Introducción

Muchos de los que se inician en la programación, lo hacen enfrentándose desde el inicio a un Lenguaje de Programación determinado. Esto generalmente, lleva al futuro programador, a hacer un uso excesivo de tiempo para poder comprender la lógica de la solución de un problema a través de la computadora. Por eso algunos abandonan la programación y otros se desgastan para resolver una tarea no muy compleja, con la consiguiente falta de productividad y eficiencia en el desarrollo de una aplicación.

Las causas de que esto ocurra, están dadas generalmente, en que se desconoce que antes de enfrentarse a resolver un problema utilizando un Lenguaje de Programación, hay que conocer la Lógica de la Programación, que es la que nos permite definir, de una manera técnica y organizada, los conceptos que nos preparan para diseñar en términos generales, la solución a problemas que pueden ser implementados a través de un lenguaje de programación.

Otros aspectos que hay que tener en cuenta antes de intentar resolver un problema con un Lenguaje de Programación son: conocer qué es la Programación, qué son los Lenguajes de Programación y cuáles son sus estructuras sintácticas y semánticas.

Todos los textos consultados sobre Lógica de Programación, la tratan apoyándose en la estructura de los ordenadores o partiendo de un lenguaje de programación en particular y no llegan a separar el aprendizaje de la programación del aprendizaje de este lenguaje.

En este libro se trata la solución de problemas, a partir de un seudo código que permite su transcripción en cualquier Lenguaje de Programación. Para ello partimos del criterio de que no son nuestros programas los que están hechos para dar trabajo a los ordenadores, sino más bien al contrario: los ordenadores están hechos para ejecutar nuestros programas. Por eso es que después de tratar todo lo referente a la Lógica de Programación es que se aborda lo concerniente a los lenguajes de programación y la codificación de programas, con el ánimo de dar ejemplos concretos y demostrar, que cuando se va aprogramar lo más importante no es el conocimiento de un Lenguaje de Programación (ya que con su uso uno se va acostumbrando a sus reglas) sino la lógica de programación que se use para desarrollar soluciones algorítmicas encaminadas a lograr un objetivo. De acuerdo con esto la diferencia entre un Lenguaje y otro serán solo sus instrucciones debido a que la lógica seguirá siendo la misma.

Se tratan a demás de forma detallada los objetos y acciones elementales de la Lógica de Programación a partir de situaciones problémicas y se ofrecen ejercicios resueltos en cada una de las temáticas así como 248 ejercicios propuestos.

El autor

CapÍtulo I. Lógica de programación.

¿Qué es la lógica?
Lógica es la Ciencia que estudia la estructura, fundamentos y uso de las expresiones del conocimiento humano.

La Lógica de Programación define, en forma precisa, los conceptos generales que permiten diseñar la solución a problemas para ser implementados a través de los lenguajes de programación. Su aprendizaje no necesita del conocimiento previo de estos lenguajes. Sólo después de dominar sus conceptos, se está en condiciones óptimas para iniciar el estudio de cualquier lenguaje de programación y con ello dar solución a problemas a través de computadoras. Por eso es importante conocer cuáles son los pasos esenciales para resolver un problema utilizando esta tecnología. Precisamente es lo primero que se aborda en este capítulo.
Si estudias detenidamente este capítulo y desarrollas los ejercicios aquí planteados, comprenderás que programar no es más que buscar soluciones lógicas a los problemas, utilizando conceptos sencillos.

Pasos para la solución de un problema con computadoras

En la solución de un problema con computadoras deben seguirse como mínimo los siguiente pasos:

1. **Análisis del problema.:** en este paso se hacen tres especificaciones:

1.1 Entrada: se seleccionan los objetos de entrada definiendo su tipo.

1.2 Proceso: se definen las operaciones o cálculos necesarios para dar solución al problema.

1.3 Salida: se declaran los objetos en los que se dará salida a los resultados de la operaciones o cálculos realizados, definiendo el tipo y precisión de los mismos

En lo adelante, para referirnos a la ejecución de estos tres pasos le llamaremos abreviadamente análisis EPS (**E**ntrada **P**roceso **S**alida).

2. **Diseño del algoritmo:** en este paso se definen en forma clara y precisa los pasos a seguir para dar solución al problema

3. **Corrida en frío del algoritmo:** El programador realiza una comprobación paso por paso del algoritmo diseñado, verificando que los resultados obtenidos coincidan con los resultados esperados.

4. **Codificación:** traducir cada paso del algoritmo a instrucciones de un lenguaje de programación.

5. **Ejecución del programa:** se ejecuta el programa elaborado en el lenguaje de programación.

6. **Comprobación del programa:** verificación de que los resultados obtenidos después de la ejecución del programa corresponden con los resultados esperados.

   De estos pasos en la solución de un problema con la computadora se tratan en este tema: el análisis del problema, diseño del algoritmo, la corrida en frío o ejecución paso a paso y se ejemplifica la codificación de algoritmos en distintos tipos de lenguajes de programación. La codificación, ejecución del programa y la comprobación del mismo están fuera del alcance de este libro.

Noción de acción, procesador, entorno, algoritmo.

Desde el surgimiento de las computadoras el hombre se ha propuesto hallar los algoritmos de solución de los diversos problemas del mundo que lo rodea para implementarlos de forma automatizada en las mismas.

El trabajo de una computadora consiste en esencia en la ejecución de un algoritmo o conjunto de algoritmos. El concepto de algoritmo, uno de los principales de la Matemática y la Informática, surgió mucho antes de las máquinas computadoras, e incluso de las calculadoras de cualquier tipo. Durante siglos se ha utilizado el concepto intuitivo de algoritmo.

Pero ¿Qué es un algoritmo?

A priori podemos entender por **algoritmo** un claro y exacto sistema de reglas que determinan una **secuencia de acciones** a realizar sobre un determinado objeto, que después de un número finito de pasos nos permite lograr el objetivo de resolver la tarea planteada.

El sistema de reglas se convierte en un algoritmo cuando puede darse en forma de instrucción a diferentes personas, desconocedoras de la esencia del problema y éstas pueden, según las instrucciones, actuar en forma similar y alcanzar la misma solución.

La palabra algoritmo proviene del nombre de un matemático árabe del siglo IX (Alheresni Khowarismi), originario de la antigua ciudad de Khowarism, hoy Kiva, situada en la antigua URSS. Formuló las reglas de las cuatro operaciones aritméticas con varios dígitos. Posteriormente este concepto comenzó a utilizarse en general para designar las secuencias de operaciones que conducen a la solución de cualquier tarea matemática.
Con el decursar del tiempo el proceso de búsqueda y formalización de algoritmos dejó de ser tarea sólo de matemáticos y se obtuvieron diferentes tipos de algoritmos. Así surgieron algoritmos para juegos como damas y ajedrez, donde los objetos son figuras y posiciones en los que se requiere seleccionar el próximo paso. En otros casos son acciones de una corriente eléctrica o de una determinada máquina o por ejemplo el algoritmo de búsqueda de una palabra en un diccionario donde se utilizan textos. Pero en todos los casos debe considerarse que los algoritmos no trabajan con **objetos** del mundo real, sino con representaciones, abstracciones de éstos. Por ello para designarlos se utilizan variables, símbolos, codificaciones.

Estos algoritmos de diversas índoles y con un sentido más amplio, reciben también el nombre de procedimiento.
Esto es realmente lo que hacen las máquinas, ejecutar formalmente los algoritmos que el hombre les entrega en forma de programas, lo cual les hace en ocasiones realizar cosas que pueden parecer lógicas e incluso "inteligentes". En esto no hay nada de extraño. Los programas realizados por el ser humano sobre la base de algoritmos comprobados exhaustivamente, pueden dar la apariencia de que las computadoras actúan igual que el hombre.
No todos los algoritmos son implementables en computadoras, por ello se distinguen dos tipos fundamentales:
Se define como algoritmos informales a todos aquellos algoritmos que no son realizables a través de una computadora. Son aquellos donde el ejecutor real es el ser humano, como el algoritmo para hacer un merengue con 3 huevos y Algoritmos computacionales aquellos algoritmos que deben ser implementados en una computadora.

Veamos como se trata en informática el concepto de algoritmo a través de la solución de un problema. Para ello consideremos el siguiente enunciado:

Hacer un merengue con 3 huevos.
Este enunciado describe la realización de cierto trabajo.
Hagamos el análisis EPS del problema anterior:

Análisis EPS
Entradas: Ingredientes y utensilios empleados:
- 3 huevos
- 1 tenedor
- 1 plato
- azúcar

Proceso: elaborar el merengue.
Salida: merengue

Para ejecutar este trabajo se pueden realizar las siguientes **acciones**:
a) Romper los 3 huevos.
b) Verter las claras en el plato.
c) Coger el tenedor.
d) Batir las claras hasta que se conviertan en merengue.
e) Verter el azúcar en el plato.
f) Batir el contenido del plato hasta que se mezcle uniformemente.

Es evidente que para ejecutar estas acciones se necesita de alguien que entienda el enunciado del problema planteado y sea capaz de resolverlo.
A cualquier entidad capaz de entender un enunciado y ejecutar el trabajo indicado en él se le llama **Procesador.**
En el caso de este enunciado cualquier persona que sepa leer y disponga de los utensilios necesarios puede ser un **Procesador.**
El **procesador** no puede ejecutar el trabajo solicitado a menos que se den las condiciones requeridas por el enunciado.
Los utensilios serán: 3 huevos, un plato, un tenedor, azúcar.
El conjunto de los objetos necesarios para la ejecución del trabajo constituye el **entorno** de dicho trabajo.
El entorno, para un procesador dado, es por tanto, específico del trabajo a realizar.
En la realización del trabajo del enunciado se distinguen distintas etapas(a, b, c, d, e, f) cada una de las cuales se denomina **acción.**

Consideremos la acción a) Romper 3 huevos. Antes de ejecutar la acción los huevos estaban enteros, después de ejecutar la acción los huevos están rotos, es decir cambiaron su estado.
La ejecución de una acción puede necesitar una observación del entorno, por ejemplo la acción d) (Batir las claras hasta que se conviertan en merengue). Esta acción debe ejecutarse hasta que las claras se conviertan en merengue.
Por regla general el procesador respeta la secuencia de las acciones: las ejecuta en el orden en que aparecen en el enunciado.
Para un procesador, una acción es **primitiva** si el enunciado de dicha acción es suficiente para poder ejecutarla sin información suplementaria.
Una acción no primitiva debe ser descompuesta en acciones primitivas.
Por ejemplo si el procesador es un niño la acción a) (romper 3 huevos)y la acción b)(Verter las claras en el plato), puede ser poco explícita. Será necesario descomponer estas acciones como se muestra a continuación:

a1: romper un huevo.
b1: verter la clara en el plato.
a2: romper un huevo.
b2: verter la clara en el plato.
a3: romper un huevo.
b:3 verter la clara en el plato.

Observe como las acciones a, y b se repiten., es decir se han descompuesto en acciones que son primitivas para el procesador niño.

Vistas las nociones de acción, entorno y procesador, se puede decir que un **algoritmo** es una secuencia de acciones primitivas(o un algoritmo ya conocido y escrito) que realiza un procesador para transformar el entorno del estado inicial dado en el estado final deseado.

Por regla general el trabajo indicado en un enunciado puede tener varias soluciones.
El algoritmo tiene un grupo de palabras que constituyen las instrucciones, que son determinadas palabras claves, así como símbolos matemáticos conocidos.
En un algoritmo se puede llamar a otro algoritmo.

Ejemplo:
Sean los algoritmo X y el algoritmo Y

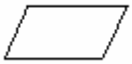
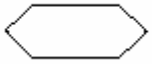
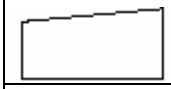
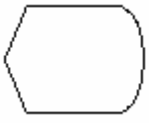
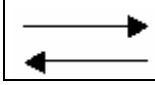
Algoritmo X	Algoritmo Y
a	a
c	b
Y	b
d	a

Si a, b, c, y, d son acciones primitivas, diremos que la acción y en el algoritmo X es una llamada al algoritmo Y. Si se ejecuta el algoritmo X, las acciones ejecutadas estarían en el orden: a, c, a, b, b, a, d.
En el algoritmo para hacer un merengue, expuesto anteriormente, se ha representado cada acción utilizando nuestras propias palabras para describirlas, sin embargo hay otras formas de hacer esta descripción.

Técnicas Para Representar Algoritmos

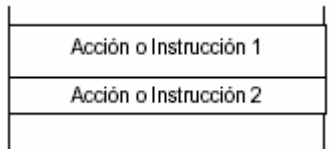
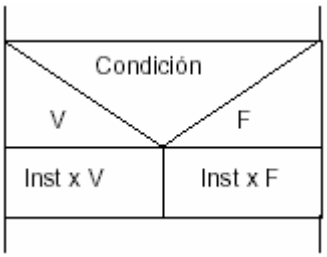
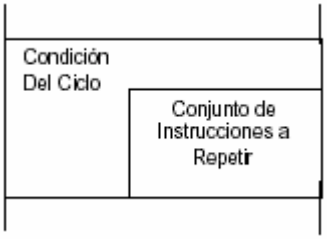
Diagramas de Flujo

Los Diagramas de Flujo parten de símbolos que nos permiten decir lo mismo que dijimos hace un momento en los algoritmos pero de una manera gráfica. Los símbolos (y el significado de ellos) que se han acordado utilizar dentro de los Diagramas de Flujo o Flujogramas son los siguientes:

	Un rectángulo representa una acción a ejecutar de manera clara y concreta.
	Este símbolo nos permite representar una Decisión. En su interior podemos escribir la condición de la cual depende la decisión y por sus extremos derecho (o izquierdo) e inferior se pueden colocar las salidas para los casos en que la condición sea Falsa o sea Verdadera.
	Este símbolo nos permite expresar un proceso de entrada o salida, teniendo en cuenta que una entrada en un algoritmo se concibe como el proceso a través del cual se recibe información y una salida es el proceso a través del cual se entrega información.
	Este símbolo permite representar la escritura de un resultado o salida.
	Este símbolo representa el Inicio ó el Fin de un Algoritmo, se escribe dentro de él la palabra Inicio o Fin y ubicarlo apropiadamente dentro del Diagrama de Flujo.
	Este símbolo permite que coloquemos en él los parámetros de inicio de una estructura repetitiva.
	Este símbolo representa una entrada de datos utilizando el teclado de la computadora. Se escriben en su interior los nombres de las variables donde queremos que se almacene el dato que entra por el teclado.
	Estos símbolo se conocen como conectores lógicos. Nos permiten representar la continuación de un Diagrama de Flujo cuando éste es tan largo que no cabe en una sola hoja.
	Este símbolo permite representar una lectura de datos. Representa una Tarjeta Perforada fue establecida cuando aún se leían los datos a través de tarjetas perforadas. Actualmente este símbolo representa sencillamente una lectura.
	Este símbolo representa una salida de datos pero escrita en la pantalla de la computadora. Es un símbolo un poco más moderno en de los diagramas de flujo.
	Las flechas son los símbolos que representan la forma de conexión entre los demás símbolos determinando el Flujo de ejecución de acciones.

Diagramas Rectangulares Estructurados

Una de las dificultades de los Diagramas de Flujo radica en que así como brinda la posibilidad de representar gráficamente el flujo de la solución a un problema también abre el espacio para que un programador desordenado ponga flechas de flujo inadecuadamente y finalmente obtenga una representación más compleja que la idea misma. Precisamente la técnica de Diagramas Rectangulares Estructurados también permite usar herramientas gráficas para representar la solución a un problema con la ventaja de que no brinda la posibilidad de que seamos desordenados en nuestra concepción. Se basa en representar todo el algoritmo dentro del marco de un rectángulo y a diferencia de la técnica anterior, se mueve básicamente con la utilización de tres símbolos que corresponden a cada una de las estructuras básicas de la lógica de programación. Estas representaciones son las siguientes:

	Para representar secuencias de instrucciones se escribe cada instrucción en una línea “enmarcada”.
	Para representar una decisión se utiliza este símbolo donde por el lado izquierdo se escriben las acciones que correspondería ejecutar en el caso de que la condición fuera Verdadera y por el lado derecho colocaríamos las acciones o instrucciones a ejecutar cuando la condición fuera Falsa.
	Para representar una estructura repetitiva sencillamente en la esquina superior izquierda del bloque correspondiente colocamos la condición y dentro del bloque colocamos las instrucciones o acciones que se debe repetir y que a su vez, por supuesto, dependen de la condición.

Seudo códigos

La tercera técnica para representar algoritmos es el Seudo código.
Qué es pues un seudo código..?

Un **Seudo código** es la representación textual de un algoritmo de manera que dicho texto se encuentre enmarcado en determinadas normas técnicas que faciliten su posterior transcripción a un lenguaje de Programación.

El seudo código tiene un grupo de palabras claves y símbolos que constituyen su vocabulario para representar las acciones esta técnica orientada hacia los algoritmos computacionales.
Para escribir un algoritmo bajo la técnica de seudo código deben seguirse las siguientes normas:
Primera Norma.- Escribir la palabra algoritmo y después de un espacio escribir el nombre del algoritmo. Es conveniente que dicho nombre haga una referencia aproximada a lo que contiene. Si a un seudo código lo llamamos seudo código X es posible que más adelante no nos sea muy claro su objetivo pero si lo llamamos seudo código Liquidar es muy factible que cada vez que lo veamos nos vamos a acordar que su objetivo era la liquidación de un determinado valor. Pero si lo llamamos seudo código LiqSalNe es muy posible que cada vez que veamos este nombre nos acordemos que ese seudo código es el que nos permite Liquidar el Salario Neto.
Segunda Norma.- Todo el cuerpo del algoritmo deberá ir “encerrado” entre las palabras *Inicio* y *Fin* indicando donde comienza y donde termina el seudo código.
Tercera Norma.- Luego de colocada la palabra **Inicio**, debemos a continuación declarar el entorno con el cual vamos a trabajar.
Cuarta norma: Las acciones se escriben después de declarar el entorno.

Implementación del Seudo código

La estructura básica de un seudo código se implementa así:

```
Algoritmo Nombre
 Inicio
 Entorno
    Acciones
Fin
```

Algoritmo Nombre: indica el inicio del algoritmo
Nombre : es el nombre del algoritmo.
Inicio: indica el inicio del algoritmo
Entorno: define los objetos de trabajo
Acciones: acciones a realizar
Fin: indica el fin del algoritmo

La utilización eficiente de esta técnica es unos de los objetivos fundamentales de este libro, para que a través de ella pueda expresar cualquier algoritmo computacional y obtenga una solución que luego sea fácilmente codificable en cualquier Lenguaje de Programación.

Cuadro Comparativo de las tres técnicas

Nombre de la Técnica	Ventajas	Desventajas
Diagramas de Flujo	a. Permite visualizar gráficamente el camino que sigue la solución a un problema. b. Por ser tan simplificado es muy entendible c. No se necesitan muchos conocimientos técnicos para utilizar esta técnica	a. Dado que los flujos (representados con flechas) pueden ir de cualquier lugar a cualquier lugar da espacio para que el diagrama llegue a ser casi inentendible b. Deben conocerse bien símbolos que se van a utilizar c. No todos los símbolos están estandarizados d. Los ciclos deben ser reinterpretados para poder ser diagramados en esta técnica e. No siempre es muy entendible f. Algunas veces la analogía entre el diagrama y la codificación en el Lenguaje de programación resulta ser compleja.
Diagrama Rectangular Estructurado	a. Permite tener un marco referencial concreto y definido para la representación de los algoritmos. b. Solo tiene tres esquemas que le permiten a su vez representar las tres estructuras básicas c. Exige orden en la representación de un algoritmo. d. Es muy entendible. e. La analogía entre la codificación y el diagrama normalmente es directa y por lo tanto muy sencilla.	a. Exige una fundamentación técnica que permita representar la solución a cualquier problema a través de las tres estructuras básicas. b. No es una técnica muy popularizada.
Seudo Códico	a. Permite expresar la solución algorítmica a un problema en nuestro propio lenguaje y casi con nuestras propias reglas. b. La codificación se facilita dado que la transcripción es directa. c. Si el programador es ordenado, esta puede llegar a ser la técnica más entendible.	a. Exige mucho orden para ser utilizada eficiente-mente. b. Exige el mantenimiento claro de los conceptos de algoritmos como tales. c. Las decisiones deben estar encasilladas dentro de los alcances de los operadores lógicos y operadores booleanos.

Cada una de estas técnicas tiene ventajas y desventajas que las hacen comparable con las demás. Debe dársele prioridad de uso a la técnica que facilite la codificación ya que la computadora no ejecutará los algoritmos escritos en estas técnicas sino escritos en términos de un Lenguaje de Programación en este sentido la técnica que más facilita la transcripción es el Seudo código.

Trace o ejecución paso a paso del algoritmo (corrida en frío)

Para analizar como las acciones de un algoritmo transforman el entorno, pasándolo de un estado a otro, se hace un seguimiento acción por acción, de las modificaciones del estado de cada uno de los objetos de dicho entorno. Esta ejecución manual realizada por el creador del algoritmo, se llama trace, ejecución paso a paso o corrida en frío y se implementa a través del siguiente esquema:

Objetos Estado inicial Acciones Estado final

Ejemplo, hagamos el trace de las transformaciones que sufren cada uno de los objetos del entorno del algoritmo Hacer un merengue con 3 huevos:

a) Romper los 3 huevos.
b) Verter las claras en el plato.
c) Coger el tenedor.
d) Batir las claras hasta que se conviertan en merengue.
e) Verter el azúcar en el plato.
f) Batir el contenido del plato hasta que se mezcle uniformemente.

Objetos	Estado inicial	Acciones	Estado final
		Algoritmo Merengue Inicio	
3 huevos	Enteros	Romper los tres huevos	Rotos
Claras	En el cascarón	Verter las claras en el plato.	En el plato
Tenedor	En la superficie de trabajo	Coger el tenedor	En la mano
Claras	Enteras	Batir las claras hasta que se conviertan en merengue.	Merengue
Azúcar	En la superficie de trabajo	Verter el azúcar en el plato	En el plato
Contenido del plato	Sin mezclar	Batir el contenido del plato hasta que se mezcle uniformemente.	mezclado

Observe en este seguimiento paso a paso, como cada acción transformó el estado de los objetos del entorno, pasándolos de un estado inicial a otro final.

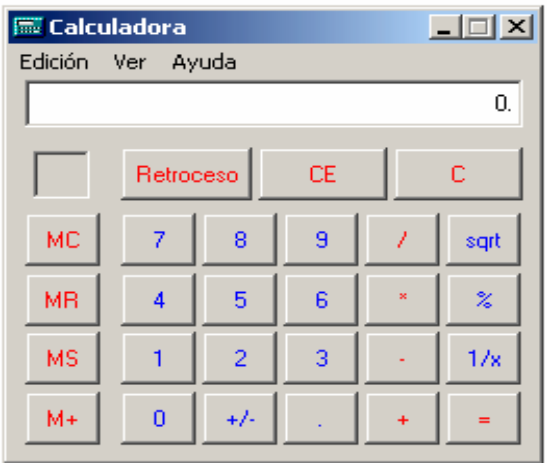
La ejecución paso a paso o trace es importante porque con ella se comprueba el funcionamiento del algoritmo. Nos brinda dos elementos muy importante en la

construcción de los algoritmos: primero nos permite saber si realmente el algoritmo cumple el objetivo o no y segundo, nos permite saber dónde está el error o los errores para ser corregidos.
Cuando se ha realizado una corrida en frío rigurosa y se han confrontados los resultados finales con el objetivo inicial solo en ese momento se podrá saber si el algoritmo estaba bien concebido o no. Solo de esta forma podemos llegar a los mismos resultados que llegaría la computadora. Esto quiere decir que si al realizar una corrida en frío vemos que nuestro algoritmo arroja unos resultados que no coinciden con el objetivo entonces si la computadora ejecutara dicho algoritmo (convertido en programa por supuesto) también arrojaría resultados errados.

Ejercicios resueltos sobre noción de acción, procesador, entorno, algoritmo.

Implemente un algoritmo para dar solución a los problemas que plantean los siguientes enunciados. Haga el análisis EPS y observe como las acciones transforman el entorno desde el estado inicial al estado final.

1. Calcule la media de tres números con una calculadora



Solución:
Análisis EPS
Entrada: 3 números.
Proceso: Calcular la media de los tres números.
Salida: un número que representa la media.

Algoritmo Media
Inicio
Entorno: calculadora
A: Pulsar C
B: Teclear el primer número
C: Pulsar +
D: Teclear el segundo número
E: Pulsar +
F: Teclear el tercer número
G: Pulsar /
H: Teclear 3
I: Pulsar =
Fin

2. Hacer un batido con 5 guayabas.

Análisis EPS
Entrada: 5 guayabas, una batidora, azúcar, hielo, agua, leche
Proceso: hacer el batido
Salida: batido

Algoritmo Batido
Inicio
Entorno: guayabas, batidora, azúcar, hielo, agua, leche
Coger las 5 guayabas y ponerlas en el baso de la batidora
Accionar la batidora para licuar las guayabas
Cuando las guayabas estén licuadas parar la batidora
Poner el azúcar en el baso de la batidora
Poner la leche en el baso de la batidora
Poner el hielo en el baso de la batidora
Accionar la batidora para licuar el contenido
Cuan el contenido se haya mezclado uniformemente parar la batidora
Fin

3. Una persona está encargada de relacionar en una hoja de notas los resultados obtenidos por 3 estudiantes en un examen. La hoja de notas esta compuesta de dos columnas; la primera con los nombres de los estudiantes relacionados por orden alfabético, en la segunda se pondrán las notas al lado de cada nombre correspondiente.

Solución:
Análisis EPS:
Entrada: exámenes, hojas de notas, lápiz
Proceso: Pasar notas
Salida: hojas de notas actualizada
Algoritmo PonerNotas
Inicio
Entorno: paquete de exámenes, hojas de notas, lápiz
A: Coger del paquete de exámenes uno y mirar el nombre y la nota.
B: Localizar en la hoja de notas, el nombre correspondiente al del examen.
C: Anotar al lado del nombre la nota.

D : Coger del paquete de exámenes uno y mirar el nombre y la nota.
E : Localizar en la hoja de notas, el nombre correspondiente al del examen.
F : Anotar al lado del nombre la nota.

G : Coger del paquete de exámenes uno y mirar el nombre y la nota.
H : Localizar en la hoja de notas, el nombre correspondiente al del examen.
I : Anotar al lado del nombre la nota
.Fin

Observe que hubo que repetir las acciones A:, B:, C: tres veces.

Resolver los ejercicios con algoritmos informales propuestos en el Capítulo II.

Siempre que vaya a resolver un problema guíese por sus razonamientos y busque siempre el camino más obvio y sencillo. No existe un problema que se resuelva con lógica cuya solución no sea sencilla. Antes de comenzar a pensar en la lógica de programación piense en su propia lógica. Diseñe las soluciones pensando en sus propias reglas y luego si ajústese a las reglas que la lógica de programación impone para facilitar la posterior codificación.

Cómo saber cuál es el camino más lógico..? Pues sencillamente la solución más obvia es la que demarca cuál es el camino más lógico. La lógica es ese conjunto de razonamientos que nos permiten solucionar fácilmente determinados problemas. Cada persona puede tener un enfoque diferente en cuanto a dicha solución y es muy importante que, cuando se trabaja en equipo, escuchar cuál es la solución de los otros. Indiscutiblemente que para cada problema ha de existir una solución óptima, obvia y además muy sencilla. ¿Porqué razón cada persona puede llegar a encontrar una solución diferente a un determinado problema..? Son múltiples las explicaciones pero se debe destacar dentro de ellas el entorno social, la preparación, el conocimiento, la convivencia y la utilización de conceptos nuevos acerca de la misma lógica, su mismo entorno personal y muchas más razones que pueden hacer que una persona vea la solución de un problema con una óptica diferente a como la podemos ver nosotros.

Lo que para una persona es absolutamente ilógico para otra es completamente lógico y es posible que ambas tengan la razón (o al menos crean tenerla) dadas sus propias condiciones personales e intelectuales. Sin embargo podemos decir que si se mira un problema con una óptica aproximadamente neutral nos podremos aproximar a la solución más sencilla y obvia. Cómo poder llegar a obtener una óptica aproximadamente neutral..? Hay una forma de acercarse a este concepto y es estudiando conceptos que nos permitan lograr este objetivo. No es fácil determinar en qué momento hemos alcanzado una lógica aproximadamente normal pero cuando el problema a solucionar puede ser resuelto con la utilización de la tecnología entonces ésta se convierte en el catalizador y en la regla de medida para saber hasta donde nuestra solución es realmente óptima o no.

