F. (Falso) en caso contrario.

Diferente de

Sintaxis X !=Y
Entrada X, Y Valores del mismo tipo de dato.
Resultado .V. (Verdadero) si X es diferente de Y y .F. (Falso) en caso contrario.

Operaciones relacionales con PSeInt

Operador	*Significado*	*Ejemplo*
Relacionales		
>	Mayor que	3>2
<	Menor que	'ABC'<'abc'
=	Igual que	4=3
<=	Menor o igual que	'a'<='b'
>=	Mayor o igual que	4>=5
< >	Distinto que	Var1<>var2

2.8 Operaciones lógicas

Los operadores lógicos se utilizan con expresiones para devolver un valor verdadero o falso (true o false). Se denominan también operadores booleanos.

Operaciones lógicas con DFD

Y Lógico (Conjunción)

Sintaxis X AND Y
Entrada X, Y Valores de tipo de dato Lógico.

Negación del AND

Sintaxis X NAND Y
Entrada X, Y Valores de tipo de dato Lógico.

O Lógico

Sintaxis X OR Y
Entrada X, Y Valores de tipo de dato Lógico.

Negación del OR
Sintaxis X NOR Y
Entrada X, Y Valores de tipo de dato Lógico.

Negación Lógica
Sintaxis NOT X
Entrada X Valor de Tipo de dato Lógico.

Operaciones lógicas con PSeInt

Operador	*Significado*	*Ejemplo*
Logicos		
& ó Y	Conjunción (y).	(7>4) & (2=1) //falso
| ó O	Disyunción (o).	(1=1 | 2=1) //verdadero
~ ó NO	Negación (no).	~(2<5) //falso

2.9 Algoritmo

Un algoritmo es conjunto ordenado y finito de pasos u operaciones que permite encontrar la solución de un problema.

Un algoritmo es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos, además es una serie de operaciones detalladas que se pueden formular de muchas formas con el cuidado de que no exista ambigüedad.

Existen dos tipos de algoritmos:

• Cualitativos: Son aquellos en los que se describen los pasos utilizando palabras.

• Cuantitativos: Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

Formalmente se define un algoritmo como un conjunto de pasos, procedimientos o acciones que permiten alcanzar un resultado o resolver un problema.

Un algoritmo tiene las siguientes características:

1. Preciso: debe indicar el orden de realización en cada paso y no puede tener ambigüedad.

2. Definido: si se sigue dos veces o más, se obtiene el mismo resultado cada vez.

3. Finito: tiene fin, es decir, un número determinado de pasos.

4. Correcto.

5. Debe tener al menos una salida y ésta debe ser tangible.

6. Debe ser sencillo y legible.

7. Eficiente y efectivo.

8.- Se ha de desarrollar en el menor tiempo posible

Ejemplo de algoritmos
Ejemplo 1.1 Problema: Se desea realizar el cálculo de la velocidad de un automóvil que recorre una distancia x en un cierto tiempo t. (Emplear la fórmula $v=\frac{x}{t}$).
a) Análisis del problema:

• Como datos de entrada se necesita el valor de la distancia (x) y el valor del tiempo (t)

• En las restricciones se observa que el tiempo no puede ser nulo pues se indetermina la operación, tampoco puede ser negativo. La distancia puede ser positiva o negativa, ya que el automóvil puede ir retrocediendo, pero no puede ser nula porque el problema indica que el automóvil recorre una cierta distancia.

Finalmente se obtiene como resultado la velocidad tras aplicar la fórmula $v=\frac{x}{t}$ donde la velocidad puede aumentar (si el resultado es positivo) o disminuir (si el resultado es negativo).

1. Inicio.

2. Leer el valor de x.

3. Si x=0 regresar al paso 2, en caso contrario ir al paso 4.

4. Leer el valor de t.

5. Si t ≤ 0 entonces ir al paso 7, en caso contrario ir al paso 6.

6. Realizar $v=\frac{x}{t}$.

7. Fin.

Prueba de escritorio

Dados los valores x=3, t=0.

1. Inicio.

2. x=3.

3. ¿x=0? NO, al paso 4.

4. t=0.

5. ¿t ≤ 0? SÍ, ir al paso 7.

7. Fin.

Dados los valores x=5, t=7.

1. Inicio.

2. x=5.

3. ¿x=0? NO, al paso 4.

4. t=7.

5. ¿t ≤ 0? NO, ir al paso 6.

6. $v = \frac{5}{7} - 0.715$

8. Fin.

2.10 Diagrama de flujo

Es la representación gráfica de las operaciones de un algoritmo. Contiene símbolos gráficos que se encuentran estandarizados. Los símbolos más comunes son:

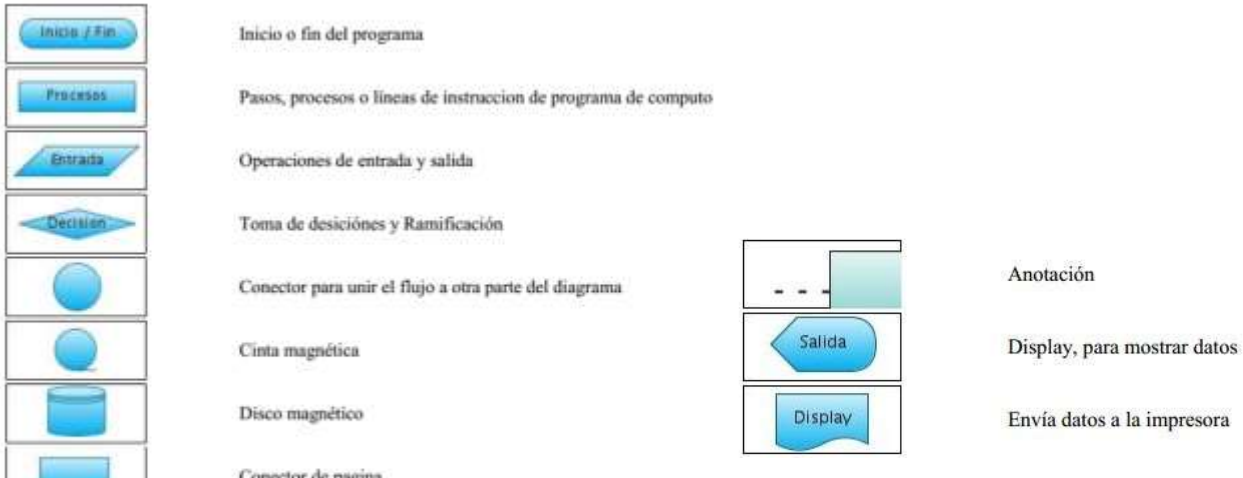


Figure 4 Simbología

2.10.1 Ventajas de los Diagramas de Flujo

• Favorecen la comprensión del proceso a través de mostrarlo como un dibujo. El cerebro humano reconoce fácilmente los dibujos. Un buen diagrama de flujo reemplaza varias páginas de texto.

• Permiten identificar los problemas y las oportunidades de mejora del proceso. Se identifican los pasos redundantes, los flujos de los reprocesos, los conflictos de autoridad, las responsabilidades, los cuellos de botella, y los puntos de decisión.

• Muestran las interfaces usuario-desarrollador y las tareas que en ellas se realizan, facilitando a los desarrolladores el análisis de las mismas.

• Son una excelente herramienta para capacitar a los nuevos empleados usuarios y también a los que desarrollan la tarea, cuando se realizan mejoras en el proceso.

2.11 Pseudocódigo

Es un lenguaje de especificación de algoritmos. El uso de tal lenguaje hace el paso de codificación final (esto es, la traducción a un lenguaje de programación) relativamente fácil.

El pseudocódigo nació como un lenguaje similar al inglés y era un medio representar básicamente las estructuras de control de programación estructurada. Se considera un primer borrador, dado que el pseudocódigo tiene que traducirse posteriormente a un lenguaje de programación. Cabe señalar que el pseudocódigo no puede ser ejecutado por una computadora.

