

**UNIVERSIDAD DE LA AMAZONIA**

**FACULTAD DE INGENIERIA  
DEPARTAMENTO DE EDUCACIÓN A DISTANCIA**

**PROGRAMA  
TECNOLOGÍA EN INFORMÁTICA Y SISTEMAS**

**COMPILADO UNIDAD TEMÁTICA  
LÓGICA Y ALGORITMOS I**

**PREPARADO POR  
YOIS S. PASCUAS RENGIFO  
Ingeniera de Sistemas  
Magíster en Ciencias de la Información y las  
Comunicaciones  
y.pascuas@udla.edu.co**

**Julio 2015**

## TABLA DE CONTENIDO

<b>INTRODUCCIÓN</b>	<b>4</b>
<b>1. CONCEPTUALIZACIÓN</b>	<b>5</b>
1.1 PROGRAMACIÓN	5
<b>1.1.1 LÓGICA</b>	7
1.2 ALGORITMOS	7
1.2.1 CARACTERÍSTICAS DE LOS ALGORITMOS	8
1.2.2 PARTES DE UN ALGORITMO	9
1.2.3 METODOLOGÍA PARA LA SOLUCIÓN DE PROBLEMAS	10
1.2.4 TÉCNICAS DE REPRESENTACIÓN DE ALGORITMOS	13
1.2.5 EJERCICIOS PRÁCTICOS DE ALGORITMOS	16
1.2.6 SCRATCH	22
<b>2. TIPOS DE DATOS</b>	<b>25</b>
2.1 CONSTANTES	28
2.2 VARIABLES	28
2.3 EXPRESIONES	30
2.3.1 EXPRESIONES ARITMÉTICAS	32
2.3.2 EXPRESIONES RELACIONALES	33
2.3.3 EXPRESIONES LÓGICAS	34
2.4 FUNCIONES	34
<b>3. TÉCNICAS PARA EL DISEÑO DE ALGORITMOS</b>	<b>36</b>
3.1 ALGORITMOS VORACES	36
3.1.1 FORMA GENERAL	36
3.1.2 EJERCICIOS PROPUESTOS	37
3.2 DIVIDIR Y CONQUISTAR	39
3.2.1 FORMA GENERAL	39
3.2.3 PROBLEMAS PROPUESTOS	40
3.3 ALGORITMOS EXHAUSTIVOS ( <i>RETROCESO, ENSAYO Y ERROR</i> )	43
3.3.1 FORMA GENERAL	43
3.3.2 PROBLEMAS PROPUESTOS	44
3.4 ALGORITMOS DE VUELTA ATRÁS ( <i>BACKTRACKING</i> )	45
3.4.1 PROBLEMAS PROPUESTOS	46
3.5 ALGORITMOS PARALELOS	47
3.6 ALGORITMOS PROBABILÍSTICOS	49
3.6.1 CATEGORÍAS	50
3.6 ALGORITMOS DETERMINÍSTICOS Y NO DETERMINÍSTICOS	52
<b>4. LENGUAJES DE PROGRAMACIÓN</b>	<b>55</b>
4.1 FORMA DE EJECUCIÓN	56
4.1.1 LENGUAJES COMPILADOS	56
4.1.2 LENGUAJES INTERPRETADOS	57
4.2 LENGUAJES DE PROGRAMACIÓN EXISTENTES	58
4.2.1 ASP.NET	58
4.2.2 PHP	58
4.2.3 VB.NET	58

4.2.4 SQL	59
4.2.5 HTML	59
4.2.6 C	59
4.2.7 C#	60
4.2.8 JAVASCRIPT	60
4.2.9 AJAX	60
4.2.10 RUBY Y RUBY ON RAILS	61
4.2.11 PERL	61
4.2.12 JAVA	61
<b>5. ESTRUCTURAS BASICAS O DE CONTROL</b>	<b>64</b>
5.1. ESTRUCTURAS SECUENCIALES	65
5.1.1. DECISIONES EN SECUENCIA	66
5.2. ESTRUCTURA DE SELECCIÓN O DECISIÓN	67
5.2.1 ESTRUCTURA CONDICIONAL SIMPLE (SI-ENTONCES / IF-THEN)	68
5.2.2 ESTRUCTURA CONDICIONAL DOBLE O ALTERNATIVA (SI-ENTONCES-SI NO / IF-THEN-ELSE)	69
5.3. ESTRUCTURAS DE REPETICIÓN O ITERACIÓN	70
5.3.1 BUCLES REPETITIVOS	72
5.3.2 INSTRUCCIÓN SELECTIVA (Case)	74
5.3.3 ESTRUCTURAS REPETITIVAS (Mientras, Repetir y Para)	75
<b>REFERENCIAS</b>	<b>79</b>

## **INTRODUCCIÓN**

Los dispositivos electrónicos están compuestos por un componente diferente al hardware, sin el cual no funcionarían; se trata del software. El caso más concreto es el computador en donde todo su potencial se ve expuesto a través de las aplicaciones de software.

El desarrollo de la unidad temática, permitirá al estudiante conocer el medio de comunicación (software - lenguajes de programación) entre el humano y la máquina, mientras construye conceptos básicos de programación, algoritmos, estructuras de datos, lenguajes de programación y aprende a solucionar problemas sencillos de su contexto, como primer paso para el desarrollo de software.

El siguiente documento es el compilado de la unidad temática de lógica y algoritmos I, del programa Tecnología en Informática y Sistemas modalidad distancia de la Universidad de la Amazonia y se identifican los elementos más relevantes para su desarrollo.

# 1. CONCEPTUALIZACIÓN

Uno de los principales objetivos que tienen los sistemas computacionales es el de ayudarle al hombre a resolver sus problemas y satisfacer algunas de sus necesidades. Para cumplir con este fin, muchas veces es necesario hacer desarrollos de software, los cuales se apoyan en la escritura de programas. El análisis y el diseño de los algoritmos es un elemento básico para la construcción de los programas por lo cual es importante disponer de buenas bases y fundamentos que nos permitan realizar de una manera fácil y rápida estos programas, y que su funcionamiento y desempeño sean satisfactorios según las necesidades y los medios con que se cuenta.

El computador no es solamente una máquina que puede realizar diferentes procesos lógicos y aritméticos que permiten generar resultados, sino que además permite diseñar soluciones a la medida, de los problemas específicos que se presentan, independientemente de si estos involucran operaciones matemáticas simples o complejas, que requieren del manejo de grandes volúmenes de datos. El diseño de soluciones a la medida de los problemas o necesidades, requiere como otras disciplinas una metodología clara y precisa que nos permita de una manera gradual y progresiva, llegar a la solución.

Estas soluciones cuando se implementan para ser utilizadas en el computador se les conocen como programas, que son una serie de operaciones, instrucciones o pasos que se realizan para llegar a un resultado, teniendo como insumo un conjunto de datos específicos, y como salida la solución al problema.

Para desarrollar software, además de conocer la metodología con que se va a afrontar dicho problema, es también importante conocer, de manera específica y clara las funciones que puede realizar el computador y las formas en que se pueden manipular los elementos o dispositivos que lo conforman, de tal manera que se haga un óptimo uso de los recursos.

## 1.1 PROGRAMACIÓN

Según la Real Academia Española programación es la acción y efecto de programar, este verbo tiene varios usos:

- Se refiere a idear y ordenar las acciones que se realizarán en el marco de un proyecto.
- A la preparación de máquinas para que cumplan con una cierta tarea en un momento determinado.

- Elaboración de programas para la resolución de problemas mediante ordenadores.

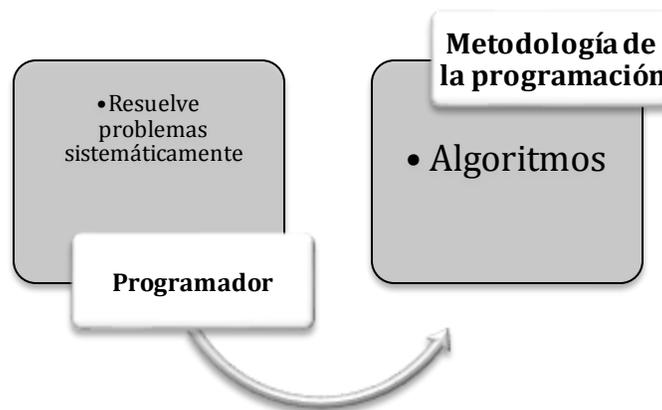
En la actualidad, la noción de programación se encuentra muy asociada a la creación de aplicaciones informáticas y videojuegos; es el proceso por el cual una persona desarrolla un programa valiéndose de una herramienta que le permita escribir el código (el cual puede estar en uno o varios lenguajes, tales como C++, Java y Python), y de otra que sea capaz de “traducirlo” a lo que se conoce como lenguaje de máquina, el cual puede ser entendido por un microprocesador. Este último paso se conoce como compilación y es necesario para que el código pueda ser ejecutado por la plataforma para la cual haya sido creado, que puede ser un ordenador, una tableta, una consola o un teléfono móvil, por ejemplo.

La totalidad del proceso de desarrollo abarca varias etapas y requiere del trabajo de diferentes especialistas. En principio, partiendo de la base de un proyecto bien organizado, es necesario dar con una idea atractiva, interesante, un problema o una necesidad que justifique el esfuerzo.

Una vez hallada la idea, se debe establecer el diseño de la misma; en otras palabras, se trata de formalizar todo aquello que se haya discutido durante la búsqueda inicial.

El programador de computadora es antes que nada una persona que resuelve problemas, por lo que para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático. A lo largo de todo el libro nos referiremos a la *metodología necesaria para resolver problemas mediante programas*, concepto que se denomina metodología de la programación. El eje central de esta metodología es el concepto de algoritmo.

Figura. Ciclo de la programación



### **1.1.1 LÓGICA**

Una definición dada para lógica lo describe como el método o razonamiento en el que las ideas o la sucesión de los hechos se manifiestan o se desarrollan de forma coherente y sin que haya contradicciones entre ellas.

## **1.2 ALGORITMOS**

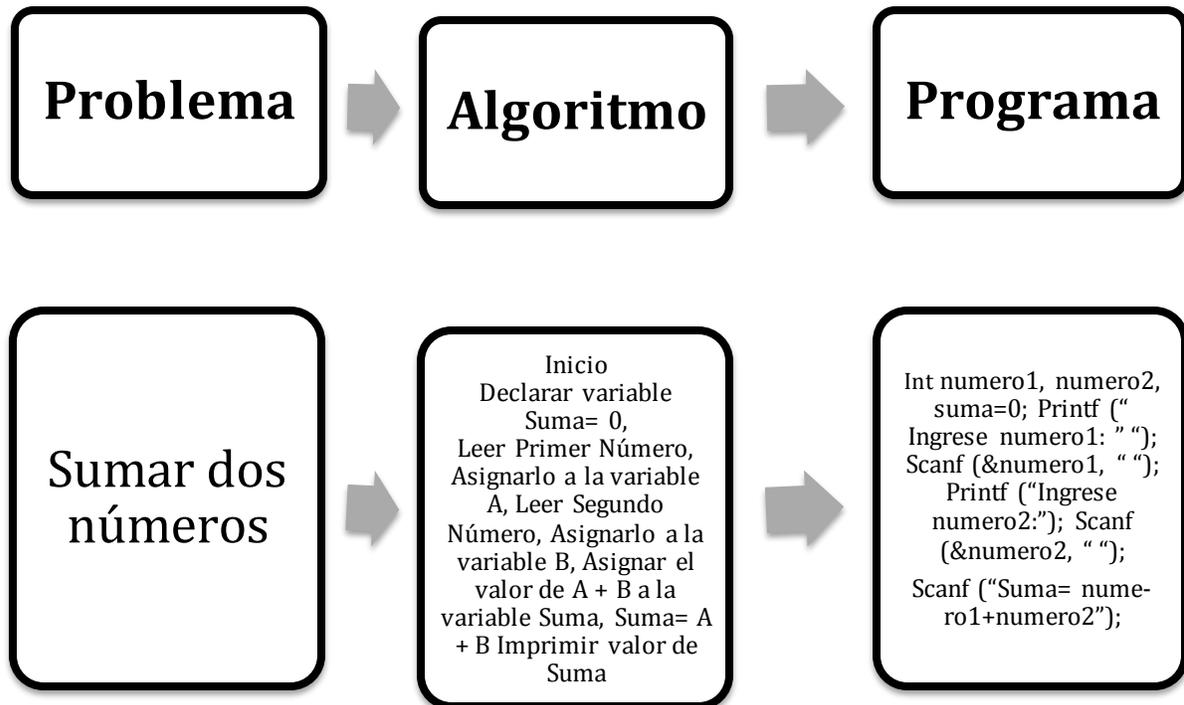
Un algoritmo es un método para resolver un problema. Aunque la popularización del término ha llegado con el advenimiento de la era informática, algoritmo proviene de Mohammed al-Khowârizmi, matemático persa que vivió durante el siglo IX y alcanzó gran reputación por el enunciado de las reglas paso a paso para sumar, restar, multiplicar y dividir números decimales; la traducción al latín del apellido en la palabra algorismus derivó posteriormente en algoritmo.

Euclides, el gran matemático griego (del siglo IV antes de Cristo) que inventó un método para encontrar el máximo común divisor de dos números, se considera con Al-Khowârizmi el otro gran padre de la algoritmia (ciencia que trata de los algoritmos).

Otro concepto de algoritmo, lo define como una secuencia de instrucciones que representan un modelo de solución para determinado tipo de problemas. O bien como un conjunto de instrucciones que realizadas en orden conducen a obtener la solución de un problema (Contextualización, 2013).

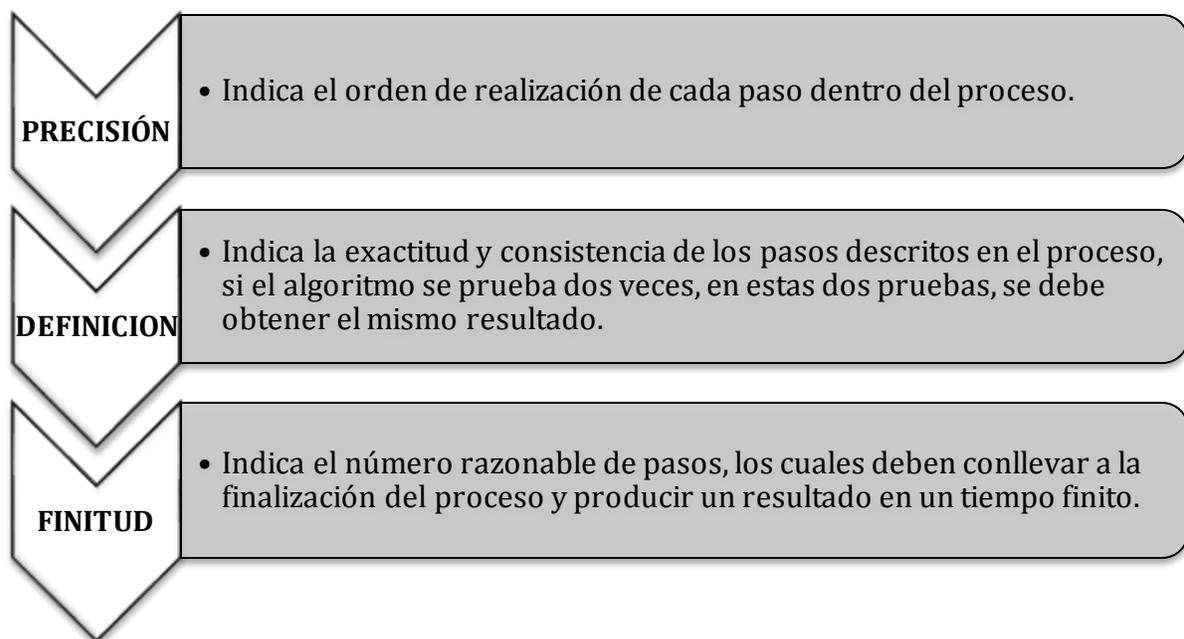
Cuando un algoritmo deba ser ejecutado por una computadora, se necesita expresar el algoritmo en instrucciones comprensibles por la computadora; para esto último, se utilizan los lenguajes de programación. Al algoritmo expresado en un determinado lenguaje de programación, se le denomina programa. Esto indica que de un determinado problema o situación dada, se elabora un algoritmo con los pasos necesarios para su solución, y si se requiere sea ejecutado por un computador, se traduce el algoritmo a instrucciones editadas en un lenguaje de programación.

Figura. Ejemplo de algoritmo para resolver la sumatoria de dos números



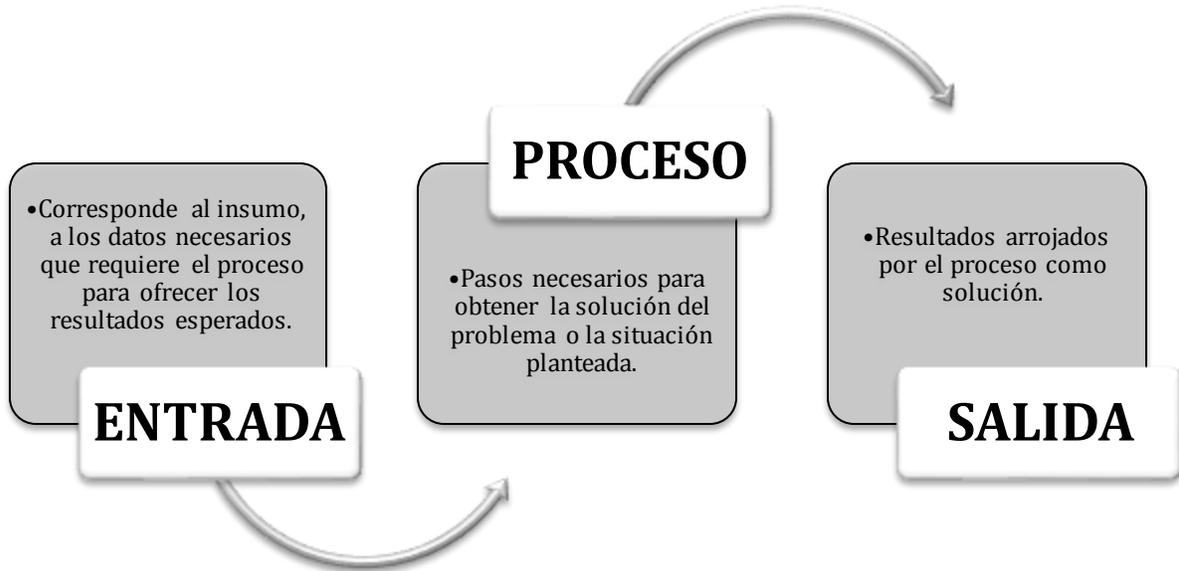
### 1.2.1 CARACTERÍSTICAS DE LOS ALGORITMOS

Las características fundamentales que debe cumplir un algoritmo son:



## 1.2.2 PARTES DE UN ALGORITMO

Todo algoritmo debe obedecer a la estructura básica de un sistema, es decir: entrada, proceso y salida. Así como se observa en la siguiente figura:



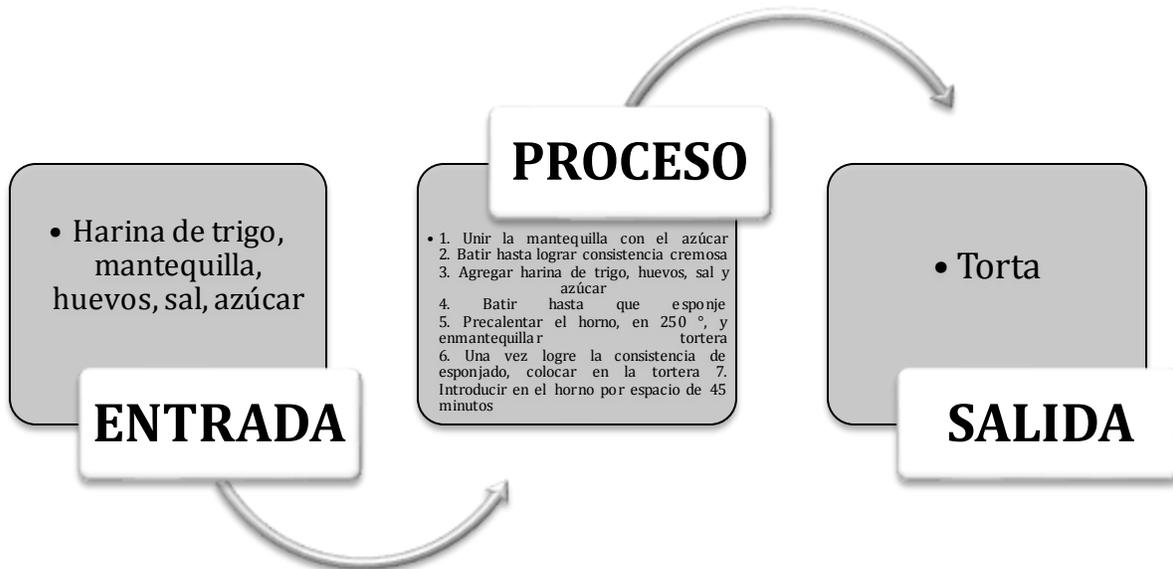
En el ejemplo del algoritmo de la sumatoria de los dos números, se tiene:

**ENTRADA:** Valores de de las variables A y B.

**PROCESO:** Asignar a la variable Suma, el valor de A mas el valor de B.

**SALIDA:** Impresión del valor de la variable Suma, que contiene la sumatoria de los valores de A y B.

La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto. En la siguiente gráfica se puede visualizar detalladamente a través de un ejemplo el algoritmo para hacer una torta, con sus respectivas partes:



### 1.2.3 METODOLOGÍA PARA LA SOLUCIÓN DE PROBLEMAS

El proceso de resolución de un problema con un computador conduce a la escritura de un programa y a su ejecución en la misma. Aunque el proceso de diseñar programas es —esencialmente— un proceso creativo, se puede considerar una serie de fases o pasos comunes, que generalmente deben seguir todos los programadores.