La lógica computacional permite buscar soluciones que pueden ser implementables con tecnología. Por esta razón es que la buena utilización de la misma nos va a permitir saber hasta donde nos hemos acercado a la solución óptima. Un problema que se solucione con lógica computacional solo tendrá una y solo una solución óptima. Es decir la solución más sencilla de implementar, la más obvia y la más entendible, a la luz de los conceptos de la misma lógica computacional.

Es muy importante que destine parte de su tiempo a resolver problemas, aunque no sean estrictamente de programación, porque en esos momentos donde se sienta a pensar detenidamente en la búsqueda de solución de un determinado problema, es cuando realmente está utilizando su cerebro. Normalmente nuestro cerebro se va acostumbrando, a buscar entre sus conocimientos las soluciones que se estén necesitando; pero muchas veces tenemos que crear soluciones y es allí donde ponemos a funcionar nuestro cerebro.

¿Cuántas veces realmente se ha sentado a crear una solución de un problema?.Son muy pocas las veces ya que en aquellas oportunidades en donde ha tratado de hacerlo y, por ventura, ha encontrado la solución es porque su cerebro ha buscado en su “biblioteca de conocimientos” alguna solución análoga a problemas parecidos y la ha “ajustado” al problema en mención.

Por tal motivo es muy importante que de vez en cuando resuelva acertijos matemáticos, problemas con dados, cartas e incluso hasta resolver adivinanzas. Este tipo de problemas le van permitiendo buscar soluciones espontáneas, originales, creadas por usted mismo y que además son solución a un determinado problema planteado. Esos juegos de lógica van haciendo que el cerebro cree soluciones y no las busque en las que ya conoce. Es muy importante que tenga en cuenta todo esto dado que en programación va a necesitar crear soluciones a problemas determinados basadas en sus conceptos y en el conocimiento que tenga de las herramientas y de los conceptos aquí planteados.

Acciones y Objetos elementales.

Formalización del entorno de un problema

Para avanzar un poco más en el estudio de las bases de la programación, es necesario aprender a formalizar el entorno de un problema. Para ello tenemos que definir un número de reglas que nos permitan transcribir sin ambigüedad y con precisión los constituyentes del entorno de un problema.

Analicemos las siguientes acciones de tres algoritmos diferentes:

A: Romper los 3 huevos.
B: Verter las claras en el plato.
C: Coger el tenedor.
D: Batir las claras hasta que se conviertan en merengue.
E: Verter el azúcar en el plato.
F: Batir el contenido del plato hasta que se mezcle uniformemente

A: Pulsar C
B: Teclear el primer número
C: Pulsar +
D: Teclear el segundo número
E: Pulsar +
F: Teclear el tercer número
G: Pulsar /
H: Teclear 3
I: Pulsar = (aparece el resultado)

A: colocarse en la mesa Nº 1
B: coger un paquete
C: Ir a la mesa Nº 2
D: Coger un paquete
E: Ir a la mesa Nº 3
F: Poner los paquetes

Se trata de acciones que hacen intervenir distintos objetos:
Huevos, tenedor, sartén, 3, tecla, mesa Nº1

Veamos de estos ejemplos algunas características de la noción de objeto:
Vemos que cada objeto tiene un **nombre**. Que se utiliza para designarlo sin ambigüedad.

Igualmente cada objeto tiene un uso particular y los usos no son intercambiables: podemos dividir con el objeto 3 pero no con el objeto tenedor. Por tanto diremos que cada objeto es de un **tipo** particular y que este tipo indica las características comunes a todos los estados posibles de dicho objeto. Por ejemplo la mesa Nº1 y la mesa Nº2 son del **tipo** "mesa"; es decir que representa en cada instante una de las mesas del grupo. Para describir un tipo es suficiente con enumerar los diferentes estados que un objeto de este tipo puede adoptar.

Ejemplo:
Tipo "mesa" { mesa Nº1, mesa Nº2..... mesa Nºn}
El primer número y 3 son del tipo numérico, para describir este tipo hay que enumerar todos los valores de números posibles. Podremos definir el tipo numérico como el conjunto de todos los números que el procesador es capaz de tratar.
Todo objeto del entorno tiene un **valor** susceptible de cambio a lo largo de la ejecución de una acción. Por ejemplo encima de la mesa Nº1 hay cierta cantidad de paquetes, el paquete de encima de la pila tiene por ejemplo el **valor** "paquete5 ", antes de ejecutarse la acción B, y el "paquete3" después de la ejecutarse la acción B. En relación a esto, observemos una particularidad importante en el cambio del valor de un objeto: a menos que hayamos anotado antes el valor precedente, es imposible recuperarlo ahora a partir del nuevo estado del entorno obtenido después de haber ejecutado la acción que atribuye el nuevo valor. Diremos que cada nuevo valor de un objeto destruye el valor precedente. Siguiendo con la noción de **valor**, vemos igualmente que ciertos objetos, tales como el objeto 3, no cambian jamás de valor.

Para resumir con una imagen las ideas anteriores, podemos decir que un objeto puede describirse como una caja con una etiqueta (su nombre), de una cierta forma (su tipo) y que contiene una información (su valor).

Del análisis anterior se deduce que un objeto es de un tipo de terminado y que puede mantener su valor constante o variable durante la ejecución de las acciones. Veamos estos conceptos.

Variables elementales

Constantes y variables

Una variables es un objeto cuyo valor es variable, tiene un nombre que sirve para designarla y un tipo (invariable) que describe la utilización posible de la variable.
Una constante es un objeto de valor invariable. Tiene un nombre que sirve para designarla y un tipo (invariable) que describe la utilización posible de la constante.

Tipos de valores elementales

Tipo numérico: es el conjunto de los valores numéricos que el procesador sabe tratar.
Ejemplo:
10
12.5
125.34

Tipo cadena: es el conjunto de las cadenas de caracteres que se pueden formar a partir de los elementos del conjunto de caracteres(letras, cifras, espacios y signos especiales) que el procesador reconoce. Para evitar confundir una cadena de caracteres con el nombre de una variable, la representaremos siempre entre comillas (“ “) delimitando el principio y el final de la cadena.

Ejemplo:
“lista”
“Apellidos, nombres”
“total a facturar”

El tipo lógico: es el conjunto de valores lógicos, **cierto** y **falso**. Una variable de tipo lógico siempre tiene uno de estos dos valores.

Variables compuestas

Una variable compuesta es una asociación de varias variables elementales. Por ejemplo si disponemos de una nota obtenida en un examen por un estudiante, para una asignatura dada, podremos colocar estas informaciones en la variable compuesta RESULTADO:

RESULTADO		
NOMBRE	ASIGNATURA	NOTA
“Ana”	“Informática”	5

En el ejemplo anterior, la variable RESULTADO está compuesta de tres variables elementales:
NOMBRE: variable de tipo cadena cuyo valor es “Ana”
ASIGNATURA: variable de tipo cadena cuyo valor es “Informática”
NOTA: variable tipo numérico, cuyo valor es 5.
Diremos que las variables NOMBRE, ASIGNATURA y NOTA son subdivisiones de la variable compuesta RESULTADO.
La variable compuesta se referencia así:
RESULTADO.NOMBRE: hace referencia a la subdivisión nombre de RESULTADO.
RESULTADO.ASIGNATURA: hace referencia a la subdivisión asignatura de ESULTADO.
RESULTADO.NOTA: hace referencia a la subdivisión NOTA de RESULTADO.
Una subdivisión puede ser ella misma una variable compuesta.

Ejemplo:

ESTABLECIMIENTO			
NOMBRE	DIRECTOR	DIRECCIÓN	
“Florería”	“Francisco Pérez”	CALLE	CIUDAD
		“Maceo”	“Tunas”

Ejemplo de cómo referenciar a CALLE:
ESTABLECIMIENTO.DIRECCIÓN.CALLE

La subdivisión DIRECCIÓN de la variable ESTABLECIMIENTO comprende las subdivisiones CALLE y CIUDAD

Del análisis anterior se concluye que el entorno de un problema(en los algoritmos computacionales) está constituido por variables y constantes.

Acción de Asignación

Para atribuir un valor a una variable se usa la notación:
V ← E
Donde V es el nombre de la variable a la que el procesador le asigna el valor.
← carácter de la acción de asignación.
E representa el valor a asignar y puede ser:

- Una constante
- El nombre de otra variable que contenga el valor
- Una expresión aritmética describiendo un cálculo a efectuar.

Ambas entidades al lado del signo de asignación ← deben ser siempre del mismo tipo.
Ejemplo:
a) Entero ← 10
b) Nombre ← “Ana”
c) Total ← suma
d) Num ← a+b

La acción **a** asigna a la variable numérica **entero** el valor 10.
Para que la acción **b** sea correcta es necesario que la variable **nombre** sea de tipo cadena ya que esta acción le asigna la cadena “Ana”.
La acción **c** le asigna a la variable **total** el valor de la variable **suma**. Ambas variables deben ser del mismo tipo.
La acción **d** asigna a la variable numérica **num** el resultado de un cálculo numérico.
Para atribuir un valor a una constante se usa la misma notación ← pero el valor a asignar tiene que ser **invariable.**

Operadores aritméticos

Para los cálculos numéricos se utilizan los operadores:

+ suma (A←B+2)
- Resta (A← B - C)
* multiplicación (A*B)
/ división con resultado decimal (A/B) (Ejemplo: 9/2=4.5
\ división entera sin decimal (A\B) Ejemplo: 9\2=4
^ potencia (A←B^2 se le asigna a A el valor de b elevado al cuadrado)

En la multiplicación se usa * para que no se confunda con la x.
En las expresiones numéricas el procesador ejecuta estos operadores según la siguiente prioridad:

Exponenciación (^)
Multiplicación y división (*, /)
División de enteros (\)
Adición y substracción (+, –)

Ejemplo:
En la acción A ← B+C/D-E*F, el procesador calcula primero C/D, después E*F y luego ejecuta la suma y la resta.
Se pueden modificar las prioridades utilizando paréntesis
Ejemplo:
En la acción A ← B+C/(D-E)*F, el procesador calcula primero D-E.

En cuanto a la precedencia de ejecución de los operadores, primero se resuelven son los paréntesis más internos o sea aquellos paréntesis que no tienen más paréntesis adentro.
Cuando, al desarrollar una corrida en frío, tenga que resolver una expresión que incluya paréntesis, operadores y variables, no vaya a suponer nada, resuelva la operación tal y como lo indica la expresión.

Ejercicios resueltos sobre variables, constante y acción de asignación:

1) Sean las variables A, B dos variables numéricas cuyos valores son 5 y 8 respectivamente.
   Elabore un algoritmo para:
   a) Calcular la suma de las variables A y B y guardar el resultado en otra variable.
   b) Multiplicar por 2 el valor de la suma de ambas variables y el resultado guardarlo en otra variable.
   Haga el análisis EPS y el trace del algoritmo.

   Solución:
   Análisis EPS
   Entrada: A, B variables numéricas
   Proceso: Calcular la suma.
   Multiplicar por 2 la suma
   Salida: R, P variables numéricas

```
Algoritmo SumaAB
Inicio
    Entorno:
    A,B, R, P variables numéricas
    A← 5
    B← 8
    R←A+B
    P←2*R
Fin
```

   Trace de este algoritmo

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmo SumaAB	
		Inicio	
A	valor indeterminado	A=5	5
B	valor indeterminado	B=8	8
R	valor indeterminado	R=5+8	13
P	valor indeterminado	P=5*13	65
		Fin	

2) Se tienen las variables numéricas X, Y, Z y Q, cuyos valores son respectivamente: 5, 9, 10, 15. Elabore un algoritmo que dé solución a las situaciones siguientes:
   a) A la suma de los valores de **X** y **Z** dividirlo por el valor **Y** y el resultado multiplicarlo por el valor de Q. Guardar el resultado en una variable.
   b) El valor de la variable Q elevarlo al cuadrado y guardarlo en una variable.
   c) Haga el trace del algoritmo

Solución:
Análisis EPS:

Entrada: X, Y, Z ,Q variables numéricas.
Proceso: Sumar, multiplicar, dividir, elevar al cuadrado.
Salida: S, C

Algoritmo Calcular
Inicio

Entorno:
X, Y, Z ,Q, S, C variables numéricas
X← 5
Y← 9
Z← 10
Q← 15
S← (X+Z)/Y*Q
C← Q^2

Fin

Trace de este algoritmo

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmo Calcular	
		Inicio	
X	valor indeterminado	X=5	5
Y	valor indeterminado	Y=9	9
Z	valor indeterminado	Z=10	10
Q	valor indeterminado	Q=15	15
S	valor indeterminado	S=(5+10)/9*15	25
C	valor indeterminado	C=15^2	225
		Fin	

3) La variable A guarda la palabra cuba**;** la variable B guarda la palabra patria. Elabore un algoritmo para intercambiar los valores de ambas variables y Haga el trace del algoritmo.

Solución:
Análisis EPS:
Entrada: A,B,C variables de cadena
Proceso: Intercambiar
Salida: A con valor de B y B con valor de A .

```
Algoritmo IntercambiarPalabras
Inicio
    Entorno: :A,B,C variables numéricas.
    A← "cuba"
    B← "patria"
    C← A
    A← B
    B← C
Fin
```

Trace:

Objetos	**Estado inicial**	**Acciones**	**Estado final**
	Algoritmo IntercambiarPalabras		
	Inicio		
A	valor indeterminado	A="cuba"	"cuba"
B	valor indeterminado	B="patria"	"patria"
C	valor indeterminado	C="cuba"	"cuba"
A	"cuba"	A="patria"	"patria"
B	"patria"	B= "cuba"	"cuba"
	Fin		

4) Sean las variables numéricas A igual 5, B igual 45, C igual 2. Conociendo que M es igual A por B y N es igual A más B menos C.
Elabore un algoritmo para calcular M dividido por C y multiplicado por N dividido por 2. Haga el trace del algoritmo.
Solución:
Análisis EPS
Entrada: A, B, C variables numéricas.
Proceso: multiplicar, sumar, restar, dividir
Salida: M, N, T

```
Algoritmo CalABC
Inicio
    Entorno: A,B,C,T,M,N variables numéricas.
    A←5
    B←45
    C←2
```

M←A*B
N←A+B-C
T←M/C*N/2
Fin

Trace de este algoritmo

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmo CalABC	
		Inicio	
A	valor indeterminado	A=5	5
B	valor indeterminado	B=45	45
C	valor indeterminado	C=2	2
M	valor indeterminado	M=5*45	225
N	valor indeterminado	N=5+45-2	48
T	valor indeterminado	T=225/2*48/2	2700
		Fin	

5) Haga el análisis EPS de este algoritmo y su trace:

Algoritmo M
Inicio
Entorno: K, S variables numéricas
K ← 4
S ← 0
S ← S + K
K ← K - 2
Fin

Solución:
Análisis EPS
Entrada: K, S variables numéricas.
Proceso: Sumar y restar
Salida: K, S

Trace de este algoritmo

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmo M	
		Inicio	
K	valor indeterminado	K=4	4
S	valor indeterminado	S=0	0
S	0	S=0+4	4
K	4	K=4-2	2
		Fin	

6) Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a . 2
b = b + 3 – c
c = $\frac{a}{2} + b^2$

Solución:
Análisis EPS
Entrada: a, b, c variables numéricas.
Proceso: Sumar, restar, dividir, elevar al cuadrado
Salida: a, b, c

Algoritmo Variablesabc
Inicio
Entorno: a, b, c variables numéricas
a←2
b←3
c←4
a←a*2
b←b+3-c
c←a/2+b^2
Fin

Trace de este algoritmo

Objetos	Estado inicial	Acciones	Estado final
		Algoritmo Variablesabc	
		Inicio	
a	indeterminado	a=2	2
b	indeterminado	b=3	3
c	indeterminado	c=4	4
a	2	a=2*2	4
b	3	b=3+3-4	2
c	4	c=4/2+2^2	6
		Fin	

7. Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = \frac{a + \dfrac{b}{a + b + \dfrac{b}{c}}}{a + \dfrac{b}{c + a}}$$

Solución:
Algoritmos
Inicio
Entorno: x, a , b, c variables numéricas
a←3
b←4
c←2

x←(a+(b/(a+b+b/c)))/(a+(b/(c+a)))
Fin

Trace de este algoritmo

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmos	
		Inicio	
a	indeterminado	a=3	3
b	indeterminado	b=4	4
c	indeterminado	c=2	2
x	indeterminado	x=(3+(4/(3+4+4/2)))/(2+(4/(2+3)))	1.2
		Fin	

Resolver los ejercicios de estructura lineal propuestos en el Capítulo II.

Cuando vaya a iniciar un algoritmo declare las variables que saltan a la vista en el mismo enunciado, no se detenga a pensar en las demás variables que va a necesitar pues éstas van surgiendo en la medida que las va necesitando en el desarrollo del mismo algoritmo.

En este libro se han explicado y se han utilizado tres tipos de datos: los datos de tipo numéricos, los datos de tipo cadena y los datos de tipo lógico. Cada uno tiene sus características técnicas que permiten manipular y manejar variables con ese tipo de datos. No olvide que siempre que almacene un valor determinado en una variable, el valor anterior se pierde pues el nuevo valor reemplaza el dato anterior. Por eso cuando necesite almacenar el dato anterior de una variables pues sencillamente tendrá que utilizar otra para que ese dato no se le pierda.

Entrada de datos.

Puede ocurrir que un valor a tratar no forme parte del entorno de un problema, para poder registrar ese valor hay que introducirlo en el entorno en el momento de la ejecución del algoritmo, para ello se utiliza la notación **leer.**

Ejemplo:

Leer V donde V es el nombre de una variable. Consideramos que toda lectura es una acción primitiva. Una lectura es una asignación.

Leer puede tener un texto antes de la variable, el cual se pone entre comillas "" (ejemplo. **Leer** "entre un número" n)

Ejemplo:

```
Algoritmo Entrada
Inicio
    Entorno:
     A, B, S: variables numéricas
      Leer A
     Leer B
      S←A+B
Fin
```

Salida de un dato

En ocasiones hay que hacer salir un valor del entorno para comunicar una información al exterior. Se trata de comunicar al exterior el valor de una variable. Llamamos **escritura** a esta operación que la consideramos una acción primitiva. Esta operación se indica por: **Escribir**

Ejemplo:

Escribir V.

Si la salida lleva un texto se escribe entre comillas y a continuación la variable así:
Escribir "el resultado es" V.

Para escribir más de variable con una sola acción **Escribir**, se escriben las variables separadas por comas. Ejemplo: Escribir A, B, C de esta forma se escribirán las variables una al lado de la otra.

La acción **escribir** V no cambia el valor de la variable; sin embargo la acción **leer** V si.

Implementemos algunos algoritmos utilizando estas notaciones:

Ejemplo:

Dado un número entero calcular la suma de los 3 primeros enteros siguientes.

Solución

Análisis EPS

Entrada: Num variable numérica para guardar el valor del número dado, Entero: variable numérica para incrementar el valor de esta variable

Proceso: sumar

Salida: S variable numérica entera que contiene la suma de los enteros

Algoritmo SumaEnteros
Inicio
Entorno:
Num, S, entero: variables numéricas
Num ← 3
S ← 0
Entero ← 1
S ← S+num+Entero
Entero ← Entero+1
S ← S+num+Entero
Entero ← Entero+1
S ← S+num+Entero
Entero ← Entero+1
Fin

A la expresión Entero ← Entero+1 se le llama **contador** porque va incrementando el valor de la variable de 1 en 1.

Ejercicios resueltos sobre entrada y salida de datos:

1. Haga el trace del siguiente algoritmo:

```
Algoritmo K
Inicio
    Entorno: X, Y, P variables numéricas
    P← 0
    X ← 4
    Y ← 3
    P ← P + X
    Y ← Y – 1
    Escribir P
    Escribir Y
Fin
```

Hacer el seguimiento de este algoritmo K.

Solución:
Trace de este algoritmo

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmo K	
		Inicio	
P	valor indeterminado	P=0	0
X	valor indeterminado	X=4	4
Y	valor indeterminado	Y=3	3
P	0	P=0+4	4
Y	3	Y=3-1	2
		4	
		2	
		Fin	

2. Dado el valor de dos variables numéricas K y Q, intercambiar sus valores. Hacer el algoritmo y su seguimiento para este enunciado.

Solución:

```
Algoritmo Intercambiar
Inicio
   Entorno: K, Q, I variables numéricas
    Leer K
    I ← K se conserva el valor de K
    Leer Q
    K ← Q
    Q ← I
Fin
```

Trace de este algoritmo para K=2 y Q=5

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmo Intercambiar	
		Inicio	
K	indeterminado	K=2	2
I	indeterminado	I=2	2
Q	indeterminado	Q=5	5
K	2	K=5	5
Q	5	Q=2	2
		Fin	

3. Dada las variables numéricas R, y T:
   a) Calcular su suma y asignársela a la variable S.
   b) Muestre fuera del entorno el resultado.
   c) Hacer el trace del algoritmo

Solución:

```
Algoritmo SumaMuestra
Inicio
    Entorno: R, T, S  variables numéricas

    Leer R
    Leer T
    S ← R+T
    Escribir S
Fin
```

Trace de este algoritmo para R=6 y T=10

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmo SumaMuestra	
		Inicio	
R	indeterminado	R=6	6
T	indeterminado	T=10	10
S	indeterminado	S=6+10	16
		16	
		Fin	

4. Dada una palabra, comunicarla al exterior del entorno. Hacer el algoritmo y su seguimiento

Solución:

Algoritmo Palabra
Inicio
Entorno: V variable de tipo cadena.
Leer V
Escribir V
Fin

Trace de este algoritmo para V=”Cuba”

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmo Palabra	
		Inicio	
V	indeterminado	V=”Cuba”	“Cuba”
		“Cuba”	
		Fin	

5. Elabore un algoritmo que, a partir de preguntar el año de nacimiento de una persona, calcule y escriba su edad.

Solución:

Algoritmo Edad
Inicio
Entorno.
Año: variable numérica para guardar el año.
Edad: variable numérica para guardar la edad.
Leer año
Edad ← 2002-año
Escribir Edad
Fin

6. Haga un algoritmo para calcular el área de un círculo. La fórmula que describe dicha área es:

$A=3.14*R^2$

A es el área
R radio del círculo

Solución:

```
Algoritmo AreaCirculo
Inicio
        Entorno:
        A: variable numérica para guardar el valor del área
        R: variable numérica para guardar el valor radio
        Pi: constante numérica
        Pi←3.14
        Leer R
        A ← Pi*R^2
        Escribir A
 Fin
```

7. Elabore un algoritmo para calcular la velocidad (V) de un móvil que se ha desplazado 240 Km en 3 horas con un movimiento rectilíneo uniforme. La fórmula que describe este movimiento es: V=S/T donde V es la velocidad, S el desplazamiento y T el tiempo.

Solución:

```
Algoritmo MovUniforme
Inicio
        Entorno:
        V: variable numérica para guardar la velocidad
        S: variable numérica para guardar el desplazamiento.
        T: variable numérica para guardar el tiempo
        S← 240 Km
        T← 3 h
        V← S/T
        Escribir V
Fin
```

8. Haga un algoritmo para calcular el promedio de las asignaturas Matemática, Física y Química.

Solución:

```
Algoritmo Promedio
Inicio
        Entorno:
        P: variable numérica para guardar el promedio
        NM: variable numérica para guardar la nota de Matemática
        NF: variable numérica para guardar la nota de Física.
       NQ: variable numérica para guardar la nota de Química.
        Leer NM
        Leer NF
        Leer NQ
        P← (Nm+NF+NQ)/3
        Escribir P
Fin
```

9. Leer un número entero de 3 dígitos y calcular la diferencia entre el primer dígito y el segundo. Haga el trace:

Solución:

```
Algoritmo Dígitos
        Inicio
        Entorno: n, d1, d2, d3, dif variables numéricas
        Leer n
        d3←n-(n\10)*10
        n←n\10
        d2← n-(n\10)*10
        n←n\10
        d1← n-(n\10)*10
        dif←d1-d2
        Escribir " La diferencia es " dif
Fin
```

Trace para n=423

Objetos	**Estado inicial**	**Acciones**	**Estado final**
Algoritmo Dígitos			
		Inicio	
n	indeterminado	n=423	423
d3	indeterminado	d3=423-(423\10)*10	
		d3=423-42*10	
		d3=423-420	3
n	423	n=423\10	42
d2	indeterminado	d2=42-(42\10)*10	
		d2=42-4*10	
		d2=42-40	2

n	42	n=42\10	4
d1	indeterminado	d1=4-(4\10)*10	
		d1=4-0*10	
		d1=4-0	4
dif	indeterminado	dif=4-2	2
		La diferencia es 2	
		Fin	

Predicado. Operadores de relación. Operadores lógicos

Noción de predicado:

Veamos este concepto a través de un ejemplo:
Sea N el conjunto de los números enteros naturales. Se define en N el siguiente enunciado: “x es un múltiplo de 3”. Este enunciado adopta el valor **cierto** para algunos valores de x (3, 6, 9 ....) y el valor **falso** para los otros (5, 7, 8, ...)

Predicado (función proposicional) es cualquier frase a la que se le puede atribuir uno y sólo uno de los elementos de un conjunto.

Por tanto la frase “x es un múltiplo de 3” no es un predicado.
Ejemplos:
Si x e y son variables numéricas, el enunciado “ x mayor que 3” es un predicado cierto para algunos valores de x, falso para el resto.

Consideremos las frases siguientes:

1. El número 3 es más grande que el número 1.
2. Nunca llueve en Francia.
3. La tierra es redonda.
4. Se debe conducir por la derecha.
5. El cubo de un número x es positivo.

Las tres primeras frases son predicados (proposiciones) : la frase 1 y 3 son ciertas, la frase 2 es falsa. Por el contrario las frases 4 y 5 no son predicados; ya que a veces son ciertas y a veces falsas.

La noción de predicado permite expresar que una condición está o no realizada en el entorno.
En la composición de algoritmos, expresamos los predicados que serán evaluados por el procesador en el momento de la ejecución del algoritmo.
Para expresar un predicado se escribe una comparación entre dos valores del mismo tipo. Una comparación tal se llama predicado elemental.

Operadores de comparación

Para las comparaciones de tipo numérico o de cadena utilizamos los siguientes operadores de comparación: **=, <, >, <=, >=, <>**. El resultado de la aplicación de estos operadores devuelven el valor **cierto** o **falso**.
En la expresión de predicados compuestos es importante el uso de los paréntesis.

Orden de prioridad de estos operadores:

Igual a (=)
Desigualdad (<>)
Menor que (<)
Mayor que (>)
Menor o igual que (<=)
Mayor o igual que (>=)

Ejemplo:
Sean X e Y dos variables numéricas cuyos valores son respectivamente 4 y 20, la tabla siguiente da los valores respectivos de algunos predicados:

Predicado	Valor
X = Y	falso
X > 3	cierto
Y < X	falso

La comparación de los caracteres se define por el orden alfabético: "A"<"B"<"C"<"D"<...<"Z"<"a"<"b"<"c"<"d"<...<"z"
Para la comparación de cadenas de caracteres la comparación se efectúa de izquierda a derecha, de la misma manera que en un diccionario, así se puede decir que la cadena "Thomas" es menor que la constante "Tomate" (aparece antes en el diccionario).

Operadores lógicos

Cuando la utilización de comparaciones no es suficiente para expresar una situación, se utilizan los **conectores lógicos** u operadores **no**(negación) **, y** (conjunción)**, o**(no exclusivo) para formar, a partir de predicados elementales, **predicados compuestos.**
El conector **no** devuelve el valor **cierto** cuando la condición es falsa y viceversa.
El orden de prioridad de estos operadores es **no, y, o.**
Consideremos las frases siguientes para comprender la función de un conector lógico:
Si Elena ha acabado su trabajo y si tiene suficiente dinero , irá al cine.
p1 p2 p3
Esta frase esta constituida por 3 proposiciones: p1, p2, p3. Las proposiciones p1 y p2 están enlazadas por la conjunción **y** ; esta permite expresar el hecho de que p3 es cierto (Elena irá al cine) si y solamente si p1 y p2 son ambas ciertas. Si p1 es falsa o si p2 es falsa, o si las dos proposiciones son falsas, p3 es entonces falsa.

Tabla de verdad de los conectores lógicos:

P	no p
cierto	Falso
falso	cierto

P	q	P y q
Falso	Falso	Falso
Falso	Cierto	Falso
Cierto	Falso	Falso
Cierto	Cierto	cierto

P	q	P o q
Falso	Falso	Falso
Falso	Cierto	Cierto
Cierto	Falso	cierto
Cierto	Cierto	cierto

La siguiente frase ilustra la función del conector **O** :

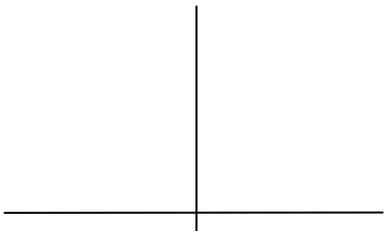
El barco partirá mañana si hay al menos 20 pasajeros o si hace buen tiempo.

Las dos condiciones no se han de cumplir obligatoriamente: el barco partirá si se cumple una o la otra de las condiciones(o las dos)

Ejemplos:
Sean **Z, K**, **P, Q** variables numéricas.
Z ← 4
K ← 4
P ← 6
Q ← 8

Predicado	Valor
No Z = K	falso
No P>Q	cierto

El conector **Y** devuelve el valor **cierto** cuando todas las condiciones son ciertas, en caso contrario devuelve **falso**



Ejemplos:
Para los valores de las variables **X** e **Y**:
X ← 2
Y ← 4

Predicado	Valor
x = 1 **Y** y = 2	falso
x<y **Y** Y>3	cierto

El conector **O** devuelve el valor **cierto** cuando una de las condiciones o ambas son ciertas, cuando ninguna es cierta devuelve **falso.**
Ejemplo:
Para los valores de las variables **X** e **Y** anteriores:
X ← 2
Y ← 4

Predicado	Valor
x = 1 **O** y = 4	cierto
x > y **O** Y < 3	falso

Orden de prioridad de los operadores
1. No
2. Y
3. O

Veamos un ejemplo más complejo:
Sean las variables numéricas. X ← 1, Y ← 4, Z ← 3
Analicemos el siguiente predicado:

(X = 1 **O** Y = 2) **Y** Z >3

(X = 1 **O** Y = 2) **Y** Z > 3
Cierto falso falso
cierto

falso

Ejercicios resueltos sobre predicado y operadores:

1. X, Y, Z, T son variables numéricas de un entorno dado. Cada una de las cuestiones de este ejercicio presenta una situación caracterizando un estado o un grupo de estados de este entorno. Se pide el predicado correspondiente.

Por ejemplo, a la situación: "Los valores de X e Y son ambos superiores a 3", corresponde el predicado: X>3 y Y>3.

Escriba el predicado correspondiente a las siguientes situaciones:

1. Los valores de X, Y y Z son idénticos
   Respuesta: X=Y y Y=Z
2. Los valores de X, Y y Z son idénticos pero diferentes del de T.
   Respuesta: X=Y y Y=Z y Z<>T
3. El valor de X está comprendido estrictamente entre los valores de Y y T y el valor de Y es inferior al de T.
   Respuesta: X>=Y y X<=T
4. Entre los valores de X, Y y Z, al menos dos valores son idénticos
   Respuesta: X=Y o Y=Z o Z=X

5. Dados dos valores numéricos comunicar si son iguales o no. Haga el algoritmo y su seguimiento.

Solución:

Algoritmo Comparación
Inicio
Entorno: A, B variables numéricas
    Leer A
    Leer B
    Escribir A = B Devolverá cierto o falso según los valores de A y B.
Fin

Trace de este algoritmo para A=12 y B=10

Objetos	Estado inicial	Acciones	Estado final
		Algoritmo Comparación	
		Inicio	
A	indeterminado	A=12	12
B	indeterminado	B=10	10
		12=10	Falso
		Fin	

Parametrización de un algoritmo

Otra forma de introducir un valor en un entorno es a través de la parametrización.
La parametrización es la **transferencia de valores entre un algoritmo y otro.**
En la introducción vimos que dentro de un algoritmo puede llamarse a otro. Cuando se llama a un algoritmo, se le pueden enviar los datos o parámetros (variables) con los que se quiere que el algoritmo llamado resuelva el problema. A esto se le llama parametrización de un algoritmo. Esto permite hacer algoritmos más claros y funcionales y lograr una programación estructurada, de manera tal que cada algoritmo resuelva una parte de un problema complejo.

Parámetro es el valor que se pasa y puede ser una variable o un valor constante.
Veamos como se hace la trasferencia de parámetros entre algoritmos.
Para que un algoritmo llame a otro se escribe el nombre del algoritmo que se llama como una acción más.
Para pasar parámetros, después del nombre del algoritmo se ponen entre paréntesis y separados por coma los parámetros.

Ejemplo:

```
Algoritmo Inicio
        Calcular (X, Y)
Fin
```

Con la acción: Calcular (X,Y) se llama al algoritmo Calcular enviándole los parámetros X e Y.
El algoritmo Calcular debe escribirse así:

```
Algoritmo Calcular (P, Q)
        S← P+Q
Fin
```

Los parámetros P y Q son los parámetros formales y X e Y los parámetros actuales. Estos parámetros pueden tener los mismos nombres; pero para mayor claridad es mejor usar otros.
Los parámetros actuales son los que se envían por parte del algoritmo que hace la llamada y los formales son los parámetros en los que se reciben los valores enviados.

Un parámetro formal es una variable elegida como parámetro en la definición de un algoritmo. Un parámetro actual es una variable o un valor constante utilizado en una llamada en lugar de un parámetro formal. En la ejecución de una llamada, la correspondencia entre un parámetro actual y un parámetro formal es definida por la posición: el primer parámetro actual corresponde al primer parámetro formal, el segundo parámetro actual corresponde al segundo parámetro formal etc.

Ejemplo en el caso anterior al parámetro P le corresponde el parámetro X y al Q le corresponde el parámetro Y.

De esto resulta, que cada llamada debe comprender tantos parámetros actuales como formales hay en la definición del algoritmo. Cada parámetro actual debe ser del mismo tipo que el parámetro formal correspondiente.
La ejecución de una llamada obliga al algoritmo a trabajar directamente con los parámetros actuales en lugar de con los formales.

Tipos de parámetros
Parámetro-datos: cuando el valor inicial es utilizado por el algoritmo y queda invariable después de la ejecución de éste.

Parámetro-resultado: cuando el valor inicial no tiene significado para el algoritmo; la ejecución del algoritmo le asigna un nuevo valor.
Parámetro-dato-resultado: es un parámetro cuyo valor inicial es utilizado por el algoritmo, pero es reemplazado al final de la ejecución por un valor-resultado.

Ejemplo:

```
Algoritmo Inicio
Inicio
        Entorno:
        A , B ,R , S variables numéricas
        R ← 0
        S←5
        Leer A,B
        CalcularSuma (A,B,R,S)
         Escribir R
         Escribir S
Fin
Algoritmo CalcularSuma (P, Q, R, Z)
Inicio
        R ←P+Q
        Z←Z+P
Fin
```

Aquí los parámetros actuales son A , B y R y los formales: P, Q, R y Z. Obsérvese que el resultado se escribe en el algoritmo que llama a los otros.
A y B son parámetro dato, R es un parámetro-resultado y S es un parámetro-dato-resultado.

Ejercicios resueltos de parametrización

1. Desde un algoritmo transfiera un número entero hacia otro algoritmo donde se calcule muestre la raíz cuadrada de dicho número.

```
Algoritmo EjercicioRaíz
   Inicio
```

```
    Entorno: n Variable numérica
    Leer n
    CalRaíz( n)
 Fin

Algoritmo CalRaíz(a)
   Inicio
   Entorno: r variable numérica
   r ← a ^ (1/2)
   Escribir "La raíz cuadrada del número es" r
Fin
```

Trace para n=4

Objetos	Estado inicial	Acciones	Estado final
		Algoritmo CalRaíz	
		Inicio	
n	indeterminado	n=4	4
		CalRaíz(4)	
		Algoritmo CalRaíz(4)	
		Inicio	
c	indeterminado	c=4^(1/2)	2
		La raíz cuadrada del número es	4
		Fin	
		Fin	

2. Leer un número entero en un algoritmo y trasferirlo hacia otro algoritmo donde se lea un nuevo número y se calcule el producto de ambos. Mostrar el resultado en el primer algoritmo. Clasifique los parámetros.

```
Algoritmo Números
    Inicio
    Entorno: n1, p Variable numérica
    Leer n1
    P←0
    CalPro( n1, p)
  Escribir p
Fin

CalPro(a,b)
    Inicio
    Entorno: n2 variable entera
   Leer n2
    b←a*n2
 Fin
```

Trace para n1=2, n2=3

Objetos	Estado inicial	Acciones	Estado final
		Algoritmo CalPro Inicio	
n1	indeterminado	n1=2	2
p	indeterminado	p=0	0
		CalPro(2,0) Algoritmo CalPro(2,0) Inicio	
n2	indeterminado	n=3	3
b	0	b=2*3	6
		Fin	
		6	
		Fin	

Clasificación de los parámetros:

n1 es un parámetro dato porque conserva su valor después de ser utilizado.

P es un parámetro resultado porque su valor no es utilizado y se le asigna un nuevo valor.

Estructuras básicas de control.

El concepto de Estructura

Una estructura se define como un esquema que nos permite representar de manera simplificada alguna idea y que bajo condiciones normales es constante. Ello significa que de alguna manera el pensamiento del ser humano, en lo que se refiere a los algoritmos, está enmarcado en algún tipo de Estructuras que no solo le permiten tener un medio más simplificado para expresar las ideas sino que además permite "restringir" un poco el horizonte de la Lógica Algorítmica. Precisamente el pensamiento se mueve entre tres estructuras básicas, todo lo que hace, sin importar que sea, cualquier acción o conjunto de acciones que haga siempre estarán enmarcadas en estas tres estructuras:

1. Secuencias de acciones
2. Decisión de acción
3. Ciclos de acciones

Estructura secuencial

Permanentemente estamos inmersos en esta estructura y generalmente primero planeamos cada secuencia de acciones (consciente o inconscientemente) antes de ejecutarlas. Cada una de las cosas que hacemos diariamente no son más que secuencias de acciones que hemos planeado para poder cumplir con nuestros objetivos en la sociedad.

Ejemplo cuando tiene que pensar que debe ir hasta la parada de la guagua a tomar el transporte lo que va organizando en su mente es una secuencia de acciones que le permitan acercarse a la parada, esperar la guagua correcta y tomarla.

Esta es la típica y evidente en cualquier proceso donde las acciones se realizan una detrás de otras, consecutivas en el tiempo.

Para escribir una secuencia de ordenes o acciones todo lo que tiene que hacer es colocar una nueva orden o una nueva acción después de la última que haya colocado. De esta manera se entiende la secuencialidad y la ordinalidad en la ejecución de esas acciones.

Podríamos decir que esta estructura es la que gobierna todo el concepto general de la programación y a la vez permite mantener el hilo lógico conceptual del diseño de algoritmos.

Ejemplos:

1. Desarrollar un algoritmo que nos permita asomarnos a una ventana, pero vamos a asumir que la ventana a donde nos queremos asomar ya está abierta y que no estamos muy distantes de ella.

Algoritmo AsomarseAVentana
Inicio
    Entorno: ventana
    Ubicar la ventana por la que nos queremos asomar
    Levantarnos del lugar en donde estemos sentados
    Avanzar hacia la ventana
    Llegar hasta tener la ventana muy cerca

```
        Asomarnos por la ventana
Fin
```

En el ejemplo dado puede ver que cada acción está antes de una y después de otra (excepto por supuesto la primera y la última). También puede notar que para que este algoritmo nos permita asomarnos a la ventana todo lo que tenemos que hacer es realizar cada acción en el orden en que están planteadas y sencillamente realizar una a la vez. Eso nos va a permitir lograr el objetivo propuesto.

2. Realizar el algoritmo para ponernos una camisa (asumimos que la camisa está en nuestro ropero doblada y abrochada)

```
Algoritmo ColocarCamisa
Inicio
        Entorno: camisa
        Dirigirnos a nuestro ropero
        Abrir el ropero
        Tomar una camisa
        Desabrocharla
        Abrir la camisa
        Meter un brazo por una de sus mangas
        Meter el otro brazo por la otra de sus mangas
        Ajustar la camisa al tronco
        Abotonarla (botón a botón)
Fin
```

Al igual que en el ejemplo anterior todo lo que tenemos que hacer es ejecutar cada acción en el orden indicado y hacerlo paso a paso y entonces podremos lograr el objetivo de colocarnos la camisa.
Puede notar que para utilizar la estructura de secuencia (que a veces parece ser tan obvia) todo lo que tenemos que hacer es ir colocando una acción tras otra y, por supuesto, ser muy racionales en el orden de dichas acciones porque en cuestión de algoritmos el orden de los factores sí altera el resultado.

3. Dados dos números enteros escribir su suma, su diferencia y su producto.

```
Algoritmo CalculaSumDifPro
Inicio
        Entorno:
        N1 variable numérica
        N2 variable numérica
        Sum variable numérica
        Dif  variable numérica
        Pro variable numérica
        Leer N1, N2
        Sum← N1+N2
        Dif ← N1-N2
        Pro ← N1*N2
```

```
        Escribir Sum
        Escribir Dif
        Escribir Pro
Fin
```

Estructuras alternativas o de decisión

Gracias a la estructura de decisión puede escoger lo que para usted sea la mejor alternativa de entre varias y hago hincapié en esto porque cuando tiene (como erradamente dicen algunos) una sola alternativa pues sencillamente no tiene alternativa y no hay caminos para escoger. La decisión se da siempre que tenga que escoger de entre, por lo menos, dos caminos lógicos.
Ejemplo: Usted ha llegado a la parada de la guagua, ve como pasan y pasan pero ninguna tiene la ruta que necesita. Por cada guagua que pasa usted le mira la ruta y al ver que no es, espera la siguiente y así sucesivamente hasta que ve llegar la que necesita.
Siempre que tengamos que utilizar la estructura de Decisiones vamos a depender de una condición. La condición es la que nos permite que podamos decidir cuál es el camino lógico correcto a tomar.

Estas estructuras seleccionan de acuerdo con una condición, una sola de las acciones que forman parte de la estructura alternativa o ramificada.
Hay estructuras alternativas simples, dobles y múltiples en dependencia de la cantidad de posibles acciones a seleccionar.
A partir un ejemplo veamos lo que se entiende por esquema condicional o estructura alternativa.
Consideremos el siguiente enunciado:

```
a; Calcular el salario de un empleado
Si el empleado es ingeniero  entonces añadir la prima de técnico
                p                              b
c: añadir la prima de antigüedad
```

La acción añadir la prima de técnico , sólo se ejecuta cuando la condición expresada por medio del predicado el empleado es ingeniero , es cierto.
El enunciado puede expresarse así:

```
a:  si p entonces b; c.
```

En el caso que el empleado es ingeniero las acciones a,b,c serán ejecutadas. En caso contrario, solo lo serán a y c
Esta estructura condicional es simple pues si la condición se cumple entonces se ejecutan las acciones, en caso contrario se sigue la secuencia normal.

Estructura condicional simple

Tiene la forma siguiente:

Si condición **entonces**
acciones
Finsi

El delimitador **Finsi** indica el final de la secuencia de acciones.
En esta estructura si la condición es verdadera, entonces se ejecutarán las acciones correspondientes si no lo es, entonces se sigue la secuencia normal del algoritmo.
Modifiquemos en el enunciado anterior , la frase condicional reemplazándola por:

a: Calcular el salario de un empleado
Si el empleado es ingeniero **entonces** añadir la prima de técnico
p b
sino pagar las horas extras
b'
c: añadir la prima de antigüedad

La secuencia de acciones es ahora la siguiente:

a; si p entonces b sino b'; c.

Si el empleado es ingeniero, las acciones a, b y c serán ejecutadas. En el caso contrario se ejecutará a, b' y c.
Esta estructura condicional es doble porque si se cumple la condición se ejecutan unas acciones, si no se ejecutan otras y después se sigue la secuencia normal.

Estructura condicional doble

La estructura condicional doble selecciona entre dos posibles alternativas y se expresa así:

Si condición **entonces**
acciones
sino
acciones
Finsi

Las acciones a ejecutar en caso de que la condición sea Verdadera estarán escritas entre la formulación de la condición y la palabra clave *Sino.* En caso que la condición sea falsa se ejecutan las acciones comprendidas entre el **sino** y el **Finsi.**

Estructura condicional múltiple

Esta estructura selecciona entre varias alternativas (más de dos) y se expresa de la siguiente forma:

Si condición **entonces**
acciones
sinosi condición **entonces**
acciones

sinosi condición **entonces**
acciones
sinosi condición **entonces**
acciones

Sino
acciones

Finsi

Para una secuencia de acciones, que comprende una estructura condicional, una acción es exterior a la estructura si aparece antes del **si** o después del **Finsi**.

Estructura caso o de selección

Esta estructura es similar a la condicional múltiple; pero más clara en su implementación.

Seleccionar Caso expresión
Caso lista de expresiones
acciones
Caso lista de expresiones
acciones

Caso sino
acciones
Finseleccionar

Expresión: cualquier expresión numérica o de cadena que de cómo resultado un número o una cadena.

Lista de expresiones: lista delimitada por comas de una o más de las formas siguientes: expresión, expresión **a** expresión.

Esta estructura permite abreviar una serie de decisiones en cascada o en secuencia. Facilita la toma de decisiones por parte del procesador. La estructura casos toma el contenido de la expresión y lo evalúa acorde con unos posibles valores ejecutando lo que se le indique en cada una de las opciones.

Ejercicios resueltos con las estructuras condicionales

1. Se quiere hacer un algoritmo para controlar el funcionamiento de un robot que controla el regadío en un campo de legumbres. Su función es regar en dependencia de la humedad relativa. Se activa cuando la humedad relativa es menor del 85 %.

Solución:

```
Algoritmo Regar
Inicio
        Entorno:
        Dispositivo para medir la humedad relativa
        Leer la humedad relativa
        Si humedad relativa <85 % entonces
                Regar
        FinSi
Fin
```

2. Hacer un algoritmo para conocer la acidez de una solución mediante un papel de tornasol. Se sabe que al introducir este papel en una solución, si toma color rojo la solución es ácida; si es azul la solución es alcalina y si toma cualquier otro color la solución es neutral.

Solución:

```
Algoritmo Acidez
Inicio
        Entorno:
        Una probeta
        Solución
        Papel de tornasol

        Verter la solución en la probeta
        Introducir el papel de tornasol en la probeta
        Si el papel toma color rojo entonces
                Escribir "la solución es ácida"
                SinoSi el papel toma color azul entonces
                Escribir "la solución es alcalina"
          Sino
               Escribir "la solución es neutral"
        FinSi
Fin
```

3. Construya el algoritmo para efectuar una llamada telefónica en un teléfono público.

Solución:

```
Algoritmo Llamada
Inicio
        Entorno:
        Teléfono
        Monedas
        Descolgar el auricular
        Llevar el auricular hasta la oreja
        Si tiene tono entonces
                Esperar que pida insertar moneda
                Insertar moneda
                Esperar que diga que puede marcar número
                Marcar el número
                Esperar respuesta
                Saludar
                Conversar
                Despedirse
                Colgar
                Si hay monedas sobrantes entonces
                        Recogerlas
                FinSi
        FinSi
Fin
```

4. Si A, B, C son tres variables numéricas de valores enteros distintos elabore un algoritmo para escribir cual de las tres variables tiene el valor más elevado. Haga el trace de este algoritmo.

Solución:

```
Algoritmo Mayor
Inicio
Entorno: A, B, C variables numéricas
  Leer A
  Leer  B
  Leer C
  Si  A>B y A>C entonces
        Escribir "A contiene el valor mayor"
         SinoSi B>A y B>C entonces
       Escribir "B contiene el valor mayor"
         SinoSi C>A y C>B entonces
       Escribir "C contiene el valor mayor"

  FinSi
```

Fin

Trace de este algoritmo (para A=3, B=5, C=8)

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmo Mayor	
		Inicio	
A	valor indeterminado	A=3	3
B	valor indeterminado	B=5	5
C	valor indeterminado	C=8	8
		¿3>5 y 3>8?	Falso
		¿5>3 y 5>8?	Falso
		¿8>3 y 8>5?	Cierto
		C contiene el valor mayor	
		Fin	

5. Elabore un algoritmo para imprimir el valor de un número cuando éste sea mayor que 3 y menor o igual a 10. Si el número no está en ese rango escribir el mensaje “número fuera de rango”. Haga el trace para este algoritmo.

Solución:

```
Algoritmo N
Inicio
  Entorno:
  N variable numérica

  Leer N
  Si  N>3 y N<=10 entonces
        Escribir N
     Sino
        Escribir  “Número fuera de rango”
  FinSi
Fin
```

Trace de este algoritmo para N=2

Objetos	**Estado inicial**	**Acciones**	**Estado final**
		Algoritmo N	
		Inicio	
N	valor indeterminado	N=2	2
		¿2>3 y 2<=10?	Falso
		“Número fuera de rango”	
		Fin	

6. En una oficina de contabilidad cada día de la semana se hace una actividad diferente de la siguiente manera:

lunes	facturación
martes	recepción
miércoles	pedidos
jueves	contabilidad
viernes	proveedores

Haga un algoritmo para dado el día de la semana escribir la actividad correspondiente. Si lee un día que no tenga actividad, escribir el mensaje “error”

Solución:

```
Algoritmo DiaDeSemana
Inicio
    Entorno:
    Día variable de cadena para leer el día de la semana
     Leer Día
Seleccionar caso Día
  Caso “lunes”
      Escribir “Facturación”
   Caso “martes”
       Escribir recepción”
    Caso ”miércoles”
        Escribir “Pedidos”
     Caso ”jueves”
         Escribir “contabilidad”
      Caso “viernes”
          Escribir “Proveedores”
       Caso Sino
      Escribir “Error”
  FinSelecionar
Fin
```

7. Haga un algoritmo para dados los nombres de dos personas escribirlas en orden alfabético.

Solución:

```
Algoritmo OrdenAlfabético
Inicio
Entorno:
    Nombre1: variable de cadena para guardar el primer nombre
    Nombre2: variable de cadena para guardar el segundo nombre
    Leer Nombre1
    Leer nombre2
    Si Nombre1< Nombre2 entonces
            Escribir Nombre1
            Escribir Nombre2
            SinoSi Nombre1 > Nombre2 entonces
                    Escribir Nombre2
                    Escribir Nombre1
      Sino
            Escribir Nombre1
            Escribir Nombre2
    Finsi

Fin
```

O Escribir Nombre2
Escribir Nombre1 porque serían iguales

8. Construya un algoritmo que dados vía parámetro, tres valores, correspondientes a las longitudes de los tres lados de un triángulo, escriba qué tipo de triángulo es según sus lados. Se sabe que:

El triángulo que tiene los tres lados iguales se llama equilátero. (es isósceles también)
El triángulo que tiene los tres lados desiguales se llama escaleno.
El triángulo que tiene dos lados iguales se llama isósceles.
Clasifique los tipos de parámetros utilizados. (parámetro-dato, parámetro-resultado, parámetro-dato-resultado).

Solución:

```
Algoritmo Entrada
Inicio
    Entorno:
    A, B, C: variables numéricas para asignarle los valores de los lados del triángulo (parámetro-dato)
    Leer A
    Leer B
    Leer C
    Triángulo (A, B, C)
Fin

Algoritmo Triángulo(A,B,C)
Inicio
    Si A=B y B=C entonces
            Escribir "Equilátero"
            SinoSi A<>B y B<>C y A<>C entonces
                    Escribir "Escaleno"
      Sino
            Escribir "isósceles
       Finsi

Fin
```

9. Leer un número entero de 3 dígitos y determinar si todos sus dígitos son pares.

Solución:

```
Algoritmo DígitosPares
        Inicio
        Entorno: n, d1, d2, d3 variables numéricas
        Leer n
        d3←n-(n\10)*10
        n←n\10
        d2←n-(n\10)*10
        n←n\10
        d1←n-(n\10)*10
        Si d1\2=d1/2 y d2\2=d2/2 y d3\2=d3/2 entonces
                Escribir "Todos sus dígitos son pares"
                Sino
                        Escribir "Todos sus dígitos no son pares"
        FinSi


Fin
```

10. Leer un número entero de 3 dígitos y determinar si el primer dígito es múltiplo del último.

Solución:

```
Algoritmo Múltiplo
    Inicio
    Entorno: n, d1, d2, d3 variables numéricas
    Leer n
    d3←n-(n\10)*10
    n←n\10
    d2←n-(n\10)*10
    n←n\10
    d1←n-(n\10)*10
    Si (d1\d3)*d3=d1 entonces
        Escribir "El primer dígito es múltiplo del último"
        Sino
            Escribir "El primer dígito no es múltiplo del último"
    FinSi

Fin
```

Resolver los ejercicios con estructuras condicionales de los propuestos en el Capítulo II.

No se ate solo a la estructura Si-Entonces-Sino pues también tenga en cuenta que la estructura casos le va a permitir simplificar más de un conjunto de decisiones y hacer que su algoritmo sea mucho más entendible y fácil de codificar.

Siempre evalúe bien las condiciones de las decisiones y realícele el trace a las decisiones tal y como se ha recomendado en todo este libro, es decir, sin suponer absolutamente nada y sin saltar ningún paso debido a que se puede encontrar, más de una vez, con que saltar una instrucción que involucre una decisión va a significar un cambio realmente profundo en el desarrollo del algoritmo y en su posterior ejecución.

Revise que las decisiones que aparezcan en su algoritmo realmente sean necesarias y omita todas aquellas que, por razones estrictamente lógicas, estén sobrando.

Las decisiones son el único camino que tenemos en la programación para escoger uno

de entre dos ramales lógicos por eso es una de las herramientas que debemos utilizar de manera eficiente pues su excesivo uso, tal como ya se expuso, representa ineficiencia en la construcción del algoritmo. Entre menos decisiones tenga un algoritmo más eficiente será la ejecución de su correspondiente programa.

Estructuras repetitivas.

Esta estructura nos permite repetir una o varias acciones una cantidad definida de veces.

Se llama iteración al hecho de repetir una secuencia de acciones o de una acción.

Un ciclo puede definirse como una estructura que nos permite repetir o iterar un conjunto de instrucciones y que tiene las siguientes características:

a. El conjunto de instrucciones debe ser finito
b. La cantidad de veces que se repita dicho conjunto de instrucciones también debe ser finita. En algunos casos esta cantidad de veces va a depender de una condición explícita y en otros casos va a depender de una condición implícita. Una condición es explícita cuando depende solamente de la misma ejecución del programa sin que sea importante la participación del usuario. Asimismo una condición es implícita cuando depende solamente de la voluntad del usuario y por lo tanto la cantidad de iteraciones o repeticiones del ciclo podría llegar a ser diferente cada vez pues sería posible que cambiara con cada usuario.
c. Deben estar claramente demarcados el inicio y el fin del ciclo.
d. Dentro de un ciclo podrá ir cualquiera de las otras estructuras que se han estudiado incluyendo otros ciclos.

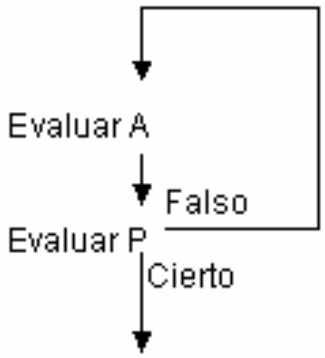
Veamos los esquemas iterativos :

Esquema iterativo Repetir hasta que:

Repetir
    Acciones
Hasta que

Este esquema iterativo repite las acciones hasta que se cumpla una determinada condición o predicado de control. El predicado se evalúa después de cada ejecución de las acciones. Por tanto las acciones son ejecutadas al menos una vez. En este tipo de esquema, el número de repeticiones no es conocido con antelación.