La ventaja del pseudocódigo es que en su uso en la planificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje específico. Es también fácil modificar el pseudocódigo si se descubren errores o anomalías en la lógica del programa, además de todo esto es fácil su traducción a lenguajes como pascal, COBOL, C, FORTRAN o BASIC.

El pseudocódigo utiliza para representar las acciones sucesivas palabras reservadas en inglés (similares a sus homónimos en los lenguajes de programación), tales como star,begin, end, stop, if- then-else, while, repeat-until….etc.

Elementos básicos de un programa y su entorno

Es recomendable seguir un esqueleto de programa como el siguiente:

1. ACCESO A BIBLIOTECAS

2. DECLARACIÓN DE CONSTANTES SIMBÓLICAS

3. DECLARACIÓN DE VARIABLES GLOBALES

4. DECLARACIÓN DE PROTOTIPOS DE FUNCIONES

5. PROGRAMA PRINCIPAL

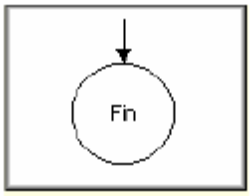
6. CÓDIGO DE LAS FUNCIONES

MÓDULO III. ELEMENTOS DEL ENTORNO DE LAS HERRAMIENTAS DFD Y PSeInt

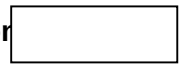
3.1 Diagrama de flujo con DFD

3.1.1 Simbologías

Objeto de Inicio: Es el primer objeto a ejecutar en cualquier algoritmo. Al ser ejecutado, el objeto Inicio transfiere el control al siguiente objeto.

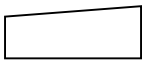


Objeto Asignación



El objeto Asignación asigna valores a campos variables. Al ser ejecutado, puede realizar hasta tres asignaciones.
El cuadro de dialogo del objeto Asignación contiene espacio para tres asignaciones, cada asignación consta de un espacio para el campo variable situado siempre a la izquierda, el símbolo de asignación y un espacio para la expresión situada siempre a la derecha. Esto indica que al campo variable se le asigna el resultado de la evaluación de la expresión. Debe realizarse por lo menos una asignación.

Objeto Lectura



El objeto Lectura permite la entrada de valores constantes desde el teclado y se los asigna a campos variables.

Objeto Salida

El objeto Salida muestra valores por pantalla. Puede ser visualizada cualquier cantidad de valores utilizando un objeto Salida.

Objeto Decisión

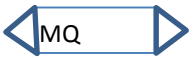
El objeto decisión selecciona el flujo a seguir de acuerdo al valor lógico de una condición. La condición debe ser siempre una expresión que al ser evaluada de como resultado un valor de tipo de dato Lógico.
Ejemplo : 3<w , x >0 AND sw = .V. , valor*15 < 300*contador.

El objeto Decisión está asociado a dos bloques de objetos ubicados a lado y lado de este, y un objeto Cierre Decisión ubicado a continuación de ambos bloques.
Si al evaluar la condición se obtiene el valor lógico .V., se ejecuta el bloque rotulado con la palabra Si, en caso contrario se ejecuta el bloque rotulado con No. En ambos casos la ejecución continúa en el objeto Cierre Decisión.

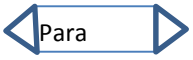
El cuadro de dialogo del objeto Decisión contiene espacio para la expresión que conforma la condición, y dos casillas por medio de las cuales se puede especificar por cual lado continuara el flujo en caso de que la condición sea verdadera.

Objeto Cierre Decisión

Este objeto delimita el cuerpo de una estructura de decisión, al culminar la ejecución de dicha estructura el control se transfiere al objeto que sigue al objeto Cierre Decisión.

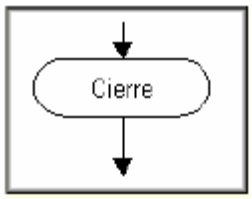
Objeto Ciclo Mientras 

El objeto Ciclo Mientras tiene como función el ejecutar un bloque de objetos mientras que una condición sea verdadera. La condición debe ser siempre una expresión que al ser evaluada de como resultado un valor de tipo de dato Lógico.
Ejemplo : 3<W , x >0 AND Sw = .V. , Valor*15 < 300*Contador.
Si al evaluar la condición se obtiene el valor .F. la ejecución del algoritmo continuará a partir del objeto que sigue al Cierre.
El cuadro de dialogo del objeto Ciclo Mientras contiene espacio para la expresión que conforma la condición.

Objeto Ciclo Para 

Su función es ejecutar un bloque de objetos mientras que la variable contadora no alcance el límite establecido por el valor final. El contador es siempre una variable de tipo de dato Real. Contiene además un valor inicial que será asignado al contador al iniciar la ejecución del ciclo, un valor final y un valor de incremento. Si el contador excede el valor final, la ejecución continuará a partir del objeto que sigue al Cierre. En caso contrario, se ejecutará el cuerpo del ciclo y el contador será incrementado en el valor indicado por el incremento.
El cuadro de diálogo del objeto para contiene espacio para la variable contador, valor inicial, valor final y el valor de incremento en su respectivo orden.

Objeto Cierre: Este objeto junto con el objeto Inicio, delimita el cuerpo del procedimiento principal. Solo existe un objeto Fin en el diagrama; la ejecución de este objeto finaliza la ejecución del algoritmo.



3.2 Barra de Herramientas de DFD

La barra de herramientas contiene los siguientes menus:

Sistemas de Menús

Menú Archivo: integra actividades que se pueden realizar con archivos como son: nuevo, abrir, guardar, guardar como, imprimir y salir
Menú Edición: integra acciones de edición de texto y objetos como son: cortar, pegar, copiar, eliminar entre otros.
Menú Objeto: contiene la simbología de objetos para integrar los diagramas de flujo
Menú Ver: zoom, anterior Subprograma, siguiente Subprograma y Depurador
Menú Ejecución: contiene objetos que nos permiten ver los resultados que se obtienen del diseño de diagrama de flujo.
Menú Depuración: contiene objetos que permiten depurar un diagrama de flujo estableciendo limites o detener.
Menú Opciones: El submenú Ángulos contiene los comandos Ángulos en Grados y Ángulos en Radianes, los cuales determinan las unidades en que serán expresados los ángulos (grados ó radianes respectivamente). Tomando en cuenta que: 1 grado = PI / 180 radianes

A continuación se describen algunos de estos menús:
Menú Objeto

Los objetos de FreeDFD, son los que aparecen en dicha sección de la barra de herramientas.

Objetos DFD

El primer botón, se denomina **Cursor**, cuando este botón se encuentra activado, se pueden hacer selecciones en el área de edición del programa.

El segundo botón, se denomina **Asignación**, y sirve para hacer definición de variables en el programa de manera estática.

El tercer botón, se denomina **Ciclo Mientras**, sirve para crear una estructura repetitiva dentro del programa y ejecutar una serie de instrucciones muchas veces seguidas, mientras se considere una condición como verdadera.

El cuarto botón, se denomina **Ciclo Para**, sirve para crear del mismo modo que el ciclo mientras una estructura repetitiva dentro del programa, para que se repitan una serie de instrucciones, solo que la condición es diferente.

El quinto botón, se denomina **Decisión**, sirve para tomar decisiones simples dentro del programa, se ingresa dentro de la "decisión", una condición y dependiendo de si esa

condición es falsa o verdadera, el programa ejecutara una serie de instrucciones diferentes en cada caso.

El sexto botón, se denomina **Lectura o Entrada**, sirve para realizar definiciones de variables de manera dinámica, en el momento en que el programa esta en ejecución.

El séptimo botón, se denomina **Salida**, sirve para mostrar salidas por pantalla de los diferentes procesos que se realizan con el programa, es el único medio que tenemos para obtener resultados.