Los pasos para la resolución de un problema o la relacionada metodología para la resolución de problemas, está relacionada con:



#### 1. Análisis del problema

La primera fase de la resolución de un problema con computadora es el análisis del problema. Esta fase requiere una clara definición, donde se contemple exactamente lo que debe hacer el programa y el resultado o solución deseada. Dado que se busca una solución por computadora, se precisan especificaciones detalladas de entrada y salida.

#### 2. Diseño del algoritmo

En la etapa de análisis del proceso de programación se determina qué hace el programa. En la etapa de diseño se determina cómo hace el programa la tarea solicitada. Los métodos más eficaces para el proceso de diseño se basan en el conocido divide y vencerás. Es decir, la resolución de un problema complejo se realiza dividiendo el problema en subproblemas y a continuación dividiendo estos subproblemas en otros de nivel más bajo, hasta que pueda ser implementada una solución en la computadora. Este método se conoce técnicamente como diseño descendente (top-down) o modular. El proceso de romper el problema en cada etapa y expresar cada paso en forma más detallada se denomina refinamiento sucesivo.

Cada subprograma es resuelto mediante un módulo (subprograma) que tiene un sólo punto de entrada y un sólo punto de salida. Cualquier programa bien diseñado consta de un programa principal (el módulo de nivel más alto) que llama a subprogramas (módulos de nivel más bajo) que a su vez pueden llamar a otros subprogramas. Los programas estructurados de esta forma se dice que tienen un diseño modular y el método de romper el programa en módulos más pequeños se llama programación modular. Los módulos pueden ser planeados, codificados, comprobados y depurados independientemente (incluso por diferentes programadores) y a continuación combinarlos entre sí.

### **3. Verificación del Algoritmo**

Verificación de algoritmos está orientado a la comprobación del correcto funcionamiento del pseudocódigo planteado. Los conceptos de verificación, prueba y depuración son en cierta medida similares y en cierta medida distintos. Las técnicas de verificación de programas no persiguen aumentar la fiabilidad del código, sino demostrar que no contiene errores.

La verificación, mejora la comprensión del código, facilita la comunicación del código, puede aumentar la fiabilidad, aunque es difícil hacer, demostraciones completas, con las herramientas adecuadas puede integrarse en el código (Verificación de algoritmos, 2009).

Para reflexionar...

1. ¿Qué cree usted, que es la programación?
2. ¿Cómo utiliza la programación en su vida diaria?
3. ¿Considera que la programación es importante, por qué?

Como complemento...

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo, el algoritmo será siempre el mismo. Así, por ejemplo, en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, inglés o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizarán sin importar el idioma del cocinero.

En las ciencias de la computación y en la programación, los algoritmos son más importantes que los lenguajes de programación o las computadoras. Un lenguaje de programación es tan sólo un medio para expresar un algoritmo y una computadora es sólo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente. Dada la importancia del algoritmo en la ciencia de la computación, un aspecto muy importante será el diseño de algoritmos, la enseñanza y práctica de esta tarea se ha denominado algoritmia.

El diseño de la mayoría de los algoritmos requiere creatividad y conocimientos profundos de la técnica de la programación. En esencia, la solución de un problema se puede expresar mediante un algoritmo.

**Ejemplo.** Algoritmo para comprar las boletas de entrada al cine.

1. Inicio
2. Seleccionar la película
3. Llegar al lugar de proyección de la película
4. Revisar la cartelera
5. Hacer la fila de pago
6. Esperar el turno
7. Solicitar la película.  
**Si** la hay
8. Entregar el dinero
9. Esperar por las boletas y la diferencia de pago
10. Retirarse  
**Si no** hay la película
11. Escoger otra película o retirarse
12. Fin

## 1.2.4 TÉCNICAS DE REPRESENTACIÓN DE ALGORITMOS

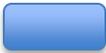
Existen diferentes técnicas para especificar los elementos de un algoritmo, dependiendo del lenguaje algorítmico que se utilice así:

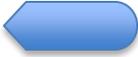
### 1.2.4.1 Diagramas de flujo

Un diagrama de flujo es la representación gráfica de un algoritmo. También se puede decir que es la representación detallada en forma gráfica de cómo deben realizarse los pasos para producir resultados.

Esta representación gráfica se presenta mediante un conjunto de símbolos que se relacionan entre sí a través de líneas que indican el orden en que se deben ejecutar cada uno de los procesos.

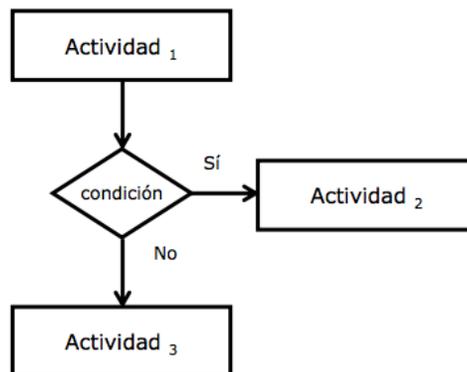
Los símbolos básicos utilizados en los diagramas de flujo son:

Nombre	Símbolo	Explicación
Inicio/Fin		Representar el inicio o el fin de un algoritmo. También puede representar una parada o una interrupción programada que sea necesaria realizar en un programa.
Proceso		Para un proceso determinado, es el que se utiliza comúnmente para representar una instrucción, o cualquier tipo de operación que origine un cambio de valor.
Entrada/Salida		Representar una entrada o salida de información, que sea procesada o registrada por medio de un periférico.
Decisión		Para la toma de decisiones, ramificaciones, para la indicación de operaciones lógicas o de comparación entre datos.
Conector		Permite enlazar dos partes cualesquiera de un diagrama a través de un conector de salida y un conector de entrada. Esta forma un enlace en la misma página del diagrama.
Conector fuera de página		Para enlazar dos partes de un diagrama pero que no se encuentren en la misma página.
Flujo del programa		Indica la secuencia del diagrama de flujo, es decir, para indicar el sentido de las operaciones

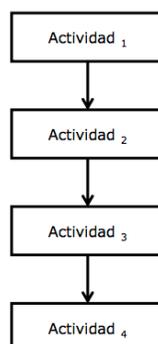
		dentro del mismo.
Salida de información impresa		Representa la salida de información por medio de la impresora.
Mostrar información en pantalla		La salida o para mostrar la información por medio del monitor o la pantalla (Diagramas de flujo, 2013)

### Recomendaciones para el diseño de diagramas de flujo:

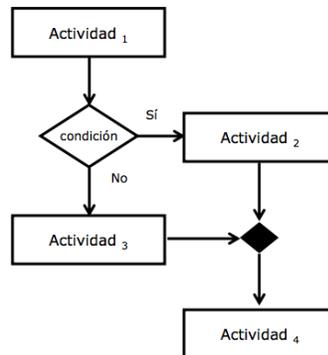
- Se deben usar solamente líneas de flujos horizontales y/o verticales.
- Se deben usar conectores solo cuando sea necesario.
- No deben quedar líneas de flujo sin conectar.
- Se deben trazar los símbolos de manera que se puedan leer de arriba hacia abajo y de izquierda a derecha.
- Todo texto ubicado dentro de un símbolo deberá ser escrito claramente.
- Los controles del flujo se representan con rombos. Dentro del rombo se indica la condición o expresión lógica a evaluar, así como se observa en la siguiente figura:



- Flujo secuencial: En este caso se ejecutan las actividades 1, 2, 3 y 4, de forma ordenada.



- Flujo condicionado: En este caso se ejecuta siempre la actividad 1. Si la condición es verdadera, entonces se ejecuta la actividad 2, en caso contrario se realiza la actividad 3. Finalmente, se ejecuta la actividad 4 (Fundamentos de la informática, 2013).



#### 1.2.4.2. Pseudocódigo

Pseudocódigo, significa escribir las instrucciones del algoritmo en lenguaje natural, tal y como lo expresamos de manera cotidiana, este procedimiento facilita su escritura en los lenguajes de programación. Según Joyanes (2003): “El pseudocódigo es un lenguaje de especificación (descripción) de algoritmos”.

Así mismo, otros autores lo definen como la mezcla de lenguaje de programación y un idioma como el español, que se emplea dentro de la programación estructurada, para especificar el diseño de un programa. Se puede definir como un lenguaje de especificaciones de algoritmos, utilizando palabras que indican el proceso a realizar. Las palabras más comunes son:

Inicio, fin, leer, escribir, si, sino, fin si, para, fin para, mientrasque, fin mientras que, repita, hasta, regresar.

Se recomienda que los pseudocódigos posean una indentación o sangría consistente a la margen izquierda, con el fin de organización el pseudocódigo.

**Ejemplo.** Algoritmo diseñado para el cálculo de la hipotenusa de un triángulo, utilizando pseudocódigo,

#### **Inicio**

Declaración de variables  
Lectura de los datos A, B y C  
Aplicar formula

Imprimir resultado

**Fin**

Nota: Como se observa se utiliza el método **Entrada-Proceso-Salida**

La escritura del algoritmo consta de:

- Identificación o cabecera
- Declaración de variables
- Cuerpo o sección de acciones

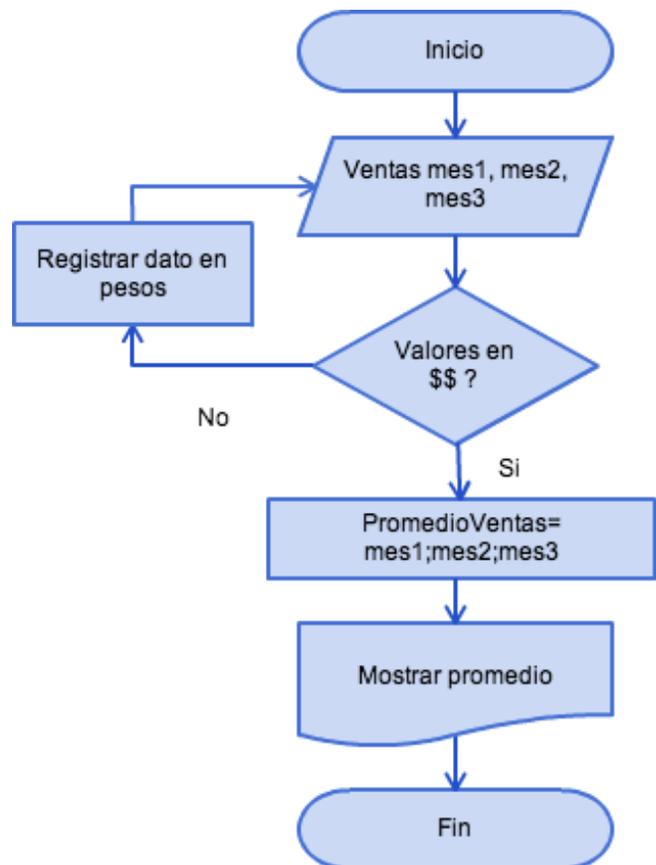
Recomendado...

Herramientas para el diseño de diagramas en línea CACOO. Disponible en:  
<https://cacoo.com/diagrams/>

### 1.2.5 EJERCICIOS PRÁCTICOS DE ALGORITMOS

**Ejercicio 1.** En una bodega de distribución del queso Caquetá, se requiere obtener el promedio de las ventas de queso durante los últimos tres meses. Elabore el diagrama de flujo y pseudocódigo que permita calcular el promedio de las ventas.

Diagrama de Flujo



Seudocódigo

Inicio

Registrar ventas mes1, mes2, mes3

Dato registrado en pesos

No Registrar nuevamente dato ventas

Si PromedioVentas (mes1,mes2,mes3)

Mostrar PromedioVentas

Fin

**Ejercicio 2.** Se requiere calcular bono vacacional para todos los empleados, de acuerdo a su salario. La empresa tiene un total de 75 empleados. Desde su inicio se define el número de iteraciones o se crea la condición necesaria para darle fin al ciclo (Ejemplos y ejercicios, 2013)

Diagrama de Flujo

Seudocódigo

**Inicio**

NoEmpleados= 0

**Si** NoEmpleados <76

**Entonces**

Calcular bono Imprimir

bono Incrementar en uno al contador  
(NoEmpleados)

**Fin Si**

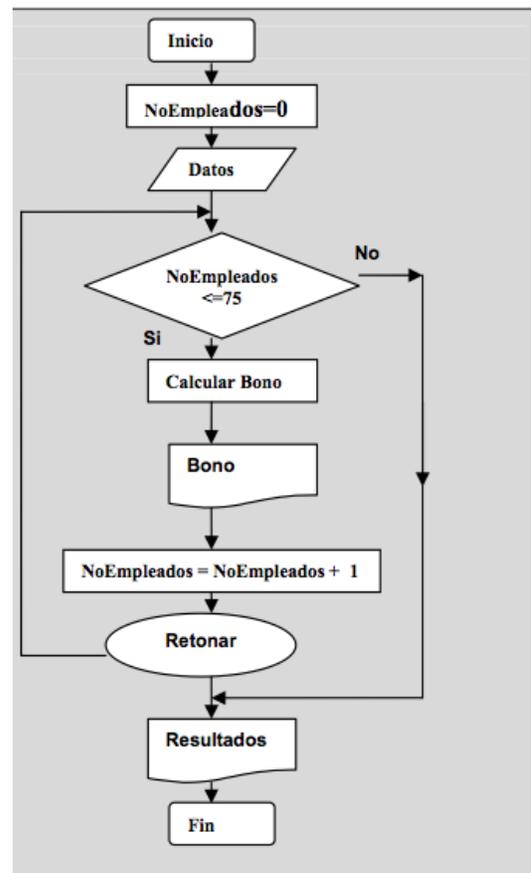
Imprimir Resultados

**Fin**

**Ejercicio 3.** Desarrolle un algoritmo que permita leer dos valores distintos, determinar cual de los dos valores es el mayor y escribirlo.

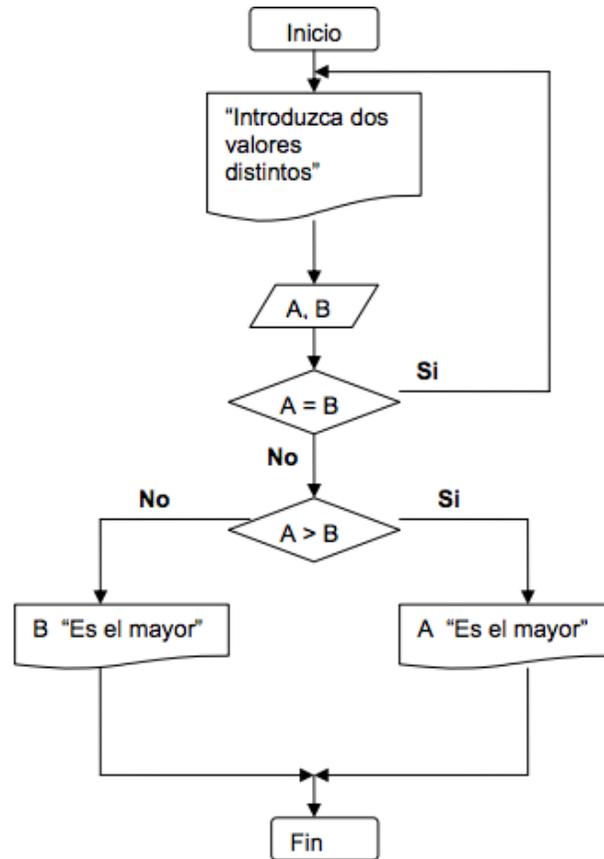
Seudocódigo

1. Inicio
2. Inicializar variables: A = 0, B = 0
3. Solicitar la introducción de dos valores distintos
4. Leer los dos valores
5. Asignarlos a las variables A y B
6. Si A = B Entonces vuelve a 3 porque los valores deben ser distintos
7. Si A>B Entonces Escribir A, "Es el mayor"



8. De lo contrario: Escribir B, "Es el mayor"
9. Fin\_Si
10. Fin

Diagrama de Flujo



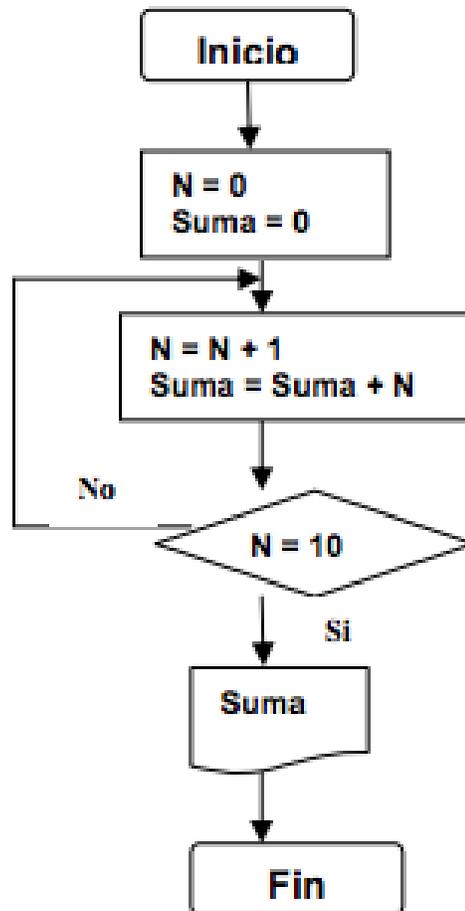
**Ejercicio 4.** Desarrolle un algoritmo que realice la sumatoria de los números enteros comprendidos entre el 1 y el 10, es decir,  $1 + 2 + 3 + \dots + 10$ .