Esquema repetir en forma gráfica.
A: Acciones.
P: Predicado de control o condición



Ejemplo:
Se quiere hacer un algoritmo que permita escribir un número indeterminado de palabras.
Solución:

```
Algoritmo EscribirPalabras
Inicio
Entorno:
Pal variable de cadena
    Repetir
        Leer Pal
        Escribir     Pal
    Hasta que Pal =”fin”
Fin
```

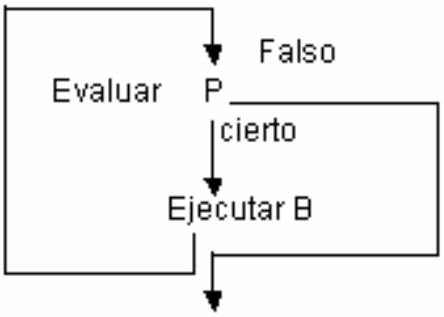
En este algoritmo se leen y se escriben palabras hasta que se lea la palabra “fin” que también será escrita antes de terminar la iteración.
El esquema Repetir no permite describir fácilmente todos los casos de iteración por ello se hace necesario este otro esquema iterativo:

Esquema iterativo Mientras:

```
Mientras predicado de control
    Acciones
FinMientras
```

Este esquema iterativo repite las acciones mientras se cumpla el predicado de control o condición. El predicado se evalúa antes de realizar las acciones, por tanto puede ocurrir que las acciones no se ejecuten. En este esquema también el número de repeticiones es desconocido.

Esquema mientras en forma gráfica:
B: acciones
P: predicado de control o condición



Ejemplo:
Se quiere escribir un número mientras sea menor que 3.

Solución:

```
Algoritmo ImprimirNúmero
Inicio
    Entorno: N variable numérica
    N ← 1
            Mientras  N<3
       Escribir N
       N←N+1
    FinMientras
Fin
```

En este algoritmo el número n se imprime dos veces (1,2), cuando toma el valor 3 termina la iteración.

Esquema iterativo Para

```
Para  V de Vi a Vf incremento Inc
    Acciones
FinPara
```

V: variable de control
Vi: primer valor de la variable.
Vf: valor final de la variable.
Inc: valor positivo o negativo del incremento
Incremento: cantidad en la que cambia V cada vez que se ejecuta el ciclo. Si no se especifica, el valor predeterminado de incremento es uno. Puede ser positivo o negativo.
El valor del argumento incremento determina la manera en que se procesa el ciclo: si el incremento es positivo o 0 V debe ser <= Vf. Si el incremento es negativo, V debe ser >=Vf
Una vez que se inicia el ciclo y se han ejecutado todas las instrucciones en la iteración, incremento se suma a V.

Esta estructura repite las acciones hasta que V=Vf .

Ejemplo:
Escribir 3 veces la palabra “cuba”

```
Algoritmo imprimir
Inicio
Entorno: Vi, Vf variables numéricas
Vi ← 1
Vf ← 3
    Para  V de 1 a 3
      Escribir  “cuba”
    Fin Para

Fin
```

Resultado:
cuba
cuba
cuba

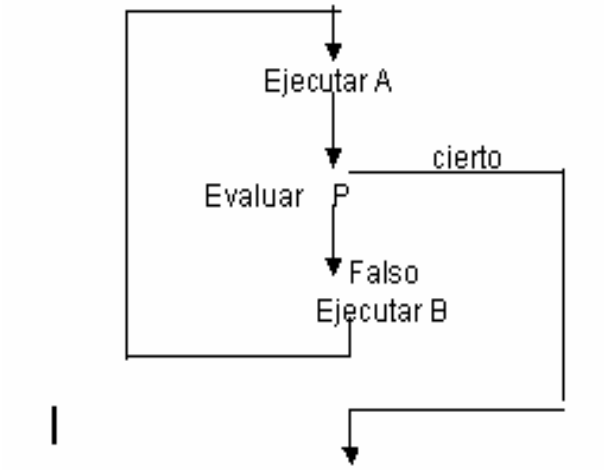
Esquema iterativo general

```
Iterar
    Acciones A
    SalirSi predicado de control o condición
    Acciones B
```

FinIterar

En este esquema iterativo las acciones se repiten hasta que se cumpla la condición expresada en SalirSi.

Esquema general en forma gráfica
A y B son acciones
P predicado de control (condición)



En este esquema iterativo se ejecutan las acciones A, luego se evalúa la condición, si es cierta termina la iteración y si es falsa se ejecutan las acciones B. Siempre las acciones A se ejecutan.
Ejemplo:
Veamos el mismo ejemplo que utilizamos en el esquema Repetir:
Se quiere hacer un algoritmo que permita escribir un número indeterminado de palabras.

Solución:

```
Algoritmo EscribirPalabras
Inicio
  Entorno:
     Pal: variable de cadena
     Iterar
            Leer Pal
            SalirSi Pal =”fin”
            EscribirPal
     FinIterar
Fin
```

Este algoritmo imprime la palabra leída solamente si es diferente de “fin” y no imprime esta última palabra.
Los esquemas Mientras y Repetir son casos particulares del esquema general.
Cuando A es una acción vacía, el esquema general es equivalente a el esquema Mientras. FinMientras.

Cuando B es una acción vacía el esquema general equivale al esquema Repetir Hasta que.

Ejercicios resueltos con estructuras repetitivas:

1. En una embotelladora de refrescos se quiere utilizar un brazo mecánico para coger una botella de un depósito, ponerla en un embudo, accionar un botón para llenarla, cuando esté llena retirarla del embudo, ponerla en la enchapadora, accionar un botón para enchaparla y luego ponerla en una estera transportadora.

Elabore un algoritmo para que el brazo realice este trabajo.

Solución:

```
Algoritmo Embotelladora
Inicio
    Repetir
            Llevar el brazo junto al depósito de botellas vacías
            Mover el brazo hasta el embudo
            Poner la botella en el embudo
            Accionar el botón para llenarla
            Cuando la botella está llena accionar el botón para dejar de llenar
            Sacar la botella del embudo
            Mover el brazo hasta la enchapadora
            Accionar el botón para ponerle la chapa
            Cuando tenga la chapa accionar el botón para detener la enchapadora.
            Mover el brazo hasta la estera.
            Poner la botella en la estera
   Hasta que no quede ninguna botella en el depósito
Fin
```

2. Hacer un algoritmo para calcular la suma de los N primeros números enteros. Haga el seguimiento del algoritmo (trace).

Solución:

```
Algoritmo SumaEnteros
Inicio
    Entorno:
    S: variable numérica para guardar la suma de los enteros.
    N: variable numérica para guardar el valor del entero
    N← 1
    S← 0
    Repetir
        S← S+N
        N← N+1
     Hasta que  N= 5
Fin
```

Trace de este algoritmo

Objetos	Estado inicial	Acciones	Estado final
		Algoritmo SumaEnteros	
		Inicio	
N	Valor indeterminado	N=1	1
S	Valor indeterminado	S=0	0
S	0	S=0+1	1
N	1	N=1+1	2
		¿2=5?	Falso
		Repetir	
S	1	S=1+2	3
N	2	N=2+1	3
		¿3=5?	Falso
		repetir	
S	3	S=3+3	6
N	3	N=3+1	4
		¿4=5?	falso
		Repetir	
S	6	S=6+4	10
N	4	N=4+1	5
		¿5=5?	cierto
		Fin	

3. Sobre un escritorio se tiene una pila de 10 paquetes de hojas fotocopiadas, cada paquete está compuesto de un centenar de hojas que corresponden a la misma página. Los paquetes no están dispuestos en la pila por orden de página, se quieren agrupar las fotocopias por un individuo (ordenador) que dispone de los objetos siguientes (entorno) para efectuar la tarea:

a) 10 paquetes de hojas.
b) Un escritorio donde están apiladas, paquete por paquete, las hojas que se han de ordenar.
c) Una precilladora.
d) 10 mesas numeradas del 1al 10. La mesa con el número 1 deberá recibir las hojas cuyo número de página sea 1, la mesa número 2 recibirá las hojas cuyo número de página sea 2 y así sucesivamente hasta la mesa número 10.

Solución:

Algoritmo Agrupar
Inicio
    Repetir
        A: Coger el paquete situado encima de la pila sobre el escritorio.
        B: Ir a la mesa cuyo número es el mismo que el de las hojas del paquete.
        C: Poner el paquete sobre la mesa.
    Hasta que se acabe la pila que hay sobre el escritorio

    Repetir
        D: colocarse en la mesa Nº 1
        E: coger una hoja de la mesa Nº 1
            Repetir
                F: ir a la mesa siguiente
                G: coger una hoja de esta mesa y ponerla en la mano debajo de las otras.
            Hasta que la mesa sea la Nº 10.
        H: precillar el ejemplar
        I: ponerlo sobre el escritorio.
    Hasta que una de las mesas quede sin hojas.

Fin

4. Se quieren repartir caramelos a cuatro niños. Estos caramelos son de tres colores diferentes, rojo, verde y azul y están en una bolsa opaca. Se quiere dar a cada niño el mismo número de caramelos de cada color. Para ello se preparan tres cajas de colores (rojo, verde y azul). Antes de la repartición se pone cada caramelo en la caja del color que corresponde al suyo. Elaborar el algoritmo correspondiente al llenado de las cajas y a la distribución de los caramelos.

Solución:
Entorno:
Una bolsa opaca.
Cierta cantidad de caramelos rojos, verdes y azules (no menos de cuatro caramelos de cada color).
Tres cajas (una roja, una verde y otra azul).
Cuatro niños.

Algoritmo Repartir
Inicio
Repetir
A: coger un caramelo de la bolsa.
B: poner el caramelo en la caja que coincida con el color del caramelo.
Hasta que la bolsa esté vacía.

Repetir
C: coger 1 caramelo de cada caja
D: darle los caramelos al primer niño.
E: coger 1 caramelo de cada caja
F: darle los caramelos al segundo niño.
G: coger 1 caramelo de cada caja
H: darle los caramelos al tercer niño.
I: coger 1 caramelo de cada caja
J: darle los caramelos al cuarto niño.
Hasta que una de las cajas tenga menos de 4 caramelos.

Fin

5. En una fábrica de juguetes se hacen carritos que se mueven (con velocidad constante) por control remoto. El motor acoplado a las ruedas puede rotar en un sentido y en el otro y tienen un mecanismo de dirección que permite girar las ruedas hacia la derecha y hacia la izquierda. Se quiere programar un mando que controle el movimiento de las ruedas de dichos juguetes. El mando debe tener botones que realicen las siguientes acciones: "Mover adelante", "Mover atrás", "Girar derecha", "Girar izquierda", "Ruedas sin giro", "Parar".

Solución:
Algoritmo Mando
Inicio
Entorno:
Mando
MoverAlante botón que al pulsarse indica que el movimiento será hacia delante.
MoverAtrás botón que al pulsarse indica que el movimiento será hacia atrás.
GirarDercha botón que al pulsarse indica que el giro será hacia la derecha.
GirarIzquierda botón que al pulsarse indica que el giro será hacia la izquierda
RuedasSinGiro botón que al pulsarse indica que las ruedas se mantendrán sin giro.
Parar botón para detener el movimiento de las ruedas.

```
Mientras (MoverAlante esté pulsado o MoverAtrás esté pulsado) y Parar no
    esté pulsado
        Si MoverAlante está pulsado entonces
            Girar motor hacia delante
            SinoSi   MoverAtrás está pulsado entonces
            Girar motor hacia atrás
        FinSi

        Si GirarDerecha está pulsado entonces
            Girar ruedas hacia la derecha
            SinoSi GirarIzquierda está pulsado entonces
                Girar ruedas hacia la izquierda
                Si RuedaSinGiro está pulsado entonces
                Mantener ruedas derechas
        FinSi
    FinMientras
Fin
```

6. Elabore un algoritmo para con ayuda de una balanza de dos platillos, encontrar una moneda falsa entre varias monedas, conociendo que todas las monedas verdaderas pesan lo mismo y la falsa un poco menos.

Solución:

```
Algoritmo Monedas
Inicio
        Entorno:
        Balanza de dos platillos
        Varias monedas verdaderas
        Una moneda falsa
    Repetir
        Poner una moneda en el platillo izquierdo
        Poner una moneda en el platillo derecho
        Esperar que se estabilice la balanza
        Si balanza equilibrada entonces
                Retirar las monedas de ambos platillos
        Finsi
    Hasta que la balanza esté desequilibrada
    Escribir  " La moneda falsa se encuentra en el platillo más alto"
Fin
```

7. Diseñe un algoritmo para programar un robot, que de acuerdo con el turno de remisión que le entregue un paciente, lo envíe a la consulta del especialista correspondiente y además contabilice la cantidad de pacientes que asistieron a cada consulta. Considere que el robot está activo hasta que llegue al paciente 200. Las especialidades posibles son ortopedia, piel y cardiología.

Solución:

```
Algoritmo Remisión
Inicio
    Entorno:
    TurnoRemisión: variable de cadena para guardar los turnos de remisión.
    CuentaOrtopedia: variable numérica para contar la cantidad de pacientes que
    asistieron a ortopedia.
    CuentaOrtopedia← 0
    CuentaPiel: variable numérica para contar la cantidad de pacientes que  asistieron a
    piel.
    CuentaPiel ← 0
    Cuentacardiología: variable numérica para contar la cantidad de pacientes que
    asistieron a cardiología.
    Cuentacardiología ← 0
    SumaPacientes: variable numérica para sumar la cantidad total de pacientes que
    asistieron a todas las consultas.
    SumaPacientes← 0
      Mientras SumaPacientes < 200 entonces
                Leer TurnoRemisión
                Si TurnoRemisión=Ortopedia entonces
                        Enviar a Ortopedia
                        CuentaOrtopedia← CuentaOrtopedia+1
                        SinoSi TurnoRemisión=CuentaPiel entonces
                                Enviar a Piel
                                CuentaPiel ← CuentaPiel+1
                    Sino
                    Enviar a cardiología
                    CuentaCardiología ←  CuentaCardiología+1
                Finsi
            SumaPacientes← CuentaOrtopedia+CuentaPiel+CuentaCardiología
       FinMientras
       Escribir “Asistieron a Ortopedia”; CuentaOrtopedia
       Escribir “Asistieron a piel”; CuentaPiel
       Escribir “Asistieron a Cardiología”; CuentaCardiología
Fin
```

8. Haga un algoritmo para hallar el duplo de la suma de 20 números.

Solución:

```
Algoritmo Duplo
Inicio
    Entorno:
    K: variable numérica para controlar la iteración
    N: variable numérica para leer un número
    S: variable numérica para guardar la suma de los 20 números.
    D: variable numérica para guardar el duplo de S
    N←0
    S← 0
    D← 0
    Para k de 1 a 20
            Leer n
            S←S+N
    Fin para
    D← 2*S
    Escribir D
Fin
```

9. Haga un algoritmo para hallar la producción total de calzado del año conociendo las producciones mensuales. Además indicar la producción promedio mensual y si se cumplió el plan del año

Solución:

```
Algoritmo ProducciónCalzado
Inicio
Entorno:
    T: variable numérica para guardar la producción total.
    Pro: variable numérica para leer la producción del mes.
    PromedioMensual: variable numérica para guardar el promedio mensual
    Plan: variable numérica para guardar el plan.
    V: variable control del ciclo.
    T← 0

    Leer Plan
    Para V de 1 a 12
            Leer Pro
            T←T+Pro
    FinPara
    PromedioMensual← T/12
    Escribir PromedioMensual
    Si T>=Plan entonces
            Escribir “Se cumplió el plan”
            Sino
```

```
            Escribir "No se cumplió"
        Finsi
    Fin
```

10. Haga un algoritmo para dadas las notas de un alumno en 5, 4, 3 ,2 convertirlas en E, B, R, M respectivamente.

Solución:

```
Algoritmo CovertirNota
Inicio
    Entorno:
    Nota: variable numérica para leer las notas
    NotaCualitativa: variable para guardar la nota cualitativa
    Repetir
            Leer Nota
            Si Nota=2 entonces
                    Notacualitativa← "M"
                    SinoSi Nota=3 entonces
                            Notacualitativa← "R"
                            SinoSi Nota=4 entonces
                                            Notacualitativa← "B"
                                    SinoSi Nota=5 entonces
                                            Notacualitativa← "E"
            Finsi
       Hasta que Nota<2 y Nota>5
Fin
```

11. Haga un algoritmo que permita dadas las notas de 20 estudiantes conocer si todos aprobaron o no. Las notas se dan en 2, 3, 4 , 5 (2 es desaprobado)

Solución:

```
 Algoritmo NotaEstudiante
Inicio
    Entorno:
    Nota: variable numérica para leer la nota del alumno, i variable numérica
    Suspenso: variable lógica para conocer si hay suspensos o no
    Para i de 1 a 20
            Leer Nota
            Si Nota=2 entonces
                    Suspenso = cierto
            Finsi
    Fin Para
    Si Suspenso = Cierto entonces
            Escribir "No aprobaron todos"
            Sino
               Escribir "Aprobaron todos"
    Finsi
Fin
```

12. Haga un algoritmo para escribir los múltiplos de 3 entre 1 y 20.

Solución:

```
Algoritmo MúltiplosDeTres
Inicio
Entorno:
    V: variable numérica para controlar el ciclo.
    Para V de 1 a 20
            Si V/3=V\3 entonces
                    Escribir V
            Finsi
    FinPara
Fin
```

Si la división con resultado decimal (/) de un número es igual a la división entera(\) de dicho número, entonces no hay resto, por lo tanto el número es divisible por 3 o múltiplo de 3.

Otra solución para este ejercicio

```
Algoritmo MúltiplosDeTres
Inicio
Entorno:
    V: variable numérica para controlar el ciclo.
    Para V de 3 a 20 Incremento 3
            Escribir V
    FinPara
Fin
```

13. Elabore el algoritmo que debe contener el programa de un robot que sea capaz de cruzar las calles sin ser arrollado.

Solución:

```
Algoritmo CruzarCalle
Inicio
        Si la calle es de sentido derecha entonces
                Mirar hacia la izquierda
                Si carro en movimiento a menos de 30 m entonces
                        Esperar que pase
                        Sino
                          Cruzar
                Finsi
        Finsi
        Si la calle es de sentido izquierda entonces
                Mirar  hacia la derecha
                Si carro en movimiento a menos de 30 m entonces
                        Esperar que pase
                        Sino
```

```
                    Cruzar
            Finsi
    Finsi


    Si la calle es de doble sentido entonces
            Repetir
                    Mirar hacia la izquierda
                    Si carro en movimiento a menos de 30 m entonces
                            Esperar que pase
                    Finsi
                    Mirar hacia la derecha
                    Si carro en movimiento a menos de 30 m entonces
                            Esperar que pase
                    Finsi
            Hasta que no se acerque ningún carro por la izquierda ni por la
            derecha.
       Cruzar
    Finsi
Fin
```

Se quiere programar un procesador que permita a un robot humanoide subir y bajar una escalera. Declare el entorno y elabore un algoritmo para este procesador.

Solución:

```
Algoritmo RobotEscalera
Inicio
    Entorno:
    Escalera
    Robot
    Ponerse junto al primer escalón
    Levantar el pie izquierdo
    Bajar el pie y ponerlo en el primer escalón
    Repetir
        Levantar el pie derecho
        Bajar el pie y ponerlo en el siguiente escalón
        Levantar el pie izquierdo
        Bajar el pie y ponerlo en el siguiente escalón
    Hasta que no quede ningún escalón
    Dar media vuelta
    Repetir
        Levantar pie izquierdo
        Bajar el pie y ponerlo en el siguiente escalón
        Levantar el pie derecho
        Bajar el pie y ponerlo en el siguiente escalón
    Hasta que no quede ningún escalón

Fin
```

14. Haga un algoritmo para escribir los cuadrados de los números pares que hay entre 1 y 50

Solución:

```
Algoritmo Cuadrados
Inicio
Entorno:
N: variable numérica para controlar el ciclo
Para N de 2 a 50 incremento 2
    Escribir N^2
FinPara
```

15. Hacer un algoritmo para que de 20 pares de números leídos escriba aquellos cuya diferencia sea menor que 5.

Solución:

```
Algoritmo Diferencia
Inicio
    Entorno:
```

```
N1, N2: variables numéricas para leer dos números
V: variable numérica para controlar el ciclo
Para V de 1 a 20
        Leer N1
        Leer N2
        Si (N1-N2) < 5 entonces
                Escribir N1
                Escribir N2
        Finsi
FinPara
Fin
```

16. Una caja de madera está llena de pelotas de diferentes tamaños (diámetro). Se quieren sacar las pelotas de esta caja. Las pelotas de 10 cm se pondrán en una caja de cartón que se ha marcado con la inscripción “10 cm” y las de 20 cm se pondrán en otra caja de cartón cuya inscripción dice “20 cm”. Las pelotas cuyos diámetros sean distintos a 10 ó 20 cm se pondrán en una bolsa plástica.

Declare el entorno y elabore un algoritmo para que un procesador haga este trabajo.

Solución:

```
Algoritmo Pelotas
Inicio
    Entorno:
    Una caja de madera
    Pelotas
    Dos cajas de cartón
    Una bolsa plástica
    Instrumento para medir diámetro
    Repetir
            Coger una pelota de la caja de madera
            Medir su diámetro
            Si diámetro = 10 cm entonces
                    Poner la pelota en la caja de 10 cm
                    SinoSi diámetro = 20 cm entonces
                            Poner la pelota en la caja de 20 cm
               Sino
                    Poner la pelota en la bolsa plástica
            Finsi
    Hasta que no queden pelotas en la caja de madera
Fin
```

17. En una fábrica se producen caramelos y se clasifica según su peso de la forma siguiente: los que pesan hasta 5 g son de baja calidad; los que pesan más de 5 g y menos de 7 g son de buena calidad; los que pesan desde 7g hasta 10 g son de alta calidad. Los de baja calidad que pesan menos de 2 g, se clasifican como desechables y el resto de éstos como aprovechables.

Se necesita:

a) Contar la cantidad de caramelos de cada clasificación.
b) Conocer la cantidad total de caramelos producidos.
c) Envasar los caramelos en cajas según su calidad.

Elabore los algoritmos parametrizados necesarios para dar solución al problema planteado. Clasifique cada tipo de parámetro. (parámetro-dato, parámetro-resultado, parámetro-dato-resultado).

Solución:

```
Algoritmo FábricaCaramelo
Inicio
    Entorno:
    PesoCaramelo: variable numérica para asignarle el valor del peso del
    caramelo.(parámetro-dato)
    CantidadTotal: variable numérica para sumar la producción total de caramelos.
    CantidadTotal← 0
    CantidadBajaCalidad. Variable numérica para contar los de baja calidad.(parámetro-
    dato-resultado)
    CantidadBajaCalidad← 0
    CantidadDesechables: variable numérica para contar los desechables. ( parámetro-
    dato-resultado)
    CantidadDesechables← 0
    CantidadAprovechables: variable numérica para contar los aprovechables.(parámetro-
    dato-resultado)
    CantidadAprovechables← 0

    CantidadBuenaCalidad: variable numérica para contar los de buena calidad.(
    parámetro-dato-resultado)
    CantidadBuenaCalidad← 0
    CantidadAltaCalidad: variable numérica para contar los de alta calidad.( parámetro-
    dato-resultado)
    CantidadAltaCalidad ← 0
    Repetir
        Leer PesoCaramelo
        Si PesoCaramelo<= 5 g entonces
            CaramelosDeBajaCalidad (PesoCaramelo, CantidadBajaCalidad,
            CantidadDesechables, CantidadAprovechables)
            SinoSi PesoCaramelo>5 g y PesoCaramelo<7 g entonces
                CaramelosBuenaCalidad(CantidadBuenaCalidad)
                Sinosi PesoCaramelo>=7g y PesoCaramelo<=10 g entonces
                    CaramelosDealtaCalidad(CantidadAltaCalidad)
        Finsi
    Hasta que pare la producción
    CantidadTotal← CantidadBajaCalidad+CantidadBuenaCalidad+CantidadAltaCalidad
    Escribir CantidadBajaCalidad
```

```
            Escribir CantidadDesechables
            Escribir CantidadAprovechables
            Escribir CantidadBuenaCalidad
            Escribir CantidadAltaCalidad
        Fin

        Algoritmo CaramelosDeBajaCalidad (PBaC, CBaC,CD,CA)
        Inicio
            CBaC ← CBaC +1
            Si PBaC < 2 g entonces
                    CD ← CD +1
                    Ponerlo en la caja de desechables
                    SinoSi PBaC >=2 g y PBaC <=5 entonces
                            CA ← CA +1
                            Ponerlo en la caja de aprovechables
            Finsi
        Fin
    Algoritmo CaramelosBuenaCalidad( CbuC)
    Inicio
            CbuC← Cbuc+1
            Ponerlo en la caja de buena calidad
    Fin
    Algoritmo CaramelosAltaCalidad( CAC)
    Inicio
            CAC ← CAC +1
            Ponerlo en la caja de alta calidad
    Fin
```

18. Se tienen 5000 tornillos de diferentes longitudes, en una caja y se necesita:
Saber cuántos hay de 3 pulgadas, cuántos de 5 pulgadas y cuántos de 7 pulgadas.
Saber cuántos tornillos se clasificaron y cuántos quedaron sin clasificar.
Elabore los algoritmos para dar solución a este problema, utilizando la transferencia de parámetros. Clasifique cada tipo de parámetro. (parámetro-dato, parámetro-resultado, parámetro-dato-resultado)

```
Algoritmo Tornillo
    Inicio
    Entorno:
    Longitud: variable numérica para asignarle el valor de la longitud del tornillo.(parámetro-dato)
    TotalClasificados: Variable numérica para asignarle el total de tornillos clasificados.
    TotalClasificados← 0
    DeTresPulgadas: variable numérica para contar los de 3 pulgadas. (parámetro-dato-resultado)
    DeTresPulgadas← 0
    DeCincoPulgadas: variable numérica para contar los de 5 pulgadas. (parámetro-dato-resultado)
    DeCincoPulgadas← 0
```

```
        DeCietePulgadas: variable numérica para contar los de 7 pulgadas. (parámetro-dato-
        resultado)
        DeCietePulgadas← 0
        Para T de 1 a 5000
                Coger un tornillo
                Leer Longitud
                Clasificar(Longitud, DeTresPulgadas, DeCincoPulgadas, DeCietePulgadas)
        FinPara
        TotalClasificados=← DeTresPulgadas+ DeCincoPulgadas+ DeCietePulgadas
        Escribir TotalClasificados
        Escribir DeTresPulgadas
        Escribir DeCincoPulgadas
        Escribir DeCietePulgadas
        SinClasificar← 5000 – TotalClasificados
        Escribir SinClasificar
Fin
Algoritmo Clasificar (L, C3, C5, C7)
Inicio
        Si L=3 pulgadas entonces
           C3← C3+1
            SinoSi L=5 pulgadas entonces
                C5← C5+1
                SinoSi L=7 pulgadas entonces
                C7← C7+1
        Finsi
Fin
```

19. Dado un número entero determinar si su primer dígito es un número primo.

Solución:

```
Algoritmo Primo
        Inicio
        Entorno: n, d, j variables numéricas, primo variable lógica
        Leer n

   Mientras n <> 0
        d ← n - (n \ 10) * 10
        n ← n \ 10
   FinMientras
   primo ← Cierto
        j ← 1
      Repetir
        J←j+1
        Si (d \ j) * j = d Y d <> 2 Entonces
           primo ← Falso
        FinSi
```

```
    Hasta que primo = Falso O j >= d^(1/2)
    Si primo=cierto entonces
       Escribir "El primer dígito del número es un número primo"
       Sino
          Escribir "El primer dígito del número no es un número primo"
    FinSi
Fin
```

20. Leer un número entero y determinar si la suma de sus dígitos es un número par

Solución:

```
Algoritmo NumeroPar
   Entorno: n, d, s Variables numéricas
   Leer   n
   Mientras n <> 0
      d ← n - (n \ 10) * 10
      s ← s + d
      n ← n \ 10
   FinMientras
       Si (s\2)*2=s entonces
            Escribir "La suma de sus dígitos es un número par"
        Finsi
  Fin
```

21. Leer un número entero y determinar cuál es el menor de sus dígitos.

```
Algoritmo MenorDígito
   Entorno: n, d, me Variables numéricas
   Leer n
   me ← n - (n \ 10) * 10
   Mientras n <> 0
      n ← n \ 10
      d ← n - (n \ 10) * 10
      Si d <= me Entonces
         me ← d
      FinSi
   FinMientras
   Escribir  "El menor de los dígitos es"; me
Fin
```

Resolver los ejercicios con estructuras repetitivas de los propuestos en el Capítulo II.

Como vimos, existen diferentes tipos de ciclos. ¿Cuál de las estructuras cíclicas es la mejor..? Comencemos por decir que en la inmensa mayoría de las veces todos los ciclos son equivalentes. Como vimos existían unos casos que no permitían una implementación de algunos ciclos pero en general la mayoría de las veces son equivalentes. Cuando

necesite un ciclo en un algoritmo simplemente utilice aquel que vea más a la mano, es decir, aquel que vea más lógico, aquel con el cual vea que le resulta más fácil implementar el conjunto de instrucciones a repetir. No se ate obligatoriamente a un esquema de ciclos pero no se extrañe si con el tiempo nota que utiliza uno o dos esquemas de ciclos muchos más que los otros.
Con los ciclos todo lo que tenemos que hacer es tener mucho cuidado de que las condiciones sean coherentes con el cuerpo de los mismos de manera que se cumpla que el conjunto de instrucciones a repetir se itere una cantidad definida de pasos. Es muy útil recordar que la buena utilización de operadores relacionales y operadores booleanos es lo que nos permite lograr unos ciclos eficientes y ante todo muy funcionales.

Noción de vector

Vamos ahora a definir una nueva estructura de objeto, frecuentemente utilizado en informática, y a la que se le aplican numerosos tratamientos iterativos.

Más de una vez habrá notado que el cajero de un Banco, para referirse a una de las personas que hacen cola para hacer una consignación puede decir “Venga la quinta persona” y habrá notado como, una persona que no estaba de primera, pasa a ser atendida por él. Varios razonamientos se involucran inconscientemente en esta situación:

1. En primera instancia sabemos que la “quinta persona” se ubica contando a partir de la primera. Esto es algo muy obvio pero para el tema que nos ocupa en el momento es importantísimo.
2. Normalmente todas las personas, sin equivocación, voltean a mirar a una sola persona que es, con toda seguridad, la que ocupa la quinta posición en la fila.
3. El “quinto puesto” en la cola tiene que estar ocupado por alguien. No puede estar vacío pues esa tuvo que haber sido una de las razones para que el cajero se refiriera a dicha persona con tanta seguridad.
4. La cola de personas no es infinita. Tiene una cantidad determinada de clientes.
5. Delante de la Quinta persona deben haber cuatro personas y después de ella pueden haber muchas o ninguna persona.
6. Si existen varias colas solo atenderá el llamado del cajero la “quinta persona” de la cola a la que él se refiera.
7. Si en algún momento otro cajero dijera “todas las personas de esa cola pásense a esta caja”. Entonces todo el conjunto de personas atendería el llamado de ese otro cajero lo cual significaría que en cualquier momento ese conjunto de personas puede ser manejado como una sola unidad.

Podemos ver cómo para ubicar efectivamente a una persona necesitamos utilizar referencias diferentes. Esto quiere decir que para ubicar a una persona dentro de una cola todo lo que tenemos que hacer es utilizar QUINTO, CUARTO etc. Nos estamos refiriendo exactamente a una persona de manera incuestionable.
Es curioso pensar que ésta persona pudo haber sido la misma solo que para referirnos a ella tuvimos que ubicarla exactamente de manera diferente en cada una de las situaciones.

Para ilustrar mejor consideremos el siguiente problema:
Una empresa recoge cada mes el importe de las ventas realizadas por cada una de sus 40 sucursales. Se quiere componer una secuencia algorítmica que calcule el total de las ventas.
Una primera solución consiste en definir 40 variables Venta0, Venta1,.....Venta39 de las que cada una tendrá por valor el importe de una de las ventas.
Si Total es la variable de acumulación se escribirá la acción:

Total← Venta0+Venta1+Venta2+.....+Venta39
En la realidad habrá que indicar explícitamente las 40 variables, ya que ningún procesador comprenderá el significado de los puntos suspensivos que hemos utilizado. En la práctica raramente se adopta una solución así: es costoso enumerar y tratar demasiadas variables.
Otro método consiste en agrupar los valores en un objeto de estructura análoga a la de la variable compuesta pero donde todas las subdivisiones son del mismo tipo y cada cuna contiene uno de los cuarenta valores:

(0)	(1)	(2)	(3)		(38)	(39)
1000	850	1700	675	----------	2000	450

Demos un nombre a este objeto: llamémosle Venta. Venta puede representarse esquemáticamente en forma de una tabla. Para designar uno de los 40 valores se utilizará su posición en la tabla ejemplo Venta(3) representa el cuarto valor (675).
Acabamos de definir un vector (Venta) en el que los 40 elementos son Venta(0), Venta(1)..... Venta(39). Para designar un elemento se utilizará el nombre del vector seguido de una expresión aritmética entre paréntesis. El valor de la expresión da la posición del elemento (índice): Venta(i), Venta(10), Venta(k+3).
En la expresión Venta(i), i debe ser una variable numérica de valor entero tal que 0<= i <=39.

Un **vector** es conjunto de variables, donde cada una de ellas puede ser referenciada utilizando su posición relativa en relación con el primer elemento de dicho conjunto.
Un vector, tiene un nombre, un tipo(tipo de dato) y un tamaño (cantidad de elementos)

Para definir un vector se pone el nombre del vector y a continuación entre paréntesis se escribe la cantidad de elementos que almacenará y seguidamente se declara el tipo de dato. Ejemplo: **Venta(39): numérico** define un vector de 40 elementos (0- 39)**.** Al definir el tamaño del vector de esta manera, el índice del primer elemento es 0. Si se quiere que el primer elemento comience en un índice diferente de 0, entonces entre paréntesis se define el índice de ese primer elemento y el índice del último, así:
Venta(1 a 40), este vector tiene 40 elementos (de 1 a 40).

Se justifica la utilización de vectores cuando se dan algunas de las siguientes razones:
1. En algún momento se necesita manejar una cantidad de datos como todo un conjunto.
2. Se necesitan almacenar datos que posteriormente se van a volver a utilizar.
3. Se necesitan realizar cálculos de manera que los resultados progresivos se vayan a necesitar más adelante.
4. Se necesita realizar un determinado proceso con un conjunto de datos, “al tiempo”.
5. Se necesita realizar cálculos tan complejos que resulte más óptimo almacenar los resultados provisionales que volverlos a calcular.
6. En general cada vez que necesitemos hacer operaciones con conjuntos de datos.
7. Siempre que se necesite desarrollar algoritmos con conjuntos de datos cuya cantidad pueda en algún momento, no dentro de una misma ejecución, cambiar.

Volviendo al problema planteado, podemos ya describir una solución diciendo que es suficiente con acumular cada elemento del vector Venta en la variable Total:

```
Algoritmo Empresa
Inicio
        Entorno:
        I: variable numérica
        Total: Variable numérica
        Venta(39): vector numérico
        Total← 0
        Para I de 0 a 39
                Total← Total+Venta(i)
        Finpara
```

En los problemas con vectores la entrada y salida de datos se hace con un ciclos para manejar cada uno de sus elementos. Con frecuencia hay que utilizar ciclos anidados para la solución de problemas.
Cuando se conoce inicialmente la cantidad de elementos del vector se dice que el vector es determinado y si no se conoce al inicio se dice que el vector es indeterminado.
Cuando el vector es indeterminado su tamaño se define con un valor que supere la cantidad probable que debe almacenar.
Veamos como determinar el mayor y el menor valor de los datos almacenados en un vector.
Para determinar el mayor valor almacenado en un vector puede utilizarse el siguiente algoritmo:

```
Algoritmo Mayor
Inicio
Entorno:
ma: numérico
i: numérico
num(4): vector numérico (el tamaño del vector se pone de acuerdo al problema.
ma ← num(0) se le asigna a ma el valor que contiene el primer elemento del vector
Para i de 0 a 4
    Si num(i) >= ma entonces
       ma ← num(i) en ma queda almacenado el mayor valor almacenado en el vector.
    Finsi
FinPara
```

Algoritmo para el menor:

```
Algoritmo Menor
Inicio
Entorno:
me: numérico
i: numérico
num(4): vector numérico (el tamaño del arreglo se pone de acuerdo al problema.
me ← num(0) se le asigna a ma el valor que contiene el primer elemento del vector
Para i de 0 a 4
    Si num(i) <= me entonces
       me ← num(i) en ma queda almacenado el menor valor almacenado en el vector.
    Finsi
FinPara
```

Los algoritmos anteriores solo guardan uno de los valores mayores o menores almacenados en el vector; pero en un vector puede haber más de un elemento que contenga el mayor o el menor valor.

¿Cómo determinar todos los elementos del vector que contienen el mayor o el menor valor?.

Para resolver este problema después de ejecutar los algoritmos anteriores, se ejecuta otro ciclo en el que se compara cada elemento del vector con el valor almacenado en la variable **ma,** si son iguales, este valor se guarda en otro vector.

El algoritmo quedaría así:

```
Algoritmo Mayor
Inicio
Entorno:
Ma(4): numérico
i: numérico
TodosMayores(4): vector numérico
num(4): vector numérico (el tamaño del arreglo se pone de acuerdo al problema.
ma ← num(0) se le asigna a ma el valor que contiene el primer elemento del vector
Para i de 0 a 4
    Si num(i) >= ma entonces
       ma ← num(i) en ma queda almacenado el mayor valor almacenado en el vector.
    Finsi
FinPara
```

Con este ciclo se almacenan todos los mayores.

```
Para i de 0 a 4
    Si num(i) = ma entonces
       TodosMayoes(i) ← num(i)
    Finsi
FinPara
```

En TodosMayoes quedan almacenados todos los elementos del vector que contienen valor mayor.

De forma similar se define el algoritmo para almacenar todos los elementos del vector que contienen el menor valor:

```
Algoritmo Menor
Inicio
Entorno:
me: numérico
i: numérico
TodosMenores(4): vector numérico
num(4): vector numérico (el tamaño del arreglo se pone de acuerdo al problema.
me ← num(0) se le asigna a ma el valor que contiene el primer elemento del vector
Para i de 0 a 4
    Si num(i) <= me entonces
       me ← num(i) en ma queda almacenado el menor valor almacenado en el vector.
    Finsi
FinPara
Con este ciclo se almacenan todos los menores.
Para i de 0 a 4
    Si num(i) = me entonces
       TodosMenores (i) ← num(i)
    Finsi
FinPara
```

Veamos como realizar la búsqueda de un valor determinado dentro de un vector.
Para realizar una búsqueda se almacena en una variable el valor a buscar y luego se recorre el vector, comparando cada uno de sus elementos con el valor almacenado en la variable, si son iguales se guarda en una variable de tipo lógico el valor cierto. Al terminar de recorrer el vector, si la variable tiene el valor cierto, el elemento existe.

Un algoritmo para la búsqueda dentro del vector puede ser el siguiente:

```
Algoritmo Buscar
Inicio
Entorno:
ElementoBuscar: numérico
Existe: lógico
i : numérico
Num(4): vector numérico
Existe← falso
ElementoBuscar← 8
Para i de 0 a 4
    Si   num(i) = ElementoBuscar  entonces
       Existe← cierto
    Finsi
FinPara
Si Existe=cierto entonces
        Escribir " el elemento 8 está en el vector"
```

```
        Sino
            Escribir " el elemento 8 no está en el vector"
FinSi
```

Ejercicios resueltos con vectores

Haga el análisis EPS y un algoritmo para resolver los siguientes problemas:

1. En una escuela se tiene la cantidad de ausencia que han ocurrido en sus 5 grupos durante los 24 días del mes. Se quiere saber cuántas ausencias hubo en dicho mes y el promedio de ausencias por día de cada grupo.

Solución

Análisis EPS

Entrada: Aus: var numérica para leer las ausencias.

Proceso: Sumar, calcular promedio.

Salida: S,P

```
Algoritmo Ausencias
Inicio
   Entorno:
   Aus: var numérica para leer la ausencia.
    I: var numérica para controlar el ciclo de los días
    S: var numérica para acumular la suma de ausencias
    P(1 a 5): vector numérico para guardar el promedio de cada grupo
    G: var numérica para controlar el ciclo de los grupos
    S← 0
    Para G de 1 a 5
          Para i de 1 a 24
              Leer Aus
              S← S+Aus
          Finpara
          P(g)← S/24
         S← 0
    Finpara

          Para G de 1 a 5
                Escribir  “El promedio del grupo” G “es" P(G)
          Finpara
Fin
```

El ciclo G controla los grupos

El ciclo i controla los días

En este ejercicio el vector P es un vector determinado porque se conoce la cantidad de elementos de este vector (5)

2. En una escuela se tiene la cantidad de ausencia que han ocurrido en todos sus grupos durante los 24 días del mes. Se quiere saber cuántas ausencias hubo en dicho mes y el promedio de ausencias por día de cada grupo. Mostrar el nombre del primer y último grupo.
   Observe que es el mismo problema del ejercicio anterior; pero que en este caso se desconoce la cantidad total de grupos.

Solución

```
Algoritmo Ausencias
Inicio
```

```
        Entorno:
        Aus: var numérica para leer la ausencia.
        I: var numérica para controlar el ciclo de los días
        S: var numérica para acumular la suma de ausencias
        P(100): vector numérico (como no se sabe el número de grupos se pone un número que se
        sepa que es mayor que la cantidad de grupos que pueda haber.
        NombreGrupo(100): vector de cadena para guardar el nombre de los grupos
        C: var numérica para contar la cantidad de grupos
        K: var para controlar el ciclo de impresión
        C← 1
        S← 0
    Repetir
        Leer NombreGrupo ( C )
         Para i de 1 a 24
            Leer Aus
            S← S+Aus ‘ acumula la suma de las notas de cada grupo
         Finpara
          P(C)← S/24 ‘ va guardando el promedio de cada grupo
          S← 0    ‘para que empiece en 0 en el otro grupo
          C← C+1
    Hasta que NumbreGrupo=”fin”
        Para K de 1 a C
                Escribir  “El promedio del grupo” K “es" P(K)
        Finpara
        Escribir “El nombre del primer grupo es” NombreGrupo(1)
        Escribir “El nombre del último grupo es” NombreGrupo(C)
Fin
```

Observe que en este caso como no se conoce la cantidad de grupos, se usa el ciclo repetir que permite poner una condición para terminar la iteración, en este caso pusimos cuando se lea la palabra “fin”.

En este ejercicio P es un vector indeterminado porque inicialmente no se conoce la cantidad de elementos de este vector.

3. Se tienen las notas del último trabajo de control de geografía de los 31 alumnos del grupo 5. Se necesita calcular el promedio del grupo y además la cantidad de alumnos que obtuvieron una nota mayor o igual que el promedio del grupo.
   Hacer un algoritmo para dar solución a este problema.
   Solución:

```
   Algoritmo Promedio
   Inicio
      Entorno:
       Nota(1 a 31): vector numérico para guardar cada nota
       S: variable numérica para sumar las notas
       P: variable numérica para guardar el promedio
       C: variable numérica para contar las notas mayo igual que el promedio.
       S← 0
```

```
    Para i de 1 a 31
        Leer Nota(i)
        S← S+Nota(i)
    Finpara
    P← S/31
    Para i de 1 a 31
        Si Nota(i)>=P entonces
            C← C+1
        Finsi
    Finpara
    Escribir "obtuvieron notas mayor o igual que el promedio" C "alumnos"
Fin
```

6. Leer 10 enteros, almacenarlos en un vector y determinar en qué posiciones del vector está el menor número par leído.

```
Algoritmo MenorPar

  Entorno i, ma, p Variables numéricas , V(1 a 10) Vector numérico
  Leer  V(i)
  FinPara
  me ← V(1)
  Para i de 1 a 10
    Si V(i) <= me y v(i)\2=v(i)/2   Entonces
      me ←V(i)
      p ←i
    FinSi
  FinPara
Para i de 1 a 10
        Si v(i)=me entonces
            Escribir "El menor está en la posición" i
        FinSi
FinPara
  Escribir "El mayor está en la posición " p
Fin
```

Resolver los ejercicios con vectores de los propuestos en el Capítulo II.

Aplicación de las tres técnicas para la solución algorítmica de un problema

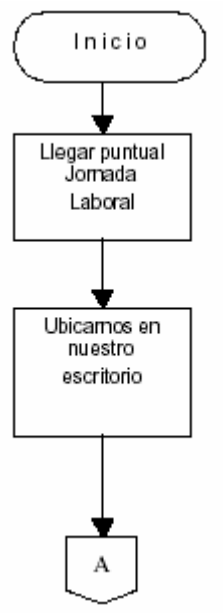
Después de haber abordado todas las estructuras básicas y haber resuelto problemas algorítmicamente, utilizando la técnica de seudo código, ilustraremos con ejemplos como hacerlo utilizando el diagrama de flujo y el diagrama rectangular:

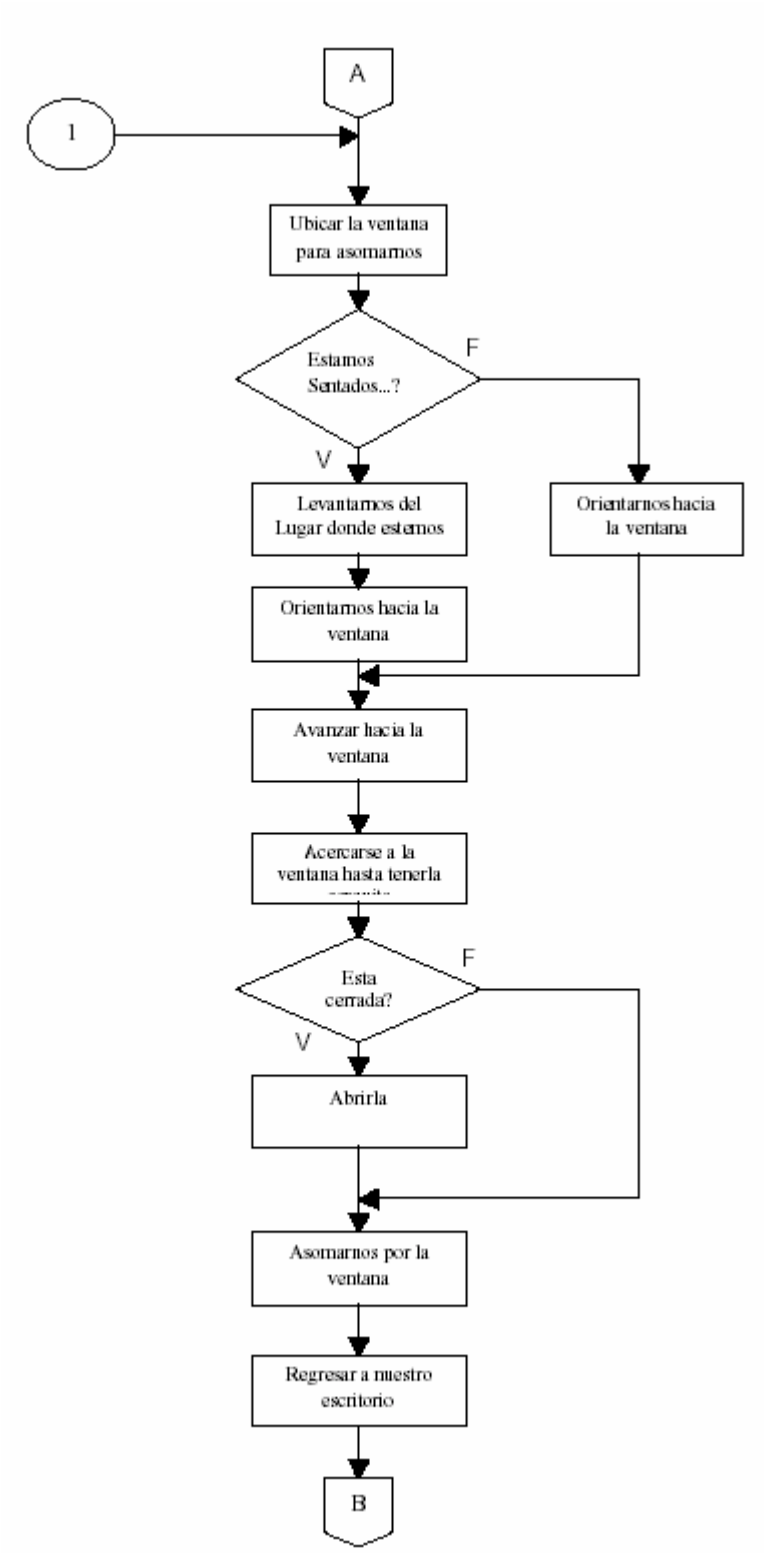
1. Hacer un algoritmo para vigilar una empresa desde una ventana asomándonos cada media hora por ella.

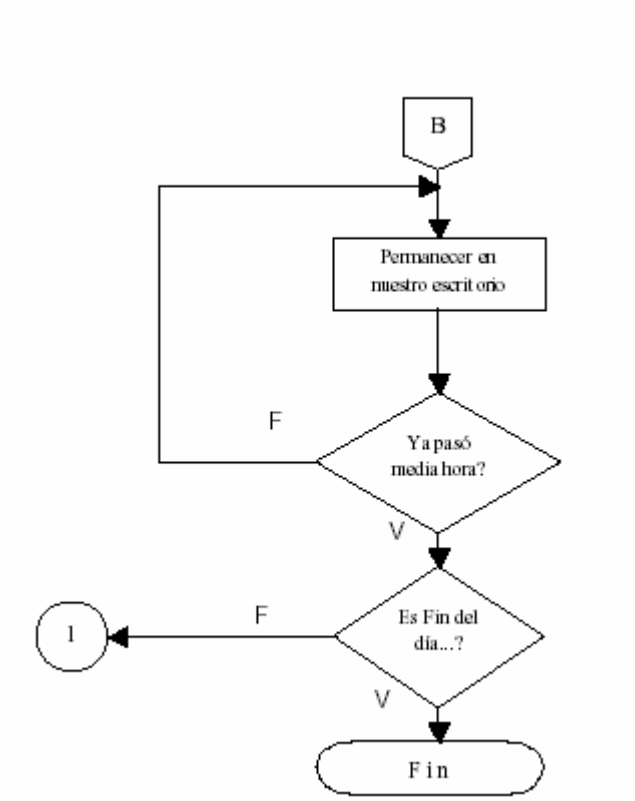
Solución con la técnica de seudo código:

```
Algoritmo VigilarEmpresa
Inicio
        Llegar puntual a la hora de inicio de la jornada laboral
        Ubicarnos en nuestro escritorio
        Mientras no sea el fin del día
                Ubicar la ventana por la que nos queremos asomar
                Si estamos sentados entonces
                        Levantarnos del lugar en donde estemos sentados
                        Orientarnos hacia la ventana
                        Sino
                           Orientarnos hacia la ventana
                          Avanzar hacia la ventana
                         Llegar hasta tener la ventana muy  cerca
                         Si está cerrada entonces
                                        Abrirla
                                         Asomarnos por la ventana
                                          Regresar a nuestro escritorio
                                         Mientras no haya pasado Media Hora
                                          Permanecer en nuestro escritorio
                                         Fin_Mientras
                                Finsi
                        Finsi
        Fin_Mientras
Fin
```

Solución utilizando la simbología de Diagramas de Flujo:
Se ha hecho el diagrama en varias páginas para ilustrar el uso de los conectores lógicos.




A
1
Ubicar la ventana para asomarnos
Estamos Sentados...?
F
V
Levantarnos del Lugar donde estemos
Orientarnos hacia la ventana
Orientarnos hacia la ventana
Avanzar hacia la ventana
Acercarse a la ventana hasta tenerla
Esta cerrada?
F
V
Abrirla
Asomarnos por la ventana
Regresar a nuestro escritorio
B



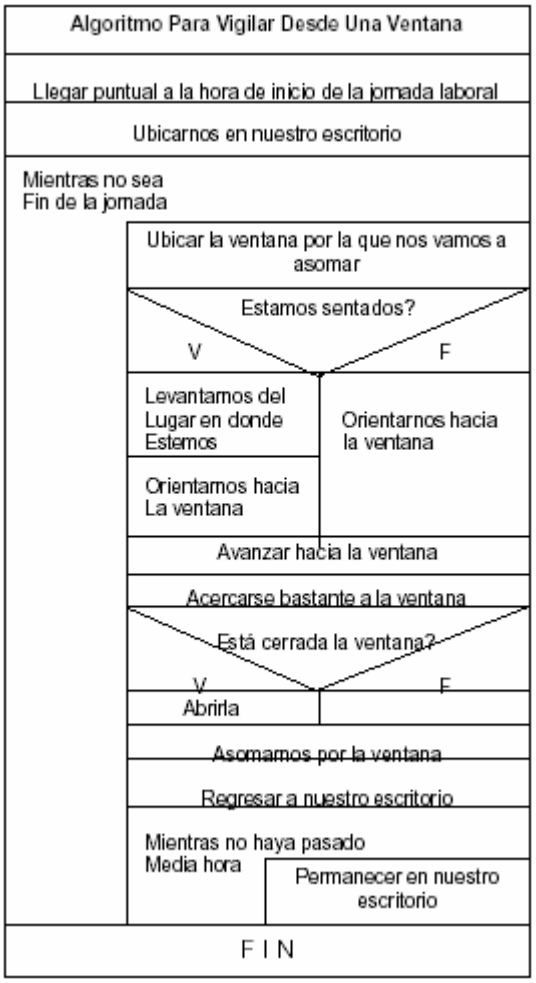
Cabe destacar algunos detalles significativos en este Diagrama de Flujo:

1. Toda decisión, como es obvio, tiene dos caminos: Un camino nos lleva a la acción o las acciones a realizar en el caso de que la respuesta a la pregunta sea Verdadera y el otro camino es el que nos dice que debemos hacer en caso de que la respuesta a la pregunta sea Falsa.
2. Lo que en el seudo código eran ciclos, en el diagrama se cambiaron por decisiones en donde uno de los caminos se devuelve (instrucciones atrás obviamente). Al realizar un seguimiento de este Diagrama de Flujo notará que se podrá devolver tantas veces como lo permita la condición de la decisión que queda al final y que solo se va a salir de ese ciclo cuando la condición sea Verdadera o sea que el ciclo se mantiene mientras la condición sea Falsa lo cual concuerda con la teoría de los ciclos.
3. En la última decisión, el camino Falso nos lleva a una burbuja que tiene un número 1 adentro. Número que también está al principio del diagrama pero con la flecha en el otro sentido (es decir, no saliendo del diagrama sino entrando a él). Se utiliza esta notación solo para simplificar un poco el Diagrama de Flujo.
4. Con el Diagrama de Flujo puede ver un gráfico de la solución y con ello hacerse una idea clara de la secuencia de pasos que necesitaría para alcanzar el objetivo.
5. Siempre que vaya a desarrollar un Diagrama de Flujo trate de ser muy organizado y muy estético, pues no se olvide que si vamos a representar un algoritmo computacional (en donde se busca que la computadora logre un objetivo por nosotros) al momento de la trascripción será muy importante el orden que haya tenido en la utilización de esta técnica.
6. Cuando diseñe un ciclo, no se olvide verificar que, lógicamente, la decisión por la cual reemplace el ciclo al momento de diseñar su diagrama de flujo tenga el mismo

comportamiento es decir permitan que bajo las mismas condiciones una acción o un conjunto de acciones se repitan una cantidad finita de veces.

7. Si el algoritmo que tiene para lograr este mismo objetivo es diferente, tenga presenta que el Diagrama de Flujo también va a ser diferente ya que éste es un reflejo gráfico de aquel.
8. Es muy importante que sepa que el solo hecho de cambiar la llegada de una determinada flecha, cambia completamente el algoritmo. Puede notar que la utilización de los símbolos resulta ser una tarea muy simplificada, pero lo que si es delicado es la colocación de las flechas ya que ellas son las que representan el sentido con que se va a “mover” el flujo de nuestro lógica.

Solución con el diagrama rectangular:


Algoritmo Para Vigilar Desde Una Ventana
Llegar puntual a la hora de inicio de la jornada laboral
Ubicarnos en nuestro escritorio
Mientras no sea
Fin de la jornada
Ubicar la ventana por la que nos vamos a asomar
Estamos sentados?
V
F
Levantarnos del Lugar en donde Estemos
Orientarnos hacia la ventana
Orientarnos hacia La ventana
Avanzar hacia la ventana
Acercarse bastante a la ventana
Está cerrada la ventana?
V
F
Abrirla
Asomarnos por la ventana
Regresar a nuestro escritorio
Mientras no haya pasado
Media hora
Permanecer en nuestro escritorio
F I N

Es imperante hacer algunas precisiones acerca de esta diagrama:
a. Puede notar que la correspondencia entre nuestra idea y su representación (bajo esta técnica) es mucho más exacta que en el caso del Diagrama de Flujo en donde hubo que hacer algunos pequeños cambios lógicos para que el diagrama correspondiera a la solución planteada.
b. La técnica de diagramación rectangular estructurada obliga a ser mucho más ordenado y no da ningún espacio para que nuestro algoritmo sea inentendible dado que las estructuras son relativamente rígidas.
c. Para la utilización de esta técnica solo tenemos que conocer tres símbolos y con ellos representamos todo lo que queramos dado que nuestra lógica se basa en esas tres estructuras.
d. Enmarcar nuestra idea en un rectángulo nos brinda una concepción más concreta de la solución planteada.
e. Realizar una corrida en frío con un diagrama basado en esta técnica se reduce a seguir la secuencia de instrucciones y (al igual que con los diagramas de flujo) a realizar una a una y tal como están allí las instrucciones o acciones, las decisiones y la revisión de las condiciones de los ciclos.

2. Hacer un algoritmo para saber si un número es par.

Seudo código:

```
Algoritmo NúmeroPar
Inicio
        Entorno:
        Num: variable entera
        Leer num
        Si num < 0 entonces
                Escribir  “El número debe ser positivo”
                SinoSi num / 2 * 2 = num entonces
                     Escribir  “El número leído es par”
           Sino
             Escribir “El número leído no es par”
        Finsi
Fin
```

Diagrama de flujo:

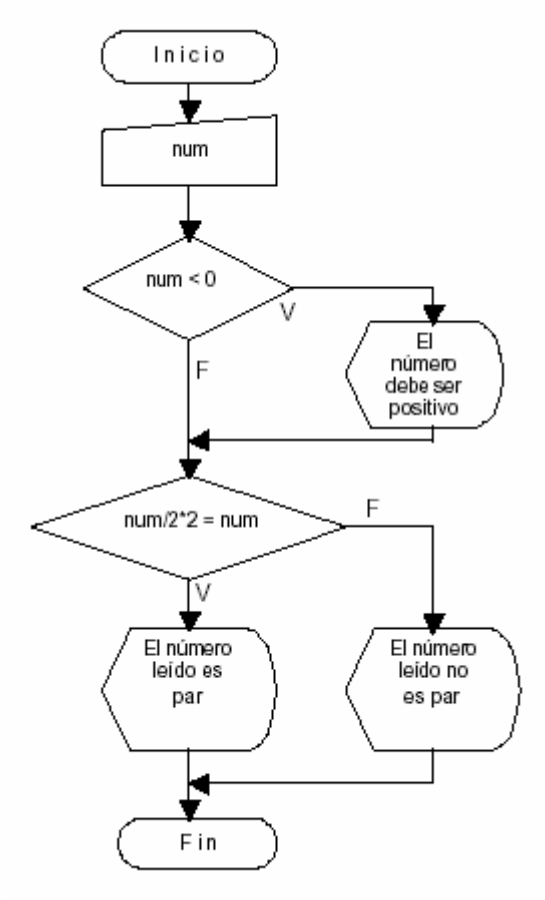
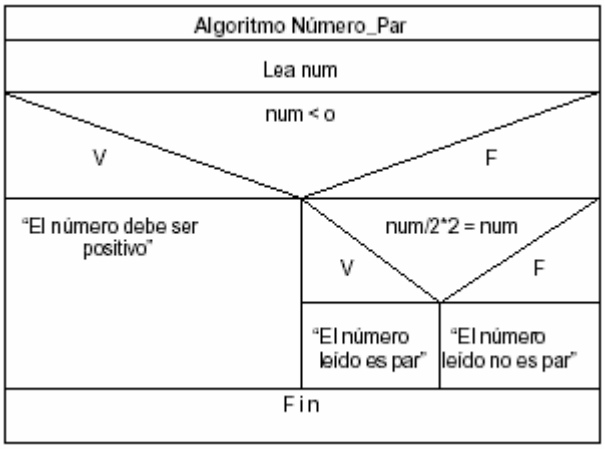


Diagrama rectangular estructurado:



Capítulo II. Ejercicios Propuestos

*En los ejercicios de este capítulo el orden de los dígitos de un número se toman de izquierda a derecha.

Ejercicios con algoritmos informales

Elabore un algoritmo que permita:

1. lavar una camisa en una lavadora.
2. sentarse en un auto.
3. poner una inyección.
4. hacer una costura en una máquina de coser.
5. poner un bombillo en el lugar de otro fundido..
6. hacer una pizza.
7. hacer una maraca.
8. colocarse una camisa.
9. sacar la cámara de la goma de una bicicleta.
10. elevar una cometa.
11. armar una casa de campaña.
12. pesar un artículo.
13. cambiar la zapatilla a una pluma de agua.
14. entrar a una casa que está con llave.
15. envolver un regalo.
16. cambiar el caset a una grabadora.
17. hacer un triángulo de madera.
18. freír un huevo.
19. poner un telegrama.
20. hacer un avión con una hoja de papel.
21. hacer un pastel.
22. hacer un depósito en su cuenta de ahorros.
23. ir de la casa al trabajo.
24. quitar el corcho a una botella con el saca corchos.
25. manejar una bicicleta.
26. fregar un baso.
27. mirar por un telescopio.
28. abrir una lata de conserva.
29. parquear un vehículo.
30. ponerse los zapatos.
31. quitarse la camisa.
32. quitarse los zapatos.
33. tirarse desde un avión con un paracaídas.
34. hacer una ensalada de tomates.
35. tomar una fotografía.
36. regar un cantero de verduras.

Algoritmos computacionales

Ejercicios de estructura lineal

1. Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + 3
b = b + 4 – a
c = a + b + c
a = a + c
c = c + 3 - b + 2

2. Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + 10
b = b + 5 – c
c = c + 4 + b
d = d + b + a
a = a + 1
b = b + c
c = b + c
d = b + b

3. Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + 4
b = b + 2
a = a + 10
b = b – 25
a = a – 20
b = b + 5
a = a + 4
b = b + 2
a = a + 10
b = b – 10

4. Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + b
b = a - b
c = a + b
d = a - b
a = a - b
b = a + b
c = a - b
d = a + b

5. Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a - 5
b = b + 6
a = a + 18
b = b – 23
a = a – 21
b = b - 5
a = a - 4
b = b - 2
a = a + 10
b = b + 10

6. Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + b – c + d
b = a + b – c + d
c = a + b – c + d
d = a + b – c + d
a = a + b – c + d
b = a + b – c + d
c = a + b – c + d
d = a + b – c + d

7 Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + 15
b = b + 12
c = a * c

8.- Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + 1
b = b + 2
c = c + 3

9.- Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + b - 5
b = a + b - 5
c = a + b - 5
a = a + 5 * $\frac{b}{2}$
b = a + 5 * $\frac{b}{2}$
c = a + 5 * $\frac{b}{2}$

10 Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + a
b = b + b
c = c + c
a = a + b + c
b = a + b + c
c = a + b + c

11 Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:
a = a + 5
b = a + 3
c = a + 2
a = b + 4
b = b + 5
c = c + 8

12.- Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + c
b = a + c
c = a + c
a = c + 5
b = c + b
c = a + b + c

13.- Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + a
b = b + a
c = c + a
a = a + a
b = b + a
c = c + a

14.- Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a – b
b = b – c
c = c – a
a = a – 1
b = b – a
c = c + a – b

15.- Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

a = a + b
b = a – b
c = a * b
a = a – b
b = a + b
c = a * b

16.- Haga el algoritmo y la corrida en frío para conocer los valores finales de las variables contenidas en las siguientes expresiones:

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

17.- Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = \frac{a + \frac{b}{c}}{\frac{a}{b} + c}$$

18.- Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = \frac{a + b + \frac{a}{b}}{c}$$

19.- Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = \frac{\frac{a}{a + b}}{\frac{a}{a - b}}$$

20.- Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = \frac{a + \frac{b}{a + b + \frac{b}{c}}}{a + \frac{b}{c + a}}$$

21.- Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = \frac{a + b + c}{a + \frac{b}{c}}$$

22.- Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = \frac{a + b + \frac{c}{d * a}}{a + b * \frac{c}{d}}$$

23.- Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = \frac{a + \frac{b}{c} + d}{a}$$

24.- Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = \frac{\frac{a}{b} + \frac{b}{c}}{\frac{a}{b} - \frac{b}{c}}$$

25.- Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = a + \frac{a + \frac{a + b}{c + d}}{a + \frac{a}{b}}$$

26.- Haga el algoritmo y la corrida en frío para conocer el valor de x en la siguiente ecuación:

$$x = a + b + \frac{c}{d} + \frac{\frac{a}{b - c}}{\frac{a}{b + c}}$$

27. Dado un número entero de tres dígitos escribir su último dígito.
28. Dado un número entero de tres dígitos, escribir cada uno de sus dígitos.
29. Leer un número entero de tres dígitos y calcular la suma de sus dígitos.
30. Leer un número entero de tres dígitos y calcular el producto del primer dígito por el último y el resultado dividirlo por el segundo dígito.
31. Dados tres valores numéricos enteros se quiere conocer si son distintos.
32. Dados los valores de X=2, Y=3 y Z=4, escribir el valor que devuelven los siguientes predicados:
    a. Y=X

b. Y< X
c. X<>Y
d. Y>X
e. X<4
f. Y>=3
g. X>2 y Y<4
h. X=Y o Z=X
i. Z>Y y X<=Z
j. X=10 o Y<Z y X=2
k. Z=X y Y=3 y X<>0
l. X<<0 o Y<>0 o Z<>0
m. Y=3 y X=2 y Z=3
n. X=2 y Z>2 o Y<4
o. NO X=2
p. NO Y=Z

33. NO Z=2 y NO X=2
34. Y=X o Z=4 y NO Y>4

35. Desde un algoritmo transfiera un número entero hacia otro algoritmo donde se calcule y muestre el cuadrado de dicho número. Clasifique los parámetros.
36. Leer dos números en un algoritmo y transferirlo hacia otro algoritmo donde se intercambien sus valores. Clasifique los parámetros.
37. Leer un número entero en un algoritmo y transferirlo hacia otro algoritmo donde se lea otro número entero y se calcule y muestre la suma de ambos números. Clasifique los parámetros.
38. Leer un número entero de 2 dígitos en un algoritmo y transferirlo a otro algoritmo donde se lea un nuevo numero entero de 2 dígitos y se calcule y muestre la diferencia entre la suma de los dígitos del primer número con la suma de los dígitos del segundo número. Clasifique los parámetros.

39. Dados los valores de X="Cuba", Y="Patria" , Z="Pueblo" escriba los valores que devuelven los siguientes predicados

    a. Y=Z
    b. X> "Curso"
    c. X="Cubo"
    d. Z<X
    e. Y>X
    f. Z<"Poblado"

40. X, Y, Z, K son variables numéricas de un entorno dado. Exprese el predicado elemental correspondiente a cada una de las siguientes situaciones:

    a. Los valores de X, Y, Z y K son idénticos.
    b. Entre los valores de X, Y, K al menos dos son idénticos.
    c. El valor de X supera al de Y pero está por debajo del de Z.
    d. El valor de K es mayor o igual que el de Y pero menor o igual que el de Z
    e. Los valores de X, Y, Z, K son diferentes.

41. Leer un número entero de 3 dígitos en un algoritmo y transferirlo hacia otro algoritmo donde se calcule el promedio entero de sus dígitos. Mostrar el resultado en el primer algoritmo. Clasifique los parámetros.

42. Leer un número entero en un algoritmo y transferirlo a otro algoritmo donde se le extraiga su mitad. Mostrar el resultado en el primer algoritmo. Clasifique los parámetros.

43. Desde un algoritmo pase las 3 notas de un alumno a otro algoritmo donde se calcule el promedio de sus notas. Mostrar el resultado en el primer algoritmo. Clasifique los parámetros.

44. Leer un número entero de 3 dígitos en un algoritmo y transferirlo a otro algoritmo donde se establezca una comparación entre el primer y último dígito de dicho número. Mostrar el resultado en el primer algoritmo. Clasifique los parámetros.

45. Leer un número entero de 2 dígitos en un algoritmo y transferirlo a otro algoritmo donde se lea un segundo número entero de 2 dígitos y se calcule el promedio de los dígitos de los dos números. Mostrar el resultado en el primer algoritmo. Clasifique los parámetros.

46. Leer en un algoritmo el valor del radio de un círculo y calcular el área de dicho círculo en otro algoritmo. El área del círculo se calcula multiplicando la constante pi por el radio al cuadrado. Mostrar el resultado en el primer algoritmo. Clasifique los parámetros.

47. En un algoritmo determinar cada uno de los dígitos de un número entero de 3 dígitos y en otro algoritmo, mostrar el cuadrado de cada uno de los dígitos.

48. En el movimiento rectilíneo uniforme, la distancia recorrida se calcula multiplicando la velocidad del móvil por el tiempo de duración del movimiento. Calcule y muestre el valor de dicha distancia, en un algoritmo que reciba el valor de la velocidad y el tiempo desde otro algoritmo. Clasifique los parámetros.

Ejercicios con estructuras condicionales.

1. Leer un número entero y determinar si es un número terminado en 4.
2. Leer un número entero y determinar si tiene más de 3 dígitos.
3. Leer un número entero y determinar si es negativo.
4. Leer un número entero de dos dígitos y determinar a cuánto es igual la suma de sus dígitos.
5. Leer un número entero de dos dígitos y determinar si ambos dígitos son pares.
6. Leer un número entero de dos dígitos y determinar si un dígito es múltiplo del otro.
7. Leer un número entero de dos dígitos y determinar si los dos dígitos son iguales.
8. Leer dos números enteros y determinar cuál es el mayor.
9. Leer dos números enteros de dos dígitos y determinar si tienen dígitos comunes.
10. Leer dos números enteros de dos dígitos y determinar si la suma de los dos números origina un número par.
11. Leer dos números enteros de dos dígitos y determinar a cuánto es igual la suma de todos los dígitos.
12. Leer un número entero de tres dígitos y determinar a cuánto es igual la suma de sus dígitos.
13. Leer un número entero de tres dígitos y determinar si al menos dos de sus tres dígitos son iguales.
14. Leer un número entero de tres dígitos y determinar en qué posición está el mayor dígito.
15. Leer un número entero de tres dígitos y determinar si algún dígito es múltiplo de los otros.
16. Leer tres números enteros y determinar cuál es el mayor.
17. Leer tres números enteros y mostrarlos ascendentemente.
18. Leer tres números enteros de dos dígitos cada uno y determinar en cuál de ellos se encuentra el mayor dígito.
19. Leer un número entero de tres dígitos y determinar si el primer dígito es igual al último.
20. Leer un número entero de tres dígitos y determinar cuántos dígitos pares tiene.
21. Leer un número entero de tres dígitos y determinar si alguno de sus dígitos es igual a la suma de los otros dos.
22. Leer un número entero de cuatro dígitos y determinar si la suma de sus dígitos es mayor que 10.
23. Leer un número entero de cuatro dígitos y determinar cuántos dígitos pares tiene.
24. Leer un número entero de cuatro dígitos y determinar si el segundo dígito es igual al penúltimo.
25. Leer un número entero de 3 dígitos y determinar si tiene el dígito 1.
26. Leer un número entero y determinar si es igual a 10.
27. Leer un número entero y determinar si es múltiplo de 7.
28. Leer un número entero y determinar si termina en 7.
29. Leer un número entero menor que mil y determinar cuántos dígitos tiene.
30. Leer un número entero de dos dígitos, guardar cada dígito en una variable diferente y luego escribirlas.

31. Leer un número entero de 4 dígitos y determinar si tiene más dígitos pares o impares.
32. Leer dos números enteros y determinar cuál es múltiplo de cuál.
33. Leer tres números enteros y determinar si el último dígito de los tres números es igual.
34. Leer tres números enteros y determina si el penúltimo dígito de los tres números son iguales.
35. Leer dos números enteros y determinar si la diferencia entre los dos es un número par.
36. Leer dos números enteros y determinar si la diferencia entre los dos es un número divisor exacto de alguno de los dos números.
37. Leer un número entero de 4 dígitos y determinar si el primer dígito es múltiplo de alguno de los otros dígitos.
38. Leer un número entero de 2 dígitos y si termina en 1 mostrar su primer dígito, si termina en 2 mostrar la suma de sus dígitos y si termina en 3 mostrar el producto de sus dos dígitos.
39. Leer un número entero y si es múltiplo de 4 mostrar su mitad, si es múltiplo de 5 mostrar su cuadrado y si es múltiplo de 6 mostrar su primer dígito. Asumir que el número no es mayor que 100.

Ejercicios con estructuras repetitivas

1. Hacer un algoritmo que permita determinar la cantidad de días en los que las precipitaciones caídas estuvieron por encima de los 44 mm.
2. Elaborar un algoritmo que permita obtener una tabla de las raíces cuadras a partir del número en que se quiera comenzar el cálculo y hasta el valor de la raíz que se desee.
3. Escriba un algoritmo que pregunte en que sentido se desea realizar la conversión de radianes a grados o de grados a radianes, que pida además los valores del principio y e final de la tabla e imprima la tabla de resultados.
   Para convertir grados a radianes se utiliza la fórmula:
   ValorRadianes=3.14*ValorGrados/180
   Para convertir de radianes a grados:
   ValorGrados=180*ValorRadianes/3.14
4. Elabore un algoritmo que permita generar los productos del 1 al 10 de un número entero determinado.
5. Haga un algoritmo que escriba los múltiplos de un número A, mientras los mismos no excedan de 100.
6. Escriba un algoritmo que permita, dado el nombre y la edad de N trabajadores, escribir el promedio de edad.
7. Se tiene un listado de los trabajadores de una empresa con la siguiente información: Nombre, sexo, edad, categoría ocupacional. Se quiere calcular el promedio de edad de las mujeres y cuántos técnicos trabajan en la empresa.
8. Escriba un algoritmo para dada las notas de n alumnos escribir la mayor de ellas.
9. Haga un algoritmo que permita dados los tiempos logrados por un ciclista expresados en minutos, determinar cuál fue su mejor tiempo.
10. Durante la presentación de una película se le ha entregado a cada espectador una planilla, donde debe indicar si le gustó o no la película, y al abandonar el cine, la persona debe depositar la planilla en una caja. Se necesita conocer, al concluir la función del día, a cuántas personas le gustó o no la película.
11. Leer un número entero de dos dígitos menor que 20 y determinar si es primo.
12. Leer un número entero de dos dígitos y determinar si es primo y además si es negativo.
13. Leer un número entero de dos dígitos y determinar si sus dos dígitos son primos.
14. Leer un número entero de tres dígitos y determinar cuántos dígitos primos tiene.
15. Leer un número entero menor que 50 y positivo y determinar si es un número primo.
16. Leer dos números enteros y si la diferencia entre los dos es menor o igual a 10 entonces mostrar todos los enteros comprendidos entre el menor y el mayor de los números leídos.
17. Leer dos números enteros de dos dígitos y si la diferencia entre los dos números es par mostrar la suma de los dígitos de los números, si dicha diferencia es un número primo menor que 10 entonces mostrar el producto de los dos números y si la diferencia entre ellos terminar en 4 mostrar todos los dígitos por separado.
18. Leer un número entero y mostrar todos los enteros comprendidos entre 1 y el número leído.
19. Leer un número entero y mostrar todos los pares comprendidos entre 1 y el número leído.

20. Leer un número entero y mostrar todos los divisores exactos del número comprendidos entre 1 y el número leído.
21. Leer dos números enteros y mostrar todos los enteros comprendidos entre ellos.
22. Leer dos números entero y mostrar todos los números terminados en 4 comprendidos entre ellos.
23. Leer un número entero de tres dígitos y mostrar todos los enteros comprendidos entre 1 y cada uno de los dígitos.
24. Mostrar todos los enteros comprendidos entre 1 y 100.
25. Mostrar todos los pares comprendidos entre 20 y 200.
26. Mostrar todos los números terminados en 6 comprendidos entre 25 y 205.
27. Leer un número entero y determinar a cuánto es igual la suma de todos los enteros comprendidos entre 1 y el número leído.
28. Leer un número entero de dos dígitos y mostrar todos los enteros comprendidos entre un dígito y otro.
29. Leer un entero y mostrar todos los múltiplos de 5 comprendidos entre 1 y el número leído.
30. Mostrar los primeros 20 múltiplos de 3.
31. Escribir el resultado de sumar los primeros 20 múltiplos de 3.
32. Mostrar el promedio entero de los *n* primeros múltiplos de 3 para un número entero *n* leído.
33. Promediar los *x* primeros múltiplos de 2 y determinar si ese promedio es mayor que los *y* primeros múltiplos de 5 para valores de *x* y *y* leídos.
34. Leer dos números entero y mostrar todos los múltiplos de 5 comprendidos entre el menor y el mayor.
35. Leer un número entero y determinar si es primo.
36. Leer dos números enteros y determinar si la diferencia entre los dos es un número primo.
37. Leer un número entero de 2 dígitos y si es par mostrar la suma de sus dígitos, si es primo y menor que 17 mostrar su último dígito y si es múltiplo de 5 y menor que 30 mostrar el primer dígito.
38. Leer un número entero y si es menor que 100 determinar si es primo.
39. Leer un número entero y si es múltiplo de 3 determinar si su último dígito es primo.
40. Leer un número entero y determinar cuántos dígitos tiene.
41. Leer un número entero y determinar a cuánto es igual la suma de sus dígitos.
42. Leer un número entero y determinar cuántas veces tiene el dígito 1.
43. Leer un número entero y determinar si la suma de sus dígitos es un número primo.
44. Leer un número entero y determinar a cuánto es igual la suma de sus dígitos pares.
45. Leer un número entero y determinar a cuánto es igual el promedio entero de sus dígitos.
46. Leer un número entero y determinar cuál es el mayor de sus dígitos.
47. Leer 2 números enteros y determinar cuál de los dos tiene mayor cantidad de dígitos.
48. Leer 2 números enteros y determinar cual de los dos tiene mayor cantidad de dígitos primos.
49. Leer un número entero y determinar a cuánto es igual el primero de sus dígitos.
50. Leer un número entero y mostrar todos sus componentes numéricos o sea aquellos para quienes el sea un múltiplo.

51. Leer números enteros hasta que digiten 0 y determinar a cuánto es igual el promedio de los números terminados en 5.
52. Leer números hasta que digiten 0 y determinar a cuanto es igual el promedio entero de los número primos leídos.
53. Dado el número 248, determinar cuál es el número primo más cercano por debajo de él.
54. Generar los números del 1 al 10 utilizando un ciclo que vaya de 10 a 1.
55. Leer dos números enteros y determinar a cuánto es igual el producto mutuo del primer dígito de cada uno.
56. Mostrar la tabla de multiplicar del número 5.
57. Generar todas las tablas de multiplicar del 1 al 10.
58. Leer un número entero y mostrar su tabla de multiplicar.
59. Se define la serie de Fibonacci como la serie que comienza con los dígitos 1 y 0 y va sumando progresivamente los dos últimos elementos de la serie, así: 0 1 1 2 3 5 8 13 21 34..... Utilizando el concepto de ciclo generar la serie de Fibonacci hasta llegar o sobrepasar el número 10000.
60. Leer un número entero de dos dígitos y determinar si pertenece a la serie de Fibonacci.
61. Determinar a cuánto es igual la suma de los elementos de la serie de Fibonacci entre 0 y 100.
62. Determinar a cuánto es igual el promedio entero de los elementos de la serie de Fibonacci entre 0 y 1000.
63. Determinar cuántos elementos de la serie de Fibonacci se encuentran entre 1000 y 2000.
64. Leer un número entero y calcularle su factorial.
65. Leer un número entero y calcularle el factorial a todos los enteros comprendidos entre 1 y el número leído.
66. Leer un número entero y calcular el promedio entero de los factoriales de los enteros comprendidos entre 1 y el número leído.
67. Leer un número entero y calcular a cuánto es igual la sumatoria de todos los factoriales de los números comprendidos entre 1 y el número leído.
68. Utilizando ciclos anidados generar las siguientes parejas de enteros

0 1
1 1
2 2
3 2
4 3
5 3
6 4
7 4
8 5
9 5

69. Utilizando ciclos anidados generar las siguientes ternas de números

1 1 1
2 1 2
3 1 3

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

70. Utilizando ciclos anidados generar las siguientes parejas de números
0 1
1 1
2 1
3 1
4 2
5 2
6 2
7 2

71. En una base de campismo se alquilan bicicletas de diferentes tamaños (20, 26 y 28). El tiempo de alquiler de cada tipo no cuesta lo mismo. Al terminar la jornada se quiere saber el tiempo de utilización de cada tipo de bicicleta, cuánto se recaudó por cada tipo y cuánto fue la recaudación total. Haga un algoritmo para dar solución a este problema.

Ejercicios con vectores

1. En una biblioteca se tienen 20 libros distintos de Computación, de los cuales se quiere conocer cuántas veces ha sido solicitado cada uno de ellos. Se tiene como información las solicitudes hechas en los últimos seis meses y se indica en cada una el número del libro solicitado.
2. Un circo hizo un total de 20 actos diferentes. Al concluir cada función se recogió una planilla por espectador donde se indicaba cuál o cuáles actos habían sido los más gustados(podían marcar hasta tres). Se quiere determinar cuál fue el acto más gustado, la cantidad de votos que obtuvo y el total de votos.
3. En un supermercado cada producto tiene una tarjeta que indica el código del artículo. Cuando el usuario pasa por la caja, introduce la tarjeta del artículo comprado en un robot y éste inmediatamente indica cuánto tiene que pagar. Al concluir el turno de trabajo se quiere que el robot imprima los códigos de los artículos que se vendieron junto con la recaudación hecha por artículo y el total de toda la venta. Haga el algoritmo para que el robot pueda realizar todas estas funciones.
4. Durante los juegos Centroamericanos y del Caribe efectuados en la Habana y dentro de las competencias de atletismo, se celebró el evento de maratón, con la participación de un número considerable de corredores. Según iban arribando a la meta, los jueces iban recogiendo el número de la camiseta y la hora de llegada. Después del arribo del último corredor, se entregó dicha información para ser procesada junto con los nombres de cada corredor, país y número de camiseta. Se quiere obtener:
   a) Medallista de oro, plata y bronce.
   b) Indicar si se rompió o no, el record de la competencia y quién o quienes lo rompieron.

5. Del último Campeonato de pelota se tiene la actuación de bateo de cada uno de los 25 jugadores que participaron (veces al bate y hits conectados) por cada juego del campeonato (51 juegos). Se necesita calcular el promedio de bateo de cada jugador y escribirlo junto con el nombre del jugador, además conocer los jugadores cuyo promedio de bateo fue mayor de 300.
6. Leer 10 enteros, almacenarlos en un vector y determinar en qué posición del vector está el mayor número par leído.
7. Almacenar en un vector 10 números enteros y determinar en qué posición del vector está el mayor número primo almacenado.
8. Cargar un vector de 10 posiciones con los 10 primeros elementos de la serie de Fibonacci y mostrarlos.(Se define la serie de Fibonacci como la serie que comienza con los dígitos 1 y 0 y va sumando progresivamente los dos últimos elementos de la serie, así: 0 1 1 2 3 5 8 13 21 34.....)
9. Llenar un vector con los 10 números primos comprendidos entre 100 y 300. Luego mostrarlos.
10. Leer dos números enteros y almacenar en un vector los 10 primeros números primos comprendidos entre el menor y el mayor. Luego mostrarlos.
11. Guardar 9 números enteros en un vector y determinar en qué posiciones se encuentra el número mayor.

12. Poner en un vector 8 números enteros y determinar en qué posiciones se encuentran los números terminados en 4.
13. Colocar 7 números enteros en un vector y determinar cuántas veces está repetido el mayor.
14. Entrar 6 números enteros, en un vector y determinar en qué posiciones se encuentran los números con más de 3 dígitos.
15. Situar 5 números enteros en un vector y determinar cuántos números, de los almacenados allí, tienen menos de 3 dígitos.
16. Asignar 4 números enteros, a un vector y determinar a cuánto es igual el promedio entero de los datos del vector.
17. Incorporar 10 números enteros, a un vector y determinar si el promedio entero de estos datos está almacenado en el vector.
18. Incluir 9 números enteros, en un vector y determinar cuántas veces se repite el promedio entero de los datos dentro del vector.
19. Almacenar 8 números enteros en un vector y determinar cuántos datos almacenados son múltiplos de 3.
20. Llenar un vector con 9 números enteros y determinar cuáles de los datos almacenados son múltiplos de 3.
21. Cargar 7 números enteros en un vector y determinar cuántos números negativos hay.
22. Poner 10 números enteros en un vector y determinar en qué posiciones están los números positivos.
23. Leer 8 números enteros, almacenarlos en un vector y determinar cuál es el número menor.
24. Colocar 5 números enteros en un vector y determinar en qué posición está el menor número primo.
25. Rellenar un vector con 16 números enteros y determinar en qué posición está el número cuya suma de dígitos sea la mayor.
26. Entrar 10 números enteros en un vector y determinar cuáles son los números múltiplos de 5 y en qué posiciones están.
27. Situar 18 números enteros en un vector y determinar si existe al menos un número repetido.
28. Asignar 30 números enteros a un vector y determinar en qué posición está el número con más dígitos.
29. Depositar 13 números enteros en un vector y determinar cuántos de los números depositados son números primos terminados en 3.
30. Incorporar 12 números enteros en un vector y calcularle el factorial a cada uno de los números incorporados almacenándolos en otro vector.
31. Almacenar 10 números enteros en un vector y determinar a cuánto es igual el promedio entero de los factoriales de cada uno de los números almacenados.
32. Llenar con 25 números enteros un vector y mostrar todos los enteros comprendidos entre 1 y cada uno de los números almacenados en el vector.
33. Cargar 12 números enteros en un vector y mostrar todos los enteros comprendidos entre 1 y cada uno de los dígitos de cada uno de los números almacenados en el vector.
34. Guardar 30 números enteros en un vector. Luego leer un entero y determinar si este último entero se encuentra entre los 30 valores almacenados en el vector.

35. Poner 18 números enteros en un vector. Luego leer un entero y determinar cuantos divisores exactos tiene este último número entre los valores almacenados en el vector.
36. Colocar 11 números enteros en un vector. Luego leer un entero y determinar cuántos números de los almacenados en el vector terminan en el mismo dígito que el último valor colocado.
37. Entrar 13 números enteros en un vector y determinar a cuánto es igual la suma de los dígitos pares de cada uno de los números entrados.
38. Situar 16 números enteros en un vector y determinar cuántas veces en el vector se encuentra el dígito 2. No olvide que el dígito 2 puede estar varias veces en un mismo número.
39. Asignar 8 números enteros a un vector y determinar si el promedio entero de dichos números es un número primo.
40. Incorporar 24 números enteros a un vector y determinar cuántos dígitos primos hay en los números leídos.
41. Almacenar 34 números enteros en un vector y determinar a cuántos es igual el cuadrado de cada uno de los números almacenados.
42. Cargar 20 números enteros en un vector y determinar si la semisuma entre el valor mayor y el valor menor es un número primo.
43. Guardar 30 números enteros en un vector y determinar si la semisuma entre el valor mayor y el valor menor es un número par.
44. Poner 40 números enteros en un vector y determinar cuántos números de los almacenados en dicho vector terminan en 15.
45. Colocar 38 números enteros en un vector y determinar cuántos números de los colocados en dicho vector comienzan con 3.
46. Entrar 17 números enteros en un vector y determinar cuántos números con cantidad par de dígitos pares hay almacenados en dicho vector.
47. Situar 12 números enteros en un vector y determinar en qué posiciones se encuentra el número con mayor cantidad de dígitos primos.
48. Asignar 19 números enteros a un vector y determinar cuántos de los números asignados en dicho vector pertenecen a los 100 primeros elementos de la serie de Fibonacci.
49. Leer 10 números enteros, almacenarlos en un vector y determinar cuántos números de los almacenados en dicho vector comienzan por 34.
50. Almacenar 14 números enteros en un vector y determinar cuántos números de los almacenados en dicho vector son primos y comienzan por 5.
51. Cargar 20 números enteros en un vector y determinar en qué posiciones se encuentran los números múltiplos de 10. No utilizar el número 10 en ninguna operación.
52. Guardar 40 números enteros en un vector y determinar en qué posición se encuentra el número primo con mayor cantidad de dígitos pares.
53. Poner 17 números enteros en un vector y determinar cuántos números terminan en dígito primo.
54. Colocar 15 números enteros en un vector y determinar cuántos números de los almacenados en dicho vector comienzan en dígito primo.

Capítulo III. Lenguajes de Programación

Hasta ahora hemos considerado un procesador como una entidad abstracta y susceptible de reconocer objetos (constantes, variables) y de ejecutar ciertas acciones sobre los valores de estos objetos, conforme a un algoritmo dado; es decir capaz de efectuar ciertas operaciones sobre la información contenida en estos objetos. Para elaborar un algoritmo, no es necesario definir un físicamente el instrumento que permite ejecutar este algoritmo. Basta con conocer las acciones primitivas.

La segunda parte de la resolución informática de un problema dado consiste en adaptar el algoritmo de manera que pueda ser reconocido por la máquina en la cual va a ser ejecutado. La noción algorítmica que hemos utilizado no es comprendida por los procesadores de una computadora. Dado un algoritmo, es necesario expresarlo en el lenguaje que utilice el procesador, para que pueda ejecutarlo. Es la finalidad de la codificación. Los objetos del entorno serán escritos con ayuda de las declaraciones expresadas con el vocabulario del lenguaje, conforme a las reglas de gramática que lo rigen. Cada una de las acciones primitivas que componen el algoritmo será codificada en una o varias instrucciones expresadas en este mismo lenguaje. La sucesión de declaraciones e instrucciones constituye el programa correspondiente al algoritmo.

Se pueden formular varias interrogantes:

¿Comprende la máquina varios lenguajes?

¿ Debe tener en cuenta la codificación las características de la máquina o solamente las del lenguaje elegido?

Lenguaje de programación es cualquier lenguaje artificial que puede utilizarse para definir una secuencia de instrucciones para su procesamiento por un ordenador o computadora (programa), son el vehículo de comunicación entre el hombre y la computadora.

Los lenguajes de programación se clasifican en:

- Lenguaje de máquina.
- Lenguaje de bajo nivel o ensamblador.
- Lenguaje de alto nivel.

Lenguaje de máquina

El microprocesador sólo entiende el lenguaje llamado Lenguaje-Máquina que está formado por un conjunto de instrucciones elementales, por lo general consisten en cadenas de números al final reducidos a ceros y unos (código binario, o sea dígitos 0 y 1). Es propio de cada computadora, ya que está relacionado con el diseño del hardware de la misma (dependiente de la máquina).

Ejemplo:

Código de operación	Dirección	Significado
00010101	10000001	Carga contenido de la dirección 129 en el acumulador
00010111	10000010	Sumar contenido de la dirección 130 al acumulador
00010110	10000011	Almacenar el contenido del acumulador en la dirección 131

La escrituras de los programas en Lenguaje-Máquina resulta larga y pesada. Con el fin de facilitar la tarea del programador, se dispone de otros lenguajes más fáciles de manejar. Estos programas no podrán ser ejecutados directamente por la máquina hasta después de haber sido traducidos en el lenguaje propio de esta máquina. El traductor del texto en un lenguaje determinado a un texto en Lenguaje- Máquina se llama compilador, y la acción de traducción que efectúa es la compilación. En el caso más simple que el lenguaje simbólico esté próximo al lenguaje máquina, se hablará de ensamblaje más que de compilación y de ensamblador más que de compilador.

Los lenguajes de programación han nacido de la necesidad de una comunicación más fácil entre el hombre y la máquina. El inconveniente del Lenguaje- Máquina, aparece muy rápidamente, a partir de la realización de los primeros ordenadores: por un lado la escritura de programas en tales lenguajes es poco práctica, por otro ésta sólo puede dirigirse a un ordenador dado; cada ordenador tiene su propio lenguaje, incompatible con el de otros modelos. Se han llegado a concebir lenguajes mejor adaptados al hombre y que permiten que un mismo lenguaje esté implementado en varias máquinas diferentes. El gran abismo que separa al hombre del ordenador no ha desaparecido de un día para otro. La adaptación de la máquina al hombre se ha producido progresivamente, una breve historia de los lenguajes de programación muestra esta evolución:

Lenguaje Ensamblador

En este lenguaje cada instrucción corresponde a una instrucción-máquina y las direcciones de memoria son designadas por un nombre simbólico y no por su valor. Esto permite una mayor flexibilidad en la escritura del programa. Estos lenguajes son específicos de un ordenador dado y su utilización requiere de un buen conocimiento de las características de la máquina.
Como los lenguajes ensambladores son dependientes de la máquina, todo programa escrito en un lenguaje ensamblador particular tendrá que ser reescrito si se va a ejecutar en otro tipo de computadora.

La computadora no entiende directamente el lenguaje ensamblador, por lo que un programa escrito en este lenguaje tiene que ser traducido al lenguaje de máquina por un programa llamado *ensamblador* para que pueda ser ejecutado por la computadora.

La escritura de programas en lenguaje ensamblador es larga y difícil; pero permite obtener programas más rápidos que los escritos en lenguajes de más alto nivel, con los cuales es difícil explotar al máximo ciertos recursos de la máquina.

Ejemplo de código en ensamblador:

Código de operación	Dirección	Instrucción del lenguaje ensamblador
00010101	10000001	LOAD A
00010111	10000010	ADD B
00010110	10000011	STORE C

Lenguaje de alto nivel

Permite a los programadores escribir instrucciones en un lenguaje más familiar para ellos y que contiene notaciones matemáticas comúnmente utilizadas (independiente de la máquina).
Ejemplo:

Código de operación	Dirección	Instrucción del lenguaje ensamblador	Instrucción en el lenguaje de alto nivel
00010101	10000001	LOAD A	
00010111	10000010	ADD B	C=A+B
00010110	10000011	STORE C	

Con este tipo de lenguajes, la programación es más fácil para los usuarios ya que este no necesita tener conocimiento de la arquitectura de la computadora.
Como ocurre con los lenguajes ensambladores, la computadora no entiende directamente lenguaje de alto nivel, por lo que un programa escrito en este lenguaje tiene que ser traducido a lenguaje de máquina por un programa llamado *compilador* para que pueda ser ejecutado por la computadora.
Como los lenguajes de alto nivel son independientes de la máquina, cualquier programa escrito en un lenguaje de alto nivel particular (que tiene una versión estandarizada) puede ser ejecutado en cualquier computadora.
Ejemplo de lenguajes de alto nivel:
Java
C
C++
COBOL
FORTRAN

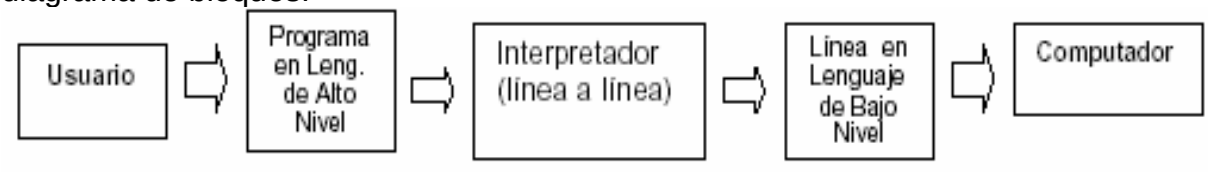
PROLOG
LISP
PL/I
SMALLTALK
ADA
VisualBasic

Delphi

Los lenguajes de alto nivel se clasifican en Interpretadores y compiladores

Lenguajes Interpretados

Son aquellos lenguajes de programación donde existe un programa interpretador que no es más que un programa que "coge" nuestro programa y lo convierte línea a línea a Lenguaje de Bajo Nivel y así mismo lo va ejecutando (o sea línea a línea). Estos lenguajes tienen de inconveniente que si el programa tiene un error en una de las últimas líneas solo cuando el interpretador llega hasta allá es que se detecta, luego de haberse ejecutado todo un gran bloque de instrucciones.
Podríamos esquematizar la utilización de lenguajes interpretados con el siguiente diagrama de bloques:

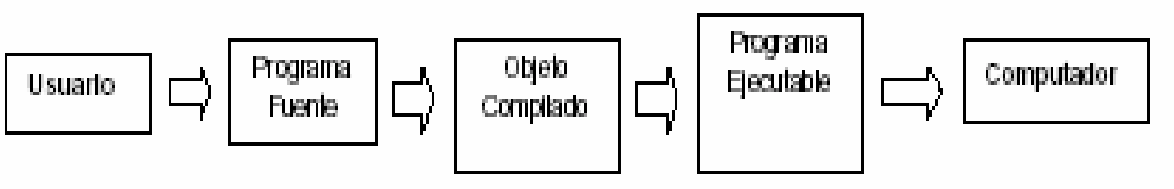


Se incorpora un elemento adicional como es el interpretador (línea a línea) que le facilita el trabajo al programador pues éste ya puede utilizar instrucciones más entendibles.

Lenguajes Compilados

La dificultad presentada por los Lenguajes Interpretados en cuanto al hecho de que convertían línea a línea el programa y así lo ejecutaban estableció el riesgo de que en programas comerciales, en donde es fácil encontrar 15000 o 20000 líneas en un programa, era demasiado costoso a nivel empresarial y demasiado riesgoso a nivel técnico saber que en cualquier momento por un error determinado el programa podía ser interrumpido (a abortado como técnicamente se le dice) debido a que el interpretador iba revisando línea a línea. Ello llevó a pensar en un esquema mucho más práctico y menos arriesgado tanto para programadores como para empresarios: los lenguajes compilados que son aquellos lenguajes donde un programa llamado compilador toma TODO el programa que hemos escrito (que normalmente se denomina Programa Fuente), lo revisa y solo hasta cuando esté completamente bien, solo hasta allí lo convierte a su equivalente en Lenguaje de Bajo Nivel para ser ejecutado por la computadora. De esta manera se reducía el riesgo de evaluación que representaban los Lenguajes Compilados y se podía saber si TODAS las instrucciones del programa eran correctas. De hecho era de suponer que, bajo condiciones normales, el programa entonces no se abortaría cuando estuviera en su momento de ejecución (cosa que no es 100% cierta). Lo que sí se puede decir es que la cantidad de interrupciones de los programas elaborados con Lenguajes

Compilados se redujo notoriamente frente a las interrupciones que sufrían los programas elaborados con Lenguajes Interpretados.
De allí que cuando se compila el programa fuente se genera un segundo programa (que solo es entendible por la computadora) conocido como Objeto Compilado y cuando se va a ejecutar entonces este programa es “organizado” de manera que cada instrucción sea perfectamente entendible y ejecutable por la computadora. A este nuevo programa se le conoce como Programa Ejecutable. De esta manera podemos resumir el proceso de utilización de los Lenguajes compilados a través del siguiente diagrama de bloques:



Estructura sintáctica y semántica de los lenguajes de programación

Características generales de la sintaxis

Los lenguajes de programación tienen un sistema de notaciones para el intercambio de información entre el programador y la computadora.
En la determinación y valoración de la sintaxis se tienen en cuenta cuatro características fundamentales: legibilidad, comodidad para la escritura,, facilidad para la compilación y no ambigüedad de la sintaxis del programa y de la interpretación del significado de este programa.

Legibilidad

Un programa es legible en la medida en que refleje en forma clara las estructuras de datos y algoritmos del programa. Así por ejemplo es más legible la instrucción ALGOL:
If B then I_1 else I_2 que la secuencia de instrucciones FORTRAN:
If .NOT. B Goto 1
I_1
Goto 2
1 I_2
2.....
aún cuando ambas significan lo mismo.
En una concepción moderna de la programación, esta legibilidad debe aspirar a hacer que el programa sea auto documentado: es decir que se explique por si solo sin ningún tipo de documentación adicional.
La legibilidad de un programa se facilita cuando la sintaxis del lenguaje incluye posibilidades tales como: instrucciones estructuradas, utilización de palabras claves, a la vez que una nomenclatura nemotécnica de estas, posibilidades de inclusión de comentarios, símbolos nemotécnicos para los operadores, longitud no acotada para los identificadores, etc. Por su puesto que todas las facilidades que se introduzcan en la sintaxis no pueden garantizarnos esta legibilidad porque no pueden liberarnos de una mala programación, al igual que una buena guitarra no garantiza un buen concierto.
La sintaxis aumenta la claridad de un programa, en la medida en que las diferencias en la sintaxis reflejan diferencias en la semántica. Así por ejemplo las diferencias entre una condición condicional, un ciclo y una instrucción selectiva tipo **case,** se reflejan en la sintaxis de la mayoría de los lenguajes de programación. Sin embargo, estas diferencias se reflejan más claramente en algunos lenguajes que en otros.
Hay lenguajes con sintaxis muy simple pero que no facilita la legibilidad del programa, porque debido a esta simplicidad cualquier recurso debe ser expresado en forma muy detallada. Ejemplos de lenguajes con esta características son el LISP y SNOBOL. En estos casos se dice que la sintaxis es frágil, en el sentido de que pequeños errores u omisiones en la sintaxis pueden representar serio errores en el significado. Así por ejemplo la, la omisión o inclusión en lugar equivocado de un paréntesis en LISP puede provocar una interpretación equivocada.

Facilidad para la escritura
Aquellas características de la sintaxis que pueden hacer un programa de fácil lectura, a menudo entran en contradicción con las facilidades para la escritura. La facilidad para la escritura presupone poder obviar una buena cantidad de símbolos y notaciones. Por ejemplo, la posibilidad de convenios de sintaxis que asumen determinadas interpretaciones por omisión, es decir implícitamente, no es conveniente desde el punto de vista de la claridad.
Algunos lenguajes son muy explícitos en la sintaxis, facilitándose la lectura, sin embargo son demasiado agotadores para la escritura de los programas. Debe buscarse un balance adecuado entre la facilidad para la escritura y la facilidad para la lectura a la hora de definir la sintaxis de un lenguaje.

Facilidad de traducción
Una tercera característica debe estar siempre presenta a la hora de decidir y valorar los aspectos anteriores: facilitar la tarea de traducción del programa a un código ejecutable.
Un elemento importante en esta facilidad para la traducción es la regularidad y la consistencia de las estructuras.

No ambigüedad
Al definir un lenguaje de programación se aspira a que sea tal que, una construcción hecha por el programador tenga una interpretación única.

Conjuntos básicos de símbolos de un lenguaje de programación
El conjunto básico de caracteres de un lenguaje de programación es el conjunto de símbolos que, en última instancia, componen el texto de un programa. Por lo general, este conjunto básico incluye las letras, los dígitos decimales y un conjunto de símbolos adicionales, como por ejemplo: “”,, _, *, (,), :, etc. Existen varios conjuntos estándares de caracteres, los más conocidos son ASCII e ISO que se diferencias fundamentalmente en el conjunto adicional de símbolos y en la codificación de estos.
Este conjunto básico de caracteres o símbolos se denomina **alfabeto** o **vocabulario** del lenguaje.

Gramática
La gramática es un mecanismo para describir un lenguaje. En este sentido una gramática para un lenguaje de programación sirve para que el usuario del lenguaje conozca las estructuras sintácticas de este y las pueda utilizar correctamente sin cometer errores

Elementos sintácticos básicos de los lenguajes de programación

identificadores
Para los identificadores la sintaxis más comúnmente utilizada es una cadena de letras y/o dígitos que empiezan con letra. En algunos lenguajes se establecen restricciones sobre la longitud de los identificadores. Esta restricción en longitud de los identificadores se debe

fundamentalmente al espacio dedicado en memoria para guardar el nombre del identificador.

Símbolo de las operaciones(operadores)

Para denotar los operadores aritméticos de suma y resta, en la mayoría de los lenguajes se utilizan los caracteres + y - , por lo general, el resto de los operadores, pueden variar bastante entre un lenguaje y otro. Lo más usual para la división es el símbolo /.

Los caracteres utilizados para los operadores dependen del hardware, es decir el conjunto de caracteres básicos disponibles. Algunos lenguajes de programación tienen un conjunto básico especial de caracteres.

Palabras claves

Las palabras claves se denotan esencialmente mediante identificadores y sirven para representar a operadores, acciones, objetos predefinidos en el lenguaje, etc. Por ejemplo: If, then, else, While, Repeat, y and en PASCAL ; DO, SUBROUTINE, COMMOND en FORTRAN.

Constantes

Las constantes son los valores numéricos o de cadena de caracteres invariables, las constantes de cadena se ponen entre comillas o apóstrofes generalmente. Ejemplo: 0,1 ,2 ,3, 4, 5; “cuba”.

Comentarios

En la mayoría de los lenguajes se tiene la posibilidad de incluir comentarios en los programas. El objetivo de los comentarios es facilitar la lectura y comprensión del programa. Se representan generalmente encerrando el contenido entre llaves, apóstrofes , asterisco y otros.

Delimitadores

Los delimitadores se utilizan, por lo general para mejorar la legibilidad del programa y para facilitar el análisis sintáctico y semántico, por ejemplo para que el análisis se haga de determinada forma, como los paréntesis en las expresiones.

Los delimitadores más usados: la coma por lo general listas de identificadores o expresiones; el punto y coma utilizado para separar instrucciones como en Pascal; dos puntos utilizado para separar un nombre de un objeto (en Pascal, en Basic se utiliza para escribir dos instrucciones en la misma línea). En algunos casos son utilizadas palabras claves como delimitadores como es el caso de Bejín y End en ALGOL y Pascal que sirven para agrupar a un conjunto de instrucciones.

Las reglas sobre el uso del espacios varía de un lenguaje a otro. Lo más común es el uso anterior del espacio como separador de identificadores.

Los lenguajes con formato libre permiten que las instrucciones puedan ser ubicadas en cualquier parte de la línea. En los lenguajes con formato fijo los símbolos en cada instrucción deben ocupar una determinada posición en la línea; por ejemplo en FORTRAN las cinco posiciones están reservadas para las etiquetas y si la instrucción no tiene etiqueta estas cinco líneas se dejan en blanco.

Evolución de los paradigmas de programación

Como todas las ramas del conocimiento humano, la programación también ha ido avanzando haciendo que ésta sea cada vez más simplificada para el programador y brindando día a día más herramientas técnicas que permitan la utilización de las computadoras de una manera sencilla y simplificada. También se han experimentado algunos cambios en cuanto a la concepción del mundo de la programación y sus efectos y utilización en el mundo de la informática.

Cuando comienzan las computadoras o lo que en esos tiempos era una computadora no existían, como era obvio, las facilidades tecnológicas que hoy existen, razón por la cual el concepto de programación se realizaba a través de circuitos eléctricos y electrónicos directamente de tal manera que los "programadores" de aquellos tiempos eran unos expertos en electrónica ya que las soluciones las construían con partes como tales. Esa programación se conoció como ***Programación Directa ó Real*** dado que el contacto entre el programador y la máquina era directo y requería un altísimo conocimiento técnico no solo de partes electrónicas sino también de lo que en ese entonces era la programación a bajo nivel.

Poco a poco la tecnología fue avanzando permitiendo, en este campo, que el ser humano tuviera cada vez más y mejores herramientas de trabajo. Fue entonces cuando se pensó en la programación tal y como se concibe en el día hoy, es decir, permitir que a través de órdenes dadas a la computadora éste pudiera realizar tareas a altas velocidades. Este concepto comenzó a ser trabajado y poco a poco empezaron a surgir en el mercado Lenguajes de programación como tales. Estos Lenguajes permitían realizar un gran número de tareas con la simple utilización correcta de determinadas instrucciones. La metodología de la utilización de estas instrucciones fue en esos tiempos algo muy libre razón por la cual a esta etapa de la programación se le conoció como ***Programación Libre***. Bajo esta técnica de programación, la persona que estuviera al frente de la computadora podía realizar todas las tareas que pudiera o más bien que el lenguaje le permitiera basado solamente en su lógica propia aplicada a la libre utilización de dichas instrucciones. A pesar de que en principio esta forma de utilizar los Lenguajes de Programación fue la solución para muchos problemas y de que el mundo había comenzado a ser más eficiente en el tratamiento de la información gracias precisamente a la utilización de órdenes para programar las computadoras, los problemas no esperaron para dejarse venir.

Cuando un programador se sentaba con su lógica propia a resolver un problema utilizando las instrucciones que un Lenguaje de Programación le permitía muchas veces (y casi siempre) llegaba a soluciones que solamente él entendía y cuando este programador era sacado de la empresa o se retiraba o se moría entonces la empresa se veía en la penosa obligación de conseguir otro programador que en la mayoría de los casos lo que hacía era volver a hacer todo lo que el primero había hecho pero con su propia lógica quedando la empresa en manos de este nuevo programador y teniendo previsto el gran problema que se originaría cuando éste se retirara o se muriera.

Fue allí en donde comenzó a pensarse en la Lógica Estructurada de Programación o más bien se comenzó a pensar que los programas por diferentes que fueran obedecían a una serie de normas que eran comunes en cualquier algoritmo, término que se comienza a acuñar técnicamente en esa época. A través de muchos estudios se llegó a la conclusión

que la lógica de programación se basaba solo en tres estructuras como son las secuencias, las decisiones y los ciclos. Se puso a prueba esta teoría y se descubrió que era cierta pues ningún algoritmo se salía de estas tres estructuras.
Pensar en unas estructuras básicas del pensamiento al momento de la programación facilitó enormemente el hecho de que el programa desarrollado por un programador fuera entendido sin mayores complicaciones por otro. También esta forma de programación restringió el desorden de algunos programadores porque le colocó unos límites a la lógica computacional que era la que había que utilizar cuando se necesitara escribir un programa. A esta forma de trabajo se le llamó ***Programación Estructurada*** que no es más que la técnica a través de la cual se utilizan los Lenguajes de Programación utilizando las estructuras básicas y permitiendo que los programas sean mucho más entendibles ya que no son concebidos al libre albedrío del programador sino basado en unas normas técnicas. Esta técnica de programación comenzó a tomar mucha fuerza en el desarrollo de la programación debido precisamente a que ya un programador podía tomar los programas de otro y entenderlos con muchísima facilidad. Se desarrollaron Lenguajes que permitieran precisamente la sana utilización de estas estructuras y a éstos se les llamó Lenguajes Estructurados. Estos lenguajes casi obligaban al programador a no salirse del marco conceptual de las estructuras básicas.
Como ya se dijo, la Programación Estructurada por su claridad en la codificación, permitía que un programador pudiera modificar el programa hecho por otro. De ahí surgió la conveniencia de dividir el programa en partes independientes, en los cuales pudieran trabajar distintos programadores por separado. Es así como surge la **Programación Modular** que plantea la necesaria división del problema principal en partes independientes llamadas módulos, de manera que estos puedan ser analizados y programados por separado e incluso por distintas personas.
Un módulo es una función, un procedimiento o el cuerpo de un programa que realiza una función perfectamente definida.
Los lenguajes modulares (Ada, Modula2..) tienen una definición más fuerte de lo que es un módulo. Consideran que es una unidad compilable de forma independiente y que posee una interface oficial que consiste en la declaración de las facilidades del módulo visibles a otros módulos, también posee detalles de implementación invisibles al mundo externo.
La máxima expresión de la programación modular se logra con la creación de Unit y Bibliotecas.
El mundo y el ser humano ávido de soluciones para sus necesidades utilizaron esta técnica de programación estructurada por mucho tiempo sin cuestionarla, hasta que las mismas necesidades de programación comenzaron a cuestionar lo que hasta ese momento había funcionado tan perfectamente. Se partió de la teoría que la programación no es más que una interpretación del mundo real y su simulación a través de una computadora por tal motivo se pensó en aproximar mucho más los conceptos de programación al mundo real y fue allí en donde se encontró que todo lo que nos rodea son objetos que tienen características y sirven para algo. Por ejemplo un lápiz tiene peso, color, longitud, espesor, textura y muchas otras características. Al mismo tiempo un lápiz sirve para escribir, para separar una hoja de un libro, para señalar un punto, para dibujar etc. Esta concepción llevó a una gran revolución en la historia de la programación pues se crearon dos vertientes dentro de la lógica de programación: La programación estructurada que ya definimos y la ***Programación Orientada A Objetos*** por medio de la cual se podía

modelar el mundo en la computadora tal y como es. Su aporte principal era el concepto de objeto. Qué es un objeto..?
En términos generales un objeto no es más que un ente informático que tiene características (técnicamente llamadas atributos) y que sirve para algo (técnicamente se dice que tiene métodos).
Así se creó este concepto y se comenzaron a utilizar los objetos (en programación) que como ya dijimos no son más que tipos de datos con atributos y métodos propios. Toda una teoría se comenzó a derivar de esta nueva concepción del mundo y se fue aplicando poco a poco en la programación. Se empezaron a descubrir relaciones entre objetos, operaciones entre objetos y en general conceptos nuevos en cuanto a lo que inicialmente no habían sido más que los objetos. Mientras se desarrollaba esta teoría y se ponía en práctica en muchos de los Lenguajes de Programación comerciales también se seguía utilizando la técnica de programación estructurada pues estas dos técnicas no eran excluyentes. Dependía del programador que tuviera una verdadera concepción acerca del problema que quería solucionar la decisión de saber por cuál técnica de programación (programación estructurada o programación orientada a objetos) era más apropiado resolverlo.
Ello exigía simultáneamente que el programador no solo conociera muy bien los conceptos de programación sino que también conociera muy bien el problema que iba a solucionar y las características de cada una de las técnicas de programación. Tenía que ser un profesional integral de la programación pues ahora no solo se necesitaba que supiera de computadoras o de electrónica o que se supiera las instrucciones de un simple lenguaje. Ahora tenía que conocer teoría y combinarlas de manera que pudiera llegar a la mejor solución aprovechando la tecnología existente. Cuando comenzó a tomar fuerza la teoría de la programación orientada a objetos no todos los Lenguajes de Programación (o más bien no todos sus compiladores) estaban acondicionados para que aceptaran la nueva forma de programar.
De esta manera también era necesario que el programador supiera si el problema que iba a solucionar a través de un programa era implementable fácilmente con el Lenguaje de Programación que tuviera a la mano pues debe saber que no es fácil inducir la compra de un Lenguaje de Programación (o sea de su compilador) en una empresa cuando todo el sistema de información está basado en otro Lenguaje de Programación. Esta filosofía de programación fue cogiendo mucha fuerza y con ella se fueron fortaleciendo los lenguajes que habían iniciado la aceptación de esas nuevas características. Empresas fabricantes que hasta ese momento habían sido competitivas se convirtieron en verdaderos imperios de la informática. La programación definitivamente había dado un salto impresionante hacia la solución de muchos problemas que eran, en algunos casos, más complejos de resolver con programación estructurada que con programación orientada a objetos.
Poco a poco algunos fabricantes de Lenguajes de Programación se fueron introduciendo en el mercado y aprovechando las características de la nueva técnica de programación fueron dejando de lado, de alguna manera, la programación estructurada que algunos libros han llamado erróneamente Programación Tradicional. En ese avance tecnológico y con el ánimo de entregar al mercado de la programación más y mejores herramientas de trabajo se empezó a manejar un concepto muy importante en programación como es el concepto de interfaz. Una interfaz no es más que la forma como puede mostrar la información por medio de algún dispositivo de salida. Ya se sabía que entre más clara y entendible fuera la información podría decirse que los programas serían mejores ya que lo

que finalmente el usuario de un programa necesitaba era que la información que le arrojaba una computadora fuera claramente entendible.
Se fue notando, por parte de las empresas fabricantes de Lenguajes de computadoras, como el tiempo de un programador se iba en su mayor parte en el diseño de las interfaces o sea en el diseño de la presentación de los datos. Por tal motivo se pensó que, en unión con la teoría de programación orientada a objetos y con las herramientas que ella facilitaba, se hacía necesario diseñar lenguajes de programación que facilitaran el diseño de interfaces para que el programador invirtiera su tiempo mejor en el diseño de procesos o de manipulación y tratamiento de datos. Fue entonces cuando entraron al mercado los Lenguajes Visuales y se incorporó al desarrollo de la programación la ***Programación Visual*** que no es más que una forma de programar en donde se cuenta con una gran cantidad de herramientas prediseñadas para facilitar, precisamente, el diseño de interfaces. Este tipo de programación ha llevado a que en el mundo de la informática y exactamente en la programación se llegue a unos resultados mucho más convenientes y mejores a nivel técnico pues en la actualidad se pueden obtener aplicaciones de computadoras mucho más entendibles y manejables por el usuario gracias a la filosofía incorporada por la Programación Visual.
A este nivel la programación requería menos conceptos técnicos y más lógica de programación que era lo que realmente se necesitaba para desarrollar un programa. Es normal ver como una persona con unos modestos conocimientos de computación puede, a través de Lenguajes Visuales, desarrollar aplicaciones verdaderamente útiles y además muy bien presentadas. Lo que poco a poco se fue afianzando fue la necesidad de tener unos conceptos de lógica de programación bien fundamentados para poder aprovechar de una manera eficiente los recursos que la informática le entregaba a la computadora.
Como se busca modelar con la computadora al mundo que nos rodea y en ese avance la tecnología cada vez se ha ido mejorando más y más se espera que dentro de muy poco se podrá hablar de una ***Programación Virtual*** en donde el programador pueda ver en tres dimensiones (3D) todo el escenario que necesita para crear sus aplicaciones.

Actualmente hay una gran cantidad de lenguajes de programación, vamos a citar cronológicamente los más conocidos por su amplia utilización y por el papel que han jugado en la evolución de la programación.

1. ForTran (FORmula TRANslator) traductor de fórmulas creado por IBM en 1956 para el microprocesador IBM 704. Este lenguaje se extendió a todos los ordenadores. En el 58 se dio una versión Fortran II y en el 62 Fortran IV. Fue uno de los lenguajes más conocido y más utilizado, principalmente para la resolución de problemas de cálculo numérico. Perduró durante 25 años aproximadamente a pesar de la nuevas posibilidades que brindaban lenguajes más jóvenes.
2. Algol 60 (Algorithmic Language) lenguaje algorítmico, desarrollado en 1960 por un comité internacional de informáticos, se introdujeron nuevos conceptos de programación y por ello influyó en la concepción de los lenguajes que le siguieron.
3. Cobol(Common Business Oriented Language) Lenguaje común orientado a negocios, compañías. Nació de las reflexiones de un grupo de usuarios y de constructores de ordenadores en 1960 a iniciativa de la defensa nacional americana para definir un lenguaje que permitiera escribir programas compresibles para los no informáticos. Se extendió ampliamente su uso.
4. APL (A Programming Language) lenguaje de programación, fue desarrollado en 1962 por K. Iverson. Se implementó en sistemas interactivos, es decir que permiten el diálogo entre el usuario y la máquina. Este lenguaje utiliza una gran cantidad de símbolos y caracteres especiales con un significado muy particular; permitiendo expresar diferentes acciones con un mínimo de signos. Las estructuras algorítmicas de este lenguaje son muy diferentes a las de los otros. Fue muy criticado por la ausencia de las estructuras de control y por consiguiente por la oscuridad que se deriva de ello. Es muy poderoso para los cálculos vectoriales y matriciales.
5. PL/1 (Programming Languag 1) concebido por IBM en 1964 para el microprocesador IBM 360 con la finalidad de resolver, a la vez, los problemas científicos y de gestión. Reúne los conceptos más significativos de los lenguajes que le han precedido(Fortran, cobol, Algol 60). No tuvo mucho éxito debido a su complicación.
6. BASIC (Beginner's All purpose Symbolic Instruction Code) Código de instrucciones simbólicas para todo principiante. Fue desarrollado en 1965 en Dartmuth College, es un lenguaje muy fácil de aprender, por esta razón está muy extendido.
7. Algol 68 fue desarrollado en 1968 por un grupo de trabajo que estaba designado para darle mantenimiento al Algol 60 y cuyo objetivo fue producir un lenguaje universal. No fue más importante que el Algol 60 y no fue muy utilizado.
8. Pascal (llamado así en homenaje al filósofo francés Blaise Pascal). Fue concebido en 1969 por Niklaus Wirth. Destinado en principio a la enseñanza de la programación, se hizo rápidamente muy popular, no solo en la universidades, sino también en la industria. Se encuentran en Pascal numerosos conceptos del algol 60.
9. Ada (en homenaje a ada Lovelace, hija del poeta Byron y amiga de Babbage). Fue desarrollado por un equipo bajo la dirección Jean Ichbiah, en el marco de un contrato con el Ministerio de Defensa de los Estados Unidos. Fue destinado a ser empleado en pilotaje de aviones, de aparatos espaciales etc.

LENGUAJE	ORIGEN DEL NOMBRE	AÑO	USOS/COMENTARIOS
Lenguajes de programación			
FORTRAN	FORmula TRANslation (Traducción de fórmulas)	1954	Diseñado en un principio para usos científicos y de ingeniería, se trata de un lenguaje compilado de alto nivel que hoy se utiliza en numerosos campos. Precursor de diversos conceptos, como las variables, las instrucciones condicionales y las subrutinas compiladas por separado.
COBOL	COmmon Business-Oriented Language (Lenguaje simbólico de programación orientado a aplicaciones comerciales)	1959	Lenguaje de programación semejante al idioma inglés, que hace hincapié en las estructuras de datos. De amplia utilización, principalmente en empresas.
ALGOL	ALGOrithmic Language (Lenguaje ALGOL algorítmico)	1960	Primer lenguaje de programación procedural estructurado, utilizado sobre todo para resolver problemas matemáticos.
LISP	LISt Processing (Procesamiento de listas)	1960	Lenguaje de programación orientado a la generación de listas, utilizado principalmente para manipular listas de datos. Lenguaje interpretado que suele utilizarse en las investigaciones y está considerado como el lenguaje estándar en proyectos de inteligencia artificial.
APL	A Programming Language (Un lenguaje de programación)	1961	Lenguaje interpretado que utiliza un amplio conjunto de símbolos especiales y que se caracteriza por su brevedad. Utilizado fundamentalmente por los matemáticos.
PL/1	Programming Language 1 (Lenguaje de programación uno)	1964	Diseñado para combinar las principales virtudes del FORTRAN, COBOL y ALGOL, se trata de un lenguaje de programación complejo. Compilado y estructurado, es capaz de gestionar errores y de procesar multitareas, y se emplea en entornos académicos y de investigación.
BASIC	Beginners All-Purpose Symbolic Instruction Code(Código de instrucciones simbólicas multipropósito para principiantes)	1965	Lenguaje de programación de alto nivel, utilizado con frecuencia por programadores principiantes.
LOGO	Derivado del griego logos, 'palabra'.	1968	Lenguaje de programación que suele utilizarse con niños. Presenta un sencillo entorno de dibujo y varias prestaciones de mayor nivel del lenguaje LISP. Fundamentalmente educativo.
PILOT	Programmed Inquiry, Language Or Teaching (Consulta, lenguaje o aprendizaje de investigación programada)	1969	Lenguaje de programación utilizado fundamentalmente para crear aplicaciones destinadas a instrucciones asistidas por computadora. Se caracteriza por utilizar un mínimo de sintaxis.
FORTH	Lenguaje de cuarta (FOuRTH) generación	1970	Lenguaje estructurado e interpretado de fácil ampliación. Ofrece una alta funcionalidad en un espacio reducido.
Pascal	Blaise PASCAL, matemático e inventor del primer dispositivo de computación.	1971	Lenguaje compilado y estructurado basado en ALGOL. Agrega tipos y estructuras de datos simplificando la sintaxis. Al igual que el C, se trata de un lenguaje de programación estándar para microcomputadoras.
C	Predecesor del lenguaje de programación B, fue desarrollado en Bell Laboratory, en 1972	1972	Lenguaje de programación compilado y estructurado, que suele utilizarse en numerosos lugares de trabajo porque sus programas pueden transferirse fácilmente entre distintos tipos de computadoras.
ADA	Augusta ADA Byron (Lady Lovelace)	1979	Derivado de Pascal, utilizado principalmente por los militares.
Modula-2	MODUlar LAnguage-2, diseñado como fase secundaria de Pascal (diseñados ambos por Niklaus Wirth)	1980	Lenguaje que hace hincapié en la programación modular. Es un lenguaje de alto nivel basado en Pascal, que se caracteriza por la ausencia de funciones y procedimientos estandarizados.

Ha podido notar que en el mundo actual existe un despliegue de tecnología que no solo facilita sino que también condiciona la vida del ser humano. Los lenguajes de programación no son la excepción dentro de esa carrera comercial en la cual se encuentran inmersas todos fabricantes de tecnología. Con gran frecuencia se encuentra uno con la decepción de que los lenguajes de programación que existen en el mercado son modificados y, teóricamente, mejorados con tanta vertiginosidad que uno escasamente alcanza a conocer algunas de sus bondades cuando ya las están cambian por otras.

Por eso no hay que preocuparse de que la tecnología va avanzando mucho más rápido de cómo avanzan nuestros conocimientos, pues lo que nos corresponde es tratar de estar a la vanguardia en dichos avances. Cómo se hace para estar al día con todos los lenguajes de programación..?

Pues si en algún momento ha tenido este propósito es verdaderamente imposible.

Permanentemente trate de mantenerse actualizado en cuanto a los avances del Lenguaje de Programación con el que usualmente trabaja. Sin embargo tenga en cuenta que por más que se estudie es muy posible que siempre esté un poco rezagado de las actualizaciones de los lenguajes de programación dado que esta tecnología nos llega luego de que ha pasado algún tiempo de prueba y otro de comercialización en los países industrializados. La ventaja de trabajar con un solo lenguaje de programación es que de una u otra forma estará, por la simple necesidad de uso del mismo, mucho más actualizado que lo norma.

Ejemplos de codificación de algoritmos en diferentes Lenguajes.

1. Hacer un algoritmo para saber si un número es par.

Seudo código:

```
Algoritmo NúmeroPar
 Inicio
        Entorno:
        Num: variable entera
        Leer num
        Si num < 0 entonces
                Escribir  “El número debe ser positivo”
                SinoSi num / 2 * 2 = num entonces
                     Escribir  “El número leído es par”
           Sino
             Escribir “El número leído no es par”
        Finsi
Fin
```

Codificación en Visual Basic

```
Sub NumeroPar( )
        Dim Sum as integer
        num=InputBox(“Entre un número”)
        If num<0 then
                MsgBox  “El número debe ser positivo”
                Elseif num/2*2=num then
                BsgBox “El número leído es par”
           Else
                BsgBox “El número leído no es par”

        End if
End Sub
```

Codificación en Pascal

```
Program NumeroPar;
      Var
       num: integer;
       Begin
            Readln(num);
               If  (num<0) then
                   Writeln('El número debe ser positivo´);
                       Else
                           If (num/2*2=num) then
                              Writeln ('El número leído es par');
                                   Else
                                   Writeln('El número leído no es par´);
End.
```

Codificación en C

```
#incude <iostream.h>
void main()
{
        int num;
        cin>>num;
        If num(<0)
                Cout << "El número debe ser positivo";
        Else
                If (num/2*2 = = num)
                        Cout << "El número leído es par";
                Else
                        Cout << "El número leído no es par";
}
```

Codificación en Cobol

```
IDENTIFICATION DISION
PROGRAM_ID. NUMERO_PAR.

ENVIROMENT DISION.
CONFIGURATION SECTION.
SOURCE –COMPUTER. CLON.
OBJECT-COMPUTER. CLON

DATA DIVISION.
WORKING-STORAGE SECTION.
01 NUM        PIC 99.
PROCEDURE DIVISION.
INICIO.
      ACCEPT NUM LINE 10 POSITION 10 NO BEEP
      IF NUM IS LESS THEN 0 THEN
         DISPLAY “EL NUMERO LEIDO DEBE SER POSITIVO” LINE 12 COL 10
     ELSE
         IF NUM/2*2 IS EQUAL TO NUM THEN
             DISPLAY “EL NUMERO LEIDO ESPAR” LINE 12 COL 10
         ELSE
              DISPLAY “EL NUMERO LEIDO NO ES PAR” LINE 12 COL 10
STOP RUN.
```

2. Haga un algoritmo para saber si los últimos dos dígitos de dos números enteros son iguales.

seudo código

```
Algoritmo Campara_Ult_Digs
Inicio
        Entorno:
        num1, num2, ud1,ud2 variables enteras
          Escribir “digite dos números”
         Leer num1
        Leer num2
        Si num1<0 entonces
                num1← num1*(-1)
                SinoSi num2<0 entonces
                        Num2←num2*(-1)
        Finsi
        Ud1←num1 – (num1\10)*10
        Ud2← num2 – (num2\10)*10
        Si ud1=ud2 entonces
                Escribir “el último dígito de un número es igual al último dígito del otro”
                Sino
                    Escribir “el último dígito de un número no es igual al último dígito del otro”

        Finsi
Fin
```

Codificación en Visual Basic

```
Sub Campara_Ult_Digs
        Dim num1 as integer, num2 as integer, ud1 as integer, un2 as integer
        Print “digite dos números”
        Num1=inputBox( )
        Num2=InputBox( )
        If num1<0 then
                Num1=num1*(-1)
                ElseIf  num2<0 then
                        Num2=num2*(-1)
        End if
        Ud1=num1- (num1\10)*10
        Ud2=num2- (num2\10)*10
        If ud1=ud2 then
                Print el último dígito de un número es igual al último dígito del otro”
                Else
                    Print “el último dígito de un número no es igual al último dígito del otro”
        End if
End Sub
```

Codificación en Pascal

```
Program compara_Ult_digs;
        Var
        num1,num2,ud1,ud2: integer;
        Begin
                Writeln (‘Digite dos números enteros?);
                Readln (num1,num2);
                If num1< 0 then
                        num1:=num1*(-1);
                If num2< 0 then
                        num2:= num2*(-1);
                ud1:= num1-(num1\10)*10;
                ud2:= num2-(num2/10)*10;
                If ud1= ud2 then
                        Writeln (‘El último dígito de un número es igual al último del  otro’);
                Else
                        Writeln (‘El último dígito de un número no es igual al último del  otro’);
End.
```

Codificación en C

```
#include <iostream.h>

vold main()
}
        int num1, num2, ud1,ud2
        coul<< “Digite dos números”;
        cin >> num1, num2;
        if (num1<0)
                num1=num1*(-1);
        if (num2<0)
                num2=num2*(-1);
        ud1=num1-(num1\10)*10;
        ud2=num2-(num2\10)*10;
        if (ud1= = ud2)
                cout << “El último dígito de un número es igual al último del  otro”;

        else
                cout << “El último dígito de un número no es igual al último del  otro”;
{
```

Codificación en Cobol

```
IDENTIFICATION DISION
PROGRAM_ID. COMPARA_ULT:GIGS.

ENVIROMENT DISION.
CONFIGURATION SECTION.
SOURCE –COMPUTER. CLON.
OBJECT-COMPUTER. CLON

DATA DIVISION.
WORKING-STORAGE SECTION.
01 NUM1     PIC 99.
01 NUM2     PIC 99.
01 UD1      PIC 99.
01 UD2      PIC 99.

PROCEDURE DIVISION.
INICIO.
        DISPLAY “DIGITE DOS NÚMEROS ENTEROS” LINE 10 COL 10
        ACCEPT NUM1
        ACCEPT NUM2
        IF NUM1 IS LESS THEN 0 THEN
                COMPUTE NUM1=NUM1*(-1)
        IF NUM2 IS LESS THEN 0 THEN
                COMPUTE NUM2=NUM2*(-1)

        COMPUTE UD1=NUM1-(NUM1\10)*10
        COMPUTE UD2=NUM2-(NUM2\10)*10

        IF UD1 IS EQUAL TO UD2 THEN
                DISPLAY  “El último dígito de un número es igual al último del  otro” LINE 20
                COL 10
        ELSE
                ESCRIBA “El último dígito de un número no es igual al último del  otro”; LINE
                20 COL 10
                STOP RUN.
```

3. Dado un número entero escribir cuántos dígitos tiene.

Seudo código

```
Algoritmo Cuenta_Dígitos
Inicio
        Entorno:
        Num, cd var enteras
        Escribir “digite un número entero”
        Leer num
        Cd← 0
        Mientras num <>0
                Num← num/10
                Cd← cd+1
        FinMientras
        Escribir “ El número digitado tiene” cd “dígitos”
Fin
```

Codificación en Visual Basic

```
Sub Cuenta_Dígitos
        Dim num as integer, cd as integer
        Print “Digite un número entero”
        Num=Inputbox( )
        Cd=0
        While num<>o
                Num=num/10
                Cd=cd+1
        Wend
        Print “ El número digitado tiene”; cd; “dígitos”
End Sub
```

Codificación en C

```
#include <iostream.h>

vold main()
}
        int num, cd;
        cout<< “digite un número”;
        cin>> num;
        cd=0;
        while (num ! =0)
        {
                num=num/10;
                cd ++;
        }
        cout << “El número digitado tiene “<< cd << “dígitos”;
}
```

Codificación en cobol

```
IDENTIFICATION DISION
PROGRAM_ID. CUENTA-DIGITOS

ENVIROMENT DISION.
CONFIGURATION SECTION.
SOURCE –COMPUTER. CLON.
OBJECT-COMPUTER. CLON

DATA DIVISION.
WORKING-STORAGE SECTION.
01 NUM       PIC 99.
01 CDPIC 99.
01 UD1       PIC 99.

PROCEDURE
INICIO.
        DISPLAY “DIGITE UN NÚMERO ENTERO “ LINE 10 C0L 10
        ACCEPT NUM LINE 10 COL 50

        COMPUTE CD=0
        PERFORM CONTEO UNTIL NUM=0.
        DISPLAY “EL NUMERO DIGITADO TIENE” LINE 14 COL 10
        DISPLAY CD                              LINE14 COL 10
        DISPLAY “DIGITOS”
        STOP RUN.
CONTEO
COMPUTE NUM=NUM/10
ADD 1 TO CD
```

En la demostración que se ha hecho con la codificación en estos tres lenguajes puede notar que la utilización de cualquier lenguaje de programación se reduce casi a reemplazar algunas palabras claves por las que corresponden en el Lenguaje pero la lógica como tal permanece allí intacta, es decir que en el fondo, la utilización de cualquier Lenguaje es lo mismo sin importar cuál sea.
Siempre tenga en cuenta que lo importante, lo realmente importante, lo verdaderamente importante es la lógica con la cual usted desarrolle sus algoritmos.

Bibliografía

1. Joëlle Biondi y Gilles Clavel, Introducción a la Programación. Editora Masson, SA, 1985.
2. Trejos Buriticá, Omar Ivan La Esencia de la Lógica de Programación .Obra de Editorial Papiro, 1999
3. Miguel Katrib Mora, Lenguajes de Programación y técnicas de compilación. Editora. Pueblo y Educación, 1988.
4. Daniel D. McCracken, Métodos Numéricos y Programación FORTRAN. EditoraRevolucionaria, 1967.
5. A. Mesa Enriquez, Introducción a la programación en PL/1. Editora Pueblo y Educación, 1984.
6. A. Tijonov, Conferencias de Introducción a las Matemáticas Aplicadas. Editorial Mir, 1989.
7. A. I. Saltykov, G.L. Semashko, Programación Para todos. Editorial Mir, 1989.
8. A. Krinitski, Algoritmos a Nuestro Alrededor. Editorial Mir, 1984. BARANOVY, S.P. Pedagogía.__ La Habana: Ed. Revolucionaria, 1989.
9. BIXIO, CECILIA. Enseñar a aprender.__ Buenos Aires: Ediciones Homo-Sapiens, 1998.
10. CASTILLEJO, J.L. Efectos de la Informática en la Estructura Cognitiva de los alumnos.__ En Educar para el Siglo XXI. España, 1998.
11. JIMENO SACRISTÁN, J. Teoría de la Enseñanza y desarrollo del currículo.__ Madrid: Ed. Anaya, S.A, 1987.
12. La informática en la educación.__ En Zona educativa No 20. Argentina, 1998.
13. MITJÁNS MARTÍNEZ, A. Pensar y crear.__ La Habana: Educación. Academia, 1995.
14. NÉRESI, I.G. Hacia una didáctica general dinámica.__ Buenos Aires: Ed. Monta, 1996.
15. PETER, J. La reflexión. Un concepto clave en la educación de profesores.__ En Revista de educación Nro. 282.__ España, enero-abril, 1987.
16. PISONI, FERNÁNDO J. La aventura de aprender a aprender informática.__ En Aula Hoy No 12. Argentina, 1998.
17. Talízina, N. Formación de los modos de actividad cognoscitiva.__ P. 201-222.__ En Psicología de la Enseñanza.__Moscú: Ed. Progreso,1988.
18. VAQUEROS SÁNCHEZ, Antonio. La tecnología en la educación. TIC para la enseñanza, la formación y el aprendizaje.__ Madrid, 1998.
19. Oruzep, Lógica Dialéctica. Moscú: Ed. Progreso,1988
20. Andreiev, Problemas Lógicos del Pensamiento Científico. Moscú: Ed. Progreso,1988

Índice

Instituto Superior Pedagógico
“Pepito Tey”
Las Tunas

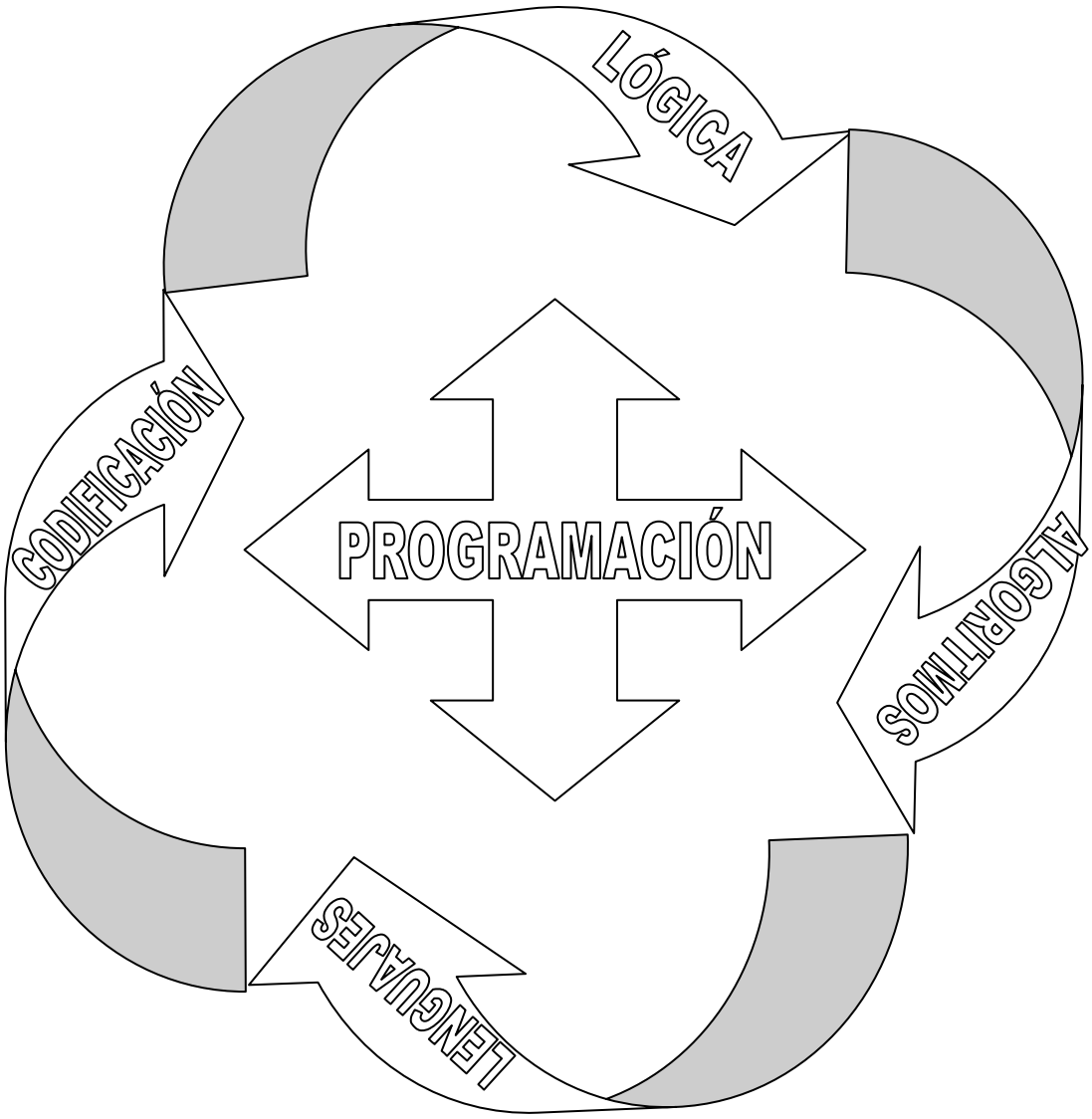
Facultad de Ciencias
Departamento de Informática

Lógica de Programación

Autor: Lic. Félix Tamayo Silva

Octubre del 2002.

Lógica de Programación



Félix Tamayo Silva