El octavo botón, se denomina **Llamada**, sirve para hacer llamadas a otros subprogramas.

Menú Ejecución

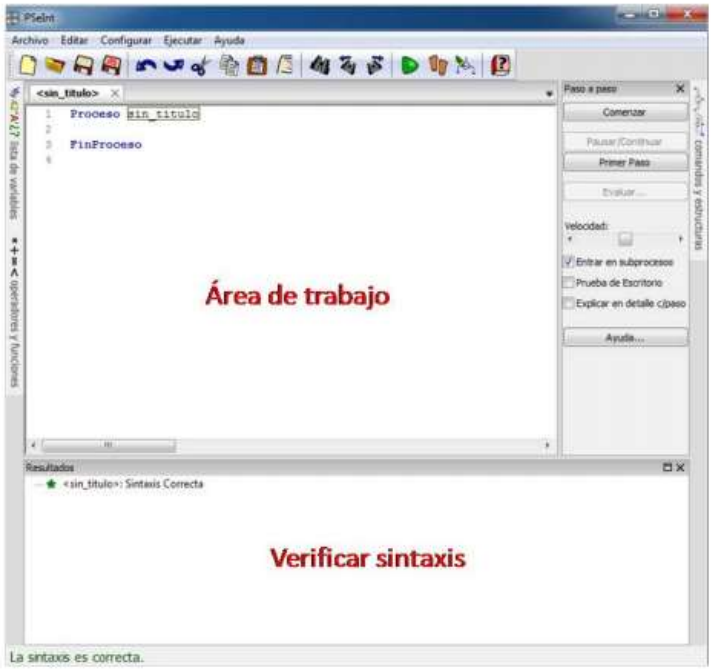
Ejecución | Ejecutar ( F9)
Este comando coloca la acción actual en Ejecución y da paso a la ejecución del algoritmo.
En caso de que la acción actual sea Edición, se realizará primero una revisión del diagrama para encontrar errores de sintaxis.
En caso de que un error sea encontrado, un mensaje será desplegado indicando el tipo de error y el objeto en el que se presentó. Si el diagrama está libre de errores, se ejecutará el algoritmo a partir del objeto Inicio.

Ejecución | Pausar ( PAUSA)
Este comando hace una pausa en la ejecución del algoritmo colocando la acción actual en Depuración Paso a Paso.
El comando solo estará disponible cuando la acción actual sea Ejecución.

Ejecución | Detener ( CTRL + PAUSA)
Este comando coloca la acción actual en Edición, deteniendo la ejecución ó depuración del algoritmo. Este comando estará disponible cuando la acción actual sea diferente de Edición.

3.3 Barra de Herramientas de PSeInt

La interfaz y el área de trabajo



← Barra de Menú
← Barra de Acceso rápido

Las funciones: botones

Abre un nuevo documento

Busca un fichero (archivo)

Guardar y guardar como

Deshacer y Rehacer respectivamente

Cortar

Copiar y pegar

Corregir indentado

Buscar

Ejecutar el algoritmo

Ejecutar paso a paso

Dibujar diagrama de flujo

Ayuda/contiene algunos ejemplos

MÓDULO V. SENTENCIAS CONDICIONALES Y/O SELECTIVAS

Las estructuras selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelan denominar también estructuras de decisión o alternativas.

La computadora tiene la capacidad de tomar decisiones es decir, de hacer pruebas para escoger el curso adecuado de acción según el resultado de esas pruebas. A esta capacidad se le llama selección. Supóngase por ejemplo, que las pelotas de béisbol tienen un precio unitario de $6.50 (pesos) cuando se compra al menos diez, y de $7.00 si se compran menos de diez. Para calcular el costo de una compra, la computadora debe examinar si se están comprando al menos diez pelotas, para escoger la fórmula adecuada. La computadora efectúa estas selecciones evaluado expresiones booleanas esto es, expresiones que tienen valores de true (verdadero) o false (falso)

5.1 Sentencias Simples

La selección simple se usa cuando se quiere que la computadora efectúe una acción condicionalmente es decir, sólo cuando cierta condición es verdadera.

FORMATO GENERAL PARA LA ESTRUCTURA DE DECISION SIMPLE:

Si entonces

```
Si (CONDICION) entonces
      sentencia;
```

```
Si (CONDICION) entonces

   Sentencia 1;
   Sentencia 2;
      :
      .
   Sentencia  N;
```

Si la condición es verdadera, la computadora ejecuta el enunciado de acción y continúa después con el siguiente en el programa. Si la condición es falsa, la computadora continúa directamente con la siguiente instrucción del programa. Por ejemplo, se permiten los siguientes enunciados:

Si (edad >= 18) entonces escribe ‘puede entrar’

Si (a+b >= c) entonces contador ← contador + 1

La representación gráfica de esta estructura se puede observar en la siguiente figura:

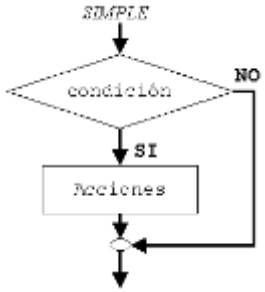


Figura 6 Sentencia simple

Observemos la figura:

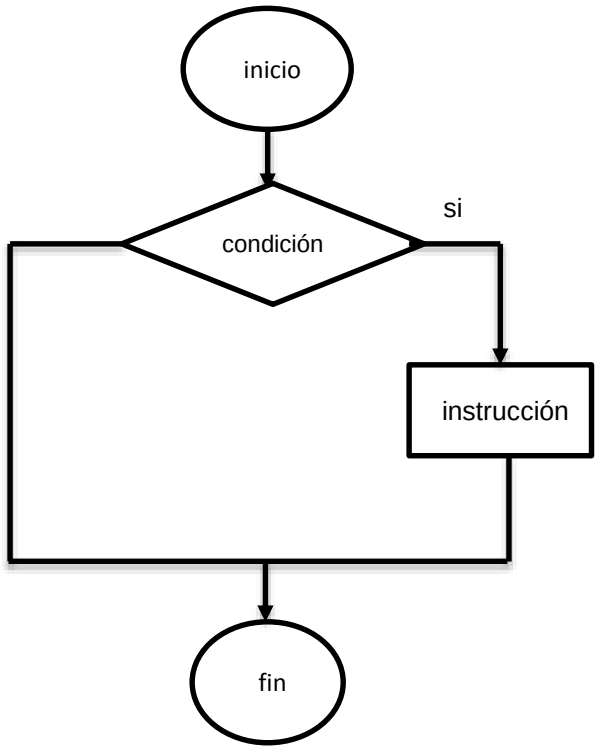
Las flechas representan en qué sentido va ejecutándose el programa.

El rombo simboliza la evaluación de la condición lógica. Observe que de dicho rombo salen dos flechas, etiquetadas como Sí y No. El programa saldrá de ese rombo por una u otra según el resultado de la condición del rombo.

La caja cuadrada representa la sentencia (o grupo de sentencias) que se ejecutan si la condición es verdadera.

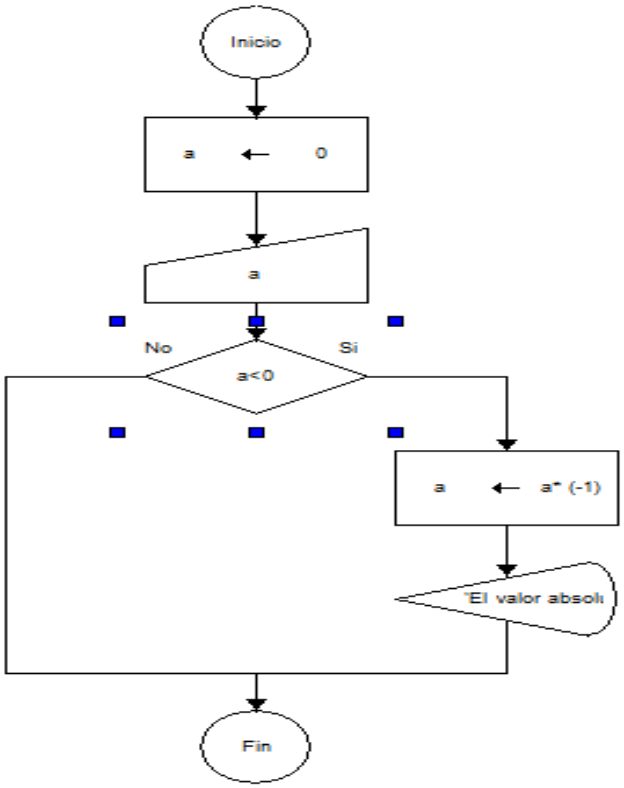
5.2 Estructura de decisión simple con DFD

En DFD la estructura de decisión simple se muestra en la siguiente figura:



Ejemplo:

LEER UN NUMERO E IMPRIMIR EL VALOR ABSOLUTO DE ESTE NUMERO .



5.3 Sentencias Compuestas

Cuando se presenta la elección tenemos la opción de realizar una actividad u otra. Es decir tenemos actividades por el verdadero y por el falso de la condición. Lo más importante que hay que

tener en cuenta que se realizan las actividades de la rama del verdadero o las del falso, NUNCA se realizan las actividades de las dos ramas.

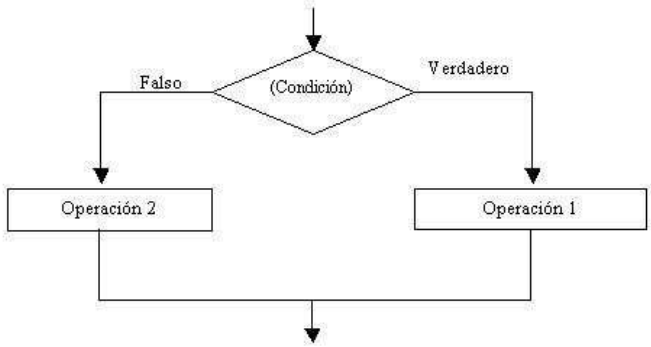


Figura 7 Estructura condicional compuesta

SELECCIÓN ENTRE DOS ALTERNATIVAS

FORMATO GENERAL PARA LA ESTRUCTURA DE DECISION COMPUESTA:

```
Si (CONDICION) entonces
   sentencia 1;
   sentencia 2;
```

```
Si (CONDICION) entonces
        Sentencia 1;
        Sentencia 2;
:
.
        Sentencia  N;
 de lo contrario
      Otras Sentencias;
:
.
```

En la selección simple la computadora hace algo o no hace nada, dependiendo del resultado de la prueba. En la selección doble, la computadora efectúa una prueba y después realiza algo en cualquier caso. Si la condición es verdadera, la computadora ejecuta la sentencia 1. Si es falsa, la computadora ejecuta la alternativa de lo contrario.

PREGUNTA: Cuando se ejecuta el fragmento

```
si (calif >= 70) entonces
  imprime 'aprobado'
de lo contrario
    imprime 'reprobado'
```

¿qué se imprime para cada valor dado de calif?

(a) calif 58 (b) calif ← 74

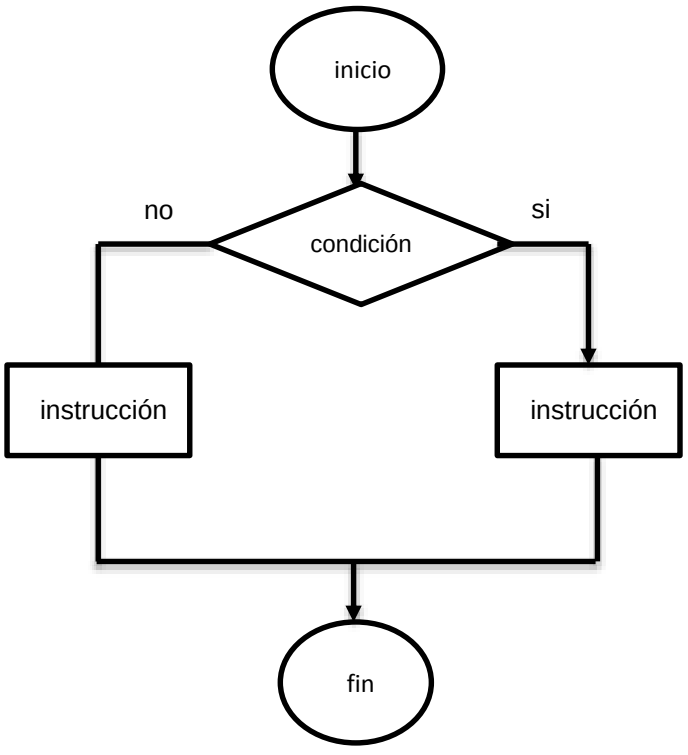
PREGUNTA: Supóngase que el precio unitario de las pelotas de béisbol es de $ 6.50 (pesos) si se compran diez o más, y de $ 7.00 en caso contrario. Complétese la prueba correspondiente si entonces del siguiente pseudocódigo

Imprime 'escriba el número comprado de pelotas: '
Leer número
Si (_____) entonces
costo ← numero*6.5

imprime 'pelotas cuestan ', numero, costo

5.4 Estructura de decisión compuesta con DFD

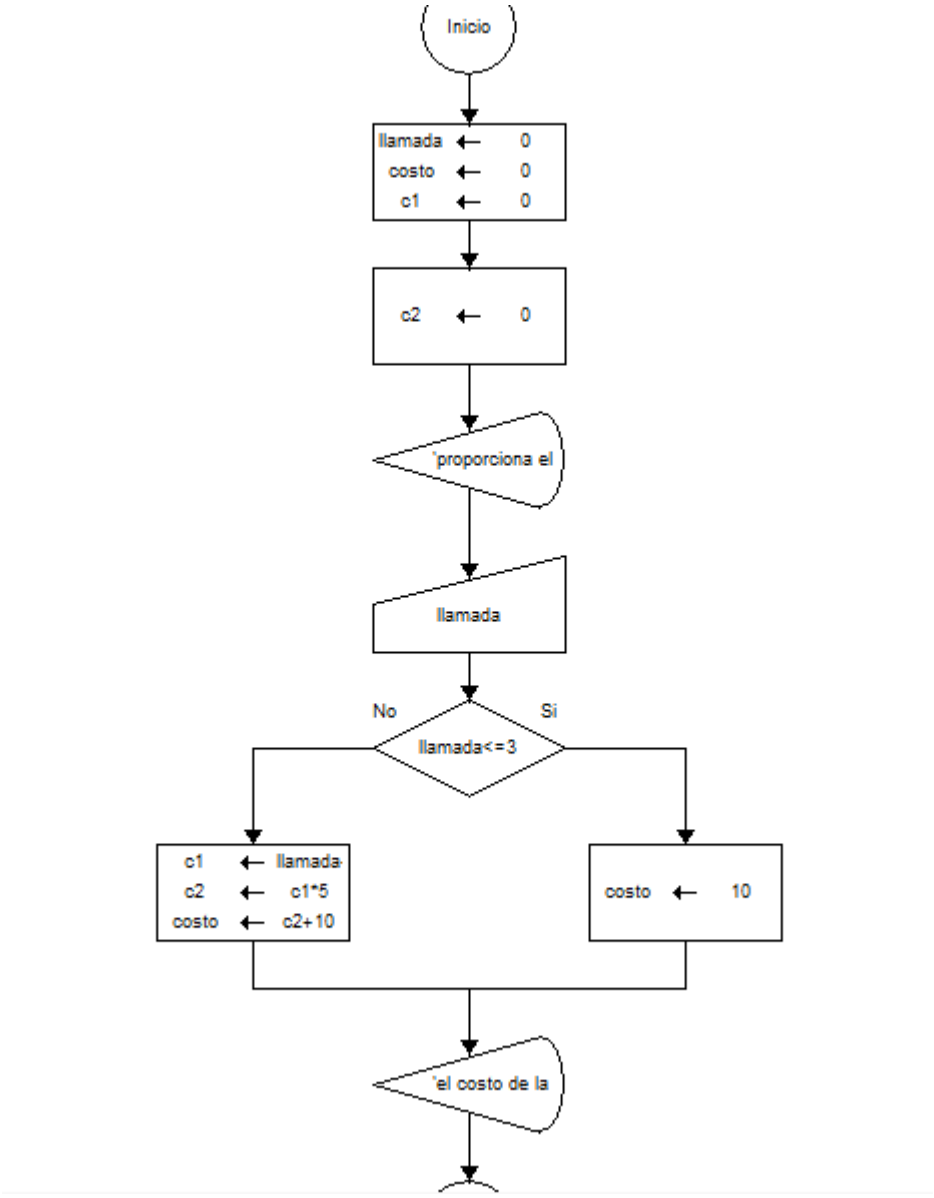
En DFD la estructura de decisión compuesta se muestra en la siguiente figura:



Ejemplo:

HACER UN DIAGRAMA DE FLUJO QUE DETERMINE E IMPRIMA LA CANTIDAD TOTAL A PAGAR POR UNA LLAMADA TELEFÓNICA TENIENDO EN CUENTA LO SIGUIENTE:

A TODA LLAMADA QUE DURE MENOR O IGUAL A 3 MINUTOS TIENE UN COSTO DE 10 PESOS, CADA MINUTO ADICIONAL A PARTIR DE LOS 3 PRIMERO CUESTA 5 PESOS.



Ejercicios de estructuras de decisión compuesta con DFD

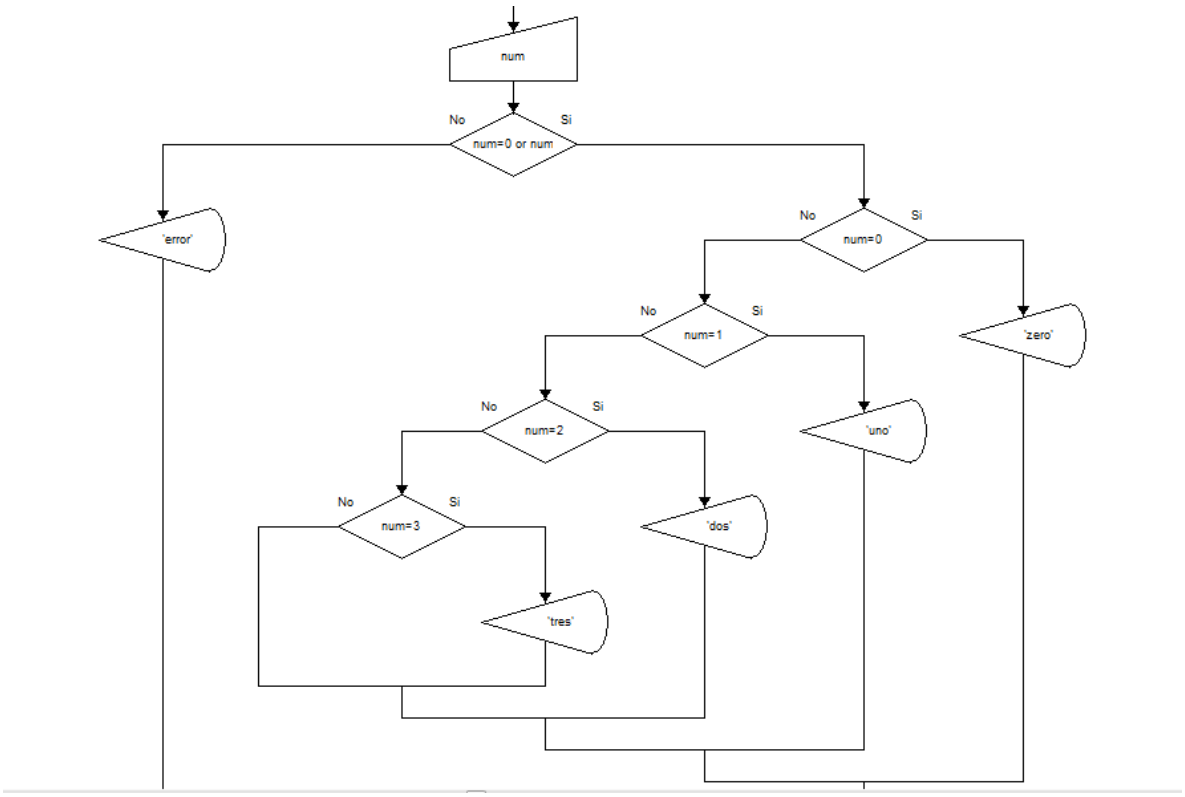
Av. Universidad 1001, Chamilpa, Cuernavaca, Morelos, México,
C.P. 62209 Tel. (777) 329 7000 y 329 7041 Ext.3148

5.4 SI anidados y estructuras selectivas múltiples

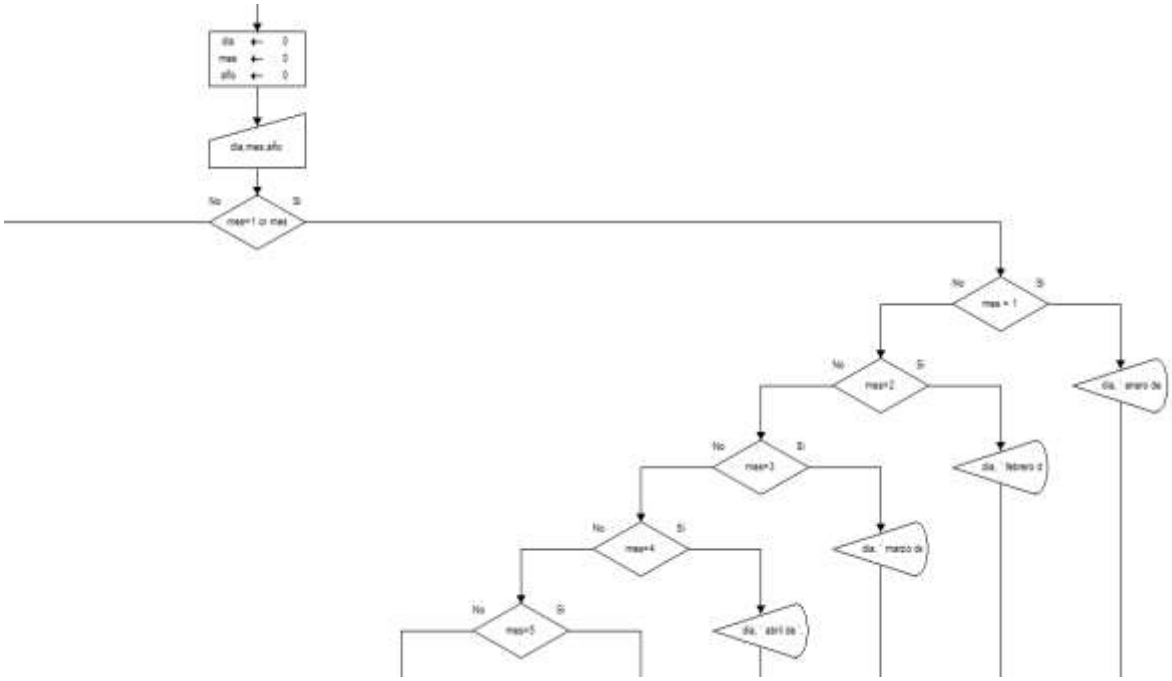
En algunos casos puede requerirse que la computadora escoja de una lista de más de dos opciones

Si nota>=85 entonces imprime 'bueno'
de lo contrario
si (nota<85 AND nota>=70) entonces imprime 'regular'
de lo contrario
si nota<70 entonces imprime 'malo'

LEER UN NUMERO ENTRE EL RANGO DEL 0 AL 3 E IMPRIMIR EL NUMERO EN INGLES, SI EL NUMERO ESTA FUERA DE ESTE RANGO MARCAR UN ERROR.



HACER UN DIAGRAMA DE FLUJO QUE LEA EL DÍA, EL MES Y EL AÑO LOS TRES DEBEN SER DE TIPO NUMÉRICO E IMPRIMIR LA FECHA CON EL SIGUIENTE FORMATO. Por ejemplo: salida del diagrama de flujo: 4 MAYO DE 1992



Ejercicios de estructuras de decisión múltiple con DFD

5.5 Estructura de decisión Simple con PSeInt

Son Estructuras de Control (Proceso) que se dividen en: condicionales y repetitivas.

Condicionales

- **Si-Entonces**

Si-Entonces (If-Then)

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

Si <condición>Entonces
<instrucciones>
Sino

<instrucciones>
FinSi

Al ejecutarse esta instrucción, se evalúa la condición y se ejecutan las instrucciones que correspondan: las instrucciones que le siguen al Entonces si la condición es verdadera, o las instrucciones que le siguen al Sino si la condición es falsa.

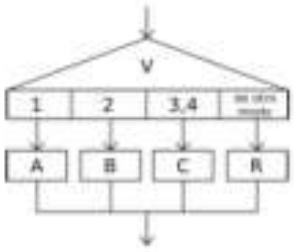
La condición debe ser una expresión lógica, que al ser evaluada retorna Verdadero o Falso.

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

Ejercicios de estructura de decisión simple con PSeInt

5.6 Selección Múltiple con PSeInt

Selección Múltiple (Select If)
La secuencia de instrucciones ejecutada por una instrucción Según depende del valor de una variable numérica.



Según <variable> Hacer
<numero1>:<instrucciones>
<numero2>,<numero3>:<instrucciones>
<...>
De Otro Modo: <instrucciones>
Fin Según

Esta instrucción permite ejecutar opcionalmente varias acciones posibles, dependiendo del valor almacenado en una variable de tipo numérico.
Al ejecutarse, se evalúa el contenido de la variable y se ejecuta la secuencia de instrucciones asociada con dicho valor. Cada opción está formada por uno o más números separados por comas, dos puntos y una secuencia de instrucciones.
Si una opción incluye varios números, la secuencia de instrucciones asociada se debe ejecutar cuando el valor de la variable es uno de esos números.

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

Ejercicios de estructura de decisión compuesta con PSeInt

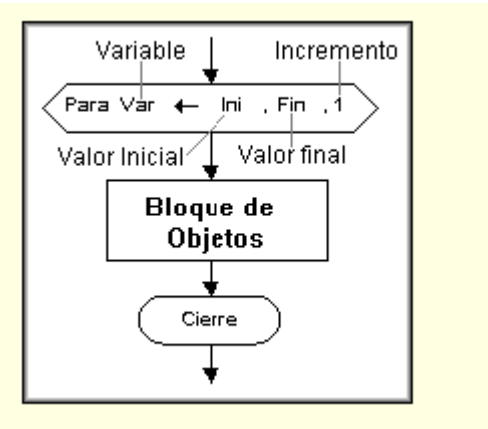
MÓDULO VI SENTENCIAS REPETITIVAS

En los programas que se han considerado hasta ahora, la computadora no ha ejecutado ninguna instrucción más de una vez. Una posibilidad poderosa de la computadora es su capacidad de ejecutar varias veces el mismo grupo de líneas. Este es el proceso de ciclos, y el grupo de líneas se ejecutan varias veces se llama el cuerpo del ciclo.

Un ciclo es ventajoso cuando se ha de repetir esencialmente la misma tarea un número especificado de veces. En tales casos, basta codificar una sola vez la tarea- como cuerpo del ciclo. Por ejemplo, en un programa para calcular e imprimir cheques para mil empleados, no se necesita incluir 1,000 grupos distintos de líneas para llevar a cabo la tarea de calcular e imprimir los cheques. En vez de ello se puede usar un solo grupo de líneas, como el cuerpo de un ciclo, que se ha de repetir 1,000 veces.

6.1 Estructuras repetitivas PARA con DFD

Su función es ejecutar un bloque de objetos mientras que la variable contadora no alcance el límite establecido por el valor final. El contador es siempre una variable de tipo de dato Real. Contiene además un valor inicial que será asignado al contador al iniciar la ejecución del ciclo, un valor final y un valor de incremento. Si el contador excede el valor final, la ejecución continuará a partir del objeto que sigue al Cierre. En caso contrario, se ejecutará el cuerpo del ciclo y el contador será incrementado en el valor indicado por el incremento.



Nota: Para que entre el ciclo la condición debe ser verdadera.

EJEMPLOS:

```
Para  I←    1 , 5 , 1
  Imprime  I, ' * 2  = ' , 2*I
Cierre para
```

Salida del programa

```
1*2 = 1
2*2 = 4
3*2 = 9
4*2 = 16
5*2 =25
```

Cuerpo del ciclo. El cuerpo del ciclo en un ciclo para es la línea o grupo de líneas que se van a ejecutar un número especificado de veces. En el ejemplo siguiente, el cuerpo del ciclo es la única instrucción.

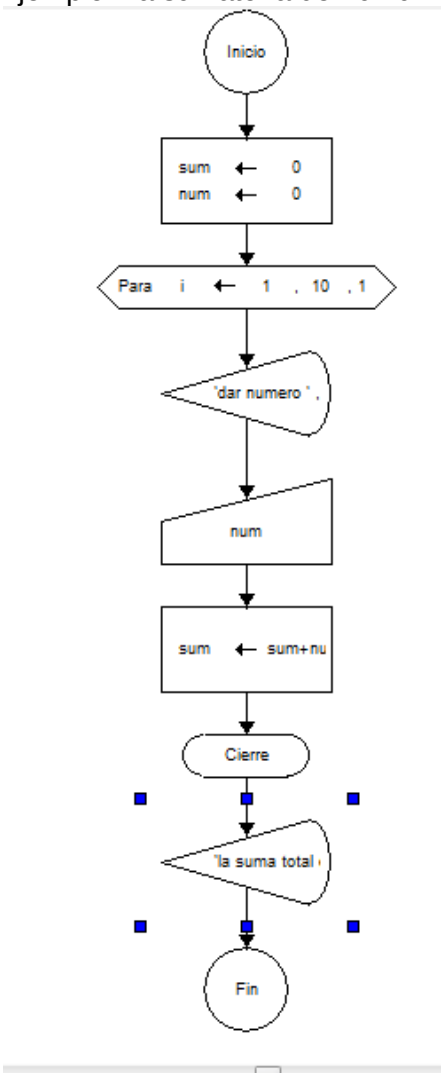
EJEMPLO

```
Para  I←    1 , 3 , 1
  Imprime  'juan perez'
Cierre para
```

La siguiente es una versión incorrecta de un segmento de programa que trata de calcular el promedio de cinco números de entrada:

```
Para  I←    1 , 5 , 1
        suma←   0
        imprime 'escriba numero'
        leer numero
        suma←    suma + numero
cierre para
prom ←    suma/5
imprime 'El promedio es ', prom
```

¿Cuál es el error del pseudocódigo?
Ejemplo: La sumatoria de 10 números

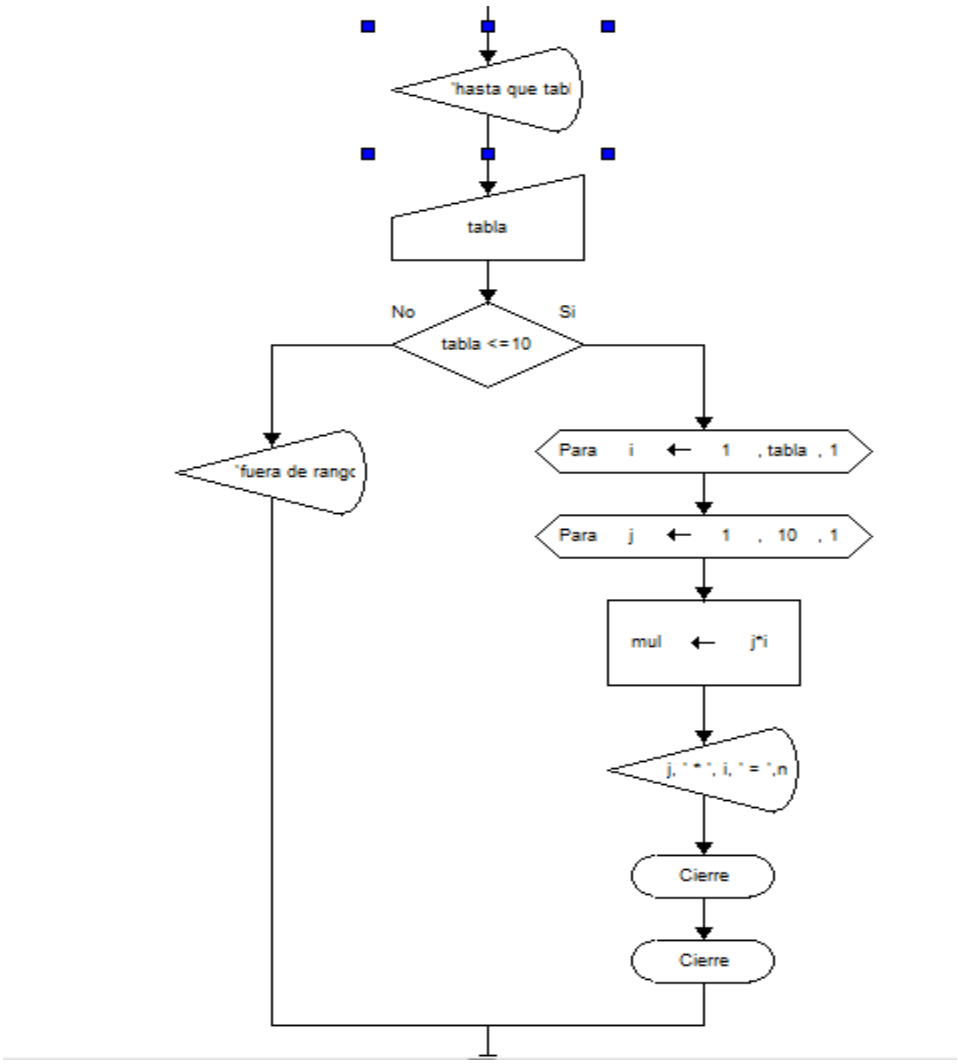


6.2 Ejemplo de un ciclos PARA anidados.

En el ejemplo siguiente, el cuerpo del ciclo para, n se ejecutará tres veces primero con n fija en 2, después con n fija en 3, y finalmente con n fija en 4.

```
Para n←    2 , 4 , 1
        Para i←    6 , 8 , 1
                Imprime n,i
        Cierre para
        Imprime 'hola'
Cierre para
```

EJEMPLO: DIAGRAMA DE FLUJO QUE REALIZA LAS TABLAS DE MULTIPLICAR COMO DATO DE ENTRADA SE ESTABLECE HASTA QUE NUMERO DE TABLAS DESEA DESARROLLAR COMO LIMITE DEL 1 AL 10



Ejercicios de estructura repetitiva PARA con DFD

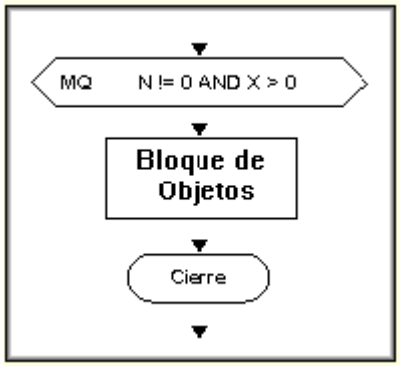
6.3 Estructura MIENTRAS con DFD

FORMATO GENERAL CICLO MIENTRAS

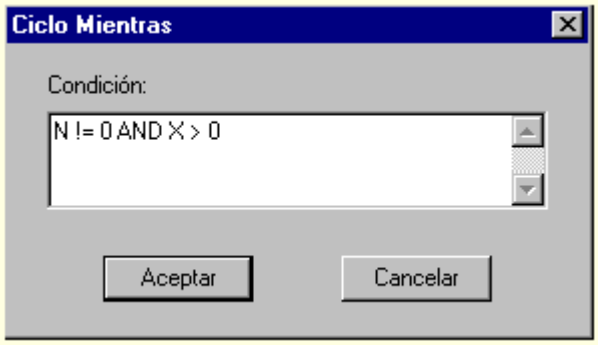
MIENTRAS (CONDICION)
    SENTENCIA
CIERRE DEL MIENTRAS

MIENTRAS (CONDICION)
    SENTENCIA 1
    SENTENCIA 2
    ….
    SENTENCIA N
CIERRE MIENTRAS

DFD



El objeto Ciclo Mientras tiene como función el ejecutar un bloque de objetos mientras que una condición sea verdadera. La condición debe ser siempre una expresión que al ser evaluada de como resultado un valor de tipo de dato Lógico.
Ejemplo : 3<W , x >0 AND Sw = .V. , Valor*15 < 300*Contador.
Si al evaluar la condición se obtiene el valor .F. la ejecución del algoritmo continuará a partir del objeto que sigue al Cierre.



El cuadro de dialogo del objeto Ciclo Mientras contiene espacio para la expresión que conforma la condición.

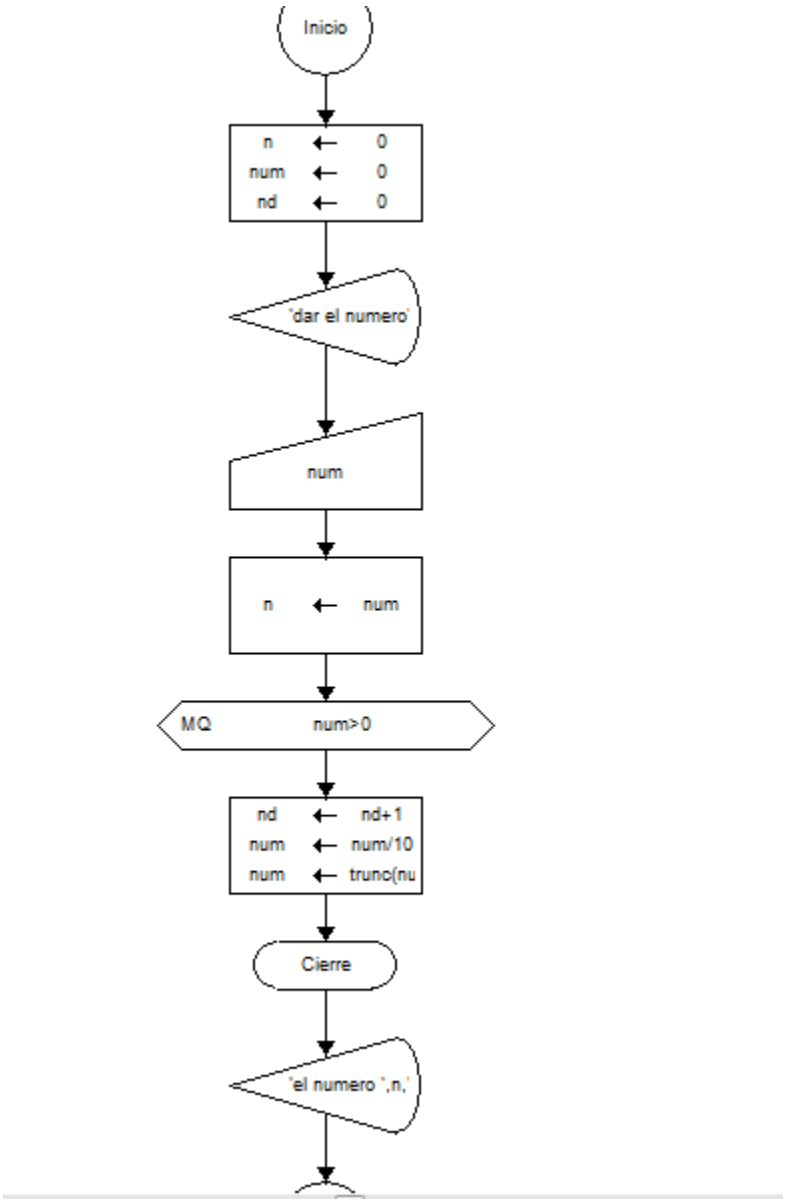
La computadora empieza probando la condición mientras. Si la condición es verdadera (diferente de verdadero), se ejecuta todo el cuerpo del ciclo. Luego se pasa el control al principio para volver a probar la condición. La primera vez que al probar la condición resulte falso (0) , la computadora sale del ciclo. Por ejemplo:

i ← 1	Salida
mientras i<9	1
imprime i	3
i← i + 2	5
Cierre mientras	7
Imprime ‘hasta luego’	9
	hasta luego

Extrae la Parte Entera

Sintaxis TRUNC(X)
Entrada X Valor de tipo de dato Real.
Resultado Parte entera de X.
Ejemplo : TRUNC(3.7) Retorna 3.
TRUNC(-4.5) -4.

EJEMPLO: HACER UN DIAGRAMA DE FLUJO QUE LEA UNA CANTIDAD E IMPRIMA EL NUMERO DE CIFRAS DEL NUMERO
DATOS DE ENTRADA: 123
SALIDA: El numero 123 tiene 3 cifras



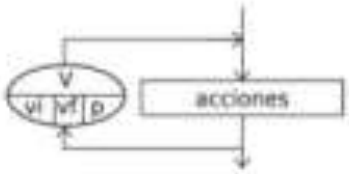
6.4 Estructuras Repetitivas con PSeInt

En PSeInt las estructuras repetitivas son las siguientes:

- **Mientras**
- **Repetir**
- **Para**

6.5 Estructuras repetitivas PARA con PSeInt

Para (for)



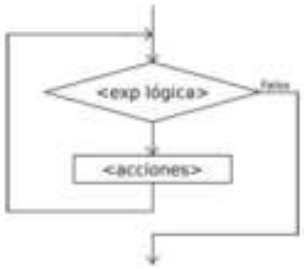
La instrucción Para ejecuta una secuencia de instrucciones un número determinado de veces.

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

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

6.6 Estructura MIENTRAS con PSeInt

Mientras Hacer (while)



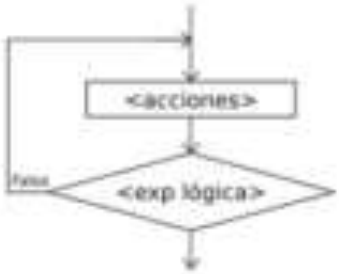
La instrucción Mientras ejecuta una secuencia de instrucciones mientras una condición sea verdadera.

Mientras <condición>Hacer
<instrucciones>
FinMientras

Al ejecutarse esta instrucción, la condición es evaluada. Si la condición resulta verdadera, se ejecuta una vez la secuencia de instrucciones que forman el cuerpo del ciclo.
Al finalizar la ejecución del cuerpo del ciclo se vuelve a evaluar la condición y, si es verdadera, la ejecución se repite. Estos pasos se repiten mientras la condición sea verdadera.
Note que las instrucciones del cuerpo del ciclo pueden no ejecutarse nunca, si al evaluar por primera vez la condición resulta ser falsa. Si la condición siempre es verdadera, al ejecutar esta instrucción se produce un ciclo infinito.
A fin de evitarlo, las instrucciones del cuerpo del ciclo deben contener alguna instrucción que modifique la o las variables involucradas en la condición, 13 de modo que ésta sea falsificada en algún momento y así finalice la ejecución del ciclo.

6.7 Estructura Repetir Hasta Que (do-while) con PSeInt

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.

Ejemplos con PSeInt:

Adivina Numero: sencillo juego en el que el usuario debe adivinar un número aleatorio.

```
II Juego simple que pide al usuario que adivine un número en 10 intentos
Proceso Adivina_Numero
        Intentos <-9;
        Números secreto <-azar(100)+1;
        Escribir “Adivine el número (de 1 a 100)”
        Leer núm_ingresado:
        Mientras num_secreto<>num_ingresado y intentos >0 hacer
                    Escribir “muy bajo”;
  Si
Escribir “Muy alto”;
                FinSi
                Escribir “Le quedan” “intentos,” “intentos”
                Leer num_ingresado:
                Intentos <-intentos -1
FinMientras
Si intentos =0 entonces
Escribir “el número era:”, número secreto
Sino
Escribir “exacto usted adivino en -1 intentos” intentos”
FinSI
FinProceso
```

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

```
II Busca los dos mayores de una lista de N datos
Proceso mayores
            Dimensión de datos[200];
            Escribir “ ingrese la cantidad de dato”, i, “:”;
            Leer n;
            Para i<-1 Hasta n Hacer
                    Escribir “Ingrese el dato “,i,”:”;
                    Leer datos[i];
             FinPara
                Si datos[i]>datos[2] Entonces
                        may1<-datos[1];
                        may2<-datos[2];
              Sino
```

```
                    may1<-datos[2];
                    may2<-datos[1];
                FinSI
                Parai<-3Hasta n Hacer
                    Si datos[i]>may1 Entonces
            may2<-may1;
            may1<-datos[i];
        Sino
Si datos[i]>may2 Entonces
                    May2<-datos[i];

                FinSi
            FinSi
        FinPara

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

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

```
II cargar datos
Escribir “Ingrese el lado 1:”;
Leer  l1;
Escribir “Ingrese el lado 2:”;
Leer l2;
Escribir “Ingrese el lado 3:”;
Leer l3

II encontrar la hipotenusa (mayor lado)
Si l1>l2 Entonces
        Cat1<-l2
        Si  l1>l3 Entonces
                hip<-l1;
                cat2<-l3;
         Sino
             hip<-l3;
 cat2<-l1
         FinSI
 FinSi
II ver si cumple con Pitágoras
```

```
        Si hip^2 = cat1^2 + cat2^2 Entonces
                II calcular área
                area<-(cat1*cat2)/2;
                escribir "El area es: ",area;
        Sino
            Escribir "no es un triángulo rectángulo. ";
    FinSi
    FinProceso
```

4. Promedio: Ejemplo básico de uso de un acumulador y la estructura de control PAR para calcular el promedio de un conjunto de valores .

```
II Calcular el promedio de una lista de N datos.
Proceso promedio
    Escribir  "Ingrese la cantidad de datos:";
    Leer n;

   Acum<-0;

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

     Prom<-acum/n;

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

BIBLIOGRAFIA

1. Luís Joyanes Aguilar, Fundamentos de Programación Algoritmos y Estructuras de Datos, Edit. Mc Grow hill
2. Luís Joyanes Aguilar, Metodología de la programación, Edit. Mc Graw hill
3. Osvaldo Cairo, Metodología de la Programación I, Edit. Alfaomega
4. M.R. Bores Rangel, Computación metodología lógica computacional y programación, Edit. Mc Graw Hill
5. Peter Norton, Introducción a la Computación, Edit. Mc Grow Hill