Seudocódigo

1. Inicio
2. Declaración de variables:  $N = 0$ ,  $\text{Suma} = 0$
3. Asignación Contador :  $N = N + 1$
4. Asignación Acumulador:  $\text{Suma} = \text{Suma} + N$
5. Si  $N = 10$  Entonces
6. Escribir Suma
7. De lo contrario, Repetir desde el paso 3
8. Fin\_Si

9. Fin

Diagrama de Flujo



**Ejercicio 5.** Desarrolle un algoritmo para la empresa Constructora Tecnovivir Casas C.A., que le permita calcular e imprimir la nómina para su cancelación a un total de 50 obreros calificados a quienes debe cancelar por horas trabajadas. La hora trabajada se pautó en 30.000 Bolívares.

Seudocódigo

1. Inicio

2. Declaración de Variables:

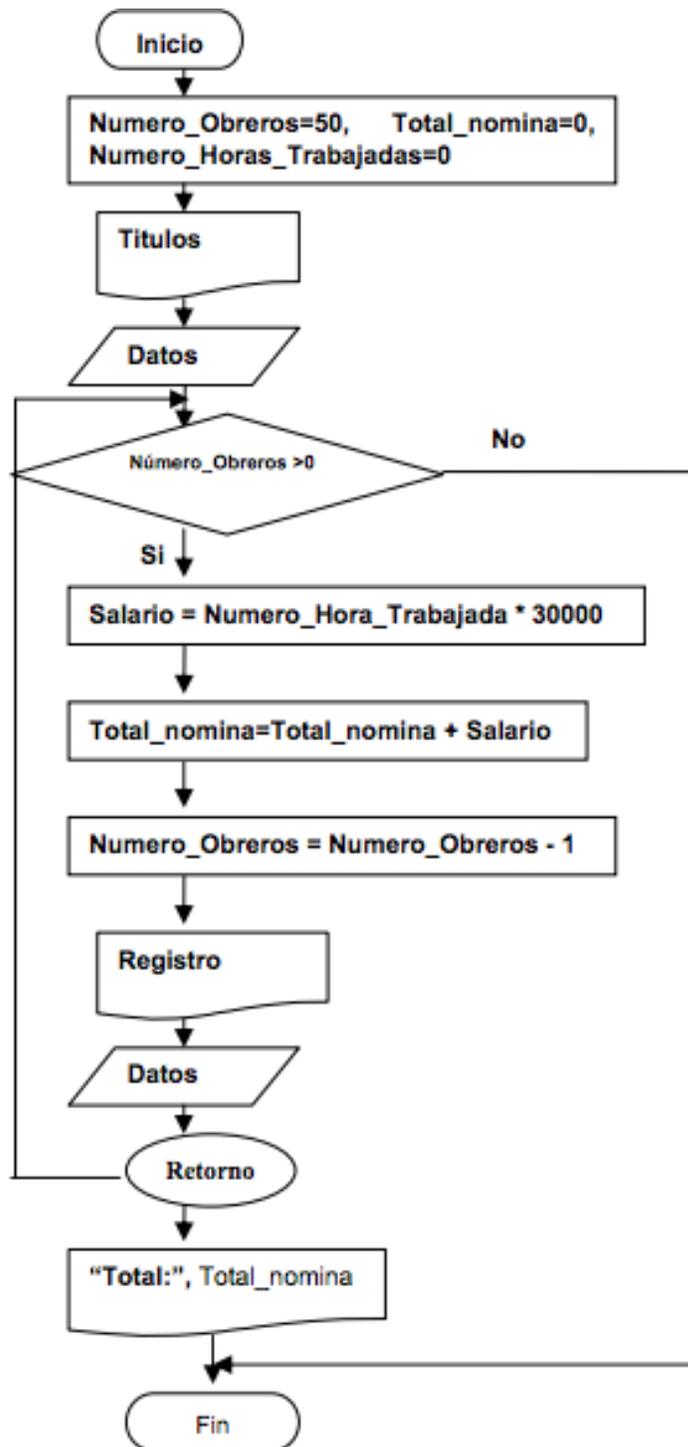
Numero\_Obreros =50

Numero\_Hora\_Trabajadas = 0

Total\_nomina = 0

3. Imprimir líneas de títulos de la nómina
4. Leer Datos
5. Mientras Numero\_Obreros>0
6. Salario = Numero\_Hora\_Trabajada \* 30
7. Total\_nómina= Totalnómina + Salario
8. Numero\_Obreros = Numero\_Obreros - 1
9. Imprimir Registro
10. Leer Datos
11. Fin\_Mientras
12. Imprimir "Total : ", Total\_nómina
13. Fin

## Diagrama de Flujo



**Ejercicio 6.** Escriba un algoritmo que calcule el área de un rectángulo.

Inicio ( Variables lado1, lado2, area, contador, estado)

Contador  $\square$ 0, estado  $\square$ "s"

Mientras contador  $\leq$  5 y estado = "s" haga

Escriba "Digite el primer lado"

Leer lado1

Escriba "Digite el segundo lado"

Leer lado2

Area  $\square$ lado1 \* lado2

Escriba "El área del rectángulo es: ", area

Contador  $\square$ contador + 1

Escriba "Desea continuar:"

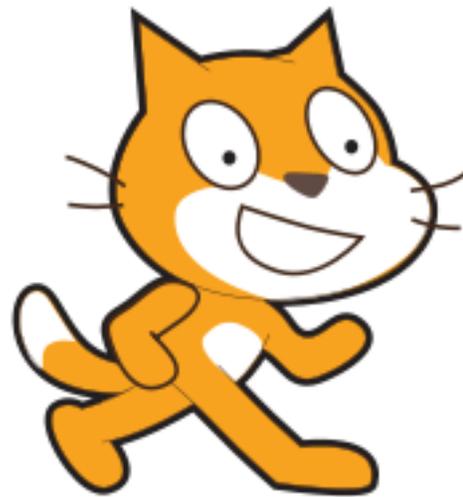
Leer estado

Fin (mientras)

Fin (inicio)

### 1.2.6 SCRATCH

Scratch es un entorno de aprendizaje de lenguaje de programación, que permite a los principiantes obtener resultados sin tener que aprender a escribir de manera sintácticamente correcta primero. Scratch está escrito en Squeak (una implementación libre de Smalltalk-80), a partir de la versión 2.0 el código es reescrito en actionscript (Adobe Flash).



Es un entorno de programación que facilita el aprendizaje autónomo. Fue desarrollado por "el grupo permanente Kindergarten" en el Media Lab del MIT (Instituto Tecnológico de Massachusetts) por un equipo dirigido por Mitchel Resnick, apareció por primera vez en el verano de 2007.

Scratch se puede instalar y redistribuir gratuitamente en cualquier ordenador con Windows, Mac OS X o Linux.<sup>4</sup> El logo de Scratch es un gato de color naranja. Se puede utilizar este para programa, tal como dice su lema: programar, jugar y crear. El nombre de Scratch se deriva de la técnica de scratching usada en el Turntablism (arte del DJ para usar los tocadiscos), y se refiere tanto a la lengua y su aplicación. La similitud con el "scratching" musical es la fácil reutilización de piezas: en Scratch todos los objetos, gráficos, sonidos y secuencias de comandos pueden ser

fácilmente importados a un nuevo programa y combinados en maneras permitiendo a los principiantes a conseguir resultados rápidos y estar motivados para intentar más (Guía de referencia, 2009)

### **Ejemplo**

Diseñar un algoritmo (seudocódigo y diagrama de flujo) para hallar el área de un triángulo rectángulo cuya Base mide 3 cm, la Altura 4 cm y la Hipotenusa 5 cm (Algoritmos y programación, 2009).

### **Análisis del problema**

Formular el problema: Ya se encuentra claramente planteado.

Resultados esperados: El área de un triángulo rectángulo.

Datos disponibles: Base, Altura, Hipotenusa, tipo de triángulo. La incógnita es el área y todos los valores son constantes. El valor de la hipotenusa se puede omitir. El estudiante debe preguntarse si sus conocimientos actuales de matemáticas le permiten resolver este problema; de no ser así, debe plantear una estrategia para obtener los conocimientos requeridos.

Determinar las restricciones: Utilizar las medidas dadas.

Procesos necesarios: Guardar en dos variables (BASE y ALTURA) los valores de Base y Altura; Guardar en una constante (DIV) el divisor 2; aplicar la fórmula  $BASE * ALTURA / DIV$  y guardar el resultado en la variable AREA; comunicar el resultado (AREA).

### **Algoritmo en pseudocódigo**

Paso 1: Inicio

Paso 2: Asignar el número 2 a la constante "div"

Paso 3: Asignar el número 3 a la constante "base"

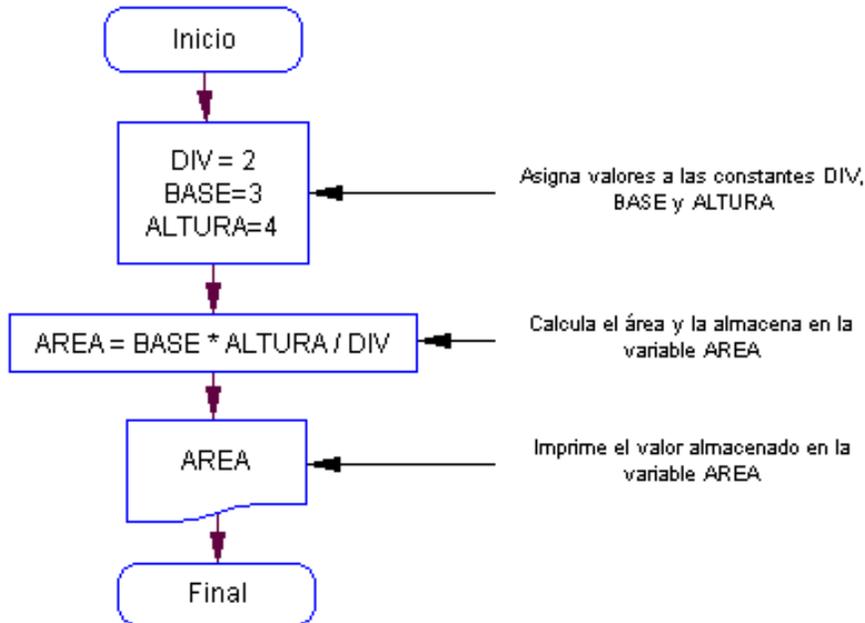
Paso 4: Asignar el número 4 a la constante "altura"

Paso 5: Guardar en la variable "área" el resultado de  $base * altura / div$

Paso 6: Imprimir el valor de la variable "área"

Paso 7: Final

## Diagrama de flujo



## Algoritmos implementado en Scratch



Recomendado...

- Investigación incidencia de scratch en el desarrollo de competencias laborales generales. Disponible en: [http://www.eduteka.org/investigacion\\_scratch\\_inem.php](http://www.eduteka.org/investigacion_scratch_inem.php)
- Aprender a programar, programar para aprender. Disponible en: <http://www.eduteka.org/codetolearn.php>

## 2. TIPOS DE DATOS

La palabra dato tiene su origen etimológico en el término latino “datum” que significa lo dado. Según la Real Academia Española un dato es un antecedente necesario para llegar al conocimiento exacto de algo o para deducir las consecuencias legítimas de un hecho. Para el campo de la informática lo define como la información dispuesta de manera adecuada para su tratamiento por un ordenador.

Los datos son símbolos que se convierten en condiciones, hechos, situaciones o valores. Un dato puede significar un número, una letra, un signo ortográfico o cualquier símbolo que represente una cantidad, una medida, una palabra o una descripción. La importancia de los datos está en su capacidad de asociarse dentro de un contexto para convertirse en información. Es decir, por sí mismos los datos no tienen capacidad de comunicar un significado y por tanto no pueden afectar el comportamiento de quien los recibe. Para ser útiles, los datos deben convertirse en información que ofrezca un significado, conocimiento, ideas o conclusiones (Programa Nacional de Formación en Sistemas e Informática, 2013).

El dato es un documento, una información, una prueba, un testimonio, cuyo conocimiento o descubrimiento permite deducir las consecuencias de un hecho determinado. Por ejemplo: “Hemos descubierto quien provocó los daños en la Universidad de la Amazonia gracias al aporte de datos que han hecho varios testigos del siniestro”.

El dato presenta una relevancia destacada, a los mismos se los considera como expresiones generales cuyas funciones son las de describir las características de las entidades sobre las cuales operan los algoritmos, en tanto, las mismas serán presentadas de determinada manera para poder ser tratadas por una computadora (Definición ABC, 2013).

Dato es la expresión general que describe los objetos con los cuales opera el algoritmo. Un tipo de datos es la propiedad de un valor que determina su dominio (qué valores puede tomar), qué operaciones se le pueden aplicar y cómo es representado internamente por el computador. Todos los valores que aparecen en un programa tienen un tipo (Tipos de datos, 2013).

Los datos pueden ser de los siguientes tipos:

- Entero: Subconjunto finito de los números enteros, cuyo rango o tamaño dependerá del lenguaje en el que posteriormente se codifique el algoritmo y de la computadora utilizada.

Los números enteros son un conjunto de números que incluye a los números naturales distintos de cero (1, 2, 3, ...), los negativos de los números naturales (... , -3, -2, -1).

Ejemplos: 465, -387, 9, 99, -1578, 125550

- Real: Subconjunto de los números reales limitado no sólo en cuanto al tamaño, sino también en cuanto a la precisión.

Son números que contienen una parte fraccionaria y, por lo tanto, incluyen el punto decimal, pueden estar precedidos del signo + o --.

Ejemplos: 17725.87, -45128.0, 158000.75, -35.58788

- Lógico: Conjunto formado por los valores Verdad (V) y Falso (F).

Se utiliza para representar las opciones (si/no) a determinadas condiciones. Ejemplo: Nacionalidad = "Colombiano" (S/N)?

Carácter (Char): Conjunto finito y ordenado de los caracteres que la computadora reconoce.

Su valor lo comprenden todos los caracteres alfabéticos, mayúsculas y minúsculas (A - Z), numéricos (0 - 9) y símbolos especiales (#,@,%,&).

- Cadena (String): Los datos (objetos) de este tipo contendrán una serie finita de caracteres, que podrán ser directamente traídos o enviados a/desde consola.

Su valor está representado por un conjunto de caracteres.

Ejemplos: "Reporte Anual de Fallas Técnicas", "1 de mayo, día del Trabajador"

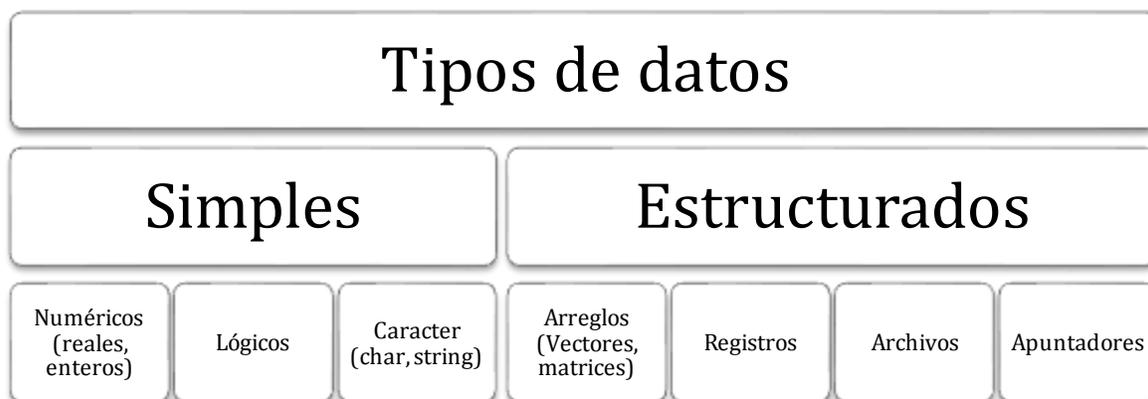
Existen datos simples (sin estructura) y compuestos (estructurados). Los datos simples son:

- Numéricos (Reales, Enteros)
- Lógicos
- Carácter (Char, String)

Los datos estructurados (definidos por el usuario) son:

- Arreglos (Vectores, matrices, arrays)
- Registros (record)
- Archivos (file)
- Apuntadores (pointer)

Figura. Resumen tipos de datos

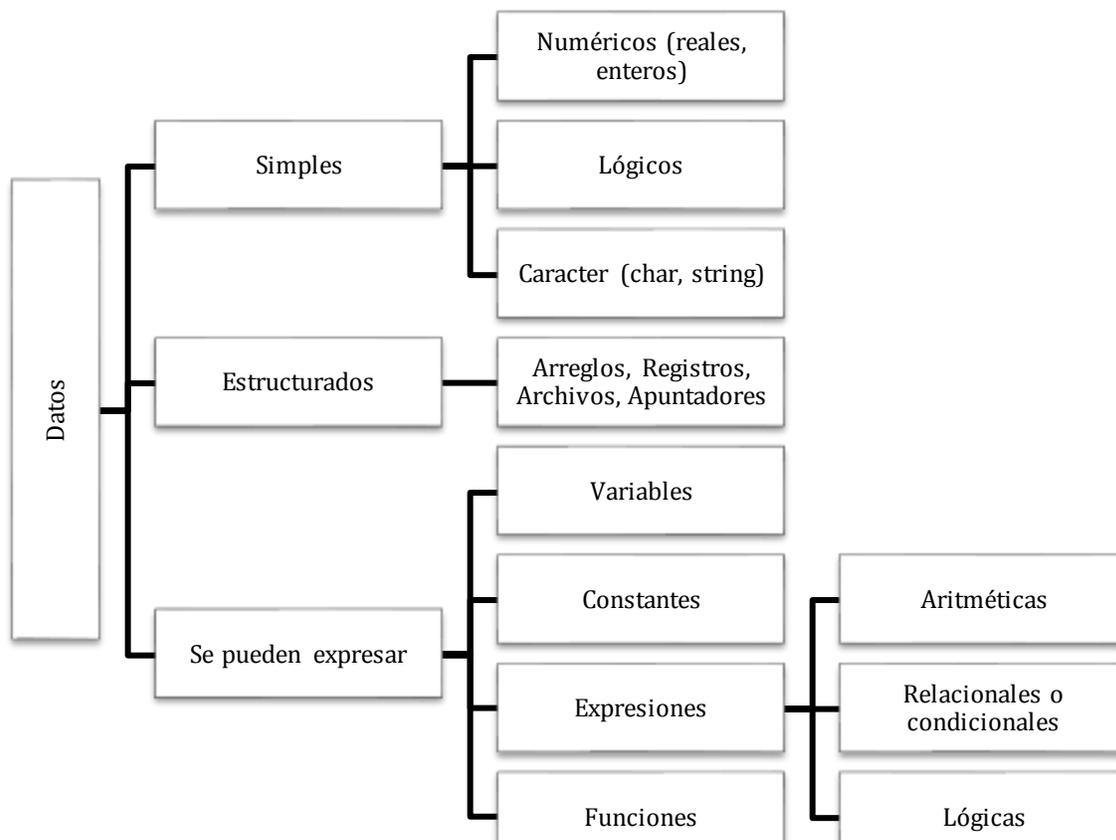


**Ejercicio.** Identifique en la tabla a continuación los tipos de datos simples y de carácter:

Dato	Respuesta	Dato	Respuesta
V		“Caquetá”	
2005		A	
-4.25		SUCRE	
3,147114		3	
0,50		“Colombia”	
10.000.000,00		F	
%		-39,78	
MIL5008		@	
“Florencia”		CAR/\$\$22	
DXLVII		1038	

Los datos pueden venir expresados como constantes, variables, expresiones o funciones. Así pues las variables y constantes son espacios de memoria creados para contener valores que de acuerdo a su naturaleza deseen mantenerse (Constantes) o que puedan variar (Variables).

Figura. Resumen datos



## 2.1 CONSTANTES

Es un dato que permanece con un valor, sin cambios, es decir constante, a lo largo del desarrollo del algoritmo o a lo largo de la ejecución del programa. Se utiliza cuando se necesita que el valor de determinada variable se mantenga durante la ejecución del programa o hasta tanto se requiera su cambio.

Existen tantos tipos de constantes como tipos de datos, por lo tanto, puede haber constantes enteras, reales (flotantes), de carácter, de cadenas de caracteres, booleanas, etc.

## 2.2 VARIABLES

Una variable es un objeto cuyo valor puede cambiar durante el desarrollo del algoritmo. Se identifica por su nombre y por su tipo, que podrá ser cualquiera, y es el que determina el conjunto de valores que podrá tomar la variable. En los algoritmos se deben declarar las variables. Cuando se traduce el algoritmo a un lenguaje de programación y se ejecuta el

programa resultante, la declaración de cada una de las variables originará que se reserve un determinado espacio en memoria etiquetado con el correspondiente identificador.

Existen tipos de constantes como tipos de datos, por lo tanto, puede haber constantes enteras, reales (flotantes), de carácter, de cadenas de caracteres, booleanas, etc. Para nombrar variables se utilizan identificadores. La declaración de una variable origina una reserva de una posición en la memoria de la computadora y que sea etiquetada con el correspondiente identificador. Se utiliza para representar un dato tipo entero, real (flotante), o una constante de carácter.

Por ejemplo:

Variable i es de tipo entero

Variable prom es de tipo real (flotante)

Variable opción es de tipo carácter

Las variables y constantes tienen básicamente, dos atributos:

Nombre: Se le asigna un nombre, en principio, para determinar que existe la variable. Este nombre debe obedecer a la naturaleza del contenido que se almacenará en ella. Por lo tanto debe orientar en relación a su contenido. Los nombres de las variables en especificaciones generales deben ser de ocho caracteres, no deben iniciarse con números ni símbolos, ni espacios en blanco.

Ejemplo de nombres de variables:

NOMBRE  
EDAD  
TOTAL  
SUELDO  
NACIONALIDAD

Ejemplo de nombres para constantes:

ISR=16.5  
AÑO FISCAL=2005  
PAIS="VENEZUELA"  
PI = 3,1416

Tipo: Es la naturaleza del dato: alfabéticos o caracteres; numéricos: enteros o reales; alfanuméricos y Lógicos. Siguiendo el ejemplo, se determina la naturaleza de las variables arriba mencionadas:

NOMBRE	CHARACTER (30)
EDAD	ALFANUMERICO
TOTAL	NUMERICO REAL
SUELDO	NUMERICO REAL
NACIONALIDAD	LÓGICO

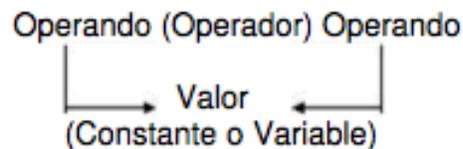
## 2.3 EXPRESIONES

Una expresión es una combinación de operadores y operandos. Los operandos podrán ser constantes, variables u otras expresiones y los operadores de cadena, aritméticos, relacionales o lógicos.

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

Por ejemplo:

$(x + y)/2$



Cada expresión produce un resultado que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.

Por ejemplo:

$+ 2 = 7$

Los operandos son 5 y 2, el signo + es el operador y 7 es el resultado de la expresión. Todo operador debe estar entre 2 operandos.

Las expresiones pueden ser simples o compuestas; las expresiones simples, son asignaciones directas a una variable o constante de un valor numérico o carácter.

Tabla. Ejemplos de expresiones simples

Nombre de la Expresión	Valor asignado	Explicación o análisis
A	=6	A la variable A se le asigna el valor de 6
B	= "DEFINITIVO"	A la variable B se le asigna por valor "DEFINITIVO"
FECHA	= dd/mm/aa	A la variable FECHA se le asigna una fecha dada.

DÓLAR	=1920	A la constante DÓLAR se le asigna el valor de 1920
PAIS	=”COLOMBIA”	A la constante PAIS se le asigna por valor “COLOMBIA”

Una expresión compuesta es la asignación a una variable o constante que surge de la unión de valores numéricos, operadores aritméticos, de comparación o lógicos. Por ejemplo:

Tabla. Ejemplos de expresiones compuestas

Nombre de la Expresión	Valor asignado	Explicación o análisis
A	=6*2	A la variable A se le asigna el valor 12, que resulta de multiplicar 6 por 2.
C	=A+B	A la variable C se le asigna el valor de sumar A + B
EDAD	= AÑOACTUAL - AÑONAC	La variable EDAD obtendrá el resultado de restarle a la constante AÑOACTUAL el valor de la variable AÑONAC
SUELDO	= DIASTRAB * SDIARIO	La variable SUELDO tomará el valor de multiplicar los días trabajados contenidos en la variable DIASTRAB por el valor de salario diario contenido en SDIARIO.

Las asignaciones se utilizan cuando:

- Se requiere que una variable contenga un valor específico
- Se requiere asignar a una variable el valor de otra variable
- Se obtiene el resultado de una expresión

Ejemplos:

Dólar = 4.30

Antes= 10

Ahora = Antes

Salario = Sueldo\_Diario \* Numero\_Dias

La forma de escribir una asignación, es:  $A = 20$  Donde la variable que esta a la izquierda toma el valor que se le asigna. Esto también indica que pierde el valor que hasta ahora tenía.

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

- Aritméticas
- Relacionales o condicionales
- Lógicas

### 2.3.1 EXPRESIONES ARITMÉTICAS

Son aquellas en donde los operadores que intervienen en ella son numéricos, el resultado es un número y los operadores son aritméticos.

Tabla. Operadores aritméticos

Nombre	Símbolo aritmético	Símbolo en algoritmo	Ejemplo	Interpretación
Suma	+	+	Si $A=10$ y $B=13$	El resultado es: 23
Resta	-	-	$A-B$	-3
Multiplicación	x	*	$A * B$	130
División	÷	/	$A**B$	100.000.000.00 0.000
Exponenciación	$b^a$	** , ^	$A/B$	0,76

Cuando los operandos son valores enteros y la operación no es una división el resultado es un número entero, pero, si al menos uno de ellos es real, el resultado es un valor real (Algoritmos – Expresiones, 2013).

A los operadores aritméticos se le puede dar las características de autoincremento (++) y autodecremento (--). El operador de incremento o decremento puede ir delante o detrás de la variable, teniendo diferente significado. Si el operador ++ se coloca después de la variable se denomina postincremento, haciendo que primero se tome el valor y después se incremente la variable.

**Ejemplo:**  $NT=T2++$ , donde NT toma el valor de T2 y luego se incrementa. Cuando el operador ++ se sitúa después de la variable, sucede lo contrario, primero se incrementa la variable y después se toma el valor, y se denomina preincremento.

**Ejemplo:**  $NT = ++T2$ , donde NT primero se incrementa y luego se asigna a NT.

No todos los operadores aritméticos, existen en todos los lenguajes, por lo tanto al codificar en determinado lenguaje, se debe investigar muy bien cual se utiliza.

Jerarquía de operaciones

1. Operador exponencial: (\*\*)
2. Operadores de multiplicación y división: (\*, /)
3. Operadores de suma y resta: (+, -)
4. Operaciones de división entera y de residuo: (div, mod)

Ejemplos

a)  $4+6*15$   
 $4 + 90$   
 $94$

b)  $5 + 7 * 3 + 2 * 4$   
 $5+ 21 +8$   
 $34$

Uso de paréntesis: Cuando una expresión aritmética posee paréntesis, se deben tener en cuenta los siguientes parámetros:

- El computador ejecuta primero las operaciones que estén dentro del paréntesis ().
- Si existen varios pares e paréntesis, se comienza a realizar las operaciones por el más interno hasta llegar al externo.

### 2.3.2 EXPRESIONES RELACIONALES

Se construyen a partir de los operadores relacionales (de relación o comparación, = , > , < , >=, <=, <>). Los operadores relacionales sirven para expresar las condiciones en los algoritmos. Las variables y constantes utilizadas son de distinta naturaleza, el resultado de una expresión lógica y relacional es de tipo lógico. El conjunto de operaciones relacionales se muestran en la siguiente tabla, se utilizan para expresar condiciones y describen una relación entre 2 variables.

Tabla. Operadores relacionales

Operadores	Significado	Ejemplo	Interpretación
------------	-------------	---------	----------------

>	Mayor que	Si A=10 y B=13 A>B	El Resultado es: FALSO
<	Menor que	A<B	VERDADERO
>=	Mayor igual	A> = B	FALSO
<=	Menor igual	A <= B	VERDADERO
=	Igual	A = B	FALSO
<>	Diferente	A <> B	VERDADERO

### 2.3.3 EXPRESIONES LÓGICAS

Las expresiones lógicas se utilizan en los programas cuando se requiere conocer o evaluar si los valores de variables o constantes determinadas cumplen con ciertas condiciones. De cumplirse o no, permiten un conjunto de instrucciones que pueden o no ejecutarse. Una expresión lógica puede ser verdadera o falsa. Los operadores lógicos se utilizan para crear las operaciones lógicas o booleanas.

Tabla. Operadores lógicos

Operador	Descripción	Resultado
A and B	AND	"true" si A y B son ambos verdaderos
A or B	OR	"true" si ambos o al menos uno de A y B son verdaderos
A == B	Igualdad	"true" si A y B son iguales
A <> B	Desigualdad	"true" si A y B NO son iguales
not B	Desigualdad	"true" si B no es verdadero (siendo B un valor o una expresión booleana)

## 2.4 FUNCIONES

En los lenguajes de programación existen ciertas funciones predefinidas o internas que aceptan unos argumentos y producen un valor denominado resultado. Como funciones numéricas, normalmente se usan:

Función	Descripción
abs(x)	Valor Absoluto
cos(x)	Coseno

sin(x)	Seno
cuadrado(x)	$x^2$
ent(x)	Parte entera
exp(x)	ex
ln(x)	Ln(x)
log(x)	Log10(x)
raiz(x)	Raiz Cuadrada
redondeo(x)	Redondear numero

Las funciones se utilizan escribiendo su nombre, seguido de los argumentos adecuados encerrados entre paréntesis, en una expresión.

### **3. TÉCNICAS PARA EL DISEÑO DE ALGORITMOS**

Desde que se construyeron los primeros computadores se viene buscado la mejor forma de escribir programas, al principio se hacían de forma empírica en donde cada programador tenía su propia técnica y básicamente se basaban en su gran experiencia para escribir programas.

Con el paso del tiempo se buscaron técnicas que permitieran el desarrollo de los programas de una manera sistemática, cumpliendo con reglas y principios, apareciendo lo que se denominó Ingeniería del software, que es un conjunto de técnicas, principios y reglas que permiten analizar, diseñar y construir programas. A continuación se muestran las técnicas más representativas para solucionar problemas:

#### **3.1 ALGORITMOS VORACES**

Se conocen como algoritmos miopes o glotones, y son aquellos que se caracterizan por tomar decisiones basados en la información que tienen a primera mano, sin tener en cuenta lo que pueda pasar más adelante; además, una vez que toman una decisión nunca reconsideran otras posibilidades, lo que ocasionalmente los lleva a caer en puntos muertos o sin salida.

Los algoritmos voraces también se caracterizan por la rapidez en que encuentran una solución (cuando la encuentran), la cual casi siempre no es la mejor. Normalmente son utilizados para resolver problemas en los que la velocidad de respuesta debe ser muy alta o en la que el árbol de decisiones de búsqueda es muy grande, no siendo posible analizar la totalidad de posibilidades.

Ejemplos típicos de problemas que se pueden resolver mediante este esquema son: búsquedas en árboles o grafos, recorridos de grafos, solución de laberintos, devolver el cambio, algunos juegos entre otros.

##### **3.1.1 FORMA GENERAL**

La estrategia general de este tipo de algoritmos se basa en la construcción de una solución, la cual comienza sin elementos y cada vez que debe tomar algún tipo de decisión, lo hace con la información que tiene a primera mano, la cual de alguna manera le permita adicionar elementos y así avanzar hacia la solución total. Cada elemento o paso de la solución se adiciona al conjunto solución y así hasta llegar a la solución final o a un

punto en el cual no puede seguir avanzando, lo cual indica que no encontró una solución al problema.

El esquema típico de una función para un algoritmo voraz es:

Funcion Voraz(C: conjunto) : conjunto

```
S ← ∅
Mientras que (C ≠ ∅ Y NO solución (S)) haga
  X ← seleccionar(C)
  C ← C - X
Fin Mientras
Si solución(S)
  Regresar(S)
Sino
  Regresar(∅)
Fin si
Fin Voraz
```

Donde C es el dominio o modelo del problema, S es la solución y X es cada una de las soluciones parciales que encuentra.

El esquema general en lenguaje C :

```
void Voraz( tipo_datos D, tipo_solucion *S)
{
  Generar la parte inicial de la solución *S
  (y las condiciones iniciales)
  while( D sin procesar del todo )
  {
    Extraer el óptimo local T de D
    Procesar T (reduciéndose D).
    Incorporar T a la solución *S
  }
}
```

### 3.1.2 EJERCICIOS PROPUESTOS

#### - **Devolver el cambio**

El planteamiento de este problema es el de dado un conjunto de denominaciones de monedas se pretende regresar una determinada cantidad de dinero haciendo uso de la menor cantidad de monedas.

**Ejemplo:** dadas tres denominaciones { 1, 4, 6 } de monedas, si quisiéramos retornar una cantidad de 8 unidades un algoritmo voraz en un primer paso examinaría entre las posibles denominaciones cual lo acerca más a la

solución final, es evidente que si restamos a la cantidad a devolver cada una de las posibles denominaciones se encuentra que la mejor aproximación es inicialmente regresar una moneda de 6 unidades, quedando únicamente por regresar 2 unidades; en un segundo paso es necesario regresar 2 unidades, siendo evidente que regresar una moneda de una unidad lo acercará aun más a la solución final, queda faltando 1 unidad, y si se repite nuevamente el proceso se encuentra que con regresar otra moneda de una unidad resolverá el problema.

El resultado final es que para regresar 8 unidades se necesitan 3 monedas, una de 6 unidades y dos de 1 unidad respectivamente para un total de tres monedas. Si se analiza el problema desde un punto de vista diferente se puede encontrar una solución que regresa la misma cantidad con un número inferior de monedas: dos de 4 unidades. (solución no voraz)

El algoritmo voraz para resolver este problema parte de la base de que se tienen las denominaciones en un arreglo ordenadas en forma descendente de mayor a menor de tal manera que permite una más fácil toma de decisiones.

Algoritmo devolver

Inicio

$C \leftarrow \{6,4,1\}$

$R \leftarrow \emptyset$

$S \leftarrow 0$

Leer(N)

Mientras que  $(S < N)$  haga

$X \leftarrow$  mayor elemento tal que  $N-X$  sea mínimo y positivo

Si no existe un  $X$  entonces

    Regresar "No hay solución"

    Fin Si

$R \leftarrow R \cup \{X\}$

$S \leftarrow S+X$

Fin Mq

Regresar R

Fin

#### – **Árbol de extensión mínima (Algoritmo de Kruskal)**

Se debe calcular el árbol de extensión mínima en un grafo no dirigido con peso, de tal manera que encuentre un conjunto de arcos que conecten todos los vértices haciendo que la suma de sus pesos sea mínima.

– **El problema del agente viajante (PAV)**

El agente viajero es una persona que debe recorrer un determinado conjunto de ciudades sin pasar dos veces por la misma. Usted debe escribir un algoritmo que dado un grafo representado mediante una matriz de  $n \times n$ , en la cual se almacenan los costos directos para ir de una ciudad a otra, encuentre mediante una técnica voraz, una ruta por la cual el agente recorre todas las ciudades. Asegúrese de no repetir ciudades para que su programa no se quede en un ciclo infinito.

– **¿Donde debe parar el viajero?**

Una persona debe viajar en su auto desde Manizales hasta Cartagena siguiendo una ruta preestablecida. Con el tanque lleno el carro puede recorrer  $n$  kilómetros; si usted dispone de un mapa, en el que se encuentran localizadas todas las estaciones de servicio y las distancias entre ellas, desarrolle un algoritmo eficiente que calcule en que estaciones debe parar.

## **3.2 DIVIDIR Y CONQUISTAR**

Es una técnica que permite encontrar la solución a un problema descomponiéndolo en subproblemas más pequeños ya sea en complejidad (más simples o sencillos) o en cardinalidad (más pequeños – menor número de elementos) y luego se unen las diferentes soluciones de cada uno de los subproblemas para conformar la solución global al problema. Cada subproblema en que se divide el problema inicial, puede ser nuevamente dividido o resuelto por otra metodología diferente hasta llegar a problemas de solución trivial o previamente conocidas.

Se debe tener en cuenta un concepto clave de la teoría de sistemas como lo es la sinergia, el cual dice: “El todo es mayor que la suma de las partes”, esto se debe a que no siempre la unión de las soluciones de los subproblemas será la solución del problema global, ya que muchas veces no se tienen en cuenta las relaciones existentes entre cada una de las partes de un problema, las cuales no pueden ser solucionadas por separado.

### **3.2.1 FORMA GENERAL**

La forma general de la técnica dividir y conquistar es la siguiente:

Función `Dividir_y_Conquistar(Problema x)`  
Inicio

```

Si x es sencillo o conocido entonces
    regresar (solución(x))
Sino
Descomponer x en problemas mas pequeños x1,x2,... xn
Para i desde 1 hasta n haga
    yi ← Dividir_y_Conquistar(xi)
Combinar los yi para obtener una solución y de x
Regresar y
    Fin si
    Fin

```

En lenguaje C, el esquema general de solución tiene la siguiente configuración:

```

Void DV( tipo_datos D, tipo_solucion *S )
{
    tipo_dato d1, d2, ..., dn;
tipo_solucion s1, s2, ..., sn;
if( se puede resolver directamente D )
    *S = Solucion directa
    else
{
    (d1,d2,dn...) = dividir(D);
DV(d1, &amp;s1 );(etc.)
DV(dn, &amp;sn);
*S = combinar( s1,s2,sn);
    }
}

```

### 3.2.3 PROBLEMAS PROPUESTOS

#### – **Busqueda de un elemento en un conjunto ordenado**

El problema de determinar si un elemento pertenece a un conjunto ordenado, es uno de los problemas clásicos que se solucionan mediante la técnica de dividir y conquistar y permite reducir la cardinalidad del conjunto de búsqueda con cada iteración del algoritmo. La clave de la solución radica en aprovechar el orden en que se encuentra el conjunto de datos, de tal manera que los elementos allí almacenados pueden ser enumerados ascendente o descendentemente desde 1 hasta  $n$  ( $n$  es la cardinalidad del conjunto). Una vez realizado este proceso comenzamos a revisar si el elemento ubicado en la mitad del conjunto  $n/2$  es el buscado, si lo es, terminamos la búsqueda o de lo contrario revisamos si el elemento buscado es mayor o menor que el ubicado en el medio y de esa

forma determinamos donde continuar la búsqueda; si el elemento buscado es menor que el del medio, continuamos la búsqueda sobre la primera mitad del conjunto, o de lo contrario lo hacemos sobre la segunda mitad del conjunto.

– **Hallar las permutaciones de un conjunto de numeros**

El problema consiste en hallar todas las permutaciones posibles para un conjunto de elementos. Una permutación de n elementos se define como una sucesión que contiene cada uno de los elementos. El número de permutaciones para un conjunto de n elementos diferentes es de n!.

**Ejemplo:** si hay un conjunto con 3 elementos {1,2,3}, el número de permutaciones será de  $3! = 6$  así:

[1,2,3] – [1,3,2] – [2,1,3] – [2,3,1] – [3,1,2] – [3,2,1]

El uso de la técnica de dividir y conquistar se presenta en intercambiar un elemento con el último y realizar de nuevo la permutación con un elemento menos.

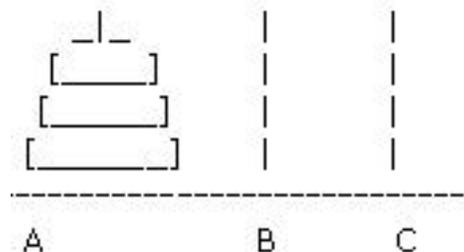
– **Las torres de hanoi**

Es un problema de solución recursiva, que consiste en mover todos los discos (de diferentes tamaños) de un eje a otro, usando un eje auxiliar, y sabiendo que un disco no puede estar sobre otro de menor tamaño. El problema de pasar n discos del eje inicial al eje final se puede dividir en el problema de pasar n-1 discos del inicial al auxiliar y luego pasar estos n-1 discos del auxiliar al final (con el mismo algoritmo).

La solución para n discos vendrá de combinar la solución de mover n-1 discos del eje inicial al eje auxiliar, de mover el último disco al eje final y de mover n-1 discos del eje auxiliar al eje final.

Solución

- 1- Mover n-1 discos de A a B
- 2- Mover 1 disco de A a C
- 3- Mover n-1 discos de B a C



Se busca desarrollar un algoritmo que permita especificar los diferentes movimientos que se deben hacer para pasar n discos de un eje a otro, cumpliendo con las reglas establecidas.

– **La exponenciación**

Sean dos números enteros  $a$  y  $n$ , se busca calcular el resultado de  $a^n$ , haciendo uso de la técnica de dividir y conquistar. La forma tradicional de resolver dicho problema, sería mediante un ciclo que se realice  $n$  veces, multiplicando el valor de  $a$ .

– **Hallar el máximo común divisor (mcd) entre dos números enteros**

El problema del máximo común divisor entre dos números, radica en encontrar el mayor número entero que los divide a los dos exactamente. El algoritmo más sencillo se construye basándose en la definición y se busca el mayor número que los divida exactamente comenzando desde la mitad del menor de los dos números, esto se debe a que el menor número que puede dividir a otro es el dos, lo que deja la mitad como el máximo divisor.

– **Adivinar el número**

Escribir un algoritmo para que permita jugar a adivinar un número; de tal manera que una persona piensa un número y la otra tratará de adivinarlo diciendo varios números. Quien pensó el número deberá decir si este es mayor, menor o igual. El algoritmo debe encontrar el número con la menor cantidad posible de intentos.

– **Búsqueda Ternaria**

Diseñe un algoritmo de búsqueda ternaria, que divida el espacio en tres partes iguales, acelerando así el proceso de búsqueda; calcule su complejidad y compárela con la búsqueda binaria.

– **Encontrar el elemento mayoritario de un vector**

El elemento mayoritario de un vector de tamaño  $n$ , es aquel que se repite más de  $n/2$  veces. Construya un algoritmo con dividir y conquistar que le permita encontrar el elemento mayoritario de un vector, si lo hay, y determine la complejidad.

– **La subsecuencia de suma máxima**

Dado un conjunto de números enteros (positivos y negativos), escribir un algoritmo que permita encontrar la subsecuencia de números (consecutivos), cuya suma sea máxima.

Ejemplo:

Dados los números  $\{-2, 3, 4, -3, 5, 6, -2\}$

La subsecuencia de suma máxima es  $\{3, 4, -3, 5, 6\}$

– **Calcular la mediana de un vector de  $n$  elementos**

La mediana es aquel elemento que ocuparía la posición  $(n + 1) \text{ div } 2$  del vector después de ordenarlo. Escriba un programa que dado un vector con  $n$  números enteros, calcule sin ordenarlo la mediana.

### **3.3 ALGORITMOS EXHAUSTIVOS (RETROCESO, ENSAYO Y ERROR)**

Los algoritmos exhaustivos son aquellos que analizan todo el espacio de búsqueda para encontrar una o todas las soluciones y garantizan que pueden encontrar una solución óptima. El retroceso o vuelta atrás es una técnica de resolución de problemas que realiza una búsqueda exhaustiva, sistemática y organizada sobre el espacio de búsqueda del problema, aplicable a problemas de optimización, juegos, búsquedas entre otros.

Se llaman algoritmos de vuelta atrás, porque en el caso de no encontrar una solución a una subtarea se retrocede a la subtarea anterior y se prueba otro camino diferente a los probados anteriormente.

Estos algoritmos se asemejan al recorrido en profundidad dentro de un grafo, siendo cada subtarea un nodo del grafo. El caso es que el grafo no está definido de forma explícita (como lista o matriz de adyacencia), sino de forma implícita, es decir, que se irá creando según avance el recorrido de la búsqueda. A menudo dicho grafo se comporta como un árbol, que no contiene ciclos, es decir, es imposible llegar a una misma solución partiendo de dos subtarear distintas.

Este tipo de estrategias tienen una premisa fundamental que afirma que si un problema tiene solución, este la encuentra. Los problemas que manejan los algoritmos exhaustivos, se caracterizan por:

- i. Se trata generalmente de problemas de optimización, con o sin restricciones.
- ii. La solución es expresable en forma de una secuencia de decisiones.
- iii. Existe una función denominada factible que permite averiguar si en una secuencia de decisiones, la solución en curso, viola o no las restricciones.
- iv. Existe una función, denominada solución, que permite determinar si una secuencia de decisiones factible es solución al problema planteado.

#### **3.3.1 FORMA GENERAL**

El esquema general de solución presenta los siguientes pasos:

- a. Vuelta Atrás hace un recorrido en profundidad del espacio de búsqueda partiendo de la raíz.

b. El recorrido en profundidad regresa sobre sus pasos, retrocede, cada vez que encuentra un camino que se ha acabado o por el que no puede continuar.

c. En un recorrido en profundidad o en anchura sobre un espacio de búsqueda se conoce de antemano el orden en que se van a generar o recorrer, sus nodos.

```
Algoritmo vuelta-atrás(etapa)
Inicio
IniciarOpciones
Repita
Opcion ← seleccionarOpcion
Si aceptable(opcion) entonces
guardar(opcion)
Si incompleta(solucion) entonces
Éxito ← vuelta-atrás(siguiete(etapa))
Si (éxito = falso) entonces
Retirar(opcion)
Fin si
Sino
Éxito ← verdad
Fin si
Fin si
Hasta (éxito = verdad OR opcion = ultimaOpcion)
Fin vuelta-atras
```

### 3.3.2 PROBLEMAS PROPUESTOS

#### – **El salto del caballo**

Se pretende recorrer un tablero de ajedrez mediante un caballo, de tal manera que iniciando en una posición determinada, recorra todas las posiciones del tablero con los movimientos o saltos del caballo en el juego de ajedrez.

#### – **Las ocho reinas**

Sobre un tablero de ajedrez hay que colocar 8 reinas de forma que ninguna de ellas se amenace. Se debe tener en cuenta que las reinas en el ajedrez pueden atacar a cualquier pieza que se encuentre ubicada a cualquier distancia sobre sus horizontales, verticales y diagonales.

Como cada reina estará en una fila diferente, podemos representar la solución con un vector de columnas donde están situadas según la fila. Para resolverlo se situaría la primera reina sobre la primera fila y se

colocaría la segunda reina en un lugar donde no amenace a las demás. Si no encontramos una solución parcial, volvemos un paso atrás y replanteamos la solución de la reina en la etapa anterior buscando una nueva ubicación y así para las demás reinas hasta lograr su ubicación en el tablero.

– **Encontrar todos los caminos mínimos en un grafo (floyd)**

Dado un grafo dirigido representado por su matriz de adyacencia y el número de nodos, se busca encontrar los caminos de menor costo entre todos los nodos que conforman el grafo.

– **El problema de la mochila**

Se pretende resolver el problema de la mochila enunciado anteriormente, pero con el fin de buscar una solución que haga uso de la estrategia de vuelta atrás, de tal manera que se calcule el valor máximo que se puede transportar, teniendo en cuenta adicionalmente que se puedan llevar varios elementos de cada uno.

### **3.4 ALGORITMOS DE VUELTA ATRÁS (BACKTRACKING)**

Dentro de las técnicas de diseño de algoritmos, el método de Vuelta Atrás (del inglés Backtracking) es uno de los de más amplia utilización, en el sentido de que puede aplicarse en la resolución de un gran número de problemas, muy especialmente en aquellos de optimización.

El diseño Vuelta Atrás proporciona una manera sistemática de generar todas las posibles soluciones siempre que dichas soluciones sean susceptibles de resolverse en etapas.

En su forma básica la Vuelta Atrás se asemeja a un recorrido en profundidad dentro de un árbol cuya existencia sólo es implícita, y que denominaremos árbol de expansión. Este árbol es conceptual y sólo haremos uso de su organización como tal, en donde cada nodo de nivel  $k$  representa una parte de la solución y está formado por  $k$  etapas que se suponen ya realizadas. Sus hijos son las prolongaciones posibles al añadir una nueva etapa. Para examinar el conjunto de posibles soluciones es suficiente recorrer este árbol construyendo soluciones parciales a medida que se avanza en el recorrido.

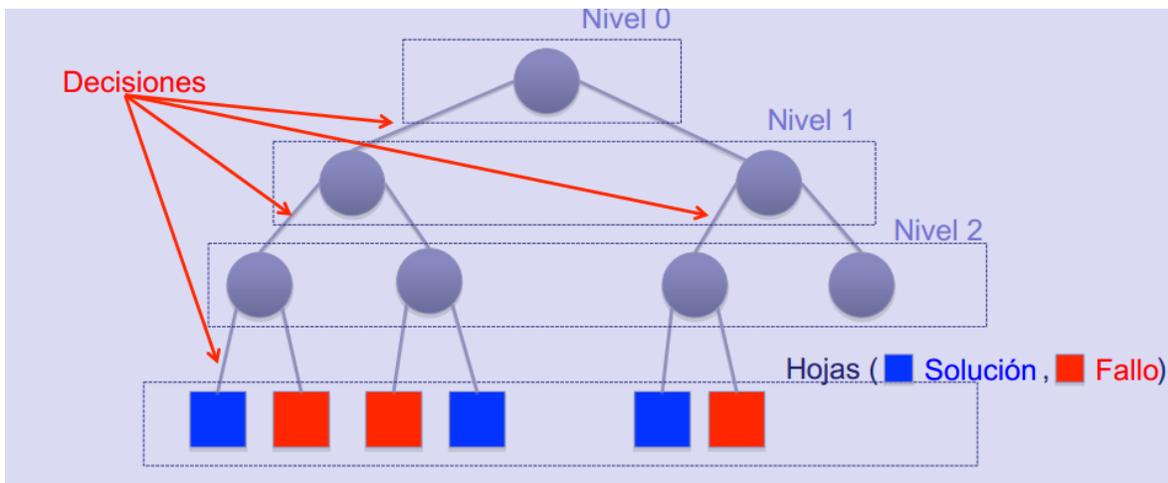
En este recorrido pueden suceder dos cosas. La primera es que tenga éxito si, procediendo de esta manera, se llega a una solución (una hoja del árbol). Si lo único que buscábamos era una solución al problema, el algoritmo finaliza aquí; ahora bien, si lo que buscábamos eran todas las

soluciones o la mejor de entre todas ellas, el algoritmo seguirá explorando el árbol en búsqueda de soluciones alternativas.

Por otra parte, el recorrido no tiene éxito si en alguna etapa la solución parcial construida hasta el momento no se puede completar; nos encontramos en lo que llamamos nodos fracaso. En tal caso, el algoritmo vuelve atrás (y de ahí su nombre) en su recorrido eliminando los elementos que se hubieran añadido en cada etapa a partir de ese nodo. En este retroceso, si existe uno o más caminos aún no explorados que puedan conducir a solución, el recorrido del árbol continúa por ellos.

La filosofía de estos algoritmos no sigue unas reglas fijas en la búsqueda de las soluciones. Podríamos hablar de un proceso de prueba y error en el cual se va trabajando por etapas construyendo gradualmente una solución. Para muchos problemas esta prueba en cada etapa crece de una manera exponencial, lo cual es necesario evitar.

Figura. Ejemplo de árbol binario (Wikipedia)



### 3.4.1 PROBLEMAS PROPUESTOS

#### - LAS $n$ REINAS

Un problema clásico que puede ser resuelto con un diseño Vuelta Atrás es el denominado de las ocho reinas y en general, de las  $n$  reinas. Disponemos de un tablero de ajedrez de tamaño  $8 \times 8$ , y se trata de colocar en él ocho reinas de manera que no se amenacen según las normas del ajedrez, es decir, que no se encuentren dos reinas ni en la misma fila, ni en la misma columna, ni en la misma diagonal.

## **- RECORRIDOS DEL REY DE AJEDREZ**

Dado un tablero de ajedrez de tamaño  $n \times n$ , un rey es colocado en una casilla arbitraria de coordenadas  $(x,y)$ . El problema consiste en determinar los  $n^2-1$  movimientos de la figura de forma que todas las casillas del tablero sean visitadas una sola vez, si tal secuencia de movimientos existe.

## **- RECORRIDOS DEL REY DE AJEDREZ (2)**

Al igual que en el problema discutido anteriormente, un rey es colocado en una casilla arbitraria de coordenadas  $(x_0,y_0)$  de un tablero de ajedrez de tamaño  $n \times n$ . Si asignamos a cada casilla del tablero un peso (dado por el producto de sus coordenadas), a cada posible recorrido le podemos asignar un valor que viene dado por la suma de los pesos de las casillas visitadas por el índice del movimiento que nos llevó a esa casilla dentro del recorrido.

## **- LAS PAREJAS ESTABLES**

Supongamos que tenemos  $n$  hombres y  $n$  mujeres y dos matrices  $M$  y  $H$  que contienen las preferencias de los unos por los otros. Más concretamente, la fila  $M[i, \cdot]$  es una ordenación (de mayor a menor) de las mujeres según las preferencias del  $i$ -ésimo hombre y, análogamente, la fila  $H[i, \cdot]$  es una ordenación (de mayor a menor) de los hombres según las preferencias de la  $i$ -ésima mujer (Vuelta atrás, 2013).

## **3.5 ALGORITMOS PARALELOS**

En las ciencias de la computación, un algoritmo paralelo, en oposición a los algoritmos clásicos o algoritmos secuenciales, es un algoritmo que puede ser ejecutado por partes en el mismo instante de tiempo por varias unidades de procesamiento, para finalmente unir todas las partes y obtener el resultado correcto.

Algunos algoritmos son fácilmente divisibles en partes; como por ejemplo, un algoritmo que calcule todos los números primos entre 1 y 100, donde se podría dividir los números originales en subconjuntos y calcular los primos para cada uno de los subconjuntos de los números originales; al final, uniríamos todos los resultados y tendríamos la solución final del algoritmo. Otro ejemplo, puede ser el cálculo de  $\pi$  en paralelo.

Por el contrario, a veces los problemas no son tan fácilmente paralelizables, de ahí que estos problemas se conozcan como problemas

inherentemente secuenciales. Como ejemplo de estos métodos tendríamos los métodos numéricos iterativos como el método de Newton o el problema de los tres cuerpos. Por otro lado, algunos problemas son difícilmente paralelizables, aunque tengan una estructura recursiva. Como ejemplo de esto último tendríamos la búsqueda primero en profundidad en un grafo.

Los algoritmos paralelos son importantes porque es más rápido tratar grandes tareas de computación mediante la paralelización que mediante técnicas secuenciales. Esta es la forma en que se trabaja en el desarrollo de los procesadores modernos, ya que es más difícil incrementar la capacidad de procesamiento con un único procesador que aumentar su capacidad de cómputo mediante la inclusión de unidades en paralelo, logrando así la ejecución de varios flujos de instrucciones dentro del procesador. Pero hay que ser cauto con la excesiva paralelización de los algoritmos ya que cada algoritmo paralelo tiene una parte secuencial y debido a esto, los algoritmos paralelos pueden llegar a un punto de saturación. Por todo esto, a partir de cierto nivel de paralelismo, añadir más unidades de procesamiento puede sólo incrementar el coste y la disipación de calor.

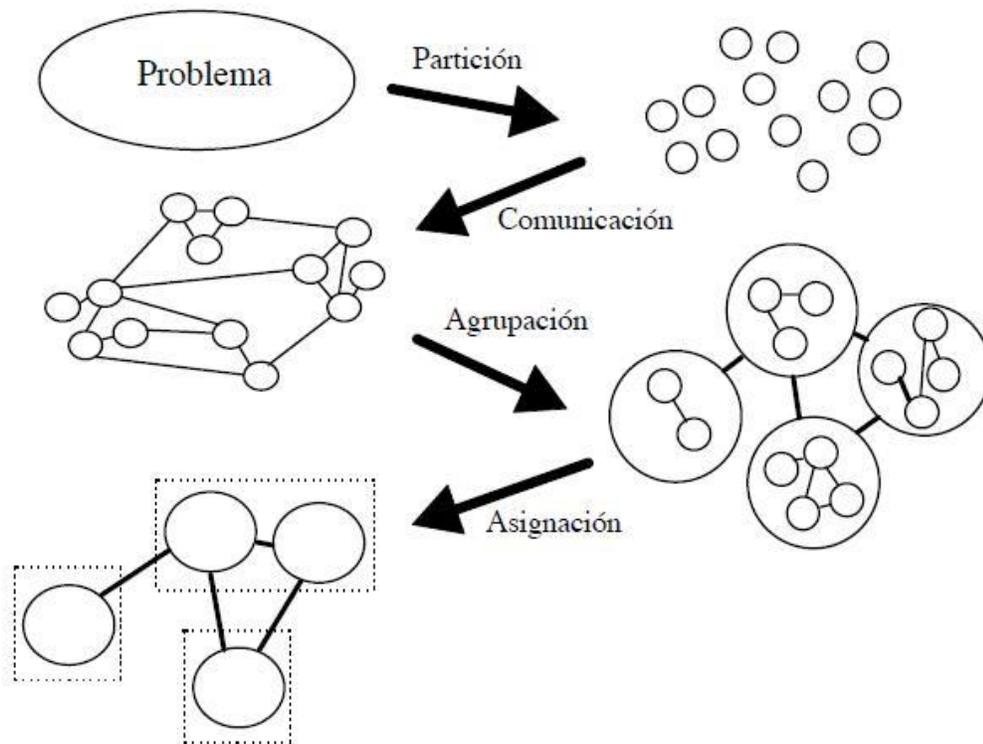
El coste o complejidad de los algoritmos secuenciales se estima en términos del espacio (memoria) y tiempo (ciclos de procesador) que requiera. Los algoritmos paralelos también necesitan optimizar la comunicación entre diferentes unidades de procesamiento. Esto se consigue mediante la aplicación de dos paradigmas de programación y diseño de procesadores distintos: memoria compartida o paso de mensajes.

La técnica memoria compartida necesita del uso de cerrojos en los datos para impedir que se modifique simultáneamente por dos procesadores, por lo que se produce un coste extra en ciclos de CPU desperdiciados y ciclos de bus. También obliga a serializar alguna parte del algoritmo.

La técnica paso de mensajes usa canales y mensajes pero esta comunicación añade un coste al bus, memoria adicional para las colas y los mensajes y latencia en el mensaje. Los diseñadores de procesadores paralelos usan buses especiales para que el coste de la comunicación sea pequeño pero siendo el algoritmo paralelo el que decide el volumen del tráfico.

Finalmente, una subclase de los algoritmos paralelos, los algoritmos distribuidos son algoritmos diseñados para trabajar en entornos tipo clusters y de computación distribuida, donde se usan otras técnicas, fuera del alcance de los algoritmos paralelos clásicos (Wikipedia).

Figura. Etapas en el diseño de algoritmos paralelos (Diseño de algoritmos paralelos, 2013)



### 3.6 ALGORITMOS PROBABILÍSTICOS

Los algoritmos probabilísticos o probabilistas son aquellos que basan el resultado devuelto en decisiones aleatorias, de tal forma que, en promedio se obtienen una buena solución al problema planteado, dada una distribución de datos de entrada. Un problema típico para hacer ver el funcionamiento de este tipo de algoritmos es el siguiente:

Se conocen dos determinados emplazamientos lo suficientemente alejados el uno del otro, al menos igual a la distancia entre cada emplazamiento y el lugar de partida. Se sabe también que en uno de los dos lugares existe un importante botín. Sin embargo, no es posible explorar un sitio primero y otro después, pues cada día que pasa, el botín se reduce en una cantidad fija. Si se hace uso de la inteligencia, podría calcularse con exactitud el lugar del botín, pero el tiempo empleado en el cálculo haría perder parte de las ganancias. Supóngase ahora que alguien ofreciera la solución a cambio de parte de las ganancias, algo inferior al tiempo de cálculo. La duda planteada sería la siguiente: ¿Cuál es la mejor solución:

calcular la ruta de forma independiente o aceptar el trato ofrecido? La solución es ninguna de las dos, pues hay una solución mejor, elegir aleatoriamente uno de los lugares.

Concretando el ejemplo, supongamos que cada localización está separada por cinco días de viaje, el cálculo de la ruta adecuada cuesta cuatro días y el trato ofrecido es dar una ganancia equivalente a tres días de pérdida. Supóngase  $x$  como el valor del botín e  $y$  como la cantidad diaria que se disminuye. Así, en el primero de los casos, se obtiene una ganancia de  $x-9y$ , mientras que si se acepta el trato, se obtiene una ganancia de  $x-8y$ . El segundo trato es claramente mejor, pero podría mejorarse. Si se escoge al azar un camino a seguir, podría acertarse o fallarse en la elección. Si se acierta, se obtiene un botín equivalente a  $x-5y$ , pero si se falla, se obtendría  $x-10y$ . Sin embargo, al haber sólo dos opciones, el caso promedio nos dice que se obtiene una ganancia de  $x-7,5y$ , mejorando los dos casos deterministas.

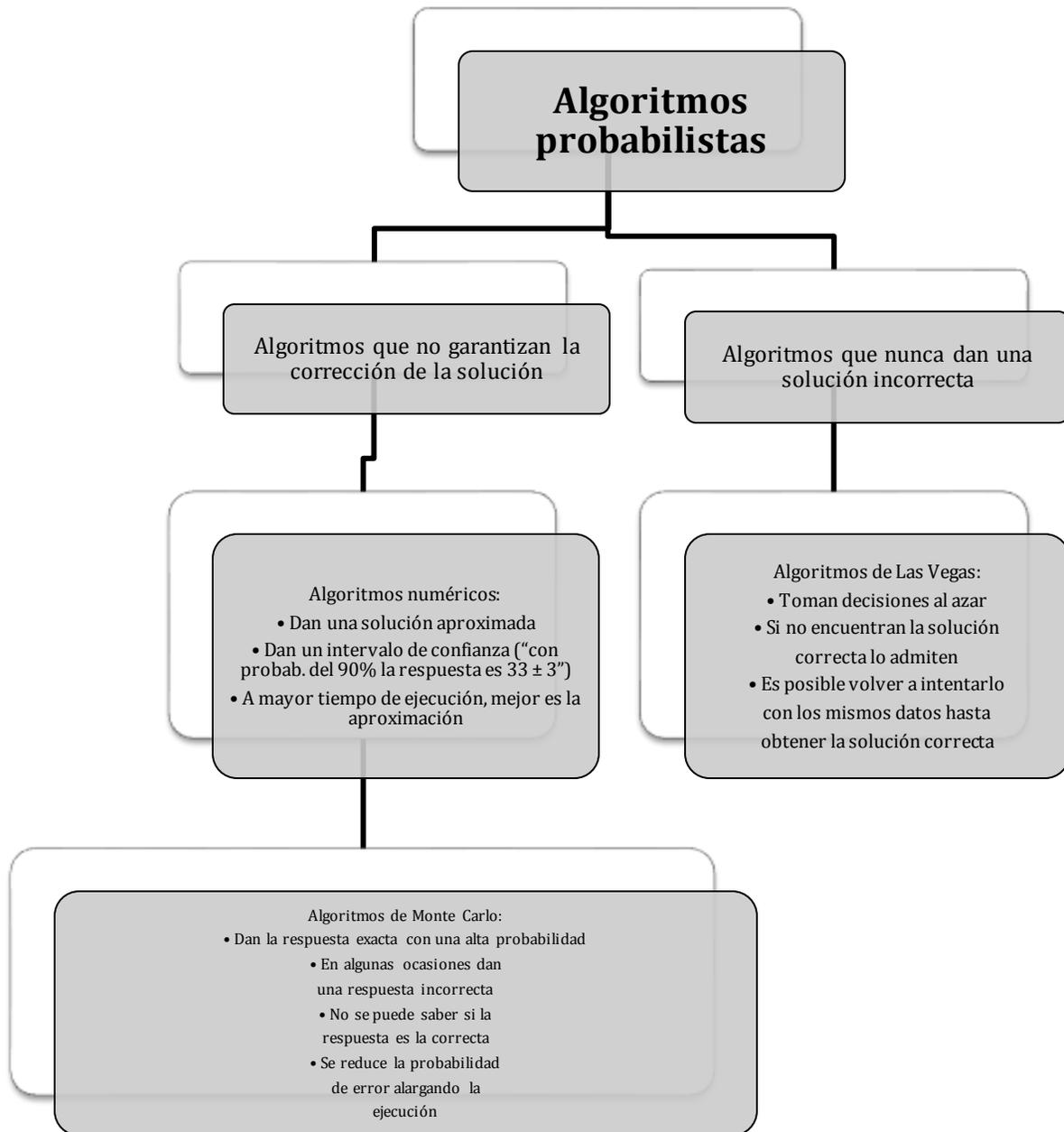
Otra ventaja de los algoritmos probabilistas sobre los deterministas consiste en que, si existen varias soluciones a un mismo problema, pueden devolver diferentes soluciones en diferentes ejecuciones sobre el mismo conjunto de datos, mientras que uno determinista ofrecerá siempre la misma solución. Así pues, tanto el tiempo de ejecución como el resultado pueden variar de una ejecución a otra.

### 3.6.1 CATEGORÍAS

#### – Algoritmos numéricos

Que devuelvan una aproximación al resultado, frecuentemente en forma de intervalo. Son útiles cuando la solución exacta es demasiado costosa (o directamente imposible de calcular, como por ejemplo, para números irracionales) y una aproximación es lo suficientemente buena. Considérese que se desea calcular el resultado de una complicada integral de varias dimensiones. Tal vez sólo se necesite una precisión de cuatro decimales, aunque la solución exacta conste de varias decenas de los mismos. Este tipo de algoritmos suelen ofrecer resultados más precisos cuanto mayor tiempo se dedica a su cálculo.

Figura. Clasificación algoritmos probabilistas (Algoritmos probabilistas, 2013)



– **Algoritmos de Monte Carlo**

Que siempre devuelven una solución aunque esta a veces no sea correcta. Son útiles cuando una aproximación no es suficiente (por ejemplo, en un problema de decisión). Cuentan con el inconveniente de no saber con

exactitud si la respuesta es acertada, pues sólo existe una cierta probabilidad de éxito. Cuantas más veces se ejecute, más seguro se estará de la corrección de la solución.

– **Algoritmos de Las Vegas**

Similares a los de Monte Carlo pero que nunca devuelven una solución errónea, con el inconveniente de que pueden no terminar o devolver solución. Esto garantiza que la respuesta sea la buena, pero no garantiza que el algoritmo funcione. Como en los casos anteriores, cuanto mayor es el tiempo dedicado al cálculo, más fiable es la solución. No debe confundirse este tipo de algoritmos con aquellos deterministas, como el simplex en programación lineal, que son muy eficientes para la mayoría de los posibles datos de entrada pero desastrosos para unos pocos. Nótese que dichos algoritmos siempre devuelven una solución correcta.

– **Algoritmos de Sherwood**

Los cuales devuelven siempre una respuesta, la cual es forzosamente exacta. Aparecen cuando un algoritmo determinista conocido es más rápido en el caso medio que en el peor. El uso del azar permite reducir, e incluso eliminar, la diferencia entre buenos y malos ejemplares (Algoritmos- Algoritmos probabilísticos, 2013)

## **3.6 ALGORITMOS DETERMINÍSTICOS Y NO DETERMINÍSTICOS**

En ciencias de la computación, un algoritmo determinista es un algoritmo que, dado un determinado insumo, siempre producirá la misma salida, con la máquina subyacente siempre que pasa a través de la misma secuencia de estados. Algoritmos determinísticos son, con mucho, el tipo más estudiado y familiar de algoritmo, así como uno de los más prácticos, ya que se pueden ejecutar en máquinas reales de manera eficiente.

Formalmente, un algoritmo determinista calcula una función matemática; una función tiene un valor único para cualquier entrada dada, y el algoritmo es un proceso que produce este valor particular como salida.

### **3.6.1 DEFINICIÓN FORMAL**

Algoritmos deterministas pueden definirse en términos de una máquina de estados: un estado describe lo que una máquina está haciendo en un momento determinado en el tiempo. Las máquinas de estado pasan de una manera discreta de un estado a otro. Justo después de que entramos en la

entrada, la máquina se encuentra en su estado inicial o estado de inicio. Si la máquina es determinista, esto significa que a partir de este punto en adelante, su estado actual determina cuál será su próximo estado, su paso por el conjunto de estados está predeterminado. Tenga en cuenta que una máquina puede ser determinista y todavía no detener o finalizar, y por lo tanto no pueden entregar un resultado.

### **Lo que hace que los algoritmos no deterministas?**

Una variedad de factores pueden causar un algoritmo para comportarse de una manera que es no determinista, o no determinista:

- Si se utiliza el estado externo que no sea el de entrada, tales como la entrada del usuario, una variable global, un valor de temporizador de hardware, un valor aleatorio, o los datos del disco almacenados.
- Si se opera de una manera que es sensible al tiempo, por ejemplo si tiene varios procesadores de escritura para los mismos datos al mismo tiempo. En este caso, el orden preciso en el que cada procesador escribe sus datos afectará el resultado.
- Si un error de hardware hace que su estado para cambiar de un modo inesperado.

Aunque los programas reales rara vez son puramente determinista, es más fácil para los seres humanos, así como otros programas de razonar acerca de los programas que son. Por esta razón, la mayoría de los lenguajes de programación y lenguajes de programación especialmente funcionales hacen un esfuerzo para evitar que los eventos anteriores a ocurrir, excepto bajo condiciones controladas.

La prevalencia de los procesadores multicore se ha traducido en un aumento del interés en el determinismo en la programación y los problemas de la no-determinismo han sido bien documentados en paralelo. Se han propuesto una serie de herramientas para ayudar a lidiar con los desafíos de tratar con interbloqueos y condiciones de carrera.

### **Problemas con algoritmos determinísticos**

Algunos problemas de algoritmos deterministas también son difíciles de encontrar. Por ejemplo, hay algoritmos probabilísticos simples y eficientes que determinan si un número dado es primo y tienen una muy pequeña posibilidad de estar equivocado. Estos han sido conocidos desde la década

de 1970; los algoritmos determinísticos conocidos siguen siendo considerablemente más lento en la práctica.

Otro problema importante con algoritmos deterministas es que a veces, no queremos que los resultados sean previsibles. Por ejemplo, si usted está jugando un juego en línea de la veintiuna que baraja la cubierta usando un generador de números pseudo-aleatorios, un jugador inteligente puede adivinar con precisión los números del generador elegir y así determinar el contenido completo de la cubierta antes de tiempo, lo que permite le engañan, por ejemplo, el Grupo de Software de Seguridad en Tecnologías de Software Fiable fue capaz de hacer esto para una implementación de Texas Hold'em Poker que se distribuye por la ASF Software, Inc, lo que les permite predecir consistentemente el resultado de las manos antes de tiempo.

Problemas similares se presentan en la criptografía, donde las claves privadas se generan a menudo utilizando como un generador. Este tipo de problema se evita generalmente el uso de un generador de números pseudo-aleatorios criptográficamente seguro (Algoritmo determinista, 2013).

## 4. LENGUAJES DE PROGRAMACIÓN

Según la definición teórica, como lenguaje se entiende a un sistema de comunicación que posee una determinada estructura, contenido y uso. La programación es, en el vocabulario propio de la informática, el procedimiento de escritura del código fuente de un software. De esta manera, puede decirse que la programación le indica al programa informático qué acción tiene que llevar a cabo y cuál es el modo de concretarla.

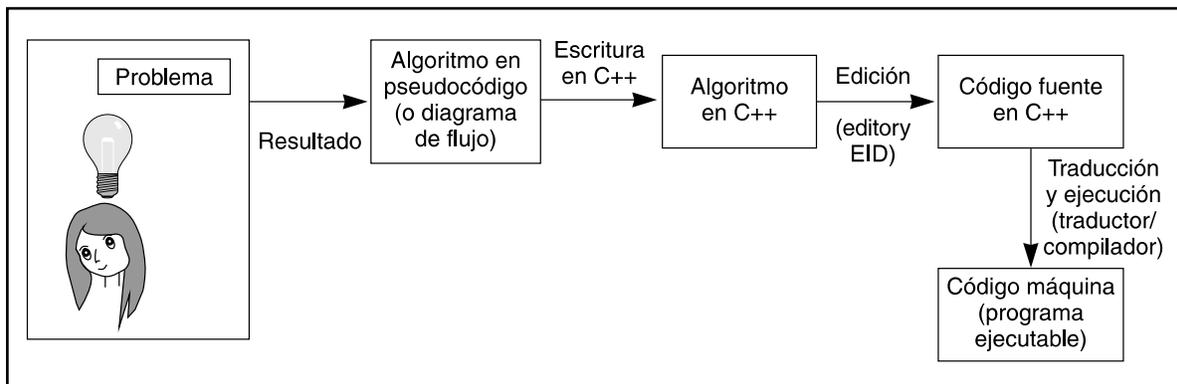
Con estas nociones en claro, se puede afirmar que un lenguaje de programación es aquella estructura que, con una cierta base sintáctica y semántica, imparte distintas instrucciones a un programa de computadora.

A la hora de establecer el origen del lenguaje de programación tenemos que hacer referencia, sin lugar a dudas, a Ada Lovelace que está considerada como la primera programadora de computadoras conocida en todo el mundo. De ahí, curiosamente que se hablara en su honor del lenguaje de programación Ada. Y es que dicha figura llevó a cabo no sólo la manipulación de una serie de símbolos para una máquina del científico británico Charles Babbage sino también la consecución del establecimiento de las instrucciones necesarias para que un computador pudiera realizar una serie de cálculos iniciales.

Dentro de lo que es el lenguaje de programación es muy importante subrayar que los profesionales que se dedican a desarrollar este trabajan con un conjunto de elementos que son los que dan forma y sentido al mismo, los que permiten que aquellos funcionen y logren sus objetivos. Entre los mismos se encontrarían, por ejemplo, las variables, los vectores, los bucles, los condicionantes, la sintaxis o la semántica estática (Lenguaje de programación, 2013)

Un programa se escribe en un lenguaje de programación y las operaciones que conducen a expresar un algoritmo en forma de programa se llaman programación. Así pues, los lenguajes utilizados para escribir programas de computadoras son los lenguajes de programación y programadores son los escritores y diseñadores de programas. El proceso de traducir un algoritmo en pseudocódigo a un lenguaje de programación se denomina codificación, y el algoritmo escrito en un lenguaje de programación se denomina código fuente.

Figura. Proceso de transformación de un algoritmo en pseudocódigo en un programa ejecutable



En la realidad la computadora no entiende directamente los lenguajes de programación sino que se requiere un programa que traduzca el código fuente a otro lenguaje que sí entiende la máquina directamente, pero muy complejo para las personas; este lenguaje se conoce como lenguaje máquina y el código correspondiente código máquina. Los programas que traducen el código fuente escrito en un lenguaje de programación (tal como C++) a código máquina se denominan traductores.

Hoy en día, la mayoría de los programadores emplean lenguajes de programación como C++, C, C#, Java, Visual Basic, XML, HTML, Perl, PHP, JavaScript..., aunque todavía se utilizan, sobre todo profesionalmente, los clásicos COBOL, FORTRAN, Pascal o el mítico BASIC. Estos lenguajes se denominan lenguajes de alto nivel y permiten a los profesionales resolver problemas convirtiendo sus algoritmos en programas escritos en alguno de estos lenguajes de programación.

## 4.1 FORMA DE EJECUCIÓN

### 4.1.1 LENGUAJES COMPILADOS

Naturalmente, un programa que se escribe en un lenguaje de alto nivel también tiene que traducirse a un código que pueda utilizar la máquina. Los programas traductores que pueden realizar esta operación se llaman compiladores. Éstos, como los programas ensambladores avanzados, pueden generar muchas líneas de código de máquina por cada proposición del programa fuente. Se requiere una corrida de compilación antes de procesar los datos de un problema.

Los compiladores son aquellos cuya función es traducir un programa escrito en un determinado lenguaje a un idioma que la computadora entienda (lenguaje máquina con código binario). Al usar un lenguaje compilado (como lo son los lenguajes del popular Visual Studio de Microsoft), el programa desarrollado nunca se ejecuta mientras haya errores, sino hasta que luego de haber compilado el programa, ya no aparecen errores en el código.

#### 4.1.2 LENGUAJES INTERPRETADOS

Se puede también utilizar una alternativa diferente de los compiladores para traducir lenguajes de alto nivel. En vez de traducir el programa fuente y grabar en forma permanente el código objeto que se produce durante la corrida de compilación para utilizarlo en una corrida de producción futura, el programador sólo carga el programa fuente en la computadora junto con los datos que se van a procesar. A continuación, un programa intérprete, almacenado en el sistema operativo del disco, o incluido de manera permanente dentro de la máquina, convierte cada proposición del programa fuente en lenguaje de máquina conforme vaya siendo necesario durante el proceso de los datos. No se graba el código objeto para utilizarlo posteriormente.

La siguiente vez que se utilice una instrucción, se le debe interpretar otra vez y traducir a lenguaje máquina. Por ejemplo, durante el procesamiento repetitivo de los pasos de un ciclo, cada instrucción del ciclo tendrá que volver a ser interpretado cada vez que se ejecute el ciclo, lo cual hace que el programa sea más lento en tiempo de ejecución (porque se va revisando el código en tiempo de ejecución) pero más rápido en tiempo de diseño (porque no se tiene que estar compilando a cada momento el código completo). El intérprete elimina la necesidad de realizar una corrida de compilación después de cada modificación del programa cuando se quiere agregar funciones o corregir errores; pero es obvio que un programa objeto compilado con antelación deberá ejecutarse con mucha mayor rapidez que uno que se debe interpretar a cada paso durante una corrida de producción (Lenguajes de programación – según su forma de ejecución, 2013)

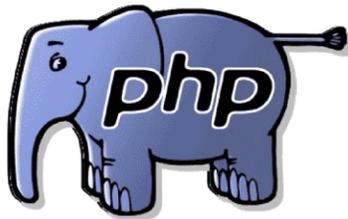
## 4.2 LENGUAJES DE PROGRAMACIÓN EXISTENTES

### 4.2.1 ASP.NET

ASP.NET es un framework para aplicaciones web desarrollado y comercializado por Microsoft. Es usado por programadores para desarrollar sitios web dinámicos, aplicaciones web y de escritorio y servicios webXML. Apareció en enero de 2002 con la versión 1.0 del .NET Framework y es la tecnología sucesora de las Active Server Pages (ASP). ASP.NET está construido sobre el Common Language Runtime, permitiendo a los programadores escribir código ASP.NET usando cualquier lenguaje admitido por el .NET Framework (Lenguajes de servidor).



### 4.2.2 PHP



PHP se considera un lenguaje interpretado (Conocido como de alto rendimiento). En origen fue creado para la creación de páginas web dinámicas. Habitualmente es usado como el código de programación del lado del servidor (server-side scripting) aunque en la actualidad se puede usar desde interfaces de línea de comandos o para crear otros tipos de programas incluyendo aplicaciones con interfaz gráfica, mediante el uso de las bibliotecas Qt o GTK+.

### 4.2.3 VB.NET

Visual Basic es uno de los lenguajes de servidor permitidos por NetFramework junto con C Sharp (C#). Ambos son lenguajes de programación dirigidos por eventos, esto quiere decir que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, bien sean definidos por el usuario o que ellos mismos provoquen.



Visual Basic fue desarrollado por Alan Cooper para Microsoft. Este lenguaje de programación es una evolución del BASIC, al que le han agregado importantes mejoras. Su primera versión fue presentada en 1991, con la intención de simplificar la programación utilizando un ambiente de desarrollo completamente gráfico que facilitara la creación de interfaces gráficas y, en cierta medida, también la programación misma.

#### 4.2.4 SQL



SQL quiere decir lenguaje estructurado de consulta o SQL (structured query language) es un lenguaje declarativo para el acceso y ejecución de operaciones en bases de datos relacionales. Se caracteriza por el manejo del álgebra y el cálculo relacional permitiendo efectuar consultas rápidas y precisas de una forma sencilla, así como también hacer cambios sobre ella. SQL permite trabajar con diferentes bases de datos como SQLServer, MySQL u Oracle.

#### 4.2.5 HTML

Al mismo tiempo existen otros lenguajes no menos importantes e igualmente necesarios como son el HTML, JavaScript, Ajax, etc.. Estos son lenguajes interpretados por el usuario en su navegador, en los que el servidor no realiza ningún tipo de interpretación. Suelen utilizarse para la creación del aspecto visible de las páginas, aunque algunos como JavaScript o Ajax, tienen capacidad para interactuar con bases de datos, servicios y aplicaciones externas.



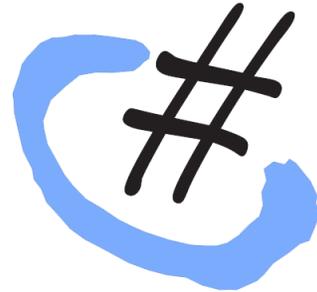
#### 4.2.6 C

Es un lenguaje de “medio nivel” pero con numerosas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Aprender C es básico mientras se aprende C se están aprendiendo conceptos básicos de lenguajes como Java o C#, además no sólo es más sencillo que estos últimos sino que comporten gran parte de su sintaxis.

#### 4.2.7 C#

C# es un lenguaje de propósito general orientado a objetos creado por Microsoft para su plataforma .NET. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes. C# fue diseñado para combinar el control a bajo nivel de lenguajes como C y la velocidad de programación de lenguajes como Visual Basic.

Es una parte esencial de la plataforma .Net, C# combina los mejores elementos de múltiples lenguajes de amplia difusión como C++, Java, Visual Basic o Delphi. De hecho, su creador Anders Heljsberg fue también el creador de muchos otros lenguajes y entornos como Turbo Pascal, Delphi o Visual J++. La idea principal detrás del lenguaje es combinar la potencia de lenguajes como C++ con la sencillez de lenguajes como Visual Basic, y que además la migración a este lenguaje por los programadores de C/C++/Java sea lo más inmediata posible.



#### 4.2.8 JAVASCRIPT



**JavaScript**

Se trata de un lenguaje de programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. Gracias a su compatibilidad con la mayoría de los navegadores modernos, es el lenguaje de programación del lado del cliente más utilizado.

La razón de mayor peso es que es utilizado por millones de páginas webs para validar formularios, crear cookies, detectar navegadores y mejorar el diseño, su fácil aprendizaje lo hace un lenguaje muy demandado.

#### 4.2.9 AJAX

AJAX no es un lenguaje exactamente su nombre viene dado por el acrónimo de Asynchronous JavaScript And XML y es posiblemente la

mayor novedad en cuanto a programación web en estos últimos años. El corazón de Ajax es el objeto XMLHttpRequest que nos permite realizar una conexión al servidor y al enviarle una petición y recibir la respuesta que procesaremos en nuestro código Javascript, estamos hablando del verdadero motor de Ajax, por ejemplo gracias a este objeto se puede desde una página HTML leer datos de una web o enviar datos de un formulario sin necesidad de recargar la página.

#### 4.2.10 RUBY Y RUBY ON RAILS

Ruby on Rails, también conocido como RoR o Rails es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby. Ruby apareció en el año 1995 y creo que su principal problema había sido la falta de documentación en otro idioma que no sea japonés. Eso se ha ido solucionando y crece la popularidad del lenguaje. Su aplicación insignia, por decirlo de algún modo parece ser RoR. Su mecanismo de gem se me parece al CPAN de Perl y al Pear de PHP.

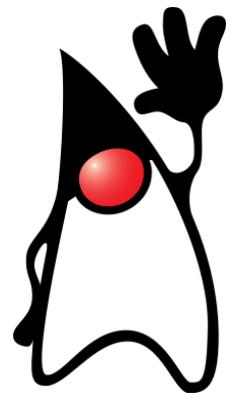
#### 4.2.11 PERL

Perl es la alternativa más popular a PHP, seguramente porque es el lenguaje más antiguo también dentro de las alternativas. En internet nos encontramos numerosos recursos que utilizan Perl, muchos de las aplicaciones “open source” requieren tener Perl instalado correctamente. Perl tiene una ventaja y es que es muy flexible, y también tiene un gran cantidad de módulos ya escritos. Bien escritos los scripts en Perl se asemejan bastante a PHP. La potencia de Perl a la hora de procesar grandes cantidades de datos lo hace realmente popular a la hora de desarrollar aplicaciones del lado del servidor, aprender Perl o PHP es básico a la hora de desarrollar aplicaciones Web (Lenguajes de programación que deberías aprender, 2013)

#### 4.2.12 JAVA

Java es una tecnología que se usa para el desarrollo de aplicaciones que convierten a la Web en un elemento más interesante y útil. Java no es lo mismo que javascript, que se trata de una tecnología sencilla que se usa para crear páginas web y solamente se ejecuta en el explorador.

Java le permite jugar, cargar fotografías, chatear en línea, realizar visitas virtuales y utilizar servicios como, por ejemplo, cursos en línea, servicios bancarios en línea y mapas interactivos. Si no dispone de Java, muchas aplicaciones y sitios web no funcionarán.



Es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir del 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados (Whats JAVA, 2013)

### En resumen...

La siguiente tabla muestra (Lenguajes de programación, 2013) una breve lista de algunos lenguajes de programación:

Lenguaje	Principal área de aplicación	Compilado/ interpretado
ADA	Tiempo real	Lenguaje compilado
BASIC	Programación para fines educativos	Lenguaje interpretado
C	Programación de sistema	Lenguaje compilado
C++	Programación de sistema orientado a objeto	Lenguaje compilado
Cobol	Administración	Lenguaje compilado
Fortran	Cálculo	Lenguaje compilado
Java	Programación orientada a Internet	Lenguaje intermediario
MATLAB	Cálculos matemáticos	Lenguaje interpretado
Cálculos matemáticos	Cálculos matemáticos	Lenguaje interpretado
LISP	Inteligencia artificial	Lenguaje intermediario

Pascal	Educación	Lenguaje compilado
PHP	Desarrollo de sitios web dinámicos	Lenguaje interpretado
Inteligencia artificial	Inteligencia artificial	Lenguaje interpretado
Perl	Procesamiento de cadenas de caracteres	Lenguaje interpretado

### En resumen...

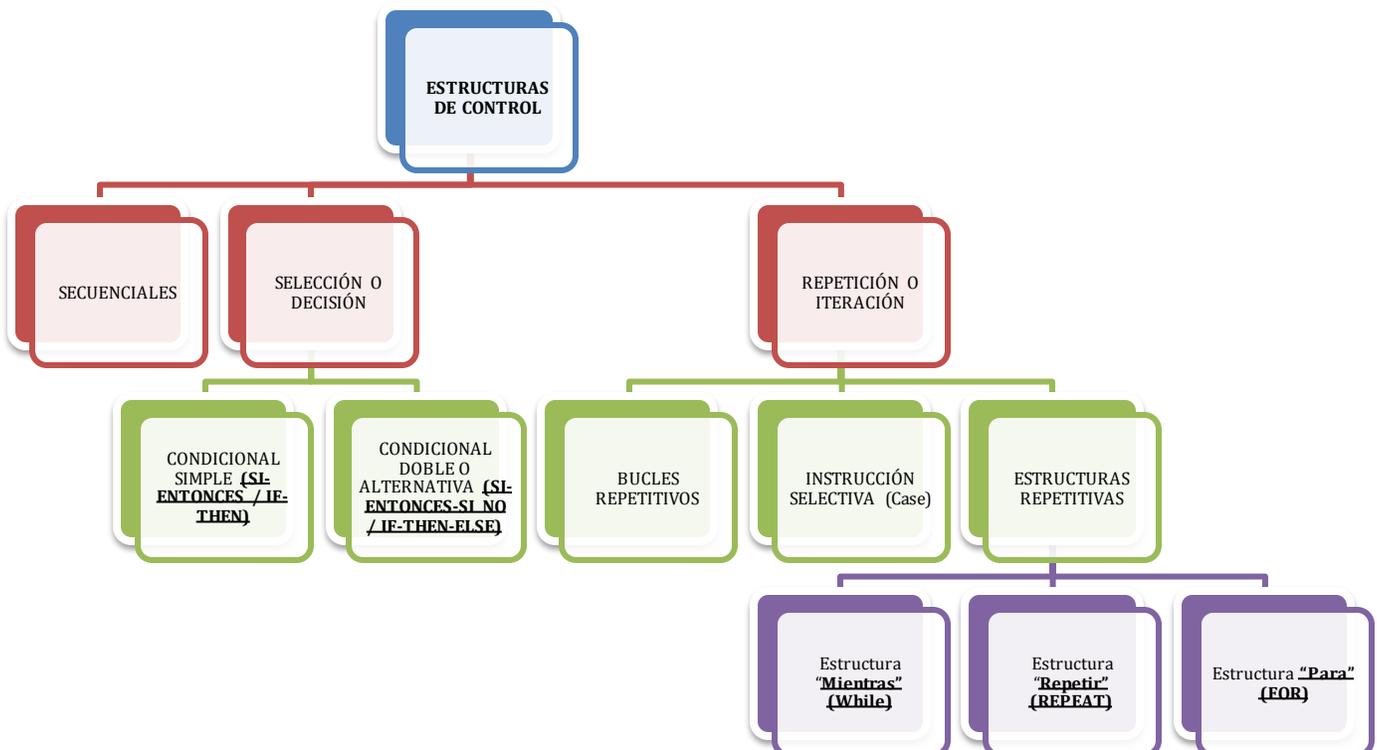
Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente.

## 5. ESTRUCTURAS BASICAS O DE CONTROL

En la elaboración de algoritmos se utilizan estructuras básicas o de control ya prediseñadas para el tratamiento de información, estas estructuras básicas traducen acciones que se realizan de acuerdo al requerimiento o al proceso necesario al cual deba someterse la información. Estas estructuras son:

- a) **Secuenciales:** cuando se requiere que una instrucción siga después de otra.
- b) **Selección o decisión:** se utiliza cuando se requiere tomar decisiones lógicas, la ejecución de las instrucciones dependerá de que se cumplan o no, una o varias condiciones.
- c) **Repetición o Iteración:** se utiliza cuando un proceso debe repetirse un número determinado o no de veces, una vez se haya establecido cierta condición para finalizar el proceso de repetición.

Figura. Resumen de las estructuras de control



Asimismo dentro de las estructuras básicas existen acciones o procesos a los cuales son sometidos los datos, entre ellos, tenemos:

- a. Asignación
- b. Condicionado (a través de las expresiones lógicas)
- c. Alternativas (estructura condicional)
- d. Iterativas
- e. De entrada y salida

## **5.1. ESTRUCTURAS SECUENCIALES**

En esta estructura una acción o instrucción se ejecuta detrás de otra en orden y secuencia. Las tareas se realizan de tal manera que debe cumplirse en estricto orden secuencial, porque la salida de una, es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.

## ESTRUCTURA SECUENCIAL

Esta estructura obedece a operaciones dadas en el siguiente orden:

Contiene operaciones de:

ASIGNACIÓN

CÁLCULO

SUMARIZACIÓN

- a) Inicio
- b) Definición e Inicialización de variables
- c) Lectura de Datos
- d) Operaciones de asignación
- e) Cálculo
- f) Sumarización o totalización
- g) Fin

Posee una entrada y una salida, la representación de una estructura secuencial en Pseudocódigo, se realiza, de la siguiente manera:

Sigamos con el ejemplo de realizar la suma de dos números:

### Entrada

Instrucciones de declaración  
inicialización de variables (Asignación)  
y de lectura de los datos de entrada

### Entrada

SUMA = 0, A = 0, B = 0  
  
Leer A y B

### Proceso

Instrucciones de Cálculo / Sumarización  
Asignación

**Proceso** (Asignar a SUMA el valor de A más el valor de B)

SUMA = A + B

### Salida

Instrucciones de totalización e Impresión

### Salida

Imprimir valor de SUMA

## 5.1.1. DECISIONES EN SECUENCIA

Se utiliza cuando se deben realizar preguntas sin que se tome en cuenta lo contrario a la condición, es decir las demás condiciones no son importantes para el objetivo de la decisión.

**Ejemplo:** Realizar un algoritmo que permita seleccionar sólo aquellas participantes del concurso Mis Venezuela que cumplan ciertas condiciones en una primera ronda de selección.

<b>Estructura</b>	“Sólo si cumplen estas cuatro primeras condiciones pasaran a la segunda ronda”.
<b>Si</b> Condición 1 se cumple <b>Entonces</b> Instrucciones a ejecutar en caso de que la condición1 sea verdadera	<b>Si</b> Rostro = “HERMOSO” <b>Entonces</b> Sumar 1 a Puntaje
<b>Si</b> Condición 2 se cumple <b>Entonces</b> Instrucciones a ejecutar en caso de que la condición2 sea verdadera	<b>Si</b> Estatura >= 1.80 <b>Entonces</b> Sumar 1 a Puntaje
<b>Si</b> Condición 3 se cumple <b>Entonces</b> Instrucciones a ejecutar en caso de que la condición3 sea verdadera	<b>Si</b> Medidas = “90-60-90” <b>Entonces</b> Sumar 1 a Puntaje
<b>Si</b> Condición 4 se cumple <b>Entonces</b> Instrucciones a ejecutar en caso de que la condición4 sea verdadera	<b>Si</b> EstudiosUniversitarios = “SI” <b>Entonces</b> Sumar 1 a Puntaje
<b>Fin Si</b>	<b>Fin Si</b>
<b>Fin Si</b>	<b>Fin Si</b>
<b>Fin Si</b>	<b>Fin Si</b>
<b>Fin Si</b> Imprimir resultados	<b>Fin Si</b> Imprimir sólo las que Puntaje = 4
<b>Fin</b>	<b>Fin</b>

La estructura secuencial es útil para aquellos procesos en los que se requiere que se cumplan con estricto orden un número determinado de instrucciones, siempre y cuando se cumpla la anterior, por lo general, la primera decisión corresponde al aspecto más determinante o general. En el caso de las postuladas la más general o determinante debería ser, la nacionalidad, si la participante es venezolana, entra a la primera selección, de lo contrario, no entra al concurso.

## 5.2. ESTRUCTURA DE SELECCIÓN O DECISIÓN

Un algoritmo se realiza para resolver un problema. Por ello, al elaborar la solución de un problema se trazan ciertas condiciones. Estas condiciones se describen a través de una estructura selectiva, también llamada de decisión o condición. Una estructura selectiva esta compuesta por una expresión lógica, si al evaluar esta expresión lógica, el resultado es “Verdadero”, es decir se cumple la condición, se realizará una secuencia de instrucciones; pero si el resultado es falso, se ejecutará otra secuencia de instrucciones.

Las estructuras selectivas, de decisión o condicionales, pueden ser:

Simple, Dobles y Anidadas o Múltiples.

### 5.2.1 ESTRUCTURA CONDICIONAL SIMPLE (SI-ENTONCES / IF-THEN)

La estructura condicional simple, **Si-Entonces** (o IF-THEN, en inglés), permite evaluar una condición determinada y si se cumple la condición ejecuta una o varias instrucciones.

Si la condición es falsa, entonces no se realizará ninguna acción.

<b>Pseudocódigo en Español</b>	<b>Pseudocódigo en Inglés</b>
<b>Si</b> <Condición> <b>Entonces</b>  <Acción N>  <b>Fin_si</b>	<b>If</b> <Condición> <b>then</b>  <Acción N>  <b>end_if</b>

<b>ESTRUCTURAS DE DECISIÓN</b>	Se utiliza cuando se requiere tomar decisiones lógicas
Pueden ser de instrucciones:	<b>Ejemplo: De acuerdo al ejercicio de la asignación del bono a empleados y aprendices, utilizando la estructura simple, tenemos:</b>
SIMPLES	
DOBLES	Bono: 0
COMPUESTAS	<b>Si Edad <math>\geq</math> 18 Entonces</b>
MULTIPLES (Case)	(Calcular Bono de empleado)
La estructura simple obedece a evaluar una condición, si se cumple se realizará una o un conjunto de instrucciones, y finalizará la ejecución del programa.	Bono = Salario * 0.50
	<b>Fin_si</b>
<b>If &lt;Condición&gt; Then</b>	Total Salario = Salario + Bono
<b>Endif</b>	

```

graph TD
    Start(( )) --> Condicion{Condición?}
    Condicion -- Verdadero --> Accion[Acción]
    Condicion -- Falso --> Exit(( ))
    Accion --> Exit
    style Start fill:none,stroke:none
    style Exit fill:none,stroke:none
  
```

Si la instrucción se diseña sólo para evaluar la condición de verdadero cuando se cumpla que el empleado es “fijo”, se estaría realizando el cálculo sólo para el personal fijo, si se requiere realizar el cálculo incluyendo a los aprendices se debe utilizar la estructura doble.

### **5.2.2 ESTRUCTURA CONDICIONAL DOBLE O ALTERNATIVA (SI-ENTONCES-SI NO / IF-THEN-ELSE)**

La estructura condicional simple es limitada porque permite la evaluación de una sola condición a la vez, la estructura condicional doble o alternativa permite evaluar una condición, la cual puede tener dos acciones, cuando se cumple, y cuando no se cumple. Si se cumple, se realizará una o un conjunto de instrucciones A, si no se cumple, se realizará una o un conjunto de instrucciones B.

## ESTRUCTURA CONDICIONAL DOBLE O ALTERNATIVA

Esta estructura obedece a evaluar una condición y en función del resultado, se realiza un conjunto de instrucciones u otras.

**Ejemplo:** De acuerdo al ejercicio de la asignación del bono a empleados y aprendices, tenemos:

(Pseudocódigo Español)

(Pseudocódigo Inglés)

**If** Condición **Then**

<Instrucciones>

**Else**

<Instrucciones>

**Endif**

**Si** Edad  $\geq$  18 **Entonces**

(Calcular Bono de empleado)

Bono = Salario \* 0.50

**Si no**

(Calcular Bono de aprendiz)

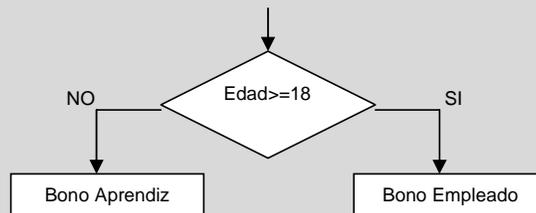
Bono = Salario \* 0.40

**Fin\_si**

Total Salario = Salario + Bono

Nota: En la data sólo están registrados empleados y aprendices

Su Diagrama de Flujo se representaría de la forma siguiente:



## 5.3. ESTRUCTURAS DE REPETICIÓN O ITERACIÓN

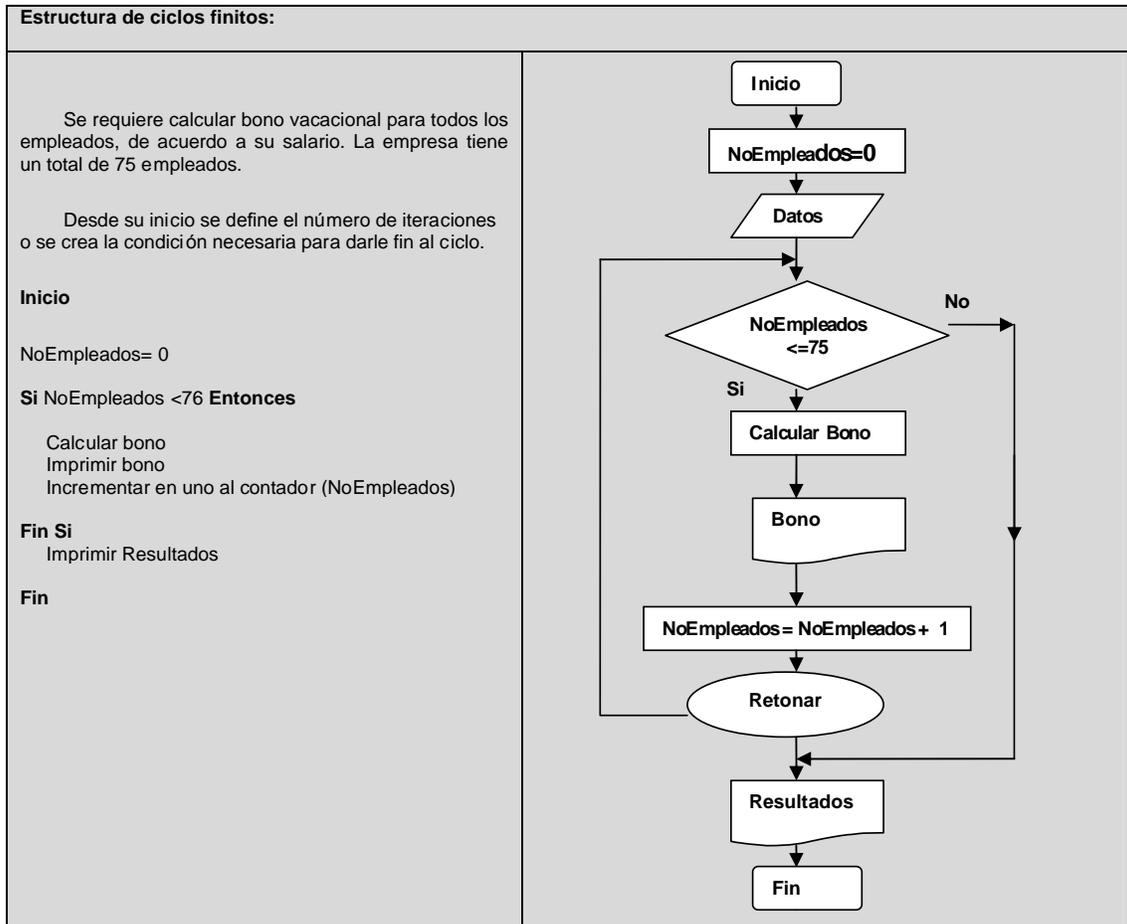
Esta estructura se utiliza cuando se debe ejecutar un conjunto de instrucciones un número repetido de veces. Al conjunto de instrucciones que se ejecutan repetidamente, un número de veces, se le llama también ciclo, bucle o lazo. El número de veces que se ejecuta se denomina **Iteraciones**; por consiguiente, una iteración, es una de las veces en las cuales se efectúan todas las instrucciones contenidas en el ciclo.

### Pasos de una estructura anidada o cíclica:

1. Entrada de datos e instrucciones previas
2. Lazo o bucle
3. Instrucciones finales o resto del proceso

#### 4. Salida de resultado

Las repeticiones deben ser finitas, no obstante puede ser que en momentos determinados no nos demos cuenta y construimos un ciclo o bucle infinito.



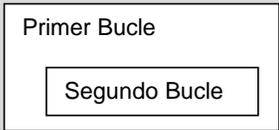
Estructura de ciclos infinitos:	
<p>Se requiere calcular el bono vacacional para todos los empleados, de acuerdo a su salario, al tiempo de vacaciones colectivas (15 días). La empresa tiene un total de 75 empleados.</p>	<p>Leer datos</p> <p>Calcular bono</p> <p>Imprimir bono</p> <p>Nota: El proceso se realizará una sola vez porque no existe una instrucción de condición que indique cuando debe finalizar.</p>

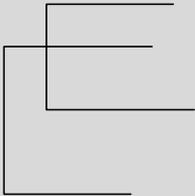
### **5.3.1 BUCLES REPETITIVOS**

Se pueden dar en tres condiciones:

- a. Estructura cíclica independiente, es cuando los bucles se realizan uno primero hasta que se cumple la condición, y sólo una vez que cumple con el primer ciclo, entra al siguiente o a instrucciones de finalización.
  
- b. Los ciclos anidados son aquellos en los que al entrar a una estructura de repetición, dentro de ella se encuentra otra, se inicia el proceso de la estructura más interna hasta que se termine y luego se continúa con la externa hasta que la condición se cumpla.
  
- c. Existen los bucles cruzados, lo cual no se recomienda, debido a que se tiene que interrumpir, es decir no finalizamos un ciclo e iniciamos el otro. Esto puede ocasionar la pérdida de control debido a que el programa podría no reconocer cual proceso se esta cumpliendo.

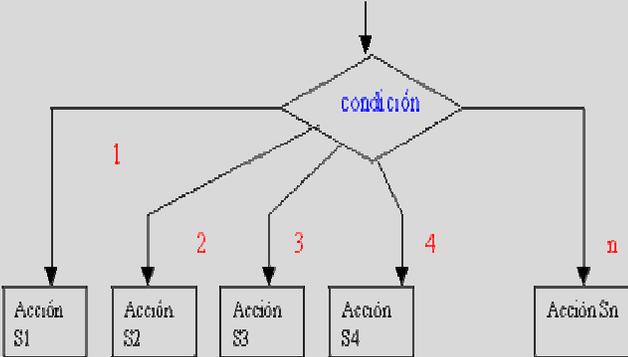
A) Ciclo Independiente	
<p>Se debe cumplir un primer ciclo y luego continuar con el siguiente.</p> <p>En el ejemplo de la selección de las candidatas al Miss Venezuela, se debe cumplir con el primer ciclo completo con todas las aspirantes y las seleccionadas pasarán al segundo proceso de selección.</p>	<p>Inicio</p> <p>Realizar primer proceso de selección</p> <p>Realizar segundo proceso de selección</p> <p>Imprimir las 25 finalistas</p> <p>Fin</p>  <pre> graph TD     Inicio --&gt; A[Primera selección]     A --&gt; B[Segunda selección]     B --&gt; C[Imprimir Resultados]     C --&gt; Fin   </pre>

B) Ciclo Anidados	
<p>Se incluye un ciclo dentro del otro, se debe completar el ciclo más interno y luego el más externo.</p>	<p>Inicio</p>  <pre> graph TD     Inicio --&gt; PB[Primer Bucle]     PB --&gt; SB[Segundo Bucle]     SB --&gt; PB     PB --&gt; Fin   </pre> <p>Fin</p>

C) Ciclo Cruzados	
<p>No es recomendable; sucede cuando en un ciclo iniciado se inicia otro ciclo de instrucciones; puede ocasionar que no se reconozca el ciclo donde suceda el proceso, o se distorsione la información.</p>	

### **5.3.2 INSTRUCCIÓN SELECTIVA (Case)**

Se utiliza cuando existen más de dos opciones posibles; se conoce también como opciones múltiples. La estructura de decisión múltiple o selectiva evaluará una expresión que podrá tomar un conjunto de valores distintos 1, 2, 3, 4, n, es decir hasta n valores. Según la elección del valor de la condición establecida, se realizará un conjunto de instrucciones.

Estructura de instrucción selectiva	Algoritmo
<p>Se requiere evaluar o ejecutar en distintas condiciones una serie de instrucciones.</p> <p>Entre <b>En Caso</b> y <b>Fin Caso</b>, se pueden incluir todas las alternativas que se considere puedan presentarse.</p> <p><b>En Inglés, se utiliza:</b></p> <p><b>Case</b> Día-semana expresión <b>Of</b></p> <p>[Lunes]: actividades 1</p> <p>[Martes]: actividades 2</p> <p style="text-align: center;">:</p> <p>[Domingo]: Salir del proceso de actividades</p> <p><b>Else</b></p> <p>Imprimir resultados</p> <p><b>End_case</b></p>	<p>Leer datos (Empleados, Día-Semana, Actividades)</p> <p><b>En caso:</b></p> <p>Día-semana = "Lunes"</p> <p style="padding-left: 40px;">Realizar actividades 1</p> <p>Día-semana = "Martes"</p> <p style="padding-left: 40px;">Realizar actividades 2</p> <p>Día-semana = "Miércoles"</p> <p style="padding-left: 40px;">Realizar actividades 3</p> <p>Día-semana = "Jueves"</p> <p style="padding-left: 40px;">Realizar actividades 4</p> <p>Día-semana = "Viernes"</p> <p style="padding-left: 40px;">Realizar actividades 5</p> <p>Día-semana = "Sábado"</p> <p style="padding-left: 40px;">Realizar actividades 6</p> <p>Día-semana = "Domingo"</p> <p style="padding-left: 40px;">Salir del proceso de actividades</p> <p><b>Fin Casos</b></p> 

### 5.3.3 ESTRUCTURAS REPETITIVAS (Mientras, Repetir y Para)

Son estructuras diseñadas para que repitan una secuencia de instrucciones un número determinado de veces, también son llamadas bucles o lazos. El número de veces que realiza el proceso se denomina iteraciones, e iteración al hecho de repetir la ejecución de una secuencia.

Entre las estructuras repetitivas se encuentran:

Inglés	Español	Evaluación y acciones
<b>While</b>	Mientras	Mientras se cumpla la condición que desencadena el proceso, las instrucciones que se encuentran dentro del ciclo While se realizarán.
<b>Repeat</b>	Repetir	Se realizará la ejecución de una o más instrucciones "Hasta" que el resultado de la expresión lógica evaluada sea "verdadero".
<b>For</b>	Para	Se realizará una secuencia de acciones un número predeterminado de veces.

### Formas de terminar con bucles o ciclos de repetición a partir de los datos de entrada

a. Preguntando antes de la iteración.	Antes de iniciar el ciclo, se comienza preguntando si existen más datos, se realiza el proceso y dentro del ciclo se vuelve a preguntar "¿Existen más datos?", de existir, sigue el ciclo, de no existir más datos culmina el ciclo, este método es tedioso si se trabaja con grandes listas de datos.
b. Encabezar la lista de datos con su tamaño.	Mientras se cumpla la condición que desencadena el proceso, las instrucciones que se encuentran dentro del ciclo While se realizarán.
c. Finalizar la lista con su valor de entrada.	Se realizará la ejecución de una o más instrucciones "Hasta" que el resultado de la expresión lógica evaluada sea "verdadero".
d. Agotar los datos de entrada.	Se comprueba que no existen más datos de entrada.

#### 5.3.3.1 Estructura "Mientras" (While)

La estructura repetitiva While, según Joyanes (2003): "es aquella en que el cuerpo del bucle se repite mientras se cumple una determinada condición", al cambiar la condición se interrumpe o se culmina con la ejecución del ciclo de instrucciones que se encuentran dentro de la estructura o el ciclo While.

Esta estructura es muy útil cuando dada una condición, se requiere asegurar se realice un conjunto de instrucciones.

Ejemplo en Inglés:	Ejemplo en Español:	Descripción:
<b>While</b> ExistenDatos = "Si"  <b>Wend</b>	<b>Mientras</b> ExistenDatos = "Si"  <i>Fin Mientras</i>	Mientras la variable que contiene la condición de verificación de existencia de datos, en este caso, se cumpla se realizarán las instrucciones que contiene el ciclo Mientras.
<b>While</b> Número <> 0  <b>Wend</b>	<b>Mientras</b> Número <> 0  <i>Fin Mientras</i>	Mientras el número que se ingrese sea distinto de cero, se realizará el conjunto de instrucciones que contiene la estructura Mientras. El ciclo culmina al introducir un 0.

## ESTRUCTURA WHILE

(Inglés)

**While** <condición>do

*Acciones*  
**Wend**

**Ejemplo:** Realizar un algoritmo que sume los primeros números hasta encontrar un número negativo.

(Español)

**Mientras** <condición> hacer

*Acciones*  
**Fin Mientras**

**Inicio**

Contador = 0  
Suma = 0

**Leer** (Número)

**Mientras** Número > 0 hacer

Contador = Contador + 1  
Suma = Suma + Número  
**Leer** (Número)

**Fin\_Mientras**

**Escribir**

'El número de enteros positivos es :' Contador

"La sumatoria es: "  
Suma

**Fin**

(Diagrama de Flujo)

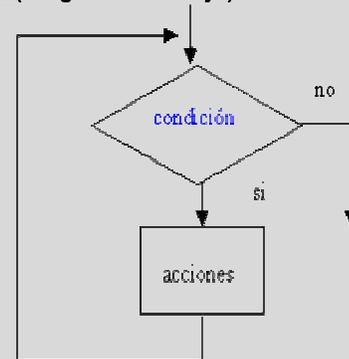
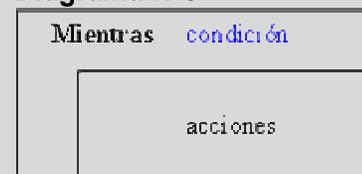
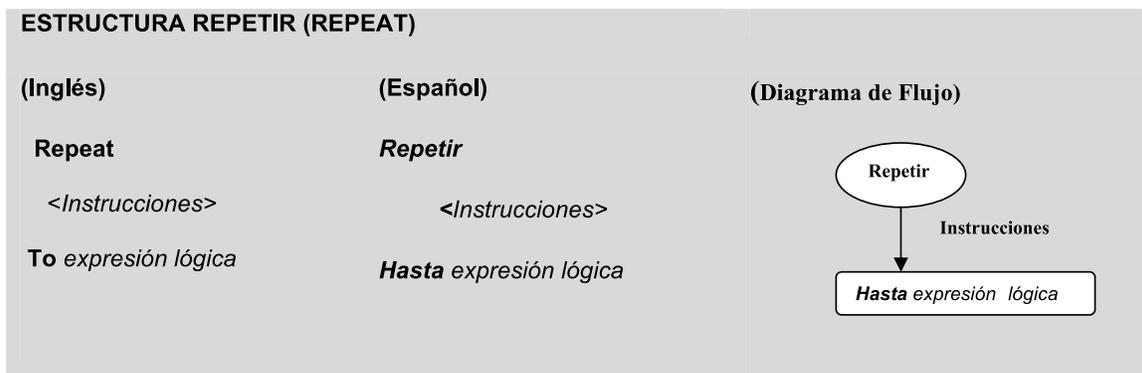


Diagrama N-S



### 5.3.3.2 Estructura “Repetir” (REPEAT)

Esta estructura permite realizar una o mas instrucciones, un número de veces hasta que se cumpla con una condición programada para que finalice el ciclo; es decir, hasta que el resultado de la expresión lógica evaluada sea “Verdadero”. Utilizar esta estructura permite que el ciclo se realice al menos una vez, debido a que la expresión lógica que debe evaluar su finalización se encuentra después del conjunto de instrucciones que contiene el ciclo.



La palabra “Repetir” indica el inicio de la secuencia de acciones que se repetirán, con la “expresión *lógica*”, indica, el fin del ciclo y hasta cuando se repetirá la secuencia.

La diferencia entre la estructura repetitiva **Mientras** y la estructura **Repeat** es que la estructura **Mientras** termina cuando la condición se hace falsa, por el contrario la estructura **Repeat** finaliza cuando la condición es verdadera. La estructura **Mientras** puede ser que no se ejecute, pero la estructura **Repeat** se realiza al menos una vez.

### 5.3.3.3 Estructura “Para” (FOR)

Al diseñar algoritmos se presentan oportunidades donde es necesario

repetir un conjunto de instrucciones un número predeterminado de veces, para esto se utiliza la estructura **Para** (For), donde la secuencia se repite bajo el control de un elemento variable que se incrementará o disminuirá cada vez que se ejecute el ciclo completo.

ESTRUCTURA PARA (FOR)		Representaciones Gráficas
<b>(Inglés)</b>	<b>(Español)</b>	
<b>For</b> variable_contador = inic inc fin  instrucciones  <b>End</b>  <b>For</b> i = 1 to 20  <Instrucciones>  <b>End</b>	<b>Para</b> <elemento de control>:= valor inicial hasta valor final <b>hacer</b>  <Instrucciones>  <b>FinPara</b>  Cuenta=1  <b>Para</b> Cuenta =1 hasta 20 <b>hacer</b>  <Instrucciones>  <b>Fin Para</b>	

## REFERENCIAS

- Algoritmia-Algoritmos probabilísticos. Fecha de consulta: 1 de Octubre 2013. Disponible en: [http://es.wikibooks.org/wiki/Algoritmia/Algoritmos\\_probabil%C3%ADsticos](http://es.wikibooks.org/wiki/Algoritmia/Algoritmos_probabil%C3%ADsticos)
- Algoritmo determinista. Fecha de consulta: 1 de Octubre 2013. Disponible en: [http://centrodeartigos.com/articulos-informativos/article\\_70664.html](http://centrodeartigos.com/articulos-informativos/article_70664.html)
- Algoritmos - Contextualización. Fecha de consulta: 3 de Octubre 2013. Disponible en: [http://ing.unne.edu.ar/pub/informatica/Alg\\_diag.pdf](http://ing.unne.edu.ar/pub/informatica/Alg_diag.pdf)
- Algoritmos – expresiones. Fecha de consulta: 3 de Octubre 2013. Disponible en: <http://es.scribd.com/doc/86588280/ALGORITMOS-EXPRESIONES>
- Algoritmos y programación. <http://www.eduteka.org/pdfdir/AlgoritmosProgramacion.pdf>
- Análisis y Diseño de Algoritmos Universidad Nacional de Colombia. Fecha de consulta: 3 de Octubre 2013. Disponible en: