Lecturas de Cátedra

Problemas y algoritmos

Un enfoque práctico

Edith Lovos

Martín Goin

EDITORIAL UNRN

Problemas y algoritmos

Lecturas de Cátedra

Problemas y algoritmos

Un enfoque práctico

Edith Lovos
Martín Goin

EDITORIAL
UNRN

Utilice su escáner de
código QR para acceder
a la versión digital

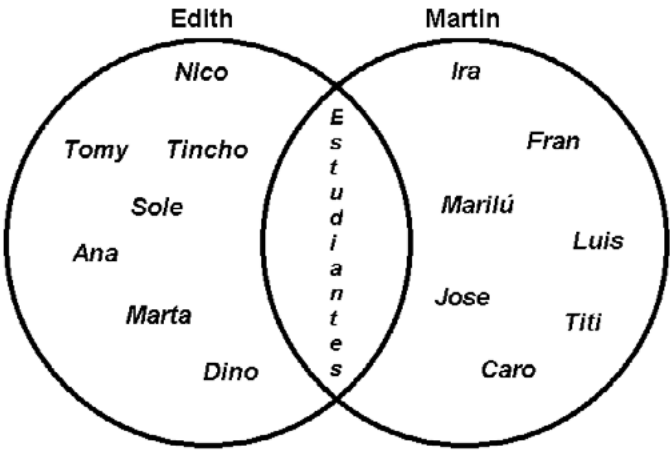
Índice

Dedicatorias **9**
Agradecimientos **10**
Prólogo **11**
Capítulo 1. Conceptos básicos de programación **13**
1. 1. Etapas en la programación 15
1. 1. 1. Definición del problema 16
1. 1. 2. Análisis del problema 16
1. 1. 3. Diseño del algoritmo 16
1. 2. Pre y poscondiciones de un algoritmo 17
1. 3. Ejercicios propuestos 18
1. 4. Expresión de algoritmos 19
Capítulo 2. Diagramación lógica **21**
2. 1. Programa FreeDFD 23
2. 2. ¿Qué son las variables? 27
2. 2. 1. Cómo darle valor a una variable 28
2. 2. 2. Intercambio de variables 29
2. 3. Estructuras de control 30
2. 3. 1. Estructura de control de decisión o selección 30
2. 3. 1. 1. Estructura de control de decisión simple 30
2. 3. 1. 2. Estructura de control de decisión doble 32
2. 3. 1. 3. Estructura de control de decisiones anidadas 35
2. 3. 1. 4. Estructura de control de decisiones independientes ... 36
2. 4. Operadores de relación 37
2. 5. Operadores lógicos 37
2. 5. 1. Conjunción lógica o producto lógico «and» 38
2. 5. 2. Disyunción lógica inclusiva o suma lógica «or» 39
2. 5. 3. Negación o complemento lógico «not» 39
2. 6. Otros operadores matemáticos especiales 43
2. 7. Problemas: Diagramación lógica – Estructura de control de decisión 48
2. 8. Estructura de control de repetición 50
2. 8. 1. Estructura de control de repetición (sentencia «para») 53
2. 8. 2. Variables como acumuladores 55
2. 8. 3. Variable contadora 55
2. 8. 4. Máximos y mínimos 62
2. 8. 5. Problemas: Diagramación lógica – Estructura de control de repetición «para» 65
2. 8. 6. Estructura de control de repetición (sentencia «mientras») 66
2. 8. 7. Problemas: Diagramación lógica – Estructura de control de repetición «mientras» 72

Capítulo 3. Pseudocódigo **73**
3. 1. PSeInt 73
3. 2. Operadores lógicos 80
3. 3. Ejercicios: pseudocódigo 87
Capítulo 4. Entorno de programación visual **89**
4. 1. Variables del programa DaVinci 93
4. 2. Conociendo la ciudad del robot 94
4. 2. Repetición 97
4. 3. Modularización 106
4. 4. Parámetros formales y reales 110
4. 5. Variables globales y locales 112
4. 6. Ejercicios Da Vinci 114

Dedicatorias

Se resume en el siguiente diagrama de Venn



Agradecimientos

Al Centro Interdisciplinario de Estudios sobre Derechos, Inclusión y Sociedad (CIEDIS), al área de Desarrollo Estudiantil y al Departamento de Asuntos Estudiantiles de la Universidad Nacional de Río Negro (UNRN).

Prólogo

Este libro surge en el marco de una actividad de formación denominada *Curso virtual de resolución de problemas usando algoritmos. Un espacio para el desarrollo de competencias* (Res. 683/2016) que se lleva adelante a través del Programa de Mejora de las Ciencias Exactas y Naturales en la Escuela Secundaria. Como resultado de esa actividad, el libro busca promover en el lector el desarrollo de su capacidad analítica y creadora, mejorando su destreza en el diseño de algoritmos que sirven como base para el desarrollo de programas.

La enseñanza y aprendizaje de la programación es una actividad compleja tanto para docentes como para estudiantes. En este sentido, este libro busca ser una herramienta de apoyo. En él se exponen numerosos ejemplos y se intenta ayudar y acompañar al lector a resolver los problemas planteados.

El libro funciona como un tutorial que explica paso a paso, las bases de la programación de modo muy sencillo. Está orientado a los primeros cursos de grado de carreras –ingenierías, profesorados, licenciaturas, tecnicaturas– que incluyan asignaturas vinculadas a la programación. De esta forma, resulta ideal para aquellos estudiantes que incursionan por primera vez en el mundo de la programación.

El material se divide en cuatro capítulos. El primero es un resumen teórico en el que se abordan los conceptos básicos de la programación. El segundo capítulo tiene por objetivo avanzar de modo progresivo en el proceso de resolución de problemas con algoritmos usando como recurso el diagrama de flujo para su representación. Con la intención de facilitar el proceso cognitivo se utilizará el *software* FreeDFD, que es de uso libre y gratuito. Este capítulo incluye cerca de 30 ejemplos prácticos que permiten probar el programa y verificar los resultados. Además aborda los siguientes temas: variables, intercambio, estructuras de control de decisión (simple, doble, decisiones anidadas e independientes), operadores (de relación, lógicos, matemáticos especiales), estructuras de control e iteración («para», «mientras»), variables contadoras y sumadoras, máximos y mínimos. El tercer capítulo presenta el pasaje del diagrama de flujo al pseudocódigo, si bien no es programable, la intención es aproximarnos al lenguaje de programación realizando una traducción del modo gráfico al código (texto). Se propone el uso del *software* PSeInt de carácter libre y gratuito. Esta herramienta, al igual que FreeDFD, permite la ejecución de los algoritmos posibilitando la autocorrección. Por último, se presenta un capítulo especial dedicado a la resolución de problemas usando el aplicativo visual DaVinci Concurrente.

De esta forma se ofrecen diversas herramientas de asistencia en el aprendizaje de los conceptos básicos de programación, sumado a un conjunto de ejemplos.

En el libro se presentan ejemplos prácticos resueltos. Además se propone un conjunto de situaciones problemáticas que le permitirán al lector desarrollar las habilidades de resolución de problemas usando algoritmos.

Capítulo 1

Conceptos básicos de programación

Fuente: Joi Ito, 2011.

Aprender a programar es programar para aprender

Mitchel Resnick (1956)

La frase pertenece al destacado profesor, físico, informático, periodista y programador. Participó del diseño y desarrollo del Scratch (lenguaje de programación para educación, 2005).

Vivimos en un mundo donde la tecnología tiene un protagonismo muy importante en nuestra vida cotidiana. Usamos cajeros automáticos, realizamos diferentes transacciones (bancarias, comerciales y otras) a través de la *web*, buscamos información en Internet, nos mantenemos comunicados a través de dispositivos móviles (celulares, tabletas, etcétera), nos ubicamos y jugamos usando tecnología de geoposicionamiento (GPS). La tecnología está íntimamente relacionada a la programación; sin esta nada tiene sentido, nada funciona. Cuando hablamos de programación, la asociamos a la palabra informática.

La Real Academia Española, define la palabra informática como:
«El conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático y racional de la información por medio de computadoras.»

Ahora bien, se deben precisar las respuestas a las siguientes preguntas: ¿Qué es información? ¿Qué significa tratar la información? ¿Qué significa que una parte del tratamiento sea automático y otra racional?

Se entiende por información a los hechos y representaciones de una situación, los cuales pueden o no tener relación. La información, para que pueda ser tratada por una máquina (computadora), necesita ser codificada en un lenguaje entendible por la máquina.

Entonces, hablar de tratamiento automático significa que será la máquina (autómata) la que llevará adelante el proceso. ¿Y cómo se logra esto? Se logra codificando el razonamiento humano a través de una secuencia de instrucciones (programa).

En la actualidad, programar no es una actividad reservada solo a unas pocas personas (técnicos, ingenieros, expertos, licenciados, etcétera), muchas lo toman como un juego, un beneficio, un desafío y hasta una afición.

Como en toda disciplina, hay áreas que son más sencillas de aprender y otras no tanto. Empezar por algo muy accesible nos dará el gusto de comenzar a programar. Luego, la creación de soluciones más elaboradas para problemas más complejos nos ayudará a incorporar conocimientos.

Mientras existan la creatividad y las ganas de aprender tendremos la libertad de programar.

Si tenemos que definir el término programación, decimos que es la acción y el efecto de programar.

Programar es lo más cercano que tenemos a un superpoder

Drew Houston (1983)

Creador de Dropbox (almacenamiento de archivos en la nube para trabajar y compartir).

Fuente: Adaptado de Financial Times, 2011.

Los programadores son los magos del futuro

Gabe Newell (1962)

Director general de Valve Software (empresa desarrolladora de videojuegos)

Fuente: Game Developers Choice Awards, 2010.

El verbo *programar* tiene varios usos. Se refiere a idear y ordenar las acciones que se realizarán en el marco de un proyecto, como por ejemplo la preparación de máquinas para cumplir con una cierta tarea específica, la preparación de un espectáculo deportivo o artístico, la preparación de datos necesarios para obtener la solución de un cálculo a través de una calculadora, el diseño del sistema y la distribución de materias para una carrera o de temas para un curso o asignatura, etcétera.

En la actualidad, la noción de programación se encuentra más asociada a la programación en ciencias informáticas. En este sentido, programar

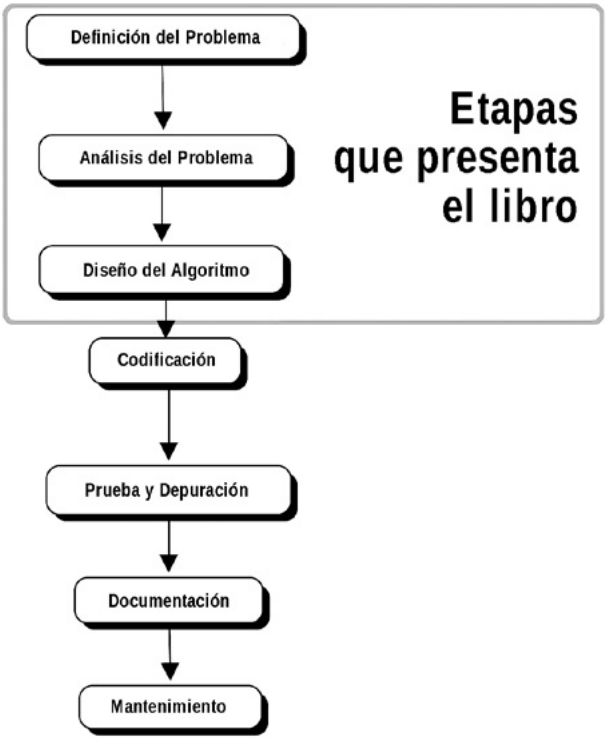
podría resumirse como el proceso por el cual un programador escribe, prueba, depura y mantiene un código a partir del uso de un lenguaje de programación. Así, aprender a programar implica aprender ciencia y tecnología. La tecnología puede verse como el conjunto de herramientas, técnicas y estándares que permiten llevar adelante la programación. Al hablar de ciencia se hace referencia a una teoría amplia y profunda que permite entender la programación. Ambos conocimientos son importantes, ya que posibilitan enfrentar los retos de la evolución tecnológica.

Por otra parte, en estos tiempos, como señalan Patricia Compañ-Rosique y otros (2015), la programación es una vía posible para el desarrollo del pensamiento computacional. Es decir, hacer uso de conceptos fundamentales de la informática para resolver problemas del quehacer cotidiano.

1.1. Etapas en la programación

En la figura 1.1 se muestran las etapas para el desarrollo de un programa:

Figura 1.1. Etapas de la programación



En este libro abordaremos las tres primeras: definición del problema, análisis del problema y diseño del algoritmo.

1. 1. 1. Definición del problema

Esta fase está dada por la especificación del problema, el cual requiere una definición clara y precisa. Es importante saber lo que se desea *que realice* la computadora; mientras esta definición no sea lo suficientemente clara, no tiene mucho sentido continuar con la siguiente etapa.

1. 1. 2. Análisis del problema

Esta fase requiere de una clara definición en la que se contemple exactamente qué debe *hacer* el programa y el resultado o la solución deseada. Entonces es necesario definir:

- datos de entrada (tipo y cantidad);
- datos de salida (tipo y cantidad);
- los métodos y las operaciones que se necesitan para procesar los datos.

En esta etapa se determina *qué* debe hacer el programa para resolver el problema, es decir, la solución al mismo.

Una recomendación muy práctica es la de ponernos en el lugar de la computadora y analizar qué es lo que necesitamos ordenar y en qué secuencia, para producir los resultados esperados.

1. 1. 3. Diseño del algoritmo

En esta etapa se determina *cómo* debe ser el proceso que lleva a la resolución del problema. La palabra algoritmo deriva del nombre de un matemático árabe del siglo IX, llamado Al-Khuwarizmi, quien describió varios métodos para resolver cierto tipo de problemas aritméticos.

El diseño de algoritmos es un recurso fundamental que permite resolver problemas relacionados con casi todas las disciplinas.

La intención de este libro es que el lector adquiera las bases necesarias para poder diseñar e implementar, de manera fácil y rápida, soluciones algorítmicas.

Definición de algoritmo: es una secuencia no ambigua, finita y ordenada de pasos para poder resolver un problema.

- No ambigua implica que cada paso del algoritmo debe poder ser interpretado de una *única* forma.
- Finita significa que la cantidad de pasos que componen el algoritmo está limitada. El algoritmo empieza y termina.

- Orden. Los pasos del algoritmo deben seguirse en una determinada *secuencia* para llegar a la solución del problema.

Resumiendo, un algoritmo se puede pensar como una receta, un conjunto de instrucciones o de especificaciones sobre un proceso para hacer algo, que cumple con las características antes mencionadas.

Ejemplos

Problema 1: Indique la manera de endulzar una taza que contiene café.

Algoritmo A: Ponerle un poco de azúcar a la taza y revolver

Algoritmo B: Agregarle una cucharadita de azúcar a la taza, revolver y degustar. Repetir el proceso hasta que quede dulce.

El algoritmo A presenta una solución ambigua al problema planteado. ¿A cuánto equivale un poco?

El algoritmo B presenta una solución adecuada al problema.

Problema 2: Desarrolle un algoritmo que describa la forma de determinar la suma de todos los números naturales.

En este caso, no es posible encontrar un algoritmo que resuelva el problema. Ya que una de las características de un algoritmo es alcanzar la solución en un tiempo finito, situación que no se cumplirá en este caso, ya que los números naturales son infinitos.

Problema 3: Llenar un pozo con piedras de un bolsón.

Algoritmo: Tomar una pala. Mientras haya piedras en el bolsón cargar la pala con piedras y volcarla en el pozo. Dejar la pala.

Esta solución es un algoritmo, ya que cada paso es no ambiguo. Por otra parte, se puede asegurar que en algún momento finalizará (es finito), aunque no se sabe cuántas paladas serán necesarias.

¿Qué sucedería si no contamos con la pala? ¿Se podría llevar adelante el proceso definido en el algoritmo?

1. 2. Pre y poscondiciones de un algoritmo

Precondición: es la información que se conoce como verdadera antes de comenzar el algoritmo.

Poscondición: es la información que se conoce como verdadera después de finalizado el algoritmo, siempre que se cumpla con las precondiciones.

Problema 4: Determinar el resto de la división entera entre dos números enteros N y M.
Precondición: N y M son números enteros. M debe ser distinto de 0.
Poscondición: el resultado será un valor entero comprendido entre 0 y M-1, que representa el resto de aplicar la división entera entre N y M.

Problema 5: Indique la manera de endulzar una taza que contiene café.
Precondición: Contar con una cucharita y azúcar *suficiente.*
Poscondición: El café quedó dulce al finalizar el proceso.

Problema 6: Determinar si el número 437 es primo.
Algoritmo: Dividir el número 437 entre cada uno de los números 1, 2, 3, 4,... 436. Si una de las divisiones es exacta (resto 0), entonces el número 437 no es primo, caso contrario es primo.
Precondición: No hay ningún requerimiento.
Poscondición: Se ha podido determinar si el número 437 es primo o no.

Preguntas: ¿Consideran que es una buena solución?
¿Qué otra/s podrían pensarse?

1. 3. Ejercicios propuestos

1. Escriba un algoritmo que permita cambiar una lámpara quemada. Indique pre y poscondiciones.
2. Escriba un algoritmo que indique cómo hacer un huevo frito. Indique pre y poscondiciones. ¿En qué cambiaría el algoritmo si quisiéramos cocinar 3 huevos en lugar de 1?
3. Escriba un algoritmo para comprar 1 kg de pan migñon en la panadería de la esquina de casa.
4. Observe las dos soluciones que se presentan a continuación e indique si son o no un algoritmo. Si no lo son, modifíquelas para que lo sean.

Algoritmo 1	Algoritmo 2
Me visto rápidamente. Me levanto por la mañana. Tomo una ducha de 10 minutos. Tomo un té con leche y me voy. Termino de ducharme a las 7:25. Llego temprano al colegio. El reloj marca las 7:15.	Traer las herramientas que voy a usar. Hacer el arreglo con esmero. Localizar el desperfecto del depósito. Pasar la factura por el trabajo hecho. Organizar cómo voy a hacer el trabajo. Ver qué tipo de arreglo necesita. Comprobar la eficiencia del arreglo.

1. 4. Expresión de algoritmos

Para el diseño de un algoritmo puede utilizarse la diagramación lógica (DL) y/o el pseudocódigo.

La diagramación lógica se basa en la representación gráfica, esquemática de un algoritmo. Es altamente intuitiva y didáctica e implica la construcción de circuitos lógicos utilizando símbolos, mientras que el pseudocódigo es la representación descriptiva y textual (en lenguaje natural, por ejemplo, el español) de las operaciones que realiza un algoritmo.

En este libro, vamos a trabajar con ambas, ya que consideramos que son un buen complemento para iniciarnos en la actividad de programar soluciones algorítmicas.

Para aquellos que incursionan por primera vez en el mundo de la programación es recomendable comenzar con la utilización del diagrama de flujo.

Recordemos que el procedimiento de desarrollo de algoritmos es independiente del lenguaje de programación que utilicemos más adelante. Es decir, una vez diseñado el algoritmo, este podrá ser luego escrito (traducido) en diferentes lenguajes de programación.

Hasta aquí podemos resumir las características de un algoritmo:

- Debe tener un punto particular de inicio.
- Debe estar definido sin ambigüedades, es decir, no debe permitir dobles interpretaciones.
- Debe ser general, es decir, soportar las variantes que puedan presentarse en la definición del problema.
- Debe ser finito en tamaño y tiempo de ejecución.

Adicionalmente, los algoritmos pueden requerir de datos de entrada para producir datos de salida.

Figura 1.2. Algoritmo

Datos de entrada: un algoritmo tiene cero o más entradas, es decir la información que recibe el algoritmo al comenzar o, dinámicamente, mientras el algoritmo se ejecuta.

Procesamiento de datos: incluye las operaciones aritmético-lógicas, cuyo objetivo es obtener la solución del problema.

Salida de resultados: permite comunicar al exterior el resultado. Un algoritmo puede producir una o más salidas.

Entonces, la computadora es una máquina (herramienta) que por sí sola no puede hacer nada, necesita ser programada, es decir, que se le suministren instrucciones u órdenes indicándole aquello que debe hacer.

Un programa es la solución a un problema inicial, así que todo comienza allí: en el *problema*. Así, dado un determinado problema es necesario idear una solución y expresarla algorítmicamente. Luego de esto, será necesario traducir el algoritmo (codificarlo) en un determinado lenguaje de programación y, por último, ejecutar el programa en la computadora. Esto es, a grandes rasgos, lo que hace el programador.

El diseño de algoritmos será tema de los próximos capítulos, comenzando con la diagramación lógica y luego con el pseudocódigo.

La computadora es el Proteo de las máquinas. Su esencia es su universalidad, su poder de simular.

Seymour Papert (1928-2016)

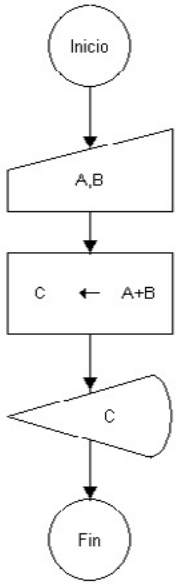
La frase pertenece al destacado científico en computación, matemática y educación. Es uno de los pioneros de la inteligencia artificial y creador del Logo (lenguaje de programación para educación).

Fuente: Adaptado de ak_mardini, 2006.

Capítulo 2

Diagramación lógica

Vamos a dar inicio a este capítulo con un ejemplo muy sencillo:

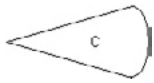


El algoritmo permite el ingreso de dos números, los suma y, por último, muestra el resultado.

Para poder comprender el concepto de este gráfico es necesario determinar el significado de cada uno de sus componentes.

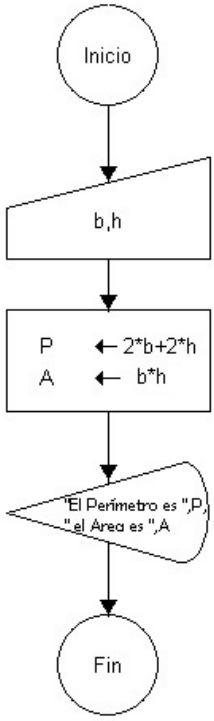
A continuación se describen cada uno de los elementos utilizados en el algoritmo:

Inicio	Comienza el algoritmo.
A,B	Ingreso de datos (en estos casos se trata de dos valores numéricos A y B).
C ← A+B	Asignación (en este caso se le asigna a la variable C la suma de A y B, es decir C = A+B).

	Salida de datos (en este caso se imprime lo que contiene la variable C)
	Termina el algoritmo.
	Las flechas indican la dirección del circuito (de ahí viene el nombre *flujo*).

La lectura de un diagrama se hace de la misma manera que la lectura de un libro (de izquierda a derecha y de arriba abajo).

Problema 1: Hallar el perímetro y el área de un rectángulo ingresando la base b y la altura h.



Es posible observar que en una misma caja de asignación se puede hacer más de una operación (en este caso para asignar el resultado del cálculo del perímetro y del área de un rectángulo).

Nota: En la salida se pueden alternar carteles (mensajes) y datos, mientras los primeros estén entre comillas y ambos separados por comas. En este caso será **"El Perímetro es ",P," y el Área es ",A.**
Ejemplo: si ingresamos b y h como 3 y 5 entonces la salida imprime lo siguiente: *El Perímetro es 16 y el Área es 15*
Atención: Las operaciones aritméticas en los algoritmos deben expresarse siempre en una línea, por ejemplo si queremos realizar la siguiente operación:

$$C = \frac{-B+9}{2A}$$

es necesario escribirla de la siguiente manera: **C = (-B+9)/(2*A)**. El asterisco * representa la multiplicación y la barra inclinada a derecha / la división.

2. 1. Programa FreeDFD

Para trabajar con diagramas de flujos, se recomienda utilizar un software libre y gratuito, FreeDFD 1.1. Este aplicativo fue desarrollado por estudiantes de la Universidad de Magdalena, Colombia, y se distribuye bajo la licencia pública GNU (GPL).

Además de ofrecer un manejo muy sencillo, permite crear, editar, ejecutar y corregir (cuando se presentan errores) el diagrama de flujo para verificar su funcionamiento.

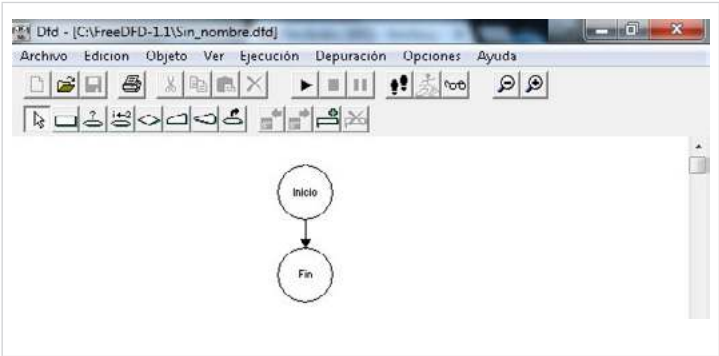
El programa es muy intuitivo, por tratarse de una herramienta gráfica, e incluye además un menú de ayuda muy completo.

En esta sección nos centraremos en el uso básico de las herramientas de diseño y depuración.

El programa puede descargarse desde el sitio http://code.google.com/p/freedfd. En Windows, el archivo comprimido llamado FreeDFD-1.1.zip, tiene que ser descomprimido en una carpeta con su mismo nombre (esto es automático). En el grupo de archivos de dicha carpeta aparecen tres ejecutables (uno para cada idioma: portugués, inglés y español), el nuestro será la aplicación ejecutable dfd-español. En sistemas operativos GNU/Linux puede instalarse usando el software Wine.

Al ingresar al programa nos encontraremos con la siguiente pantalla:

Figura 1. 3. Pantalla FreeDFD



En la parte superior tenemos el menú principal para acceder a todas las opciones posibles y, debajo, la barra de herramientas que funcionan como atajos.

Más abajo, el sector de edición de trabajo. Siempre comienza con los conectores «inicio» y «fin».

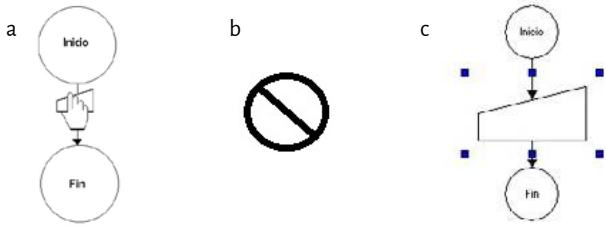
Respecto a la barra de herramientas o botones:

Figura 1. 4. Botones FreeDFD

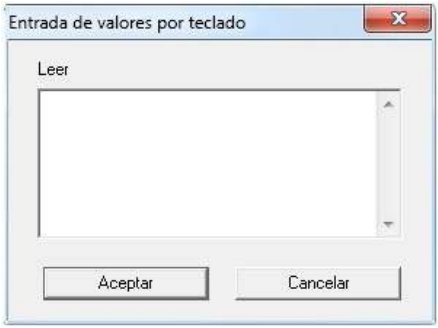
Los objetos son los elementos que permiten representar la solución al problema. De izquierda a derecha son: cursor, asignación, ciclo «mientras», ciclo «para», decisión, lectura, salida y llamada.

Por ejemplo, si queremos construir el primer diagrama de flujo del ejemplo del libro tendríamos que seguir los siguientes pasos:

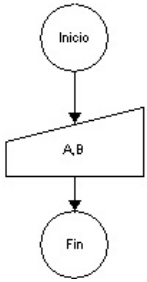
1. Para introducir entre los conectores «inicio» y «fin» el símbolo de ingreso, primero hacemos click en el ícono «lectura» de la barra «objetos» y luego nos acercamos con el mouse al destino deseado (a). La mano nos indica el sector permitido para colocar el objeto, de lo contrario aparecerá un cartel de prohibido (b).
   Luego de elegir la ubicación del objeto aparecen las marcas que indican que está seleccionado (c).



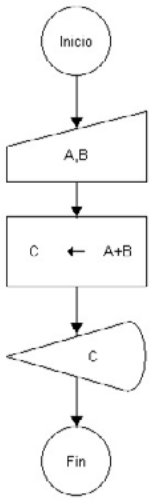
2. Haciendo doble click sobre el objeto aparece la siguiente ventana.



   Dentro de la caja escribiremos las variables de entrada, para nuestro caso: A,B
3. La entrada está lista y se verá así:



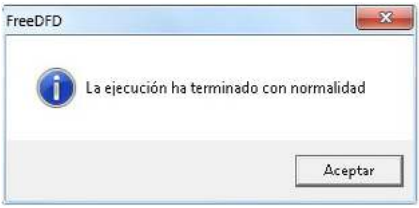
4. Luego de armar por completo el diagrama de flujo como se observa en el primer ejemplo, hay que ponerlo a prueba, aunque se recomienda primero guardar el archivo («Archivo» – «Guardar como»).



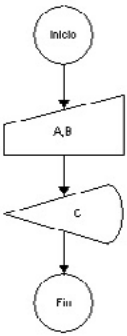
5. Hacemos un click en el botón «ejecutar» (barra de ejecución) y aparece la ventana («entrada de valores por teclado»).
   Ingresamos entonces los valores (uno por cada entrada).

6. Luego muestra la salida (el resultado final) con la siguiente pantalla:

7. Si el programa funcionó sin generar errores, FreeDFD nos mostrará la siguiente ventana:



Error de ejecución: Si creamos el siguiente algoritmo y lo ejecutamos, finalmente nos va a mostrar el error.



Y además nos va a indicar en color rojo el objeto donde se produjo el inconveniente. En este caso lo va a marcar en el objeto de salida porque la variable C no contiene valor alguno.

Nota: Sería beneficioso probar el ejemplo con el error y luego realizar y ejecutar el problema 1.

2. 2. ¿Qué son las variables?

Una variable es un espacio en la memoria de la computadora que permite almacenar temporalmente información (dato) durante la ejecución del algoritmo, y cuyo contenido puede cambiar mientras se ejecuta el algoritmo.

Para reconocer una variable es necesario darle un nombre o etiqueta, que permita identificarla.

Los nombres o etiquetas de las variables siempre deben empezar con una letra y no pueden contener espacios en blanco. Si usamos más de un carácter para su identificación empezamos con una letra y luego podemos seguir con números o letras. Está permitido usar "_" entre medio.

Ejemplos válidos: A, a , B1 , A20 , AA1 , Aa1 , B2B , Promedio , SUMATORIA , A_1 , b1_2
Ejemplos no validos: 1B , 2c , _S , ¿A, La variable

Se recomienda que el nombre de una variable represente el dato en sí y que no sea extenso, algunos ejemplos:

Para una sumatoria → S
Para dos sumatorias → S1 , S2
Para la sumatoria de edades → SE
Para contar → C
Para un promedio → P o Prom
Para el promedio de edades de mujeres → PEM

Atención: No puede existir más de una variable con el mismo nombre (identificación) en un mismo algoritmo y con distinta finalidad.

Las variables pueden contener información numérica como alfanumérica, es decir letras y símbolos, siempre y cuando estos últimos sean expresados entre comillas.

Ejemplos: A1="20" Nombre="Sofía" Simbolo="&"

En cambio, las variables que contienen números no necesitan de las comillas.

Ejemplos: G1=20 Precio=129,85 Temperatura = -0,3

Analizar la diferencia entre las variables G1 y A1 de los ejemplos anteriores. No representan la misma información, en un caso es un valor numérico y en otro una expresión alfanumérica.

Ejemplos: Calle="San Martin" Numero="201" Domicilio="Moreno 77"

Nota: Los nombres de las variables no se acentúan.

El valor de las variables y el tipo de operaciones que se puede realizar con ellas dependerá de si estas son numéricas o alfanuméricas. En el caso de las variables numéricas, las cuatro operaciones aritméticas básicas son:

suma (+), resta (–), división (/) y multiplicación (*).

En el caso de las variables alfanuméricas se puede utilizar el operador +, para concatenar. Analicemos el problema 2.

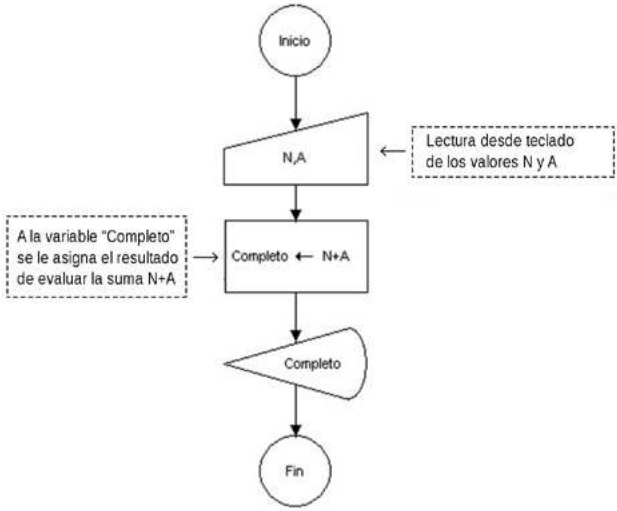
2. 2. 1. Cómo darle valor a una variable

Existen dos formas de darle valor a una variable: a través de la asignación o a través de la lectura de un valor.

En FreeDFD para asignarle un valor podemos usar el operador de asignación ← o una funcionalidad propia del aplicativo que nos permite leer un dato desde un dispositivo de entrada (ejemplo teclado).

Es importante recordar que una variable tiene un único contenido en un momento dado, es decir, un valor actual. Este valor puede cambiar durante la ejecución del programa, pero siempre será único y no quedará registro de cuáles fueron los contenidos anteriores de la variable.

Problema 2: Ingresar un nombre y un apellido en distintas variables y luego mostrar en forma concatenada el nombre seguido del apellido. Por ejemplo, si el nombre es Ana y el apellido Perez, la salida sería *AnaPerez*.



Atención: Si se ingresa **N="Manuel"** y luego **A="Belgrano"** el resultado de la suma será "ManuelBelgrano" (falta separar el nombre y el apellido), entonces podríamos dejar un espacio en blanco al final de **"Manuel "**, o bien al principio de **" Belgrano"** para solucionarlo. ¡Hagan la prueba!

Pregunta: ¿Qué sucedería si se quita la instrucción que permite la lectura de A y N?

Importante: Una cuestión problemática con las variables es su valor o contenido inicial. En algunos lenguajes si una variable se usa sin antes haberle asignado un valor, se le dará un valor por defecto, mientras que en otros lenguajes eso podría generar un error durante la ejecución del programa. Entonces para evitar estas situaciones, siempre que usemos una variable debemos darle un valor inicial. **Sugerencia:** Editar y ejecutar los problemas propuestos en el software FreeDFD para una mejor comprensión.

2. 2. 2. Intercambio de variables

El intercambio (*swap*) de los valores de dos variables no es un procedimiento que pueda hacerse en forma directa, es decir, por ejemplo: si A=8 y B=2 entonces si A=B (le asignamos a A el valor de B, se pierde el valor original de A) y si luego hacemos B=A (le asignamos a B el valor de A), finalmente los valores de A y B serán los mismos (en este caso ambos serán iguales a 2).

Para solucionar esta situación debemos usar una variable auxiliar.

Para el caso anterior será: C=A, A=B y B=C. Entonces ahora tendremos A=2 y B=8. En este caso C será la variable auxiliar.

Recordemos que solo podemos intercambiar variables que tengan el mismo tipo. ¿Qué sucede si dadas dos variables A=5 y B=7 intentamos llevar adelante las siguientes operaciones: C=A, A=B, B=C?

Importante: El tipo de una variable especifica el conjunto de valores que puede tomar y las operaciones que pueden hacerse con ella.

2. 3. Estructuras de control

Las estructuras de control permiten controlar el flujo de ejecución de las instrucciones del programa: tomar decisiones, realizar acciones repetitivas, etcétera, dependiendo de unas condiciones que nosotros mismos establezcamos.

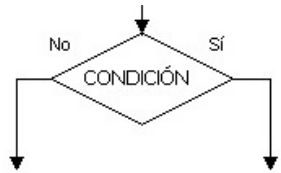
El concepto de flujo de control se refiere al orden en que se ejecutan las sentencias o acciones (instrucciones) de un programa.

En los ejemplos anteriores se ha trabajado con un flujo lineal también llamado *estructura secuencial*, así las estructuras de control (de selección y repetición) permiten alterar este orden de ejecución secuencial.

2. 3. 1. Estructura de control de decisión o selección

También llamada *de alternativa,* permite bifurcar el flujo de ejecución del programa en función de una expresión lógica o condición lógica.

Con frecuencia aparecen en algoritmos situaciones en las cuales se debe elegir un camino dependiendo de los datos de entrada y la condición impuesta.

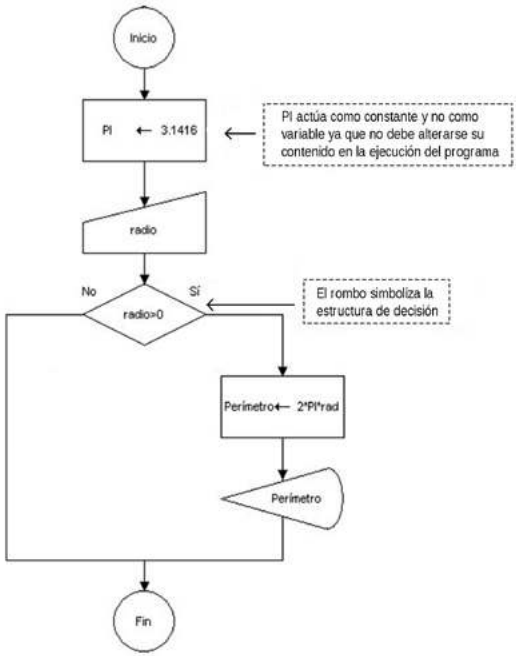
El símbolo es un rombo acostado que tiene dos posibles salidas, como se observa en la figura. Siempre tomará uno de los dos caminos (***nunca*** ambos al mismo tiempo)	

Una decisión puede clasificarse en simple, doble, anidadas y/o independientes.

2. 3. 1. 1. Estructura de control de decisión simple

Veamos un ejemplo de un algoritmo que utiliza una decisión simple.

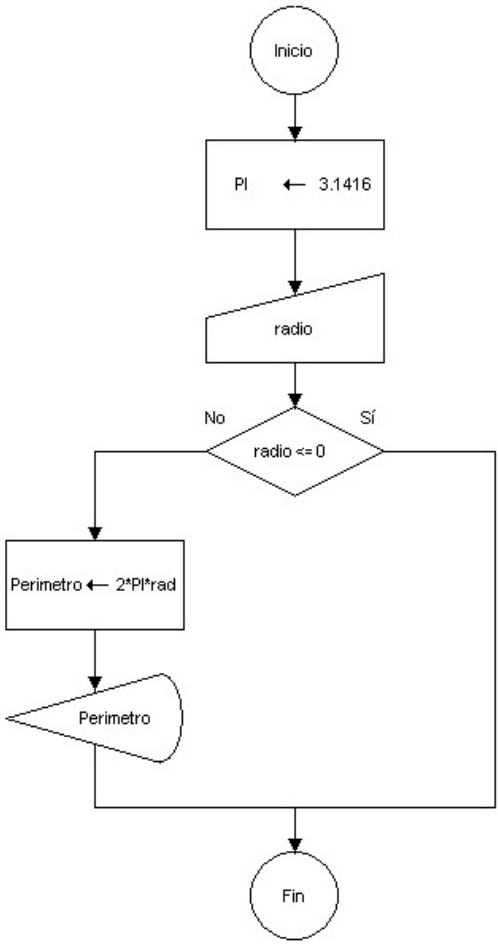
Problema 3: Mostrar el perímetro de una circunferencia, siempre y cuando el radio que se ingresa sea mayor a cero (controlar dicho ingreso).



En este caso, si al ejecutar el algoritmo se ingresa un valor de radio negativo o cero, el programa simplemente termina, ya que solo funciona cuando la condición **radio > 0** (expresión lógica) es verdadera.

Cuando se cumple la condición dentro del rombo el flujo de datos va para el lado del **Sí,** caso contrario se dirige hacia el **No**.

Atención: La estructura de control de decisión debe tener como mínimo una acción a ejecutar en caso que se cumpla la condición. En el ejemplo anterior podríamos haber establecido la condición del SI con (radio < = 0) en vez de (radio > 0), entonces nos quedaría de la siguiente manera:

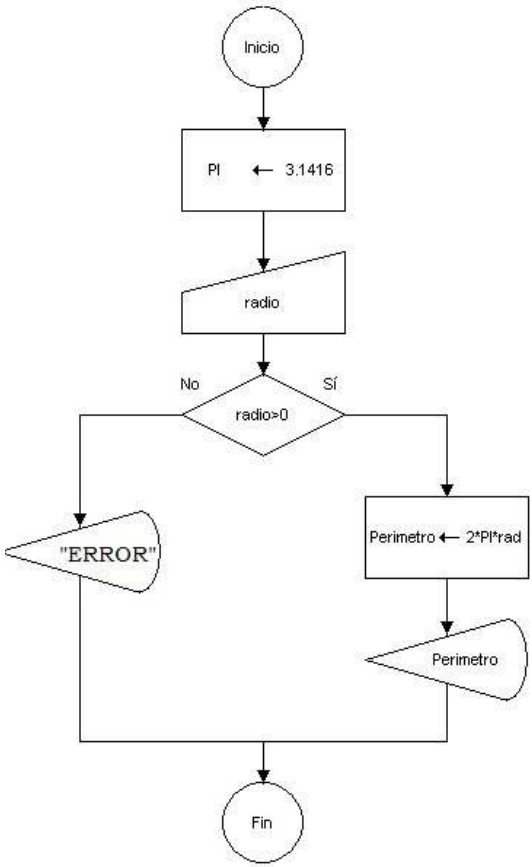


Lo cual es incorrecto.

2. 3. 1. 2. Estructura de control de decisión doble

Esta estructura es similar a la anterior con la salvedad de que se indican acciones no solo para la rama *verdadera* sino también para la *falsa*, es decir, en caso de que la expresión lógica sea cierta se ejecuta una acción o grupo de acciones y, en caso de ser falsa, se ejecuta otro grupo de acciones.

Problema 4: Ídem al ejemplo anterior pero en el caso de ingresar un radio erróneo (cero o negativo) indicarlo con el cartel "Error".

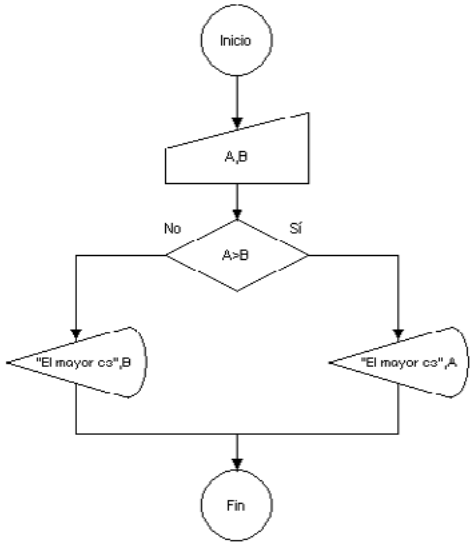


Cuando es falsa la expresión lógica (radio > 0) se ejecuta el camino que va en dirección del **No**.

Nota: Los carteles en la salida siempre van entre comillas, en este caso "ERROR".

Veamos otro ejemplo.

Problema 5: Se pide ingresar dos números y luego mostrar por mensaje cuál es el mayor.



En las salidas es posible combinar cartel y variable (texto y números). Por ejemplo, probemos ingresar por pantalla los números 3 y 7, entonces el camino que tomará el algoritmo será el **No**, al ser falsa la condición **A>B**, en consecuencia su salida mostrará *El mayor es 7*.

Nota: ¿Qué sucede si ingresamos dos números iguales? ¿Cuál sería el resultado? Lo verificamos ejecutando el algoritmo.

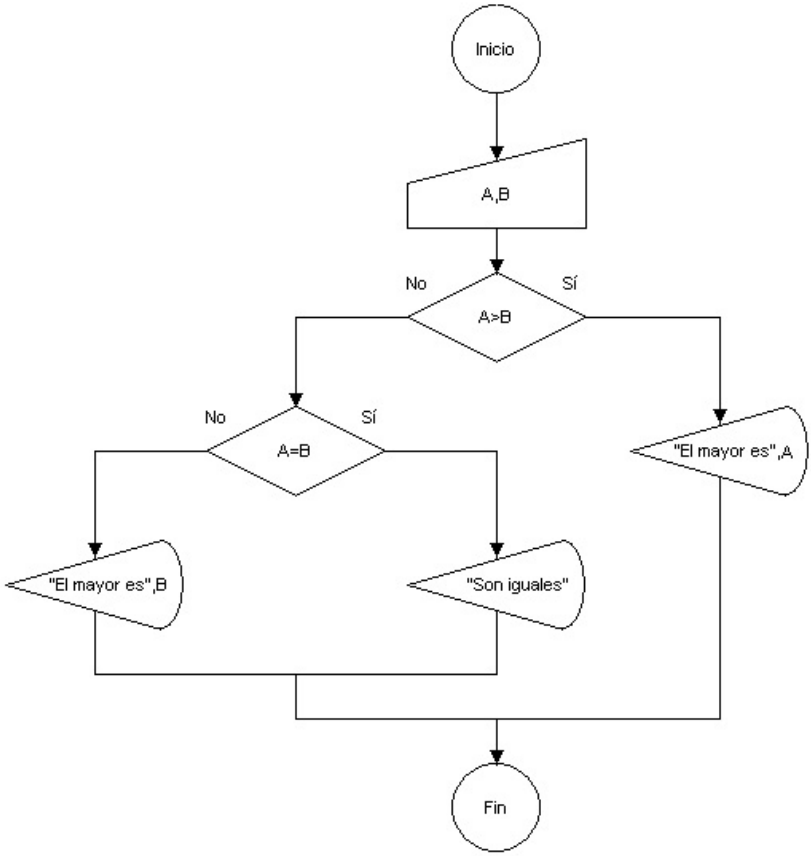
Para solucionar el problema anterior debemos utilizar *decisiones anidadas.*

En el objeto de salida del diagrama se pueden combinar texto y variables como en el caso anterior, siempre y cuando estén separados por comas.	

2. 3. 1. 3. Estructura de control de decisiones anidadas

Existe la posibilidad de tener una decisión dentro de otra, a esto se lo llama *decisiones anidadas* y se usa cuando tenemos más de una alternativa. Para resolver el problema anterior (ingreso de dos números iguales), desarrollamos las indicaciones en el siguiente problema.

Problema 6: Mostrar el número más grande (entre dos) ingresado por teclado. Si los dos números son iguales mostrar el cartel "Son iguales".

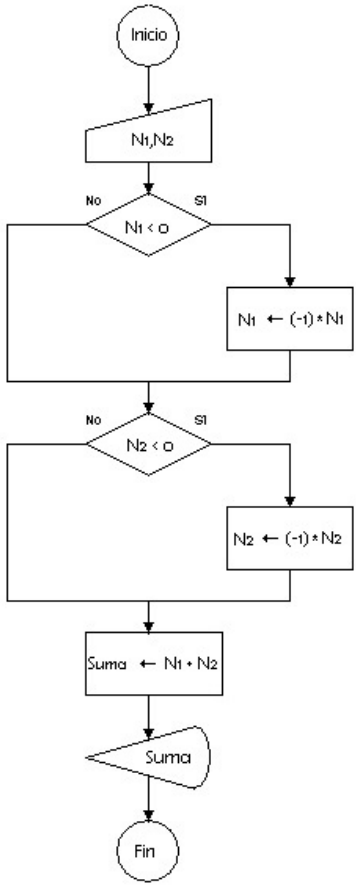


Nota: Y si la primera condición es A=B ¿Cómo cambiarías el algoritmo para que siga funcionando perfectamente? ¿Te animás a hacerlo? **Atención:** La anidación también se puede hacer dentro del camino del **Sí** y además se podría producir más de una vez.

2. 3. 1. 4. Estructura de control de decisiones independientes

Las decisiones independientes son aquellas que se establecen sin depender una de otras, es decir, sin estar relacionadas. Simplemente se ubican una debajo de la otra y en cualquier orden, ya que el orden no afectará su ejecución. Veamos un ejemplo.

Problema 7: Ingresar dos números por teclado y sumarlos. En caso que los números sean negativos, previo a la suma se debe cambiar su signo.



Nota: En este caso tenemos dos decisiones simples que son independientes (podrían estar en cualquier orden). La operación **N = (-1)*N** hace cambiar el signo de negativo a positivo.

2. 4. Operadores de relación

La siguiente tabla nos muestra los distintos operadores de relación entre dos números:

Operador	Significado	Equivalente en matemática
>	Mayor que	>
<	Menor que	<
>=	Mayor o igual que	≥
<=	Menor o igual que	≤
=	Igual	=
!=	Distinto	≠

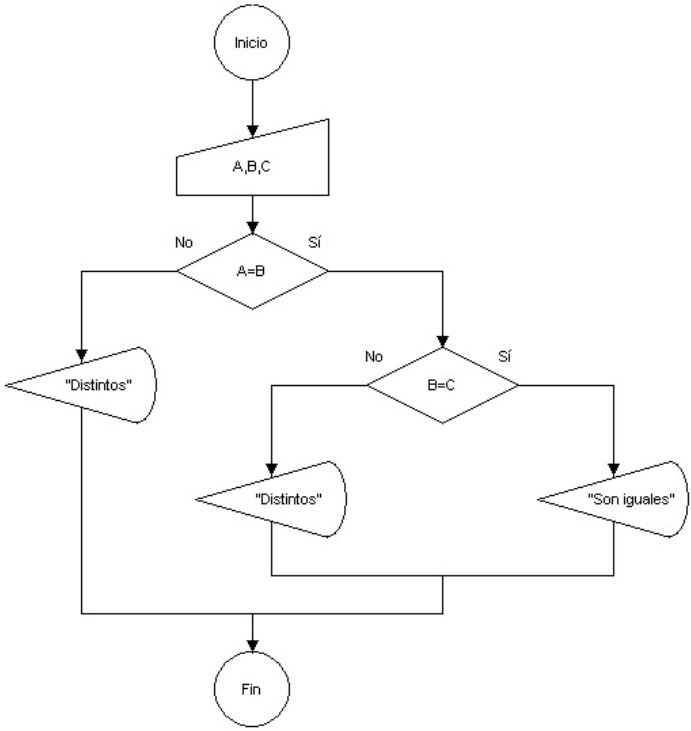
Nota: Es importante el orden en los símbolos cuando interviene el igual >= y <=.

Estos operadores nos permiten establecer comparaciones entre variables.

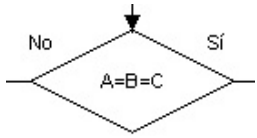
Ejemplo: Establecer una comparación alfabética entre dos variables alfanuméricas. **Provincia1="Neuquén" Provincia2="Chubut"** entonces decimos que la expresión **(Provincia1>Provincia2)** es verdadera y la expresión **(Provincia1<Provincia2)** es falsa.

2. 5. Operadores lógicos

Supongamos que quisiéramos saber si tres números son iguales utilizando decisiones. Una solución posible sería:



Es decir que una decisión depende de la anterior.

Lo incorrecto es que suceda lo siguiente, ya que las desigualdades siempre deben relacionarse de a pares.	

Para solucionarlo es necesario trabajar con los operadores lógicos «and», «or» y «not».

2. 5. 1. Conjunción lógica o producto lógico «and»

El efecto del operador «and» es la evaluación simultánea del estado de verdad de las variables lógicas involucradas.

Así por ejemplo la expresión lógica: **A and B**, será verdadera únicamente si A y B lo son. Cualquier otro estado para ambas variables dará como

resultado el valor falso, puesto que basta con que una de las dos variables tenga valor falso para que ambas no sean simultáneamente verdaderas.

Variables lógicas		Resultado A and B
A	B	
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Falso

2. 5. 2. Disyunción lógica inclusiva o suma lógica «or»

El efecto de este operador es la evaluación no simultánea del estado de verdad de las variables lógicas involucradas. Esto implica que al tener estado verdadero por lo menos una de las variables afectadas, la operación dará un resultado verdadero.

Así tendremos que la expresión: **A or B**, será falsa únicamente cuando el estado de ambas variables sea falso. En cualquier otro caso, la operación será verdadera.

Variables lógicas		Resultado A and B
A	B	
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

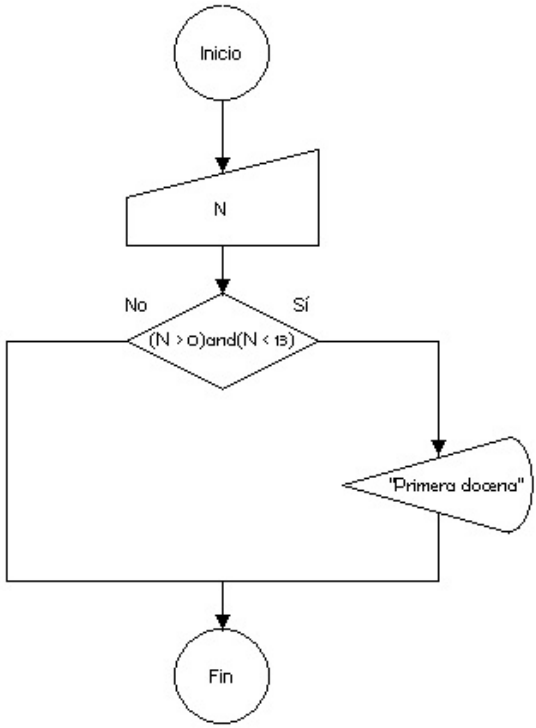
2. 5. 3. Negación o complemento lógico «not»

El efecto de este operador es negar el valor de la expresión lógica, como se indica en la siguiente tabla.

Variables lógicas A	Resultado not A
Verdadero	Falso
Falso	Verdadero

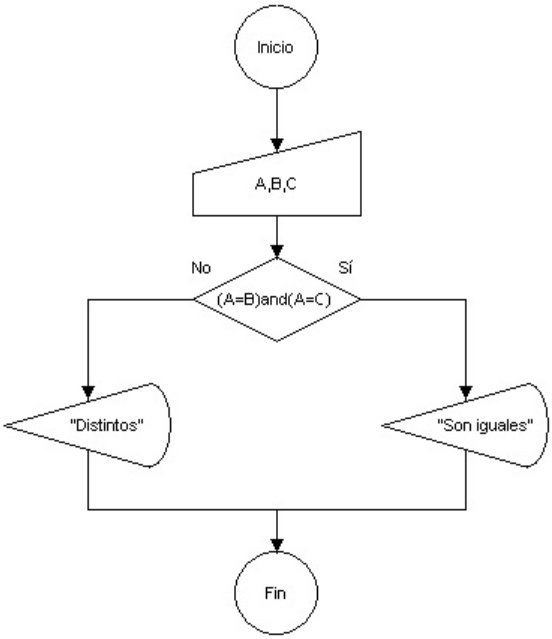
Veamos algunos ejemplos para poder comprender un poco más el concepto de operadores lógicos.

Problema 8: Solicitar al usuario un número natural y verificar que el número ingresado se encuentre dentro de la primera docena de números naturales, es decir entre el 1 y el 12.



Nota: La decisión podría reescribirse como: **(N>=1) and (N<=12)**. No es posible indicarlo como: **1<=N<=12**. Esto generaría un error, porque no sería una expresión lógica.

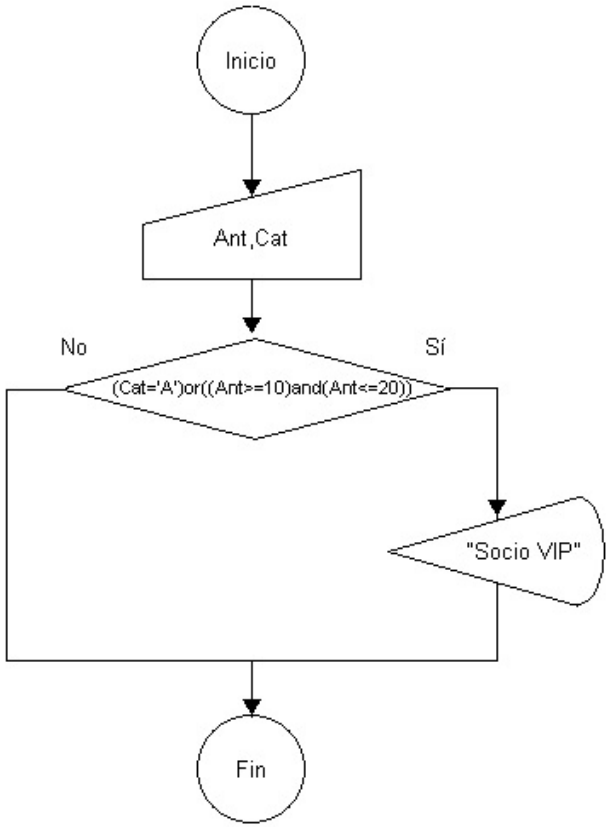
Ahora el problema de los tres números iguales se puede resolver usando el operador lógico «and» de la siguiente manera:



Nota: Si al evaluar **(A=B)** se obtiene un resultado falso entonces no es necesario evaluar la expresión **(A=C)**. Esto se denomina evaluación de circuito corto, si la primera expresión es falsa como el operador que las vincula es un **and** aunque la segunda sea verdadera el resultado será falso, de ahí que no tenga sentido realizar la evaluación.

Una condición puede expresarse en forma lógica, usando más de un operador (o, y, negación). A continuación se presenta un ejemplo.

Problema 9: Se ingresa por teclado la categoría de un socio del club deportivo Sol Naciente y su antigüedad en años. Las categorías posibles son A, B y C. Luego se desea saber si el socio ingresado tiene categoría A o su antigüedad se encuentra entre los 10 y 20 años, en esos casos se pide mostrar un cartel que exprese lo siguiente: "Socio vip".



La expresión lógica: **(Cat='A')or((Ant>=10)and(Ant<=20))** permite asegurar que el socio cumple con las condiciones impuestas.

Atención: En FreeDFD para expresar un valor alfanumérico es necesario usar comillas simples. Ejemplo: **Cat ← 'B'**. En este caso a la variable **Cat** se le asigna el valor **'B'**. En cambio **Cat='B'** expresa una comparación entre el valor de la variable y el carácter alfanumérico **'B'**. Es un error en una asignación expresar **'A' ← Cat** porque la variable debe ir a la izquierda y el valor asignado a la misma a la derecha, pero si se tratara de una condición da lo mismo: **Cat='A'** o **'A'=Cat**. ¿Qué pasaría si ingresa **'a'** (minúscula)? ¿Nuestro socio será VIP? Si no funciona, ¿cómo podríamos solucionarlo?

Es común cometer errores usando operadores lógicos, por ejemplo:

(A>B) and (A=B)
(N<10) and (N>20)

¿Cuándo se cumplirán estas condiciones?

2. 6. Otros operadores matemáticos especiales

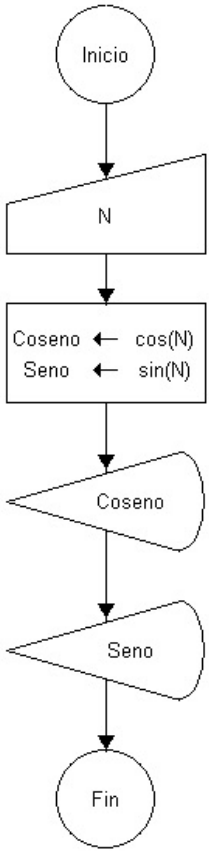
Además de los operadores básicos matemáticos como (+,-,/ y *) existen otros operadores que se irán trabajando a lo largo del texto. En este capítulo veremos los que son propios de la sintaxis de FreeDFD.

La siguiente tabla muestra algunos de los operadores matemáticos especiales, para N, N1, N2 variables numéricas:

Operación	Símbolo	Sintaxis
Potencia	^	N1^N2
Raíz cuadrada	sqrt	sqrt(N)
Valor absoluto	abs	abs(N)
Seno	sin	sin(N)
Coseno	cos	cos(N)
Tangente	tan	tan(N)
Logaritmo neperiano	ln	ln(N)
Logaritmo decimal	log	log(N)
Exponencial e	exp	exp(N)
Número aleatorio	random	random(N)
Módulo	mod	N1 mod N2

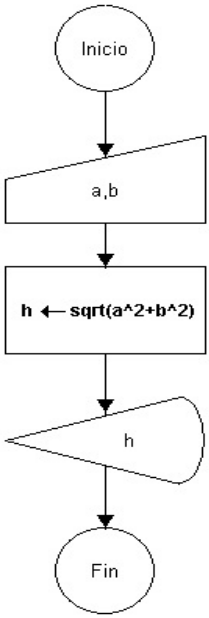
A continuación se trabajarán unos ejemplos con los operadores analizados.

Problema 10: Se lee desde el teclado el valor de un ángulo en grados. Se desea mostrar el coseno y el seno.

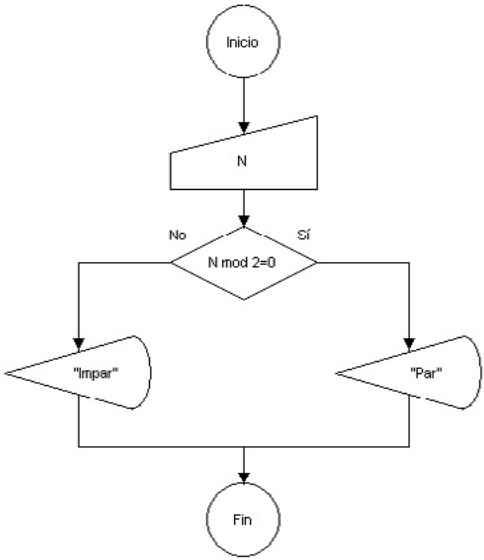


Nota: En el FreeDFD se puede optar por trabajar con ángulos en radianes o en grados simplemente clickeando en «Opciones» del menú. **Importante:** No es posible utilizar como identificadores de variables las palabras cos, sin, tan, abs, etcétera, por ser palabras reservadas del lenguaje de FreeDFD.

Problema 11: Se ingresan por teclado los catetos de un triángulo rectángulo. Se desea hallar y mostrar su hipotenusa.

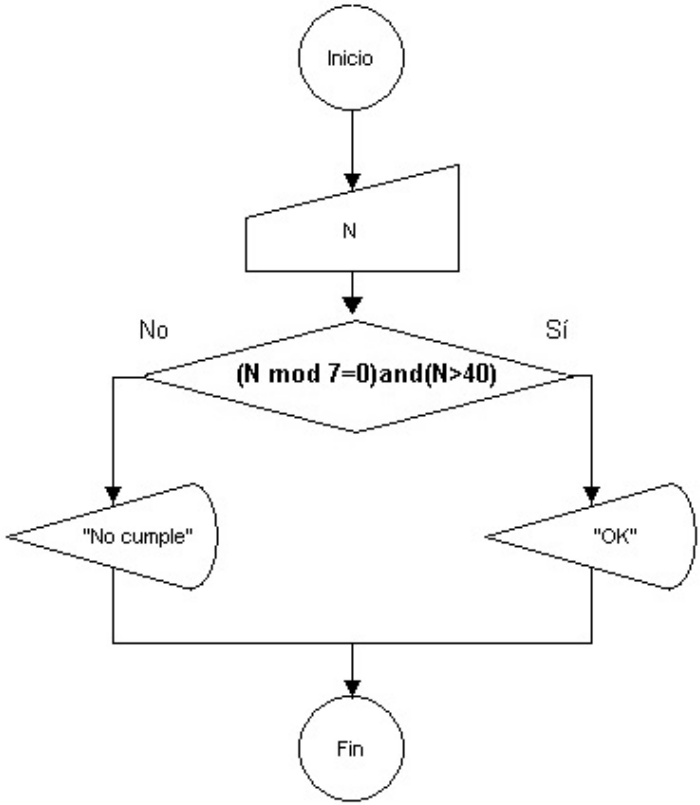


Problema 12: Ingresar un número natural por teclado. Se desea saber y mostrar si es par o impar.



El operador «mod» nos devuelve el resto de la división entre números enteros. Para este caso cualquier número natural (1,2,3,..,∞) dividido 2 nos da un 1 o un 0 (cero), entonces los números pares tendrán como resultado un 0, mientras que los impares un 1.

Problema 13: Ingresar un número entero para saber si es divisible por 7 y es mayor a 40.



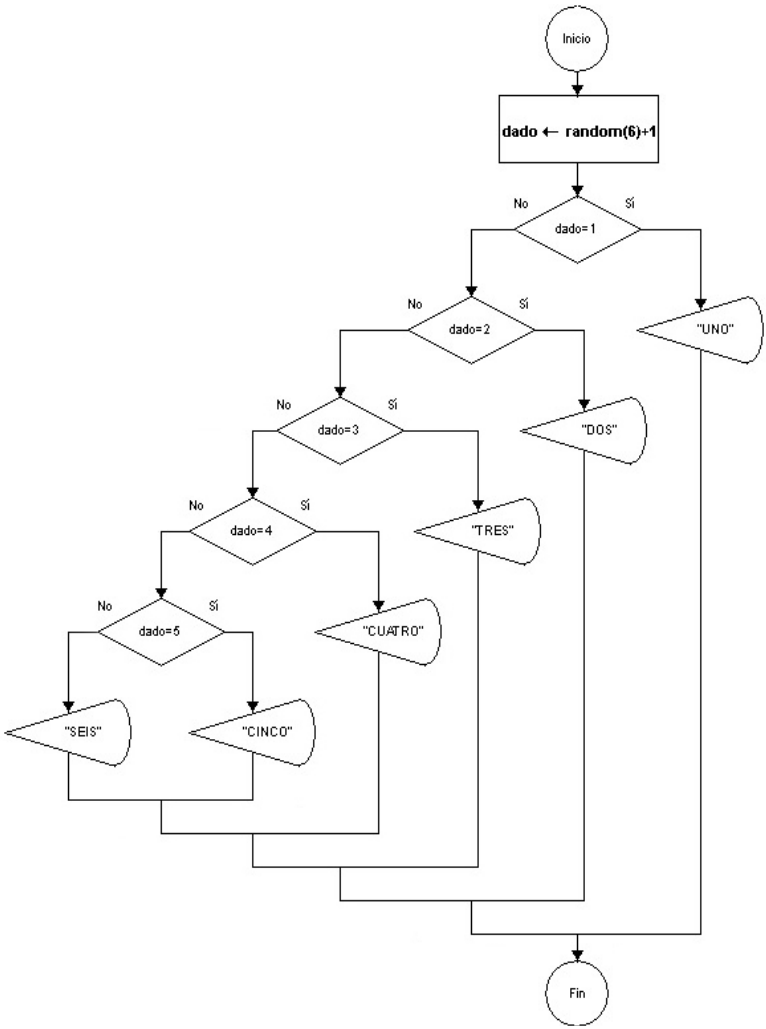
Al tratarse de dos condiciones que se requiere que sean verdaderas, es necesario utilizar el operador lógico «and».

El operador «random» sirve para generar números al azar. Por ejemplo, si necesitamos generar números enteros entre el 0 y el 9 entonces la sintaxis será **random(10)**, es decir que el valor N que se indica entre paréntesis permitirá generar números entre el 0 hasta el N-1.

La siguiente tabla muestra algunas variantes para conseguir números aleatorios:

Ejemplos	Rango de valores obtenidos
random(2)	0 y 1
radom(10)+1	Del 1 al 10
random(21)-10	Del -10 al 10
random(100)/10	Del 0,0 al 9,9
random(101)/10	Del 0,0 al 10,0
random(1001)/100-5	Del -5,00 al 5,00
random(10)*2	0, 2, 4, 6, 8, 10,…,18

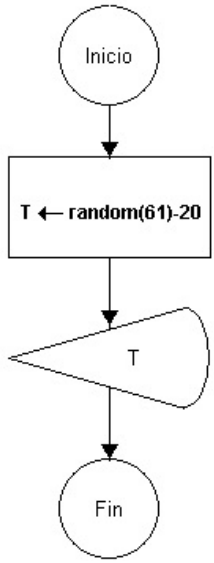
Problema 14: Mostrar en letras el número de la cara de un dado obtenido al azar.



Sabemos que son 6 las caras de un dado, entonces la expresión **random(6)+1** genera números del 1 al 6. ¿Qué podría suceder si cambiamos la primera instrucción de asignación por **dado ← random(7)**?

Nota: La última decisión del ejercicio anterior funciona por descarte ya que no necesita realizar otra consulta en el **No**, se da por descartado que se trata de un 6. En estos casos cuando se tiene N casos posibles las decisiones serán N-1.

Problema 15: Generar aleatoriamente una temperatura entre los -20º y los 40º y mostrar el resultado.



2. 7. Problemas: Diagramación lógica – Estructura de control de decisión

1. Diseñar un algoritmo que, dados dos números, muestre por pantalla su suma.
2. Realice un algoritmo que solicite dos datos: país y capital. Y luego muestre la capital del país. El cartel debe ser como lo indica el siguiente ejemplo: "Katmandú es la capital de Nepal".
3. Crear un algoritmo que muestre por pantalla el doble y el triple de un número ingresado por teclado.
4. Diseñar un algoritmo que imprima el cuadrado y el cubo de un número ingresado por teclado.

5. Diseñar un algoritmo que pida un número por teclado y luego imprima el número siguiente al ingresado.
6. Diseñar un algoritmo que genere un número aleatorio del 0 al 200, lo muestre y luego calcule y muestre el mismo número aumentado en un 30 %.
7. Diseñar un algoritmo que genere un número aleatorio del 10 al 50, lo muestre y luego calcule y muestre el mismo número disminuido en un 15 %.
8. Diseñar un algoritmo que, dados tres números enteros, calcule e imprima el promedio entre ellos.
9. Diseñe un algoritmo para ingresar dos palabras (A, B) y luego realice el intercambio de sus valores. Finalmente mostrar el contenido de A y de B.
10. Diseñar un algoritmo que imprima el área y el perímetro de un rectángulo ingresando su base y altura.
11. Realice un algoritmo que calcule el volumen de un cilindro a partir de los valores de su radio y altura.
12. Crear un algoritmo que convierta y muestre un valor ingresado en centímetros a yardas, metros, pies y pulgadas.
13. Diseñar un algoritmo que convierta y muestre la temperatura en Fahrenheit ingresando la temperatura en Celsius.
14. Diseñar un algoritmo que calcule el volumen de un cilindro dados su radio y altura (primero el programa deberá verificar si son positivas).
15. Crear un algoritmo que calcule si dos números son divisibles. Para ello, se piden un primer número y un segundo número, entonces mostrar un cartel que diga "es divisible" si el segundo número es divisible al primero.
16. Diseñar un algoritmo para calcular el porcentaje de hombres y de mujeres que hay en un grupo, dados los totales de hombres y de mujeres.
17. Diseñar un algoritmo que indique con carteles si el número ingresado es negativo, positivo o nulo.
18. Ingresar tres números y mostrar el mayor (asuma que todos son distintos entre sí).
19. Realice un algoritmo para mostrar un cartel que indique si un triángulo es «escaleno», «equilátero» o «isósceles» ingresando sus lados.
20. Diseñar un algoritmo que imprima con un cartel el número de docena («primera», «segunda» o «tercera») dado el resultado de una jugada de ruleta (del 0 al 36). Utilizar el operador lógico and.
21. Crear un algoritmo que permita obtener y mostrar la cantidad de dígitos de un número ingresado, sabiendo que el máximo permitido es 4.

22. Crear un algoritmo que muestre cada uno de los dígitos de un número ingresado por el usuario. El máximo permitido es de 4 dígitos. Por ejemplo: si se ingresa el número 187, entonces debe mostrar en un único cartel lo siguiente: "d1 = 0, d2 = 1, d3 = 8 y d4 =7".
23. Ídem al anterior pero se pide que muestre la suma de los dígitos del número ingresado (máximo 4 dígitos).
24. Diseñar un algoritmo que imprima con un cartel «Correcto» según el siguiente caso: si el numero N es múltiplo de 5 y se encuentra entre los 25 primeros números. N debe ser obtenido aleatoriamente entre números del 1 al 1000. Primero debe mostrar N.
25. Diseñar un algoritmo que ingresando un numero de 5 dígitos detecte si es capicúa y muestre un cartel «Es capicúa» o «No es capicúa» según el resultado.
26. Crear un algoritmo que muestre las soluciones de una ecuación cuadrática (usando Bhaskara), a partir del ingreso de los valores de los coeficientes del polinomio. El polinomio tiene la forma: $P(x) = ax^2+bx+c$
27. Probar con: a) $P(x) = x^2+3x+2$ b) $P(x) = 2x^2+4x+2$ c) $P(x) = 3x^2+2$

2. 8. Estructura de control de repetición

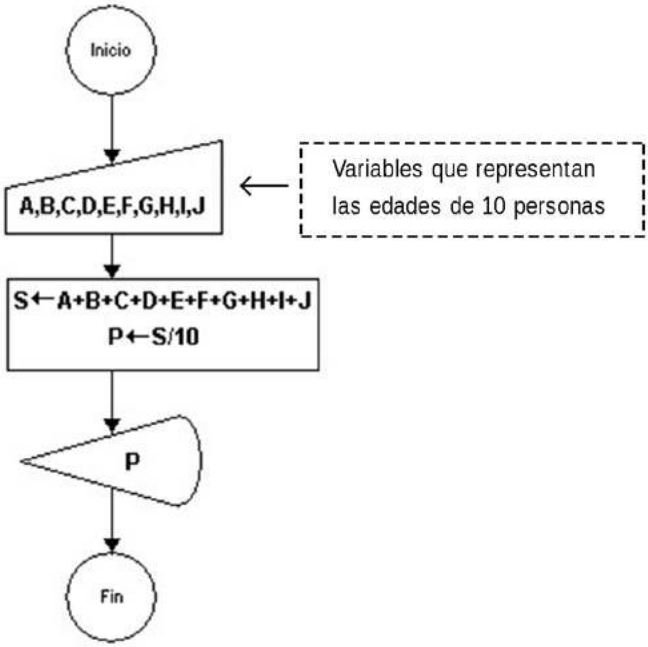
Hasta el momento, se han presentado problemas que no requieren de repeticiones de sentencias o acciones. Sin embargo al momento de resolver problemas, es muy común encontrarnos con situaciones repetitivas.

Pensemos en el siguiente problema: se desea determinar el promedio de las edades de 5 personas. Una solución posible podría ser leer 5 edades (variables) y luego calcular su promedio. Ahora bien, ¿qué sucede si en lugar de 5 personas son 100 o un valor N que ingresa el usuario?

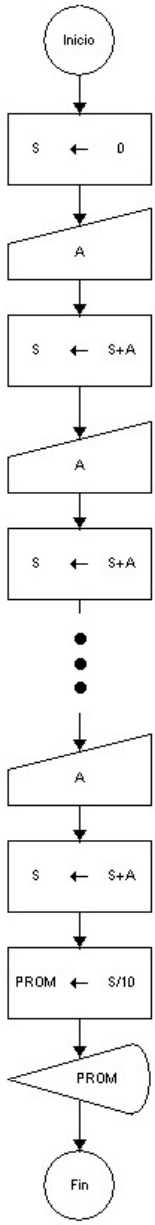
No es lo mismo hallar el promedio de tres números que el promedio de 100 o de un valor N que ingresa el usuario. Veamos justamente cómo podríamos resolver el problema propuesto con las herramientas trabajadas hasta el momento para tener una mejor idea.

Problema 16: Ingresar 10 números (edades de personas) y luego hallar y mostrar su promedio.

Solución 1



Solución 2



Nota: Por una cuestión de espacio se escribieron puntos suspensivos en reemplazo de lectura A y la asignación **S ← S + A**.

La primera solución resuelve el problema en pocas líneas, pero utiliza una gran cantidad de variables. La segunda solución utiliza pocas variables, pero muchas líneas de comando que se repiten de a pares (leer y sumar).

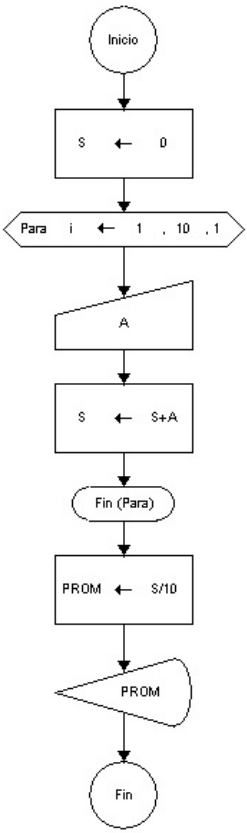
Basta imaginar si el algoritmo en vez de resolver el promedio con diez números debiera resolverlo para cien, necesitaríamos para la solución 1 cien variables y para la solución 2 más de doscientas líneas de trabajo.

Para poder tratar con este tipo de situaciones, existen las estructuras de control de repetición: «para», «mientras» y «hacer-mientras».

2. 8. 1. Estructura de control de repetición (sentencia «para»)

Cuando se desea ejecutar un conjunto de acciones un determinado número de veces, usamos la sentencia «para».

En estos casos se requiere que conozcamos por anticipado el número de repeticiones. Para solucionar el problema del ejercicio anterior, hacemos lo siguiente:

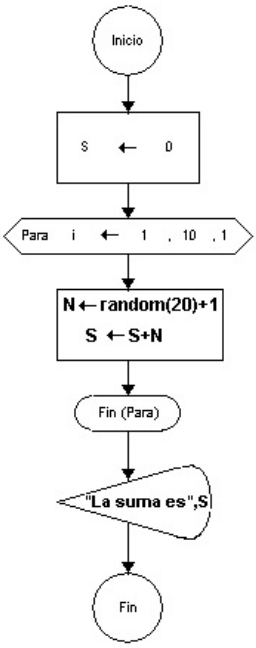


De esta manera todo lo que encierra desde **Para** hasta **Fin (Para)** se repite tantas veces como lo indica el control de repetición.

La variable **S** funciona como sumador o acumulador de **A**.

La variable i comienza a contar los ciclos (vueltas) desde 1 hasta 10 con saltos de un paso.	Para i ← 1 , 10 , 1 inicio salto fin Variable contadora de vueltas
Se repiten las instrucciones que se encuentran entre el **Para** y el **Fin (Para)**.	Fin (Para)

Problema 17: Hallar la sumatoria de 10 números generados al azar. Los números deben comprender el siguiente rango de valores [1...20].



Preguntas: Esta solución, ¿resuelve el problema? ¿Cuántas veces se ejecutan las instrucciones que se encuentran dentro del **Para**? ¿En qué cambia el algoritmo si quisiéramos hacer esto para 200 números?

2. 8. 2. Variables como acumuladores

Es una variable que, como su nombre lo indica, va a ser usada para sumar sobre sí misma un conjunto de valores. Cuando se utiliza dentro de un ciclo de repetición Para, al finalizar el mismo, esta variable contendrá la sumatoria de todos los valores que cumplen una determinada condición (también puede servir para decrementar valores variables). Es necesario haber inicializado su valor antes del comienzo de un ciclo de repetición «para».

La inicialización consiste en asignarle al sumador un valor inicial, es decir el valor desde el cual necesitamos se inicie la sumatoria (por lo general comienzan en cero).

Ejemplo: **S = 0**
S = S + N

2. 8. 3. Variable contadora

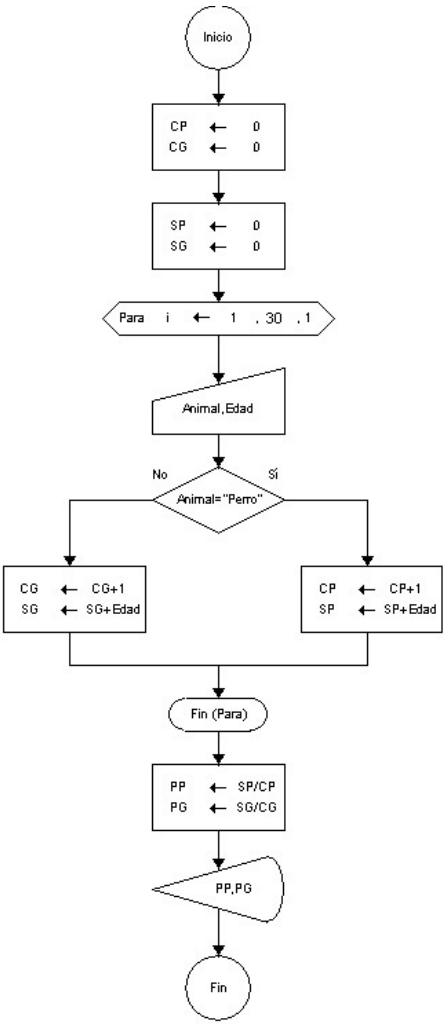
Es una variable que se encuentra en ambos miembros de una asignación a la que se le suma un valor constante. Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante cada vez que se produce un determinado suceso, acción o iteración. Los contadores se utilizan con la finalidad de registrar la cantidad de sucesos, acciones o iteraciones internas en un bucle, proceso, subrutina o donde se requiera cuantificar. Como cualquier variable es necesario inicializarla antes del comienzo de un ciclo de repetición.

La inicialización implica darle un valor inicial, en este caso, el número desde el cual necesitamos se inicie el conteo (por lo general comienzan en cero).

Ejemplos: **C=0** (inicializa la variable C en 0)
C=C+1 (incrementar)
H=H-1 (decrementar).

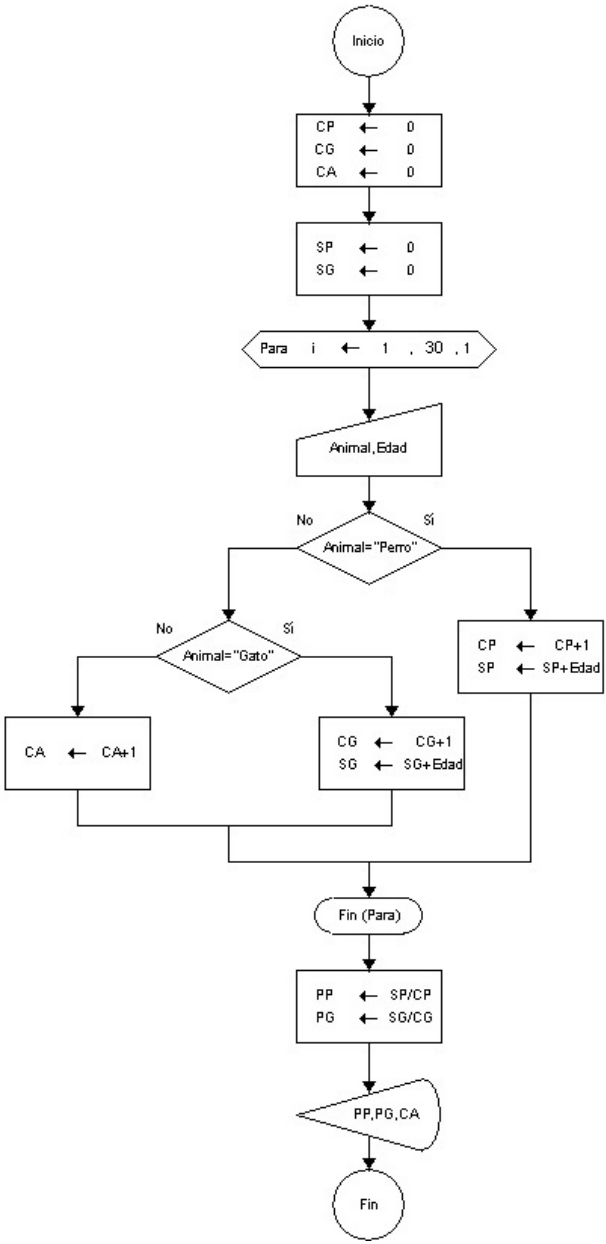
Nota: La diferencia entre un sumador y un contador es que mientras el primero va aumentando en una cantidad fija preestablecida, el acumulador va aumentando en una cantidad o valor variable. Tanto si es una variable contadora sumadora o acumuladora, es necesario asignarle un valor inicial antes de comenzar a utilizarla, por ejemplo en un bucle. Cuando se desean calcular promedios o porcentajes se requiere de ambos tipos de variables.

Problema 18: En una veterinaria se desea saber el promedio de edad de gatos y perros (por separados) que fueron asistidos durante un mes. En total se registraron 30 animales y la veterinaria solo atiende gatos y perros.



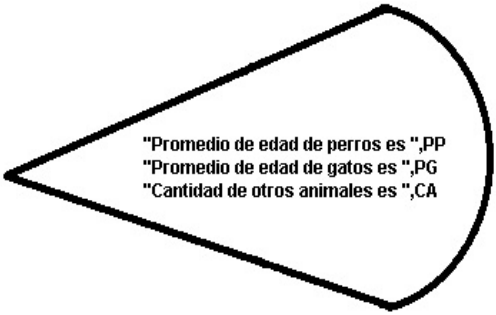
Atención: Es muy importante tomar en cuenta que los promedios siempre deben ser calculados fuera del ciclo de repetición **Para - Fin (Para)**. ¿Qué sucedería si se calculan dentro del ciclo? Recordar que el ingreso de caracteres, en este caso el tipo de animal (gato/perro) debe estar entre comillas: "Gato" y "Perro".

Problema 19: Se pide lo mismo que el problema anterior, pero la diferencia radica en que no solo la veterinaria atiende gatos y perros, puede que sean otros animales también. Justamente lo que se pide es además contar la cantidad de esos animales que no son ni gatos y ni perros.



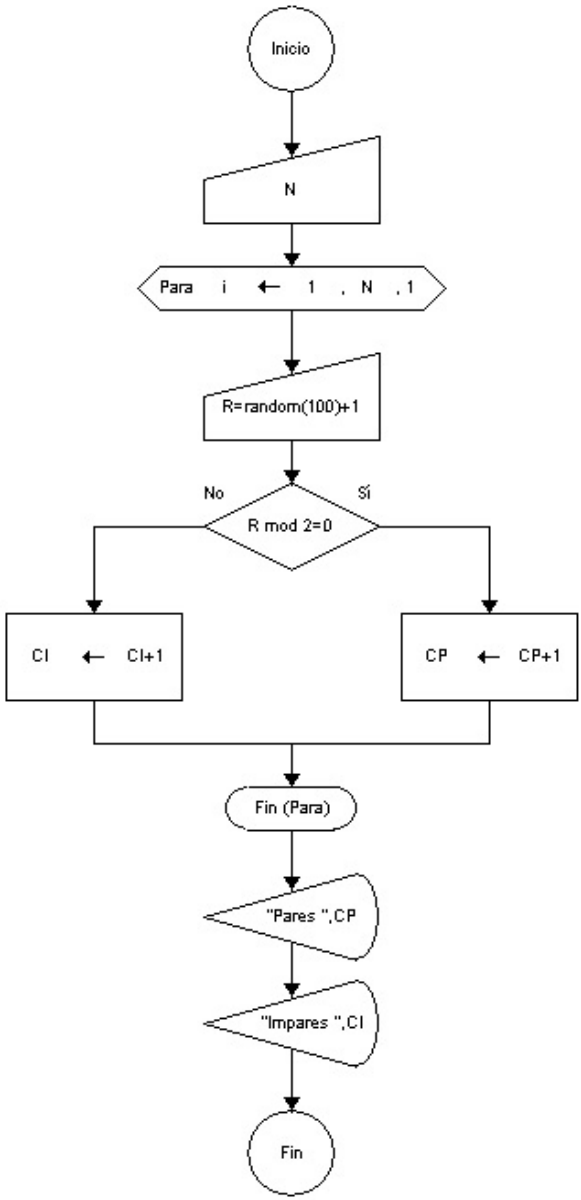
Es decir que, por descarte, otro tipo de animal caería en el último **No** de la decisión y, entonces, incrementaría en uno la variable **CA**.

Importante: Todo contador y acumulador (sumador) debe iniciarse con un número antes del comienzo del ciclo de repetición, en el caso anterior con cero.
Nota: Para una mejor comprensión en las respuestas, lo ideal sería en la salida combinar carteles y resultados, por ejemplo para el problema anterior.



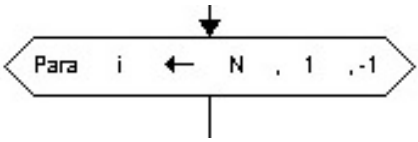
La cantidad de iteraciones del bucle «para» también se puede determinar ingresando el dato necesario. A continuación un ejercicio relacionado.

Problema 20: Solicitar al usuario que ingrese un número entero N, luego generar en forma aleatoria N números enteros comprendidos entre 1 y 100 y determinar cuántos son pares y cuántos impares.



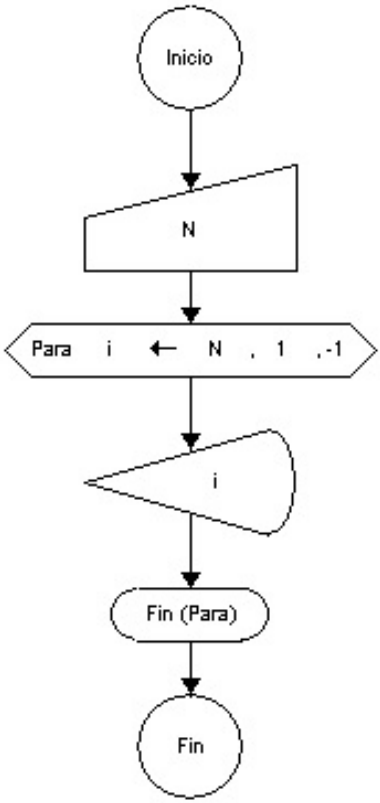
Pregunta: ¿Qué precondición debe cumplir N? ¿Qué sucede si el número que ingresa el usuario es -1?

También podemos usar el ciclo decrementando el contador de vueltas del **Para**. Desde N a 1 mientras el paso sea -1.

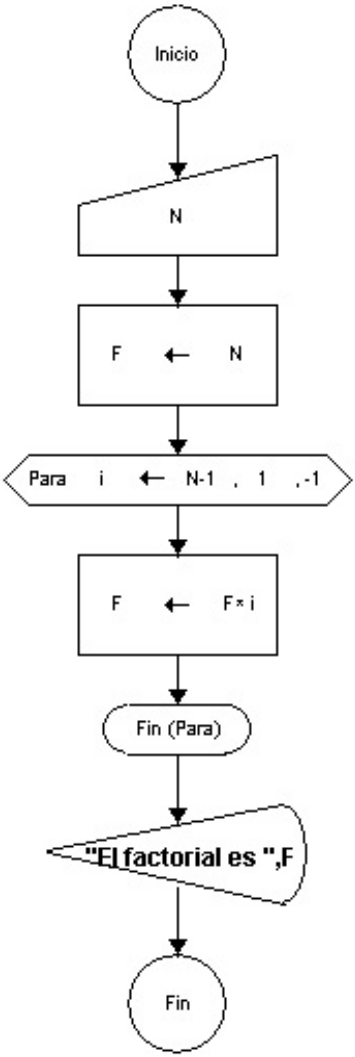


A continuación un ejemplo relacionado.

Problema 21: Solicitar al usuario que ingrese un valor N y mostrar todos los valores comprendidos entre N y 1, comenzando desde N.

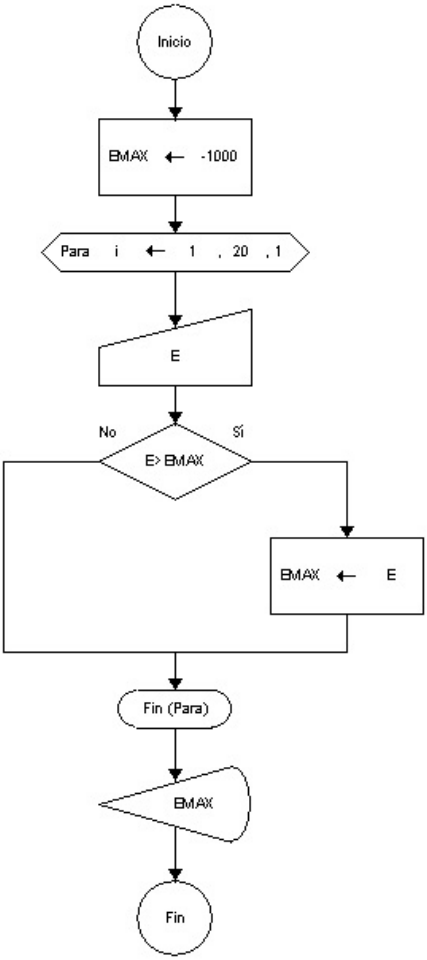


Problema 22: El factorial de un número entero se denota de la siguiente manera «n!» y su resultado es n!=n*(n-1)*(n-2)*...*1. Por ejemplo: 5!=5*4*3*2*1 siendo el resultado 120. Se pide desarrollar un programa que lee un valor N y determine su factorial.



2. 8. 4. Máximos y mínimos

Problema 23: Hallar la persona de mayor edad, sabiendo que se leen datos correspondientes a 20 muestras.



El valor inicial de la variable **EMAX** es un número absurdo por ser una edad de un número negativo (-1000). Cuando se ingresa el primer valor **E** dentro del bucle, la condición **(E > EMAX)** se cumple por ser verdadera, entonces el valor de **EMAX** cambia por el valor de la edad **E**.

A medida que se continúa ejecutando el bucle, la sentencia evalúa si aparece un valor mayor o no a **EMAX**, en caso afirmativo se le asigna al mismo el nuevo valor de **E**.

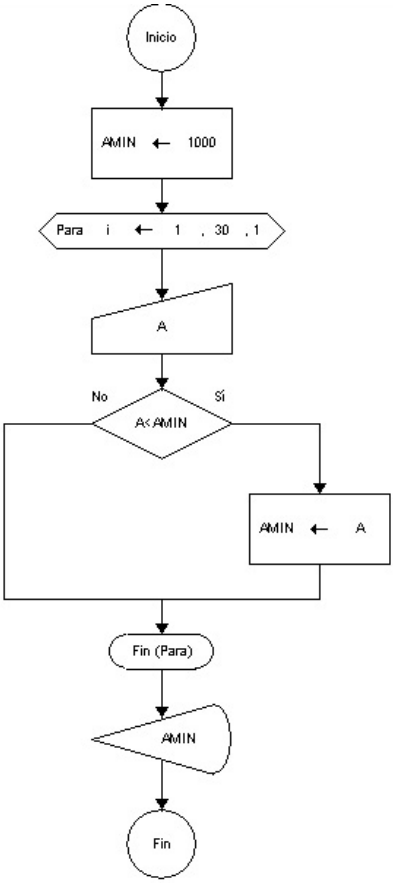
Tomar en cuenta que si se produce el caso de empate la condición es falsa, por lo tanto, no sufre efecto.

Si el ejercicio además solicitara hallar la mínima edad entre las 20 personas, entonces será necesario agregar otra variable, por ejemplo **EMIN** y asignarle un valor extremadamente grande de entrada (por ejemplo: **EMIN = 1000**) y la condición (**E < EMIN**) para la decisión.

A continuación se muestra un ejemplo de cómo hallar el mínimo.

Nota: ¿Qué alternativa sugerís, de manera de evitar la asignación de valor absurdo a **EMAX** y **EMIN**?

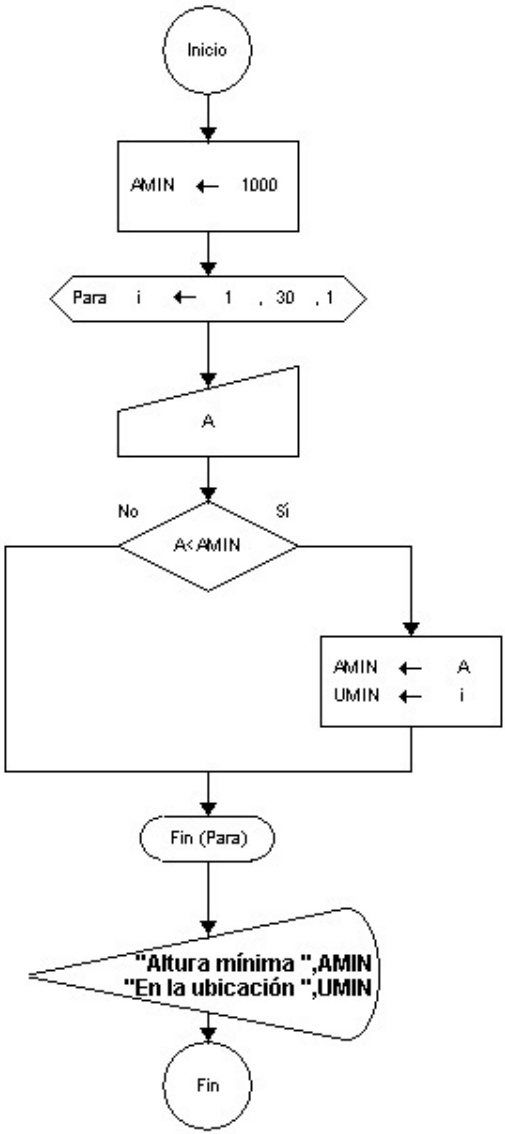
Problema 24: Hallar la persona de menor altura, sabiendo que se leen datos correspondientes a las alturas de 30 personas. El ingreso es en números enteros y en cm.



El valor absurdo de entrada será **AMIN=1000** (ninguna persona mide 10 m).

¿Qué sucede si además de registrar la mínima altura de la persona, se desea conocer en qué ubicación (orden dentro de la lista de valores leídos) se encuentra tal persona?

Problema 25: Mostrar la mínima altura registrada de un grupo de 30 personas y además en qué ubicación se encuentra.



A la variable **UMIN** se le asigna el valor de la variable de iteración **i** del **Para**. Por ejemplo, si la altura mínima se produce en la vuelta 8 significa que **i** es igual a 8.

De esta manera se tiene el registro de cuándo se produce la última condición verdadera de la decisión.

En caso de empate, el único registro que se produce es el primero que llega.

2. 8. 5. Problemas: Diagramación lógica –Estructura de control de repetición «para»

28. Mostrar por pantalla los números del 10 al 1.
29. Mostrar por pantalla las tres primeras potencias de los números del 1 al 5.
30. Dado un número, mostrar por pantalla su tabla de multiplicar (del 1 a 10).
31. Mostrar por pantalla la cantidad de personas mayores de edad (≥18) de un total de N edades leídas desde teclado.
32. Mostrar por pantalla la cantidad de mujeres y hombres (M/H) de un total de N valores leídos desde teclado. Mostrar también su porcentaje.
33. Mostrar por pantalla la cantidad de mujeres mayores de edad y la cantidad de hombres menores de edad de un total de N edades y sexo leídos desde teclado.
34. A un grupo de 10 personas se les consulta la edad y se desea calcular el promedio de edad del grupo. Mostrar el promedio y cuántas de las 10 personas son mayores de 18 años, leyendo la información desde teclado.
35. Se desea conocer el peso acumulado de 10 personas. ¿En qué cambiaría la solución si ahora son 100 personas?
36. Se desea conocer el peso promedio de 5 personas.
37. Realice un algoritmo que permita ingresar 10 edades de personas para luego hallar y mostrar el % de gente mayor de edad (≥ 18) y el % de menores de edad.
38. Crear un algoritmo que genere al azar 5 números de la ruleta (del 0 al 36) y muestre el % de números pares, % de impares y % de ceros generados.
39. Ingresar 10 temperaturas por teclado y mostrar la mayor.
40. Se ingresan 10 pares de temperaturas (T1 y T2). Hallar el promedio de las temperaturas T1 y el promedio de las temperaturas T2.

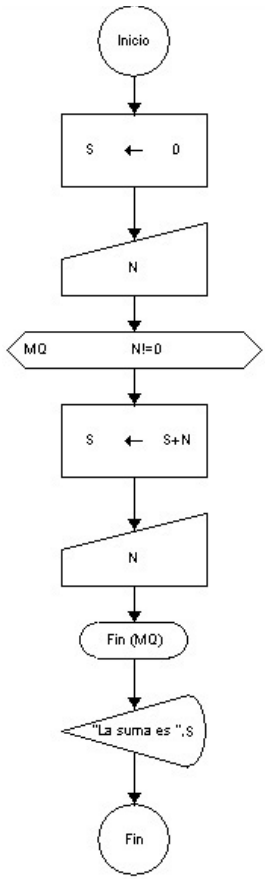
2. 8. 6. Estructura de control de iteración (sentencia «mientras»)

Esta estructura de control permite repetir una instrucción o grupo de instrucciones mientras una expresión lógica sea verdadera. De esta forma, la cantidad de veces que se reiteran las instrucciones *no necesita* conocerse por anticipado, sino que *depende* de una condición.

Lo primero que hace esta sentencia es evaluar si se cumple la condición. En caso que se cumpla se ejecuta el bucle. Si la primera vez que se evalúa la condición esta no se cumple, entonces no se ejecutará ninguna acción. En otras palabras, mientras la condición se cumpla el bucle sigue iterando, por eso es importante no caer en ciclos de repetición infinitos.

Veamos como ejemplo el siguiente problema.

Problema 26: Calcular la suma de los números ingresados por teclado hasta que se ingrese un cero.

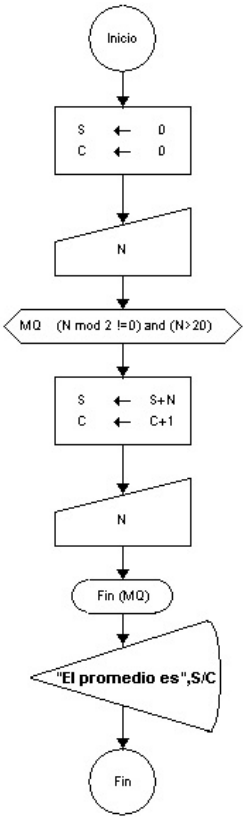


Es necesario leer el valor de la variable N antes de comenzar con el bucle «mientras», porque se debe evaluar la condición al entrar al ciclo de repetición. Luego es importante no olvidarse de volver a leer N dentro del bucle, porque si no lo hacemos caeremos en un ciclo de repetición infinito.

¿Cuál será el resultado si arrancamos ingresando un cero?

Mientras que la condición se cumpla el ciclo se sigue ejecutando. **MQ** (significa *mientras que*)	MQ N!=0 **Condición**
Se repite las instrucciones que se encuentran entre el **MQ** y el **Fin (MQ)**	Fin (MQ)

Problema 27: Hallar el promedio de números ingresados por teclado hasta que se lea un número impar o uno menor a 20.



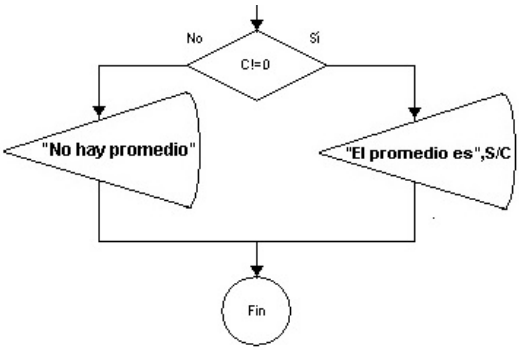
Nota: También se puede introducir en la salida la combinación de un cartel y un cálculo, como en el caso anterior donde se realiza el promedio **S/C** sin depender de otra variable: **prom ← S/C**.

¿Qué pasaría si de entrada ingresamos un número que no cumple la condición del «mientras»? Probemos con N=5 o con N=80.

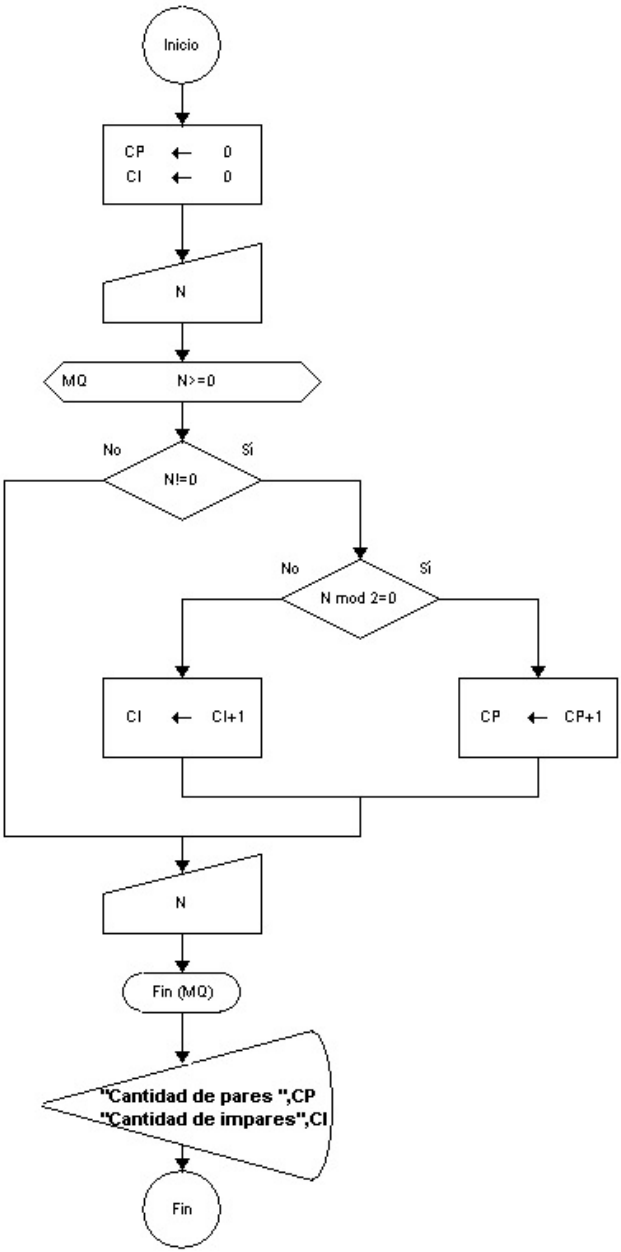
Se produce el error que muestra la figura:

Esto es así porque estamos dividiendo por cero, ya que C nunca pudo incrementarse. ¿Cómo podríamos solucionar este problema? ¿En qué casos es posible hallar el promedio?

Nota: Para poder solucionar este problema tendremos que recurrir a una decisión antes de efectuar la división, de la siguiente manera:



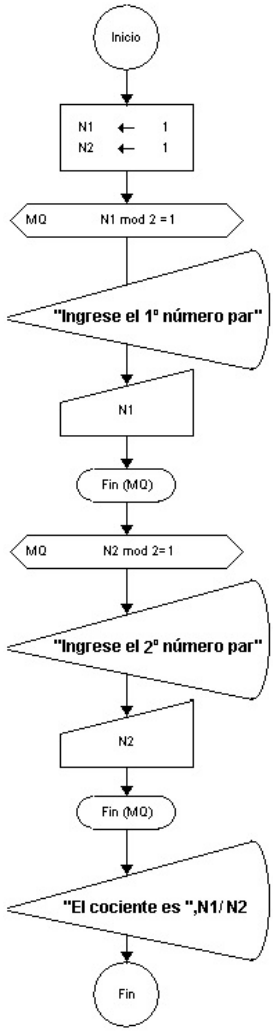
Problema 28: Se leen números que ingresa el usuario desde teclado, hasta que llega un valor negativo. Se pide que determine cantidad de impares y pares leídos. El cero no se cuenta.



Mientras la condición del ciclo de repetición cumpla con **N** mayor o igual a cero, se determina si dicho número es distinto a 0 y luego si es par o impar para incrementar **CP** o **CI** respectivamente.

Nota: No olvidarse de leer **N** antes del ciclo y dentro del mismo. ¿Qué sucedería si no leemos dentro del ciclo?

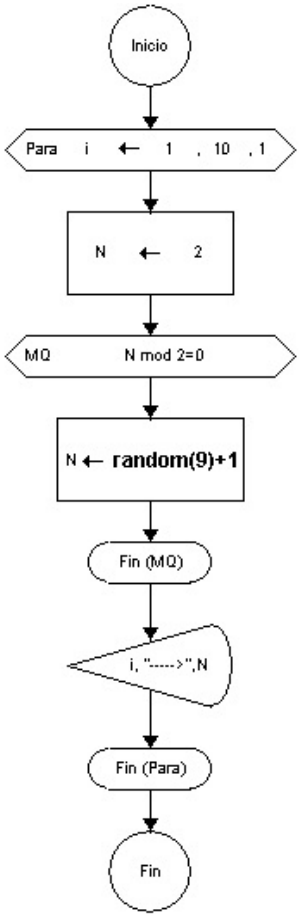
Problema 29: Diseñar un algoritmo que calcule cuánto es el cociente entre dos números (únicamente dos números pares). Si el usuario ingresa un número impar, le pide otra vez el número hasta ingresar uno que sea par.



Nota: Se asignan al inicio del algoritmo a las variables **N1** y **N2** el valor 1 (uno) para que puedan entrar a los ciclos de repetición «mientras» y así pedir el nuevo valor.
Prueben ingresar valores impares y verán como repite el ingreso hasta que se lea un valor par.

Dentro de un ciclo de repetición pueden incluirse otros ciclos de repetición como el «mientras» o el «para» y/o condiciones de selección. Es decir, las estructuras de control se pueden combinar para resolver un problema. A continuación se presenta un ejercicio con dicha combinación.

Problema 30: Diseñar usando FreeDFD un algoritmo que muestre por pantalla 10 números impares generados al azar (del 1 al 9, solo impares). Si el número obtenido al azar es par debe continuar hasta hallar un impar.



Nota: Antes de entrar al ciclo de repetición **Para** se le asigna a la variable **N** un valor 2 obligando a entrar al ciclo «mientras» hasta encontrar un número **N** impar. Esto sucede 10 veces porque se encuentra dentro del **Para**.

2. 8. 7. Problemas: Diagramación lógica – Estructura de control de repetición «mientras»

41. Diseñar un algoritmo que genere números al azar múltiplos de 5, mientras el usuario no ingresa la opción '**S**' (Salir).
42. Diseñar un algoritmo que lea un número desde teclado y determine si el mismo es primo o no.
43. Diseñar un algoritmo que lea desde teclado la información sobre altura, edad, y sexo (F/M) de los participantes de un curso. La lectura finaliza cuando se lee un valor de altura negativo. Luego calcule:
    a. Promedio de altura de las mujeres.
    b. Promedio de altura de los varones.
    c. Promedio de edad de los participantes.
44. Se ingresan 10 números cuyos valores corresponden a los de la ruleta (0,1,2,...,36), se pide hallar y mostrar por pantalla lo siguiente:
    a. Cantidad de números impares.
    b. Promedio de los números pares (no contar los ceros).
    c. Cantidad de números que se encuentran en la 2º docena (13 al 24).
    d. El número más grande.
    e. ¿En qué cambia la solución si en lugar de leer 10 números, ahora se leen números hasta que llega el valor 36?
45. Se leen desde teclado pares de temperaturas (T1 y T2) hasta que T1 sea cero. Hallar el promedio de las temperaturas ingresadas que están comprendidas entre 5° y 15° (incluidos).
46. Se leen desde teclado números hasta que la suma de los mismos llegue a 1000. Mientras tanto debe hallar:
    a. La cantidad de números múltiplos de 6.
    b. La suma de los números que se encuentran entre el 1 y el 10 (incluidos).

Capítulo 3

Pseudocódigo

Pseudo deriva del griego *seudo*, que significa 'falso', mientras que *código* proviene del latín: *codices*, *codex*. Estos se empleaban para referirse a los documentos o libros donde los romanos tenían escritas todas y cada una de sus leyes. En definitiva pseudocódigo significa 'falso lenguaje'.

Como se explicó en el capítulo introductorio, el diseño del algoritmo es independiente del lenguaje de programación, es decir, una solución a un problema puede ser escrita en diferentes lenguajes. Así, el pseudocódigo está diseñado para facilitar la comprensión de la solución algorítmica. En pocas palabras, es una herramienta que facilita el proceso de programar.

Aunque no existe una sintaxis estándar para el pseudocódigo, las diferentes versiones utilizan un lenguaje similar al lenguaje natural y, en general, su escritura exige la indentación (sangría en el margen izquierdo) de diferentes líneas. En el caso de este libro, se usará la sintaxis provista por el aplicativo PSeInt.

3.1. PSeInt

PSeInt es un software educativo libre y multiplataforma, dirigido a aquellos que quieren comenzar a incursionar en la programación y el desarrollo de la lógica. El software fue creado en el año 2003 por el ingeniero en informática Pablo Novara en la Facultad de Ingeniería y Ciencias Hídricas (FICH) de la Universidad Nacional del Litoral (UNL) de la Argentina.

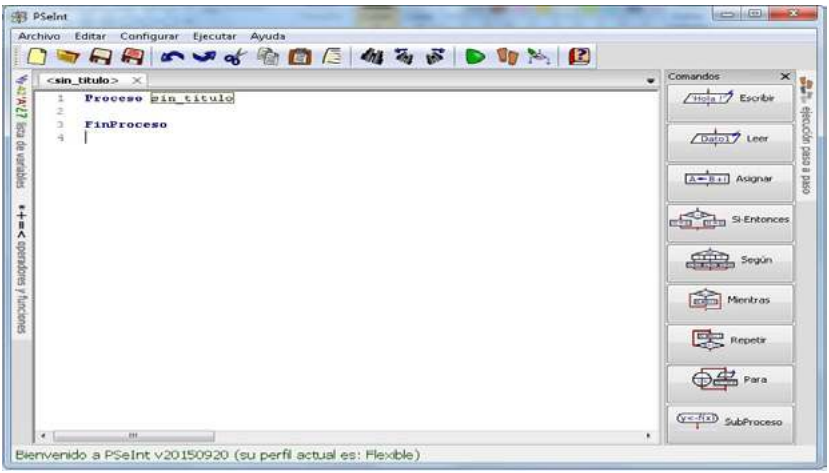
PSeInt se distribuye bajo licencia GPL (General Public License) y es uno de los softwares más utilizados en las universidades latinoamericanas para la construcción de algoritmos. Su manejo es simple e intuitivo a través de un editor de programas escritos en un pseudolenguaje en español. Su interfaz gráfica permite crear, almacenar, ejecutar y corregir fácilmente los programas.

El objetivo es permitir centrar la atención en conceptos fundamentales de aprendizaje sin detenerse en detalles para la interpretación de un compilador. De esta forma, se facilita la tarea de escritura de algoritmos y se proporciona un conjunto de ayudas y asistencias, junto a algunas herramientas que asisten a un programador novato a encontrar errores y comprender la lógica de los algoritmos. El software no deja de actualizarse y para descargarlo tenemos que dirigirnos al siguiente vínculo http://pseint.sourceforge.net/

Una vez descargado e instalado se puede ejecutar a través del icono:

La primera ventana que aparece es la que muestra la figura 2.1:

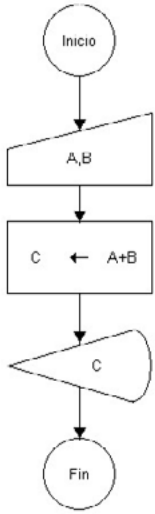
Figura 2.1. Pantalla PSeInt



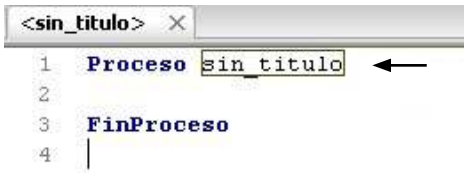
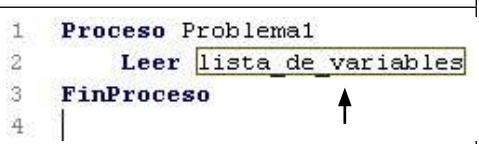
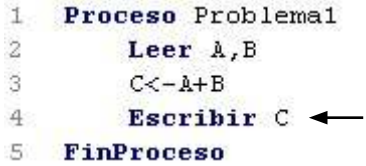
En la barra ubicada arriba de la pantalla figura el menú principal que contiene el total de las opciones; en el panel derecho, los comandos para armar el algoritmo; y, en el centro, aparece el sector de trabajo o edición.

A medida que avancemos incorporaremos los comandos necesarios para abordar cada uno de los problemas vistos en el capítulo 2 (diagrama de flujo).

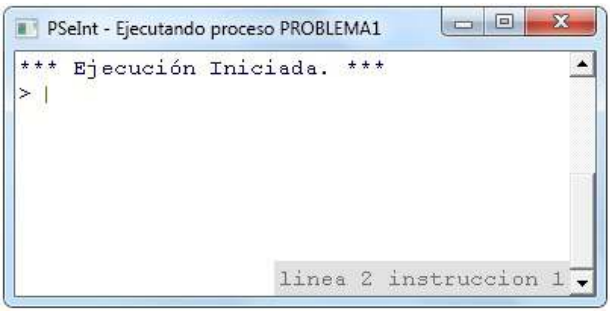
A continuación, se muestra paso a paso cómo editar y probar el primer ejercicio, en el que se ingresaban dos números enteros para hallar y mostrar la suma.



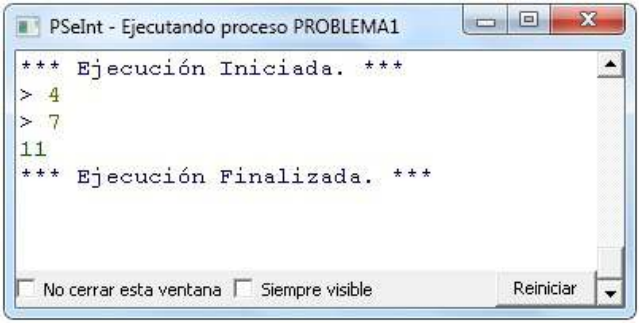
Se describen con detalle todos los pasos a seguir en PSeInt para resolver el problema 1.

1. Escribir el nombre del programa a la derecha de «Proceso», en el espacio que dice "sin_titulo" En este caso, recomendamos que escriban "Problema1" **Nota:** El título no debe contener espacios en blanco.	
2. Ubicamos el cursor en la línea 2 haciendo un click y luego elegimos el comando «Leer». 	 En la lista de variables escribimos las variables A, B.
3. Insertamos en la línea 3 el comando de asignación. 	Allí escribiremos en «variable<-expresión» lo siguiente: C<-A+B
4. Para finalizar el algoritmo, insertamos en la línea 4 el comando 	Reemplazamos en la lista de expresiones la variable C. 

Una vez terminado el programa en el sector de trabajo es necesario probarlo ejecutándolo con el ícono o directamente con la tecla F9.



Se ingresa por teclado un número Enter y luego otro número Enter. Finalmente el resultado será el siguiente:



En caso de error se muestra en un panel inferior el número del error cometido y la información detallada.

Importante:

i) Evitar dejar líneas de comando en blanco.

ii) Tener en cuenta al nombrar variables que *no* se diferencian minúsculas de mayúsculas.

iii) Las palabras reservadas del programa aparecen en *negrita*.

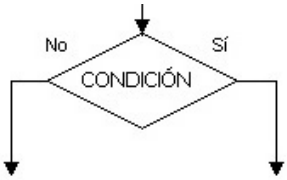
iv) Guardar los archivos.

v) Probar el funcionamiento de cada programa.

vi) Para empezar un nuevo algoritmo hacer un click en el ícono .

Nota: Realizar en PSeInt el problema 2 (del capítulo 2).

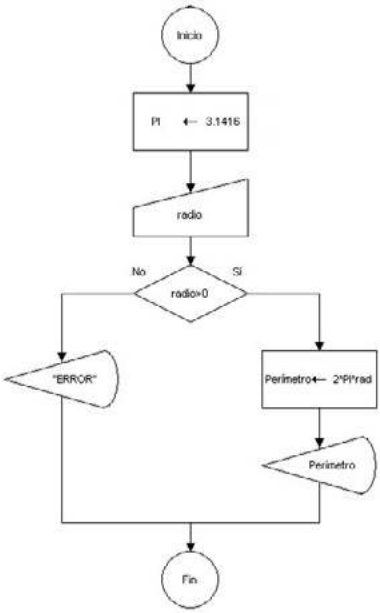
Con respecto a la estructura de control de decisión el comando del PSeInt es:




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

Atención: Es muy importante tomar en cuenta las tabulaciones que se presentan. En este caso, lo que va dentro del **Si** y **Sino** se tabula. Esta acción tiene por nombre *indentación* y PSeInt la realiza automáticamente. La indentación es lo que permite determinar un bloque de acciones que se ejecutarán juntas. Luego, cada lenguaje de programación usará sus propios símbolos para esto, por ejemplo, en el lenguaje C se usan las llaves **{}**; en Pascal, las palabras reservadas «Begin-End». Es muy importante acostumbrarse a escribir código indentado ya que facilita la lectura y la detección de errores.

Problema 3 (del capítulo 2): Mostrar el perímetro de una circunferencia, siempre y cuando el radio que se ingresa sea mayor a cero.




```
Proceso Problema3
    Leer radio
    Si radio>0 Entonces
        Perimetro<-2*PI*radio
        Escribir Perimetro
    Fin Si
FinProceso
```

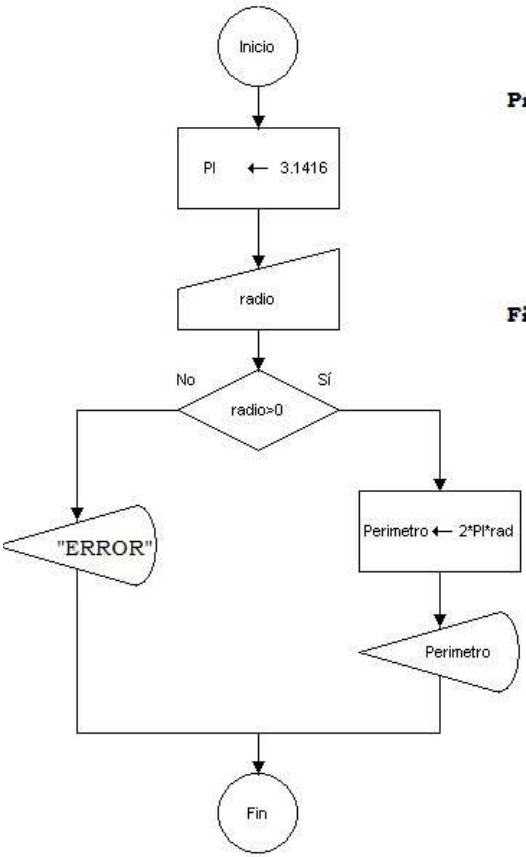
Nota: Observar que no hemos leído ni asignado ningún valor a la variable PI. En PseInt cuando se indica PI se hace referencia al valor 3,1416 viene implícito con el lenguaje.

En estos casos la estructura de control de decisión es simple, es decir que no es necesaria la instrucción **Sino**, entonces se debe eliminar simplemente seleccionando y borrando:

```
    Sino
        acciones_por_falso
```

A continuación, se muestra el problema 4, donde se presenta un caso de decisión doble.

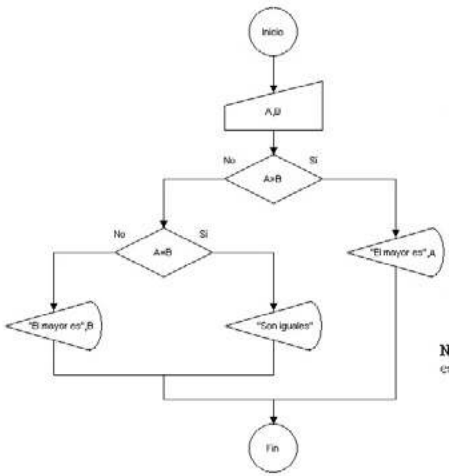
Problema 4 (del capítulo 2): Igual al ejemplo anterior pero en el caso de ingresar un radio erróneo (cero o negativo) indicarlo con el cartel "Error".




```
Proceso Problema4
   Leer radio
   Si radio>0 Entonces
      Perimetro<-2*PI*radio
      Escribir Perimetro
   Sino
      Escribir "Error"
   Fin Si
FinProceso
```

En el problema 6 se presenta un caso de decisiones anidadas.

Problema 6 (del capítulo 2): Mostrar el número más grande (entre dos) ingresado por teclado. Si los dos números son iguales mostrar el cartel "Son iguales".




```
Proceso Problema6
    Leer A,B
    Si A>B Entonces
        Escribir "El mayor es ",A
    Sino
        Si A=B Entonces
            Escribir "Son iguales"
        Sino
            Escribir "El Mayor es ",B
        FinSi
    Fin Si
FinProceso
```

Nota: PseInt nos permite directamente escribir el pseudocódigo sin la necesidad

A continuación se exhibe un caso donde las decisiones son independientes.

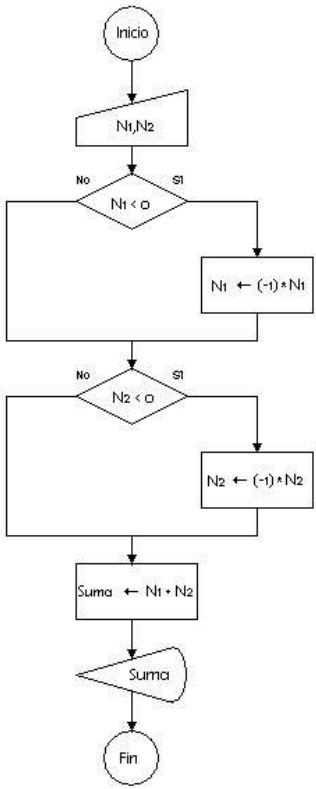
Atención: Se recomienda directamente editar el algoritmo en el sector de trabajo sin la necesidad de utilizar los comandos del panel derecho. **Importante:** A partir de ahora se incorporan comentarios en el algoritmo. Para ello, se inicia la línea con un par de barras inclinadas //. Resulta importante que en cada algoritmo, en la cabecera, escriban todas las variables en uso y el tipo al que pertenecen: alfanumérico, entero o decimal. Por ejemplo, en el caso anterior (problema 6) el programa quedaría de la siguiente manera:

```
Proceso Problema6
// A,B enteros  <——— Comentarios
    Leer A,B
    Si A>B Entonces
        Escribir "El mayor es ",A
    Sino
        Si A=B Entonces
            Escribir "Son iguales"
        Sino
            Escribir "El Mayor es ",B

        FinSi
    Fin Si
FinProceso
```

Nota: Los comentarios en PSeInt se muestran siempre en color gris.

Problema 7 (del capítulo 2): Ingresar dos números por teclado y sumarlos, con la condición de que cada uno sea positivo (de lo contrario cambiar el signo). No olvidarse de comentar las variables en uso.




```
Proceso Problema7
// N1, N2 enteros
    Leer N1,N2
    Si N1<0 Entonces
        N1<-(-1)*N1
    Fin Si
    Si N2<0 Entonces
        N2<-(-1)*N2
    Fin Si
    Suma<-N1+N2
    Escribir Suma
FinProceso
```

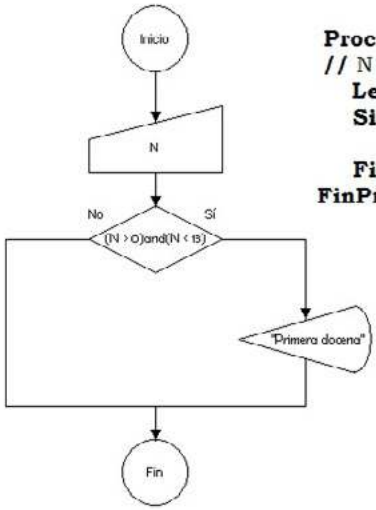
3. 2. Operadores lógicos

Los operadores lógicos en pseudocódigo se relacionan de la siguiente forma:

Diagrama de flujo	Pseudocódigo	Ejemplo
and	&	(N>10) **&** (M=='X')
or	|	(R==0) **|** (R>=100)
not	no	**no** (C<1)

A continuación, se muestran algunos ejemplos relacionados con los operadores lógicos.

Problema 8 (del capítulo 2): 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.




```
Proceso Problema8
// N entero
   Leer N
   Si (N>0) & (N<13) Entonces
         Escribir "Primera docena"
   FinSi
FinProceso
```

Realice el pseudocódigo del problema 9 (del capítulo 2).

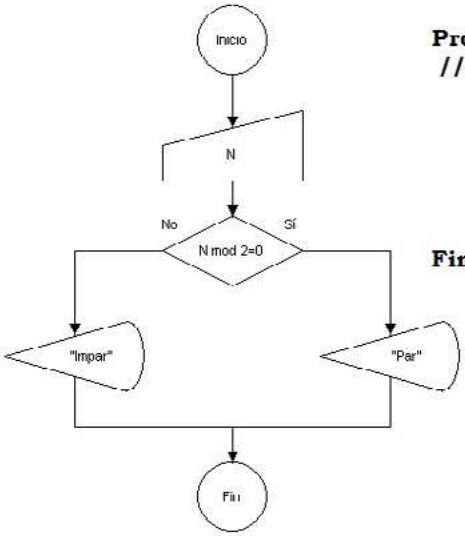
Atención: Los operadores aritméticos básicos, los de relación (>, <, =, <= y >=) y la mayoría de los operadores especiales se mantienen con respecto al diagrama de flujo. A continuación se presentan aquellos que se diferencian.

Operación	Símbolo	Sintaxis
Raíz cuadrada	raiz	raiz(N)
Seno	sen	sen(N)
Número aleatorio	azar	azar(N)

Importante: En PSeInt la desigualdad se escribe de la siguiente manera: <>

El problema 12 es un ejemplo con un operador aritmético especial.

Problema 12 (del capítulo 2): Ingresar un número natural por teclado. Se desea saber y mostrar si es par o impar.



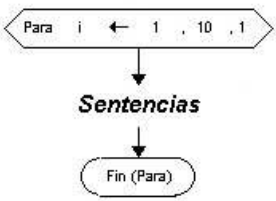

```
Proceso Problema12
 // N entero
      Leer N
      Si N mod 2 = 0 Entonces
           Escribir "Par"
      Sino
           Escribir "Impar"
      FinSi
FinProceso
```

Nota: Recuerde que el operador «mod» nos devuelve el resto de la división entre números enteros.

Realice los ejercicios 13, 14 y 15 en pseudocódigo del capítulo anterior.

Atención: Recuerde que el operador especial «random» que se utiliza en el FreeDFD, tiene su equivalente en PSeInt: «azar».

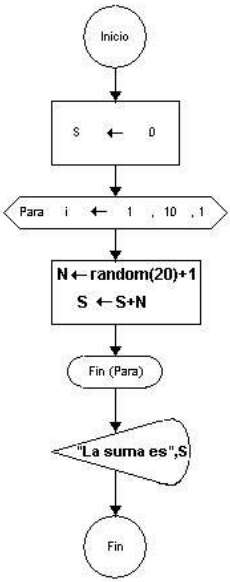
Para el caso de la estructura de control de repetición «para», el comando del PSeInt es:




```
Para i <- 1 Hasta 10 Con Paso 1
    Hacer
            sentencias
Fin Para
```

Nota: Recordar que el paso es el incremento de cada iteración o vuelta

Para el problema 17 (del capítulo 2) que consistía en hallar la sumatoria de 10 números generados al azar, los números deben comprender el siguiente rango de valores [1...20].




```
Proceso Problema17
// S,N enteros
  S<-0
  Para i<-1 Hasta 10 Con Paso
  1 Hacer
        N<-azar(20)+1
        S<-S+N
  Fin Para
  Escribir "La suma es ",S
FinProceso
```

A continuación se presenta directamente el pasaje a pseudocódigo del problema 18 (del capítulo 2). En el mismo se pedía lo siguiente: en una veterinaria se desea saber el promedio de edad de gatos y perros (por separados) que fueron asistidos durante un mes. En total se registraron 200 animales y la veterinaria solo atiende gatos y perros.

```
Proceso Problema18
  // CP,CG,SP,SG,i,Edad enteros
  // Animal alfanumérico
  // PP,PG decimales
   CP<-0
   CG<-0
   SP<-0
   SG<-0
   Para i<-1 Hasta 30 Con Paso 1 Hacer
       Leer Animal,Edad
       Si Animal="Perro" Entonces
             CP<-CP+1
             SP<-SP+Edad
       Sino
             CG<-CG+1
             SG<-SG+Edad
       Fin Si
   Fin Para
   PP<-SP/CP
   PG<-SG/CG
   Escribir "El promedio de la edad de perros es ",PP
   Escribir "El promedio de la edad de gatos es ",PG
FinProceso
```

Nota: En el problema 18 se utilizan variables de diferentes tipos; entero, decimales y alfanuméricos. Con la intención de mejorar la lectura del código se han agrupados los tipos por línea.

Recordar que el problema 19 (del capítulo 2) pide lo mismo que el problema anterior, pero la diferencia radica en que no solo la veterinaria atiende gatos y perros, puede que sean otros animales también. Justamente lo que se pide es además contar la cantidad de esos animales que no son ni gatos y ni perros.

```
Proceso Problema19
  //CA,CP,CG,SP,SG,i,Edad enteros
  // Animal alfanumérico
  // PP,PG decimales
  CP<-0
  CG<-0
  SP<-0
  SG<-0
  CA<-0
  Para i<-1 Hasta 30 Con Paso 1 Hacer
        Leer Animal,Edad
        Si Animal="Perro" Entonces
            CP<-CP+1
            SP<-SP+Edad
        Sino
            Si Animal="Gato" Entonces
                CG<-CG+1
                SG<-SG+Edad
            Sino
                CA<-CA+1
        Fin Si
    Fin Para
    PP<-SP/CP
    PG<-SG/CG
    Escribir "El promedio de la edad de perros es ",PP
    Escribir "El promedio de la edad  de gatos es ",PG
    Escribir "La cantidad de otros animales es ",CA
FinProceso
```

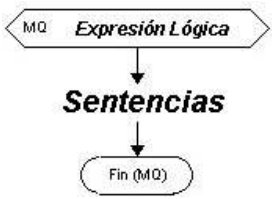
Nota: Realice los problemas 20, 21 y 22 directamente utilizando el programa PSeInt sin depender del diagrama de flujo.

El siguiente es el problema 23 (del capítulo 2) de máximos y mínimos en pseudocódigo.

```
Proceso Problema23
// BMAX,E,i, enteros
  BMAX <-  -1000
  Para i<-1 Hasta 20 Con Paso 1 Hacer
      Leer E
      Si E > BMAX  Entonces
            BMAX <-E
      Fin Si
  Fin Para
  Escribir "La edad máxima es ",BMAX
FinProceso
```

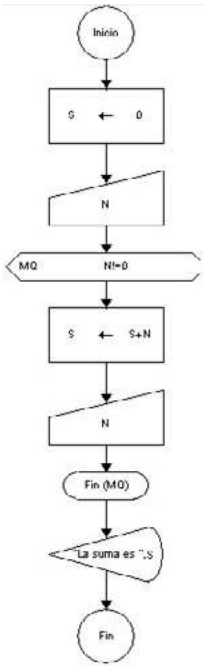
Nota: Realizar los ejercicios 24 y 25.

Para el caso de la estructura de control de repetición «mientras» el comando del PSeInt es:



Mientras expresión_lógica **Hacer**
sentencias
Fin Mientras

Tomar el problema 26 (del capítulo 2) en diagrama y pseudocódigo para comparar. En el mismo se pedía calcular la suma de números ingresados por teclado hasta que se ingrese un cero.




```
Proceso Problema26
   // S,N enteros
   S<-0
   Leer N
   Mientras N<>0 Hacer
            S<-S+N
            Leer N
    Fin Mientras
    Escribir "La suma es ",S
 FinProceso
```

Importante: Recordar que la desigualdad en el programa PSeInt se indica con el símbolo **<>**

Nota: Realizar los problemas 27, 28, 29 y 30 del capítulo anterior utilizando directamente el software PSeInt (sin la necesidad de recurrir al diagrama de flujo).

3. 3. Ejercicios: pseudocódigo

46. Se ingresan por teclado 10 pares de temperaturas (T1 y T2) para hallar el promedio de las temperaturas que están entre 5° y 15° (incluidos).
47. Se ingresan 10 números por teclado para hallar tres datos:
    a. La cantidad de números negativos.
    b. La suma de los números que se encuentran entre el 1 y el 10 (no incluidos).
    c. El promedio de todos los números.
48. Se generan números enteros en forma aleatoria entre [0 y 200] hasta que la sumatoria de los mismos sea mayor a 500. Al finalizar indicar:
    a. La cantidad de números nulos (0) leídos.
    b. La sumatoria de los números que se encuentran entre el 10 y el 100 (incluidos).
    c. El promedio de los números menores a 150.
    d. El número mayor generado.
49. La sucesión de Fibonacci genera la siguiente secuencia de números: 1, 1, 2, 3, 5, 8, etcétera. Es decir que se obtiene de sumar los dos números anteriores.
    El siguiente es el término general: $a_n = a_{n-1} + a_{n-2}$
    Se pide que diseñe un programa que le solicite al usuario un número entero positivo K y luego muestre en pantalla el K-ésimo término de la sucesión de Fibonacci. Por ejemplo, si el usuario ingresa por teclado el número 10 entonces el algoritmo debe mostrar el valor correspondiente a a_{10}
50. Realice los ejercicios 25, 37, 38, 43 y 46 del capítulo anterior en pseudocódigo.

Capítulo 4

Entorno de programación visual

Existen otras formas de aprender a programar. Una de ellas consiste en utilizar lenguajes visuales a través de entornos de desarrollo integrado (IDE) que, a diferencia de las herramientas anteriores, posibilitan la ejecución y visualización de un algoritmo al mismo tiempo.

DaVinci Concurrente, la herramienta a la que vamos a dedicar este último capítulo, es un entorno integrado de desarrollo (IDE) que facilita la comprensión de situaciones problemáticas y permite la visualización de la ejecución de los algoritmos que se resuelven. Fue desarrollado originalmente por un grupo de investigadores del Instituto de Investigación en Informática LIDI (III-LIDI) de la Facultad de Informática de la UNLP, encabezado por el licenciado Raúl Champredonde.

Luego, a través de un trabajo de tesis de grado (Aguil Mallea, 2012) de la Universidad Nacional de la Patagonia San Juan Bosco (Ushuaia), se desarrolló una segunda versión que permite la introducción a los conceptos básicos de la programación concurrente. Y esta última versión es la que vamos a utilizar aquí.

Este lenguaje está directamente ligado a la enseñanza de la programación estructurada, tanto para la etapa inicial de la programación secuencial como para los conceptos básicos de la programación concurrente.

La versión actual del aplicativo está desarrollada en lenguaje de programación Java, por lo tanto, uno de los requisitos es tener instalado el *plug-in* de Java en la computadora donde se va a ejecutar. El mismo se puede descargar en el siguiente sitio: www.java.com/getjava y el enlace para descargar el DaVinci es http://davinci-c.sourceforge.net/

Da Vinci se basa en la programación de un robot (Lubo-I) situado en una ciudad cuadrada con avenidas y calles. Este puede realizar diferentes acciones, desde movimientos dentro de los límites de la ciudad hasta interacciones con los objetos (flores, papeles y obstáculos) que se encuentran en ella.

DaVinci permite que los usuarios puedan programar los movimientos de un robot abstracto en una ciudad también abstracta. La ciudad es un cuadrado que contiene 10 calles horizontales y 10 avenidas verticales, como se puede observar en la figura 4.1. El robot camina de una esquina a la otra ejecutando las instrucciones especificadas en el programa. Durante la ejecución, el robot puede recoger o depositar dos tipos de objetos: flores y papeles. Para esto dispone de dos bolsas, una para cada tipo de objeto. Cuenta además con la posibilidad de incorporar obstáculos en las esquinas de manera de bloquear un camino, contar objetos y mostrar resultados. El lenguaje define las instrucciones

primitivas que el robot ejecuta y las construcciones necesarias que permiten una programación modular y estructurada. En Da Vinci Concurrente, un programa puede ser desarrollado en modo texto o visualmente. Cualquiera sea el modo seleccionado, todo lo que se hace en una representación, simultánea y automáticamente se reproduce en el otro modo. La verificación sintáctica y la ejecución de los programas desarrollados se llevan a cabo sobre el código en modo texto, de manera de independizar la eficiencia del lenguaje del formato visual del algoritmo (Champredonde y otros, 2000).

En el presente capítulo, se trabajan conceptos básicos de lógica proposicional vistos en capítulos anteriores, para representar condiciones complejas utilizadas en las estructuras del ambiente, aplicadas específicamente en este caso a problemas con el robot.

Ahora bien, para darle instrucciones al robot Lubo-I, es necesario conocer a qué primitivas (instrucciones) responde.

La herramienta permite configurar la ciudad de manera manual o aleatoria, para que aparezcan en las esquinas objetos como flores, papeles u obstáculos.

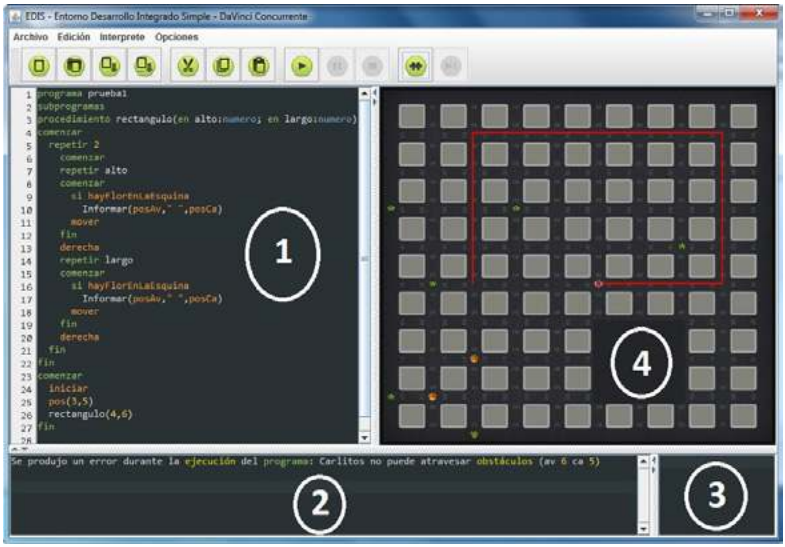
Además, el robot puede acceder y modificar los valores de variables que el sistema ofrece y dejar o no un rastro a medida que transita por la ciudad.

Al igual que PSeInt, el aplicativo DaVinci Concurrente nos brinda un lenguaje sencillo para la incorporación y el manejo de conceptos básicos de la programación estructurada y modular, empleando palabras clave, primitivas, sentencias simples o compuestas, estructuras de control, expresiones y constructores de subprogramas.

Podremos utilizar cero, uno o más robots en cada programa.

Primero empezaremos por el reconocimiento del entorno de trabajo del DaVinci, integrado básicamente por cuatro paneles: 1) de edición de código fuente, 2) el de resultados, donde se pueden visualizar los mensajes, 3) el de estado de ejecución, 4) el de la ciudad donde transitan el/los robot/s y los objetos que la componen.

Figura 4.1. Entorno gráfico del programa DaVinci



La franja superior es la barra de íconos que permiten crear, abrir y guardar programas fuentes; dentro del programa se puede cortar, copiar y pegar texto; ejecutar, pausar, continuar y frenar el funcionamiento del algoritmo, así como también depurar.

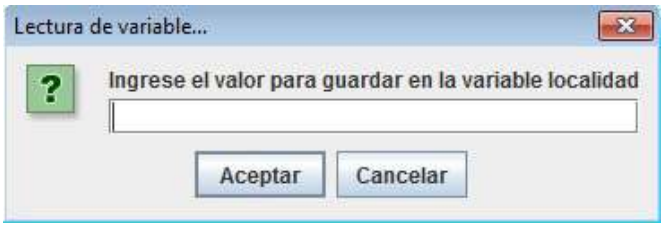
Empecemos con nuestro primer ejemplo, sin utilizar el robot por ahora. Al mismo tiempo indicaremos las diferencias con PSeInt en la escritura del algoritmo.

Recomendación: Ir probando los ejercicios propuestos a medida que avanzamos.

Ejemplo 1: Pedimos que el usuario ingrese el nombre de su localidad para mostrar por pantalla "Bienvenido a la ciudad _ _ _ _ _", donde las líneas representarían la ciudad o pueblo que se ha introducido previamente. Escribimos el algoritmo en el sector 1 (editor de código fuente).

```
programa ejemplo1
//Este es nuestro primer programa de ejemplo
variables
      localidad : texto
comenzar
      pedir(localidad)
      informar("Yo vivo en la localidad de ",localidad)
fin
```

Para ejecutar el programa se debe presionar el ícono de ejecución (barra de botones octavo de izquierda a derecha), nos aparecerá la siguiente ventana de carga:



Esto sucede gracias a la instrucción «pedir» del algoritmo.
Luego de ingresar la localidad (supongamos que ingresamos Comallo) y aceptar el paso, nos aparece en el sector 2 (panel inferior izquierdo) lo siguiente: *Yo vivo en la localidad de Comallo.*

Recomendaciones: Es importante mantener la indentación (tabulación), pues organiza y mejora la lectura del programa, aunque funcione sin ella. Notarán cambios de colores en la fuente para diferenciar las palabras reservadas del lenguaje del resto.
El nombre del programa, en este caso "ejemplo1" no debe tener espacios en blanco ni símbolos, salvo el guion de abajo "ejemplo_1". Se aconseja ir guardando el archivo con un nombre adecuado (identificatorio) siendo la extensión *punto j* (es decir *.j*, tercer ícono de izquierda a derecha).

La sintaxis del lenguaje es similar a la de PSeInt, a continuación se presenta una tabla comparativa:

DaVinci	PSeInt
Programa	Proceso
Pedir	Leer
Informar	Mostrar por pantalla
Fin	FinProceso

Además contiene las instrucciones «comenzar» y «variable» y utiliza los paréntesis () para los ingresos por teclado y salida de mensajes a diferencia de PSeInt. Para comentar una línea de código se utilizan los símbolos //. Las variables son indiferentes a minúsculas o mayúsculas.

DaVinci nos obliga a definir todas las variables que vamos a usar en nuestro programa previo a su uso. Veamos un ejemplo.

Ejemplo 2: Pedimos que el usuario ingrese los lados de un rectángulo para mostrar el perímetro y área del mismo en un único cartel de salida "El perímetro es __ y el área es __"

```
programa ejemplo2
//Hallar el perímetro y área de un rectángulo
variables
      L1: numero
      L2: numero
      P: numero
      A: numero
comenzar
      Pedir(L1)
      Pedir(L2)
      P:=2*L1+2*L2
      A:=L1*L2
      informar("El perímetro es ",P," y el área es ",A)
fin
```

Las asignaciones difieren en el símbolo ":=" (en PSeInt es) **A ← L1*L2** (en DaVinci es) **A := L1*L2**
Al igual que en el PSeInt se pueden alternar en la salida texto y variables con la instrucción «informar».

4. 1. Variables del programa DaVinci

Se pueden utilizar las variables de tipo número, texto y lógico.

Tipo	Contenido	Operaciones	
		Aritmética	Comparación
Número	Números enteros	+ - * / %	<= >= ><<> =
Texto	Combinación entre números y letras	+ (para concatenar)	<= >= ><<> =
Lógico	v/f (verdadero/falso)	& | !	= <>

Atención: Cuando operamos con enteros y obtenemos números decimales (no enteros) su salida o respuesta será truncada (solo la parte entera), por ejemplo 30,8 será 30.

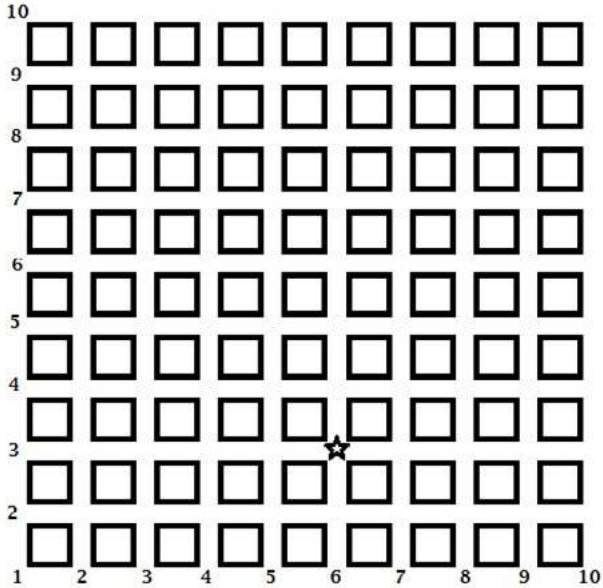
A continuación mostraremos ejemplos con la intervención del robot (Lubo-I).

4. 2. Conociendo la ciudad del robot

La ciudad de 81 cuadras está conformada por 10 avenidas y 10 calles, orientadas de modo vertical y horizontal, respectivamente.

La avenida 1 y calle 1 se ubica en la parte inferior izquierda como se muestra a continuación.

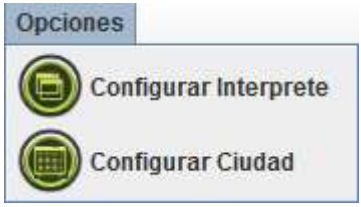
Figura 4.2. Ciudad DaVinci



Para una mejor orientación, la estrella está ubicada en la avenida 6 y la calle 3.

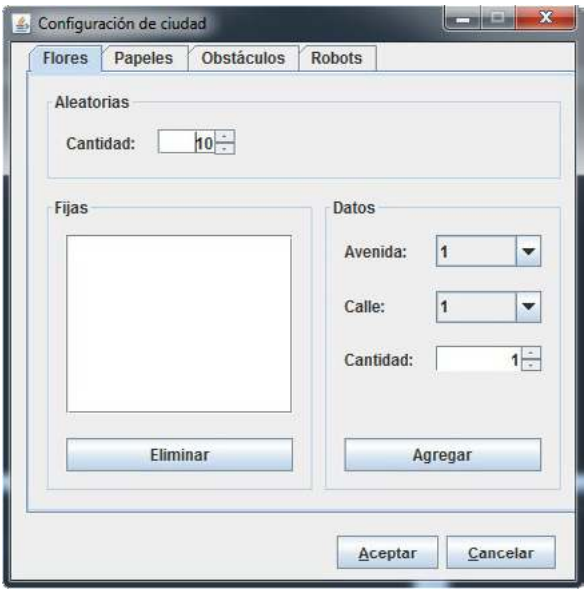
El robot siempre se mueve de esquina en esquina y podemos elegir desde donde parte.

Para configurar la ciudad y sus objetos vamos al menú superior:



Al acceder a «Configurar Ciudad» emerge la siguiente ventana:

Figura 4.3. Configuración de ciudad



En los espacios se agrega la cantidad y ubicación de flores, papeles y obstáculos que quisiéramos de entrada. Por ejemplo, si la cantidad de flores es 10, entonces debemos elegir la ubicación (avenida y calle) por cada una. Lo mismo sucede con los papeles y los obstáculos.

Atención: En una esquina se puede depositar más de un objeto (del mismo tipo o no) por ejemplo: en la avenida 4 calle 7, tres flores y dos papeles.

Ejemplo 3: Recorrer una cuadra entera desde la avenida 5 calle 7 en sentido horario llegando hasta el punto de partida.

```
programa ejemplo3
// recorre una cuadra sentido horario
comenzar
    iniciar
    pos(5,7)
    mover
    derecha
    mover
    derecha
    mover
    derecha
    mover
fin
```

En este programa no se utilizan variables. El comando «pos(5,7)» indica la posición inicial del robot, es decir, de modo genérico «pos(avenida,calle)».

Atención: Si al ejecutar el programa no se alcanza a ver el movimiento del robot, entonces podemos cambiar la velocidad con la opción de la barra del menú superior «Configurar intérprete» dándole un mayor retardo. Por defecto el robot siempre se dirige verticalmente hacia arriba. Para activar el robot debemos escribir en el programa «iniciar» luego de «comenzar».

Ejemplo 4: Hacer lo mismo que el ejercicio anterior pero en sentido contrario.

```
programa ejemplo4
// recorre una cuadra sentido antihorario
comenzar
    iniciar
    pos(5,7)
    repetir 3
      derecha
    mover
    repetir 3
      derecha
    mover
    repetir 3
      derecha
    mover
    repetir 3
      derecha
    mover
fin
```

Dado que el robot solo puede girar a la derecha es necesario moverlo 3 veces a derecha para orientar el robot a la izquierda y para eso utilizamos repetir 3.

Sin estructura de repetición	Con estructura de repetición
derecha derecha derecha	repetir 3 derecha

Atención: la instrucción **derecha**, al estar dentro de la estructura de repetición «repetir», se ubica indentada (tabulada).

Ejemplo 5: Recorrer en forma de escalera desde avenida 2 y calle 4, comenzando horizontalmente hasta llegar a la avenida 7 y calle 8 e informar la posición final.

```
programa ejemplo5
// recorre en escalera e informa
comenzar
      iniciar
      pos(2,4)
      derecha
      mover
      repetir 3
        derecha
      mover
      derecha
      mover
      repetir 3
        derecha
      mover
      derecha
      mover
      repetir 3
        derecha
      mover
      derecha
      mover
      repetir 3
        derecha
      mover
      derecha
      mover
      informar("Avenida ",posAV," calle ",posCA)
fin
```

Los comandos **posAV** y **posCA** nos brindan la posición del robot, en este caso, al finalizar el recorrido.

4. 2. Repetición

Volvamos al ejemplo 4, es posible observar que hay un conjunto de pasos que se repiten:

```
repetir 3
      derecha
mover
```

Entonces es posible reescribir el algoritmo como sigue:

```
repetir 4
      comenzar
            repetir 3
                  derecha
            mover
      fin
```

De esta forma, no solo reducimos la cantidad de comandos, sino que es más fácil determinar qué hace el programa.

Para el ejemplo 5 la solución será la siguiente:

```
repetir 4
    comenzar
        repetir 3
            derecha
        mover
        derecha
        mover
    fin
```

Ejemplo 6: Recorrer el borde de la ciudad desde avenida 1 y calle 10, mientras tanto debe contar la cantidad de esquinas en las que hay flores. Al finalizar debe informar esa cantidad.

Atención: Para realizar este ejercicio primero tendremos que configurar la ciudad poniendo flores en las esquinas. Podemos hacerlo ubicando cada una en esquinas elegidas por el usuario o bien de modo aleatorio. Supongamos que se colocan 10 flores de modo aleatorio, entonces tendremos que ir a «Configurar Ciudad» y en la solapa «Flores» elegir «10» en la sección «Aleatorias». Estas aparecerán recién cuando ejecutemos el algoritmo.

```
programa ejemplo6
// recorre el borde de la ciudad y cuenta la cantidad de esquinas con flores
variables
    c:numero
comenzar
    iniciar
    pos(1,10)
    derecha
    repetir 4
    comenzar
        repetir 9
        comenzar
            si (hayflorenlaesquina)
                c:=c+1
            mover
        fin
        derecha
    fin
    informar("La cantidad de flores halladas es ",c)
fin
```

La variable **c** de tipo número por defecto se inicializa con el valor cero para luego ser usado como contador incrementando en uno **c:=c+1** (cada vez que encuentra una flor).

La primitiva **hayflorenlaesquina** devuelve **verdadero** si en la esquina hay alguna flor o **falso** en caso contrario.

Atención: La distribución de flores aparecerá cuando ejecutemos el algoritmo y además hay que tomar en cuenta que en una esquina puede haber más de una flor. En este caso, no contemplamos la cantidad que debe haber en cada esquina, solo si existen o no flores.

Nota: Las instrucciones **comenzar** y **fin** no solo sirven para abrir y cerrar el proceso del algoritmo, se utilizan también para delimitar las sentencias dentro de las siguientes estructuras de control: «repetir», «mientras» y «si», siempre y cuando haya más de una sentencia dentro de cada una.

Ejemplo:

```
repetir 4
comenzar
    repetir 9
    comenzar
        si(hayflorenlaesquina)
            c:=c+1
        mover
    fin
    derecha
fin
informar("La cantidad de flores halladas es ",c)
```

Ejemplo 7: Igual al anterior pero tomando en cuenta la cantidad real de flores. Puede pasar que en una esquina exista más de una flor.

```
programa ejemplo7
// recorre el borde la ciudad y cuenta las flores halladas reales
variables
    c:numero
comenzar
    iniciar
    pos(1,10)
    derecha
    repetir 4
    comenzar
        repetir 9
        comenzar
            mientras(hayflorenlaesquina)
            comenzar
                c:=c+1
                tomarflor
            fin
            mover
        fin
        derecha
    fin
    informar("La cantidad de flores halladas es ",c)
fin
```

A diferencia del ejemplo anterior, reemplazamos el **si** por el control de repetición **mientras**, que se ejecuta hasta que no haya más flores en la esquina y (para no caer en un bucle infinito) debemos ir juntando flores. Esta primitiva se llama **tomarflor** y permite capturar de a una las flores para depositarlas en una bolsa.

Atención: Antes de utilizar la primitiva **tomarflor** es necesario asegurarse que exista al menos una flor. Para averiguar la existencia de flores se utiliza la primitiva **hayflorenlaesquina**.

Ejemplo 8: Recorrer el borde de la ciudad desde avenida 1 y calle 10, mientras tanto debe contar la cantidad real de flores y papeles que encuentra en el camino. Al finalizar debe informar la cantidad hallada.

Atención: Antes de continuar, debemos agregar de modo aleatorio los papeles en las esquinas de la ciudad, entrando por el menú «Opciones» – «Configurar Ciudad» y luego en la solapa «Papeles» agregamos la cantidad en la sección «Aleatorias».

```
programa ejemplo8
// recorre el borde la ciudad y cuenta las flores y papeles halladas en las esquinas
variables
    cf:numero
    cp:numero
comenzar
    iniciar
    pos(1,10)
    derecha
    repetir 4
    comenzar
        repetir 9
        comenzar
            mientras(hayflorenlaesquina)
            comenzar
                cf:=cf+1
                tomarflor
            fin
            mientras(haypapelenlaesquina)
            comenzar
                cp:=cp+1
                tomarpapel
            fin

            mover
        fin
        derecha
    fin
    informar("La cantidad de flores halladas es ",cf)
    informar("La cantidad de papeles hallados es ",cp)

fin
```

Se utilizan en este caso dos contadores: **cf** y **cp** (contador de flores y de papeles, respectivamente).

Importante: Cuando un bloque de acciones tiene más de una acción es necesario delimitar el bloque usando las palabras reservadas **comenzar** y **fin**, no siendo necesario cuando se trata de una sola instrucción. Recuerden que en DaVinci no es necesario inicializar la variable contadora/sumadora, alcanza con definirla en el sector «variables» y automáticamente toma el valor 0. Sin embargo, como buena práctica de programación, siempre que se declara una variable es importante darle un valor inicial.

Ejemplo 8b: ¿Qué sucede si deseamos que el robot cuente la cantidad de flores halladas pero sin alterar el estado de la ciudad? Es decir, dejando flores y papeles en el mismo sitio.

El robot también puede moverse *teletransportándose*, es decir, sin la necesidad de caminar a la esquina deseada, simplemente usando el comando **pos**.

A continuación, presentamos un ejemplo relacionado.

Ejemplo 9: Recorrer las avenidas de números impares (1, 3, 5, 7 y 9) de arriba hacia abajo (desde la calle 10 a la 1).

```
programa ejemplo9
// recorre solo las avenidas impares de la ciudad
variables
      avenida:numero
      calle:numero
comenzar
      iniciar
      avenida:=1
      calle:=10
      derecha
      derecha
      repetir 5
        comenzar
            pos(avenida,calle)
            repetir 9
              mover
            avenida:=avenida+2
        fin
fin
```

Atención: Primero, para dirigir el robot hacia abajo debemos girar dos veces 90° con **derecha derecha**. La estructura de repetición se efectúa 5 veces para recorrer las avenidas impares 1,3,5,7 y 9 que se incrementan de dos en dos con la asignación **avenida:=avenida+2**.

Ejemplo 10: Recorrer la diagonal principal de la ciudad (desde 1,1 a 10,10).

```
programa ejemplo10
// recorre la diagonal principal de la ciudad
variables
      avenida:numero
      calle:numero
comenzar
      iniciar
      avenida:=1
      calle:=1
      repetir 10
        comenzar
            pos(avenida,calle)
            avenida:=avenida+1
            calle:=calle+1
        fin
fin
```

Nótese que a diferencia de los demás ejercicios donde interviene el movimiento del robot, este no pinta el trayecto ya que va saltando de esquina en esquina.

Atención: Al ser la diagonal principal, se trata de recorrerla manteniendo el mismo valor de avenida y calle pos(1,1) pos(2,2) ...pos(10,10). Por lo tanto, podemos reducir las líneas de comandos de nuestro algoritmo usando solo una variable numérica, por ejemplo x, entonces pos(x,x).

Ejemplo 11: Recorrer la calle 4 en avenidas pares (2,4,6,8 y 10) e informar la cantidad de esquinas donde no hay flores.

```
programa ejemplo11
//Recorre las avenidas pares de la calle 4 e informa la no existencia de flores
variables
      c:numero
      avenida:numero
      calle:numero
comenzar
      iniciar
      avenida:=2
      calle:=4
      repetir 5
        comenzar
          pos(avenida,calle)
          si(!hayflorenlaesquina)
                c:=c+1
          avenida:=avenida+2
        fin
      informar("La cantidad de esquinas sin flores es ",c)
fin
```

Atención: Si el robot excede el límite de la ciudad, se le informa dicho error en el sector 2 de resultados (panel inferior izquierdo).

En este ejemplo, el valor de la variable calle se mantiene en 4 mientras que la variable avenida va incrementándose de a dos. El símbolo **!** representa la negación o complemento de la respuesta de **«hayflorenlaesquina»**.

DaVinci simboliza los operadores lógicos de forma similar al lenguaje de programación C.

Símbolo	Significado	Ejemplo
&	Disyunción o División	(p>4 & p<10) el valor de p se encuentra entre 4 y 10 (no incluidos)
|	Conjunción o Unión	(p>8 | p<=1) el valor de p es mayor a 8 o es menor o igual que 1
!	Negación o Complemento	!haypapelenlaesquina → No hay papel

A continuación, mostramos un ejemplo donde hay disyunción y negación al mismo tiempo.

Ejemplo 12: Recorrer todas las esquinas de las avenidas 4 y 8 e informar la cantidad de esquinas donde hay flores y no papeles.

```
programa ejemplo12
variables
      c:numero
      avenida:numero
      calle:numero
comenzar
      iniciar
      avenida:=4
      repetir 2
         comenzar
            calle:=1
            pos(avenida,calle)
            repetir 9
               comenzar
                  si(hayflorenlaesquina & !haypapelenlaesquina)
                        c:=c+1
                  mover
               fin
            avenida:=8
      fin
      informar("La cantidad de esquinas con esa condición es ",c)
fin
```

En este caso, el robot recorre las dos avenidas dejando la marca porque estamos usando la instrucción **mover**.

El ultimo objeto que nos falta ver es el *obstáculo*. A diferencia de las flores y papeles estos se deben ubicar en la ciudad de modo manual, para eso debemos ir a «Opciones» – «Configurar Ciudad» y hacer click en la solapa «Obstáculos».

Figura 4.4. Configurar obstáculos

En la figura 4. 4 hay un obstáculo en la avenida 3 y calle 5 generado a mano.

Nota: Podemos agregar una por esquina y el límite está determinado por la cantidad total de esquinas de la ciudad.

Ejemplo 13: Recorrer la ciudad (avenida por avenida) desde la posición (1,1), hasta encontrar un obstáculo, mientras tanto ir contando las esquinas que tienen flores y mostrar el resultado.

```
programa ejemplo13
//Recorre la ciudad hasta encontrar un obstáculo. Cuenta flores
variables
      flores:numero
      avenida:numero
      calle:numero
comenzar
 iniciar
 avenida:=1
 calle:=1
 mientras ((avenida<11)&(!hayobstaculo))
      comenzar
            pos(avenida,calle)
            si(hayflorenlaesquina)
                  flores:=flores+1
            calle:=calle+1
            si(calle=11)
                  comenzar
                        avenida:=avenida+1
                        calle:=1
                  fin
      fin
 informar("La cantidad de esquinas con flores es ",flores)
fin
```

Nota: Por cada paso que da el robot, se evalúa la estructura de control **mientras** y se decide si avanza o no el ciclo dependiendo de dos situaciones:

a) Que no se pase del límite de la ciudad **avenida<11**.

b) Que no aparezca en su camino un obstáculo **!hayobstaculo**.

Atención: Para probar la solución, previo a la ejecución, agregar a la ciudad por lo menos un obstáculo.

Ejemplo 14: Recorrer la ciudad de modo aleatorio hasta encontrar 6 esquinas con papeles o hasta toparse con un obstáculo. En cada paso, debe contabilizar y recolectar los papeles que vaya encontrando, utilizar la instrucción **tomarpapel**.

```
programa ejemplo14
//Recorre la ciudad hasta toparse con un obstáculo o hasta encontrar 6 papeles
variables
      papeles:numero
      avenida:numero
      calle:numero
comenzar
 iniciar
 avenida:=aleatorio(10)+1
 calle:=aleatorio(10)+1
 papeles := 0
 mientras ((papeles<6)&(!hayobstaculo))
      comenzar
            avenida:=aleatorio(10)+1
            calle:=aleatorio(10)+1
            pos(avenida,calle)
            si(haypapelenlaesquina)
                  comenzar
                        papeles:=papeles+1
                        tomarpapel
                  fin
      fin
 informar("La cantidad de esquinas con papeles es ",papeles)
fin
```

Nota: La función **aleatorio(N)** devuelve un número entero entre 0 al N-1, es decir que con **aleatorio(10)** retorna un valor entero comprendido entre 0 y 9. Si le sumamos 1 se convierte al rango entre 1 y 10, coincidiendo con los posibles números de avenidas y calles de la ciudad. **Atención:** El algoritmo finaliza cuando llega a 6 la cantidad de papeles o cuando el robot se encuentra con un obstáculo. ¿Qué sucedería si se quita la instrucción **tomarpapel**? ¿La salida del algoritmo sería la misma?

4. 3. Modularización

A lo largo de la historia de la ingeniería de sistemas de información se han desarrollado diferentes técnicas para construir programas. El primer gran avance se conoce como programación modular y complementa la programación estructurada. La programación modular utiliza abstracción de procedimientos (Di Mare, 1991). Ahora bien ¿qué significa *abstraer* y por qué este concepto resulta tan importante en la ciencia informática?

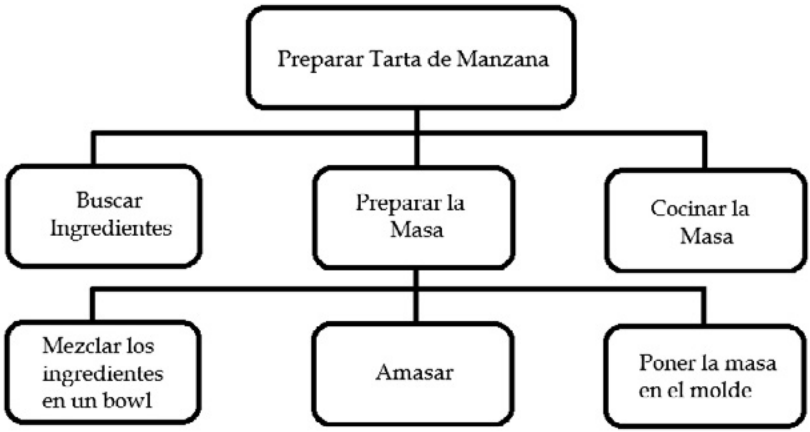
La abstracción es un proceso mental que consiste en identificar los detalles importantes cuando nos enfrentamos a un problema, mientras se ignoran los detalles irrelevantes. Este proceso permite simplificar el problema ya que la cantidad de información a manejar en un determinado momento disminuye.

La programación modular consiste en definir módulos, una especie de *cajas negras* que tienen una forma de comunicarse (interfaz) claramente definida. Usando abstracción de procedimientos el programador puede separar el *qué* hace el módulo del *cómo* lo hace.

Así, es posible descomponer funcionalmente un programa en subprogramas. El propósito es facilitar la resolución de problemas, dividiendo el problema en subproblemas mas simples, basándose en la idea «divide y vencerás». Una metodología de resolución de problemas con estas características es el diseño *top -down*.

Ejemplo: Se desea preparar una tarta de manzanas.

Figura 4.5. Ejemplo de modularización



En el ejemplo de la figura 4.5, cada caja (módulo) cumple una función específica (qué hace) y, en algunos casos, comparte información con otras (interfaz).

En general, en las soluciones modularizadas, un programa es también un módulo, llamado programa principal (en el ejemplo: preparar tarta de manzana). Este módulo se encarga de controlar todo lo que sucede y es el responsable de transferir el control a los submódulos de modo que ellos puedan llevar adelante su función y resolver el problema. Cada uno de los submódulos, cuando finaliza su trabajo, devuelve el control al módulo que lo invocó.

Los invitamos a navegar una animación desarrollada por docentes de la facultad de informática de la UNLP sobre el concepto de modularización, siguiendo este vínculo: http://weblidi.info.unlp.edu.ar/catedras/ingreso/material2013/IAI/Adicional/DemoModularizacion/MODULOS.html.

Hasta el momento, vimos ejemplos con programas a partir de la utilización de una secuencia de instrucciones de modo lineal:

```
programa ejemplo
variables
...
comenzar
...
fin
```

Muy útiles cuando trabajamos con problemas sencillos, pero cuando se nos presentan problemas más complejos es conveniente subdividirlos de acuerdo a las diferentes funcionalidades que encontramos en el problema. De esta forma, la resolución del problema general se obtendrá resolviendo cada uno de los subproblemas. Si logramos que la solución (algoritmo) a cada uno de estos subproblemas maneje su propio conjunto de datos se tendrán soluciones (algoritmos) independientes que podrán codificarse por separado y reutilizarse en otros problemas

La estructura de un programa en DaVinci ahora sería la siguiente:

```
programa ejemplo
subprogramas
procedimiento tarea1
   comenzar
   ...
   fin
procedimiento tarea2
   comenzar
   ...
   fin
variables
...
comenzar
... (aquí se hacen las llamadas (invocaciones) a los procedimientos tarea1 y tarea2)
fin
```

No es nuestra intención hacer un capítulo destinado en profundidad a modularización, pero si consideramos importante clarificar algunos conceptos. Existen dos tipos de módulos: procedimientos y funciones. En líneas generales una función es un módulo que recibe uno o más datos (parámetros) y produce un valor o resultado que es único y de alguno de los tipos permitidos en el lenguaje. Por ejemplo, **logaritmo(unNro, unaBase)** es una función que recibe un número entero y una base entera y retorna un valor numérico que representa el logaritmo en base **unaBase** del número **unNro**.

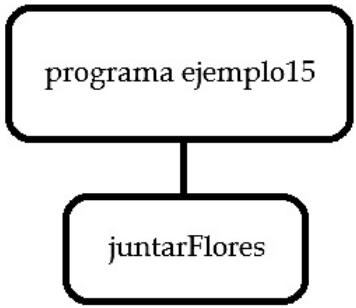
Un procedimiento, en cambio, está formado por un conjunto de sentencias o instrucciones que realizan una determinada tarea y que pueden recibir 0, 1 o más datos (parámetros) y devolver 0, 1 o más resultados.

En el caso de Visual DaVinci solo trabaja con procedimientos denominados en su sintaxis como *procesos*.

Más arriba dijimos que los módulos pueden compartir información y esto se logra a través del pasaje de parámetros. La interfaz del módulo indica qué parámetros (su tipo) y en qué orden serán recibidos.

Ejemplo 15: Juntar todas las flores de las esquinas de la avenida 6. Tenga en cuenta que podría existir más de una flor.

Figura 4.6. Procedimiento juntarflores




```
programa ejemplo15 //Utiliza un procedimiento llamado JuntarFlores
subprogramas
    procedimiento JuntarFlores
    comenzar
        mientras HayFlorEnLaEsquina
            tomarFlor
    fin
comenzar
    iniciar
    pos(6,1)
    repetir 9
        comenzar
            JuntarFlores
            mover
        fin
fin
```

Nota: Los procedimientos simplemente se invocan o llaman por su nombre en el cuerpo del programa. Y deben estar declarados previo a su uso. En este caso el procedimiento **JuntarFlores** no recibe parámetros. Su acción es solo tomar las flores de las esquinas sin contabilizarlas. **Atención:** Si quisiéramos retornar la cantidad de flores que se recogieron, debemos utilizar parámetros en el procedimiento **JuntarFlores**. Esto se hace agregando entre paréntesis el parámetro *cantidad* de tipo número. En el próximo ejemplo lo detallamos.

Ejemplo 16: Juntar todas las flores de las esquinas de la avenida 6. Informar en cada esquina la cantidad juntada y al finalizar el recorrido informar el total de flores que se recogieron. Tenga en cuenta que podría existir más de una flor en cada esquina.

```
programa ejemplo16
//Junta flores y las cuenta
variables
      c:numero
      ct:numero
       esquina:numero

subprogramas
procedimiento JuntarFlores(sa cantidad:numero)
comenzar
  cantidad :=0
  mientras HayFlorEnLaEsquina
      comenzar
         cantidad:=cantidad+1
         tomarFlor
      fin
fin

comenzar
      iniciar
      pos(6,1)
      esquina := 1
      repetir 9
            comenzar
                  JuntarFlores(c)
                  informar("La cantidad de flores de la esquina ", esquina, " es ", c)
                  ct:=ct+c
                  esquina := esquina +1
                  mover
            fin
      informar("La cantidad total de flores es ",ct)
fin
```

Nota: El procedimiento **JuntarFlores** ahora tiene un parámetro (de salida) de tipo numero **(sa cantidad:numero)**. El identificador **sa** indica que el parámetro cantidad es un parámetro de salida.
Cada vez que se invoca o llama al procedimiento, la variable se limpia comenzando nuevamente en cero y dentro del módulo cambiará dependiendo de lo que haga el programador con dicha variable.
Cuando se invoca el procedimiento en el cuerpo del programa se deben respetar los parámetros, su tipo, cantidad y ubicación. En este caso, se trata de un parámetro c, es decir **JuntarFlores(c)**, siendo **c** el resultado de cantidad.

4. 4. Parámetros formales y reales

Cuando se declara un procedimiento se habla de parámetros formales y cuando se lo invoca se habla de parámetros reales o actuales. En el ejemplo 16 el procedimiento **JuntarFlores(sa cantidad:numero)** contiene el

parámetro formal *cantidad* que está en la definición del procedimiento, mientras que **JuntarFlores(c)** contiene el parámetro real "**c**".

En general, los lenguajes de programación exigen que el número de parámetros reales y formales coincidan en tipo y cantidad. DaVinci Concurrente chequea esta condición y en caso de no cumplirse, no permite que se ejecute el programa.

Ejemplo 17: El robot debe recorrer un circuito rectangular dentro de la ciudad cuyos parámetros deben ser ingresados por el usuario (coordenada inicial y tamaño). Debe dejar la huella y comenzar a moverse hacia arriba desde el punto inicial.

```
programa ejemplo17
// Se ingresan los datos para que el robot recorra y dibuje un rectángulo
variables
      Avenida:numero
      Calle:numero
      Altura:numero
      Ancho:numero

subprogramas
procedimiento MovRectan(en AVini:numero; en CAini:numero; en alto:numero; en ancho:numero)
comenzar
      pos(AVini,CAini)
      repetir alto
            mover
      derecha
      repetir ancho
            mover
      derecha
      repetir alto
            mover
      derecha
      repetir ancho
            mover
      derecha
fin

comenzar
      iniciar
      pedir(Avenida)
      pedir(Calle)
      pedir(Altura)
      pedir(Ancho)
      MovRectan(Avenida,Calle,Altura,Ancho)
fin
```

Nota: El procedimiento **MovRectan** solo recibe parámetros de entrada, y en DaVinci esto se indica usando el identificador **en** para cada parámetro de entrada **(en AVini:numero; en CAini:numero;...)**.

Atención: Recordar que los parámetros reales, los que se usan en la invocación deben coincidir en tipo y orden con la especificación de los parámetros formales. Así mismo, para que funcione correctamente un módulo es importante verificar previo a la invocación del mismo que los valores de los parámetros cumplan con las precondiciones establecidas. En este caso, el módulo **MovRectan** no especifica ninguna precondición, entonces el módulo deberá validar que el robot no exceda los límites de la

ciudad. ¿Cómo lo resolverían?

4. 5. Variables globales y locales

Las variables que se encuentran dentro del procedimiento se llaman *locales* y tiene su alcance limitado al mismo, mientras las variables que se designan fuera de los procedimientos son *globales*, y son visibles en todo el programa. De esta forma, en cualquier subprograma podría referenciarlas o modificarlas. Es importante para preservar la independencia de los módulos que si necesitan compartir información, se haga a través del pasaje de parámetros y no referenciando a variables globales.

Los parámetros de un procedimiento pueden funcionar como entrada exclusiva, salida exclusiva o entrada/salida usando para cada caso el prefijo: **en**, **sa** o **es**. En el caso del procedimiento **MovRectan** (del ejercicio 17) usamos solo parámetros de entrada.

Veremos un caso de procedimiento con parámetros de salida.

Ejemplo 18: El robot se dirige a una esquina elegida por el usuario para contar la cantidad de papeles y flores. Además, indique pre y poscondiciones para el módulo.

```
programa ejemplo18
//El usuario elige una esquina para que el robot cuente flores y papeles
variables
      Avenida:numero
      Calle:numero
      CP:numero
      CF:numero

subprogramas
procedimiento ContarPapelFlor(sa Papeles:numero; sa Flores:numero, en av:numero, en calle: numero)
comenzar
       pos(av,calle)
      mientras haypapelenlaesquina
            comenzar
                  Papeles:=Papeles+1
                  tomarpapel
            fin
      mientras hayflorenlaesquina
            comenzar
                  Flores:=Flores+1
                  tomarflor
            fin
fin

comenzar
      iniciar
      pedir(Avenida)
      pedir(Calle)
      ContarPapelFlor(CP,CF, Avenida, calle)
      informar("La cantidad de papeles es ",CP," y la cantidad de flores es ",CF)
fin
```

Nota: El robot primero solicita la esquina elegida por el usuario y luego llama (invoca) al procedimiento **ContarPapelFlor** que tiene dos parámetros de entrada (**av**, **calle**) y dos parámetros de salida (**Papeles**, **Flores**). ¿Cuáles son las precondiciones que deben cumplir los parámetros de entrada?

Ahora veremos cómo se pueden hacer llamadas de procedimientos dentro de procedimientos.

Ejemplo 19: Se pide diseñar e implementar un programa modularizado, donde el robot recorra un rectángulo de 4 x 4 a partir de una coordenada inicial indicada por el usuario. Se pretende que el robot deje la huella de su recorrido y que comience a moverse hacia arriba desde el punto inicial, contando la cantidad de papeles y flores de cada esquina.

```
programa ejemplo19
variables
      Avenida:numero
      Calle:numero
      CP:numero
      CF:numero

subprogramas
procedimiento ContarPapelFlor(es Papeles:numero;es Flores:numero)
comenzar
      mientras haypapelenlaesquina
            comenzar
                  Papeles:=Papeles+1
                  tomarpapel
            fin
      mientras hayflorenlaesquina
            comenzar
                  Flores:=Flores+1
                  tomarflor
            fin
fin

procedimiento MovRectan(en AVini:numero; en CAini:numero; es Cpapel:numero; es Cflor:numero)
comenzar
       pos(AVini,CAini)
            repetir 4
                  comenzar
                        mover
                        ContarPapelFlor(Cpapel,Cflor)
                  fin
            derecha
            repetir 4
                  comenzar
                        mover
                        ContarPapelFlor(Cpapel,Cflor)
                  fin
            derecha
            repetir 4
                  comenzar
                        mover
                        ContarPapelFlor(Cpapel,Cflor)
                  fin
            derecha
            repetir 4
                  comenzar
                        mover
                        ContarPapelFlor(Cpapel,Cflor)
                  fin
            derecha
fin
comenzar
      iniciar
      pedir(Avenida)
      pedir(Calle)
      /*Inicializamos en 0 la cantidad de flores y papeles contados*/
      CP :=0
      CF :=0
      MovRectan(Avenida,Calle,CP,CF)
      informar("La cantidad de papeles es ",CP," y la cantidad de flores es ",CF)
fin
```

Los parámetros de entrada/salida siempre se invocarán usando variable

Pregunta: ¿Cuáles serían la pre y poscondiciones de cada módulo?

Nota: Cuando se realiza la invocación a un módulo con parámetros de entrada-salida o salida, los parámetros reales deben ser siempre variables, ya que no se copia un valor sino una referencia a una posición en memoria.

4. 6. Ejercicios Da Vinci

51. Diseñe e implemente un programa que permita que el robot recorra la ciudad completa (pasando por todas las esquinas) y dejando la huella. El robot debe comenzar su recorrido en la coordenada (1,1).
52. Igual al ejercicio anterior pero informando al final del recorrido la cantidad de esquinas que tienen flores.
53. Diseñe e implemente un programa que permita que el robot recorra la diagonal inversa desde (1,10) hasta (10,1) sin dejar marca.
54. Diseñe e implemente un programa para que el robot dibuje (dejando huella) la palabra LES. Las letras deben tener la misma altura que la ciudad.
55. Diseñe e implemente un programa que permita que el robot realice un recorrido en modo escalera desde la posición (1,10) bajando hasta (10,1) dejando huella.
56. Diseñe e implemente un programa que permita al usuario ingresar la coordenada de partida (Avenida, Calle) y luego el robot realice un recorrido en un rectángulo de 3x3 en sentido contrario al movimiento del reloj y dejando la huella.

    .....

    Nota: El programa principal debe controlar que el robot no sobrepase los límites de la ciudad.

    .....

57. Igual ejercicio anterior, pero además el programa deberá informar al finalizar la cantidad de esquinas que no tienen objetos (papel o flores).
58. Diseñe e implemente un programa que permita que el robot recorra solo las calles pares dejando huellas e informe al finalizar la cantidad (por separado) de flores y papeles que ha encontrado en las esquinas durante su recorrido.

    .....

    Nota: Tenga en cuenta que en una esquina podría haber más de un elemento (flor/papel).

    .....

59. Diseñe e implemente un programa que permita que el robot recorra solo las calles impares dejando huellas e informe al finalizar la cantidad de esquinas que tienen la misma cantidad de flores que de papeles a excepción de las que están vacías.

60. Diseñe e implemente un programa que permita que el robot recorra la ciudad con dirección al azar comenzando desde la posición (3,4), sin dejar huella y sin saltar. El azar se determina en cada esquina y el programa finaliza cuando llegue a la posición (7,7).

    Nota: Si el robot intenta irse fuera de los límites de la ciudad debe hacer un paso hacia atrás (dirección contraria al último paso).

61. Se tienen ubicados en la ciudad cuatro obstáculos en las posiciones: (2,2); (2,8); (8,2) y (8,8). Diseñe e implemente un programa, que permita que el robot recorra la ciudad al azar usando el mismo criterio que el ejercicio anterior. Cuando se encuentra con un obstáculo debe saltar hacia otro lugar también al azar. El algoritmo termina cuando se encuentra con más de dos obstáculos.

    Nota: Recordar que para ubicar los obstáculos se debe ir a «Opciones» – «Configurar ciudad» – «Obstáculos».

62. Igual al ejercicio anterior pero informando la cantidad total de flores y papeles (por separado) al finalizar el recorrido.

Listas de referencias

Lista de referencias bibliográficas

Champredonde, R., Ainchil, V., y Palacios, A. (2000). Teaching experiences in programming using the visual Da Vinci language. En *First International Congress on Tools for Teaching Logic: proceedings: University of Salamanca, June 2000* (pp. 17-21). Universidad de Salamanca.

Compañ-Rosique, P., Satorre-Cuerda, R., Llorens-Largo, F. y Molina-Carmona, R. (2015). Enseñando a programar: un camino directo para desarrollar el pensamiento computacional. *Revista de Educación a Distancia*, (46).

De Giusti, A. E., Madoz, M. C., Bertone, R. A. y Naiouf, R. M. (2001). *Algoritmos, datos y programas: con aplicaciones en Pascal, Delphi y Visual Da Vinci*. Prentice Hall.

Depetris, B. O., Aguil Mallea, D., Pendenti, H., Tejero, G., Feierherd, G. E. y Prisching, G. (2015). La enseñanza y el aprendizaje de la programación y la programación concurrente con DaVinci Concurrente. En *X Congreso sobre Tecnología en Educación & Educación en Tecnología (TE & ET). Corrientes*.

Di Mare, A. (1991). Tipos abstractos de datos y programación por objetos, Reporte Técnico PIBDC-03-91, proyecto 326-89-019, Escuela de Ciencias de la Computación e Informática, UCR, 1991. Revisión 2010. http://www.di-mare.com/adolfo/p/oop-adt.htm

García, C. E. (2012). *Algoritmos y programación I. Guía para docentes*. Fundación Gabriela Piedrahita Uribe.

Guerrero, M., Guamán, D. S. y Caiza, J. C. (2015). Revisión de herramientas de apoyo en el proceso de enseñanza-aprendizaje de programación. *Revista Politécnica, 35*(1), p. 84.

Goin, M. (2016). *Caminando junto al lenguaje C*. Editorial UNRN.

Joyanes Aguilar, L. (2008). *Fundamentos de programación*. McGraw-Hill Interamericana.

López, L. M., Amaro, S., Alonso de Armiño, A. C., Godoy, I., Leiva, M. y Piñero, J. C. (2016, May). Aplicando nuevos aspectos en la Programación de Computadoras. En XVIII *Workshop de Investigadores en Ciencias de la Computación WICC 2016, Entre Ríos, Argentina*.

Mallea Aguil, D. E. (2012). Un intérprete multiplataforma para la iniciación a la programación estructurada y concurrente. Da Vinci Concurrente. [Tesis de grado de Licenciatura]. Facultad de ingeniería de la Universidad Nacional de la Patagonia. San Juan Bosco. Ushuaia, Tierra del Fuego.

Polya, G. (2004). *How to Solve It*. Princeton Science Library Edition.

Lista de figuras

Financial Times. (2011). *Drew Hoston* [Fotografía]. https://www.flickr.com/photos/financialtimes/6478266407/in/photostream. CC-BY-2.0

Game Developers Choice Awards. (2010). *Gabe Newell* [Fotografía]. https://www.flickr.com/photos/officialgdc/4427575126/. CC-BY-2.0

Joi Ito. (2011). *Mitchel Resnick* [Fotografía]. https://commons.wikimedia.org/wiki/File:Mitchel_Resnick.jpg. CC-BY-2.0

Usuario: ak_mardini. (2006). *Seymour Papert* [Fotografía]. https://www.flickr.com/photos/mardinix/152833938/. CC-BY-SA-2.0

Problemas y algoritmos : un enfoque práctico
Edith Lovos y Martin Goin.
Primera edición. Viedma: Universidad Nacional de Río Negro, 2021.
120 p. ; 23 x 15 cm. Lecturas de cátedra

ISBN 978-987-4960-31-3

1. Algoritmo. 2. Sistemas de Información. I. Lovos, Edith. II. Título.
CDD 005.01

© Universidad Nacional de Río Negro, 2021.
editorial.unrn.edu.ar
© Edith Lovos y Martin Goin, 2021.

Queda hecho el depósito que dispone la Ley 11.723.

Diseño de colección: Dirección de Publicaciones-Editorial de la UNRN
Dirección editorial: Ignacio Artola
Coordinación de edición y edición de textos: Diego Martín Salinas
Corrección de textos: Verónica García Bianchi
Diagramación y diseño: Sergio Campozano
Imagen de tapa: Editorial UNRN, 2021.

Licencia Creative Commons
Usted es libre de: compartir-copiar, distribuir, ejecutar y comunicar públicamente esta obra bajo las condiciones de:
Atribución – No comercial – Sin obra derivada

Problemas y algoritmos
Un enfoque práctico
fue compuesto con la familia tipográfica Alegreya en sus diferentes variables.
Se editó en abril de 2021 en la Dirección de Publicaciones-Editorial de la UNRN.

Problemas y algoritmos
Un enfoque práctico

El desarrollo de la capacidad analítica y creadora para resolver problemas constituye el punto de partida para quienes desean incursionar en el mundo de la programación informática. Es esa capacidad la que permite enfocar el elemento clave del pensamiento computacional: el algoritmo.

Como un tutorial que avanza paso a paso a través de ejercicios y ejemplos, este libro busca ayudar al lector interesado a adquirir las destrezas necesarias para poder diseñar e implementar, de manera fácil y rápida, soluciones algorítmicas básicas, previo a la inmersión en el uso de uno o más lenguajes de programación en particular.

Es por ello que *Problemas y algoritmos* es un valioso material de apoyo tanto para docentes como para estudiantes de nivel medio y superior, en este último caso, que cursen las asignaturas iniciales de tecnicaturas, ingenierías, profesorados y licenciaturas orientadas o vinculadas a la informática.

LUA
EDITORIAL UNRN
Universidad Nacional de Río Negro
9 789874 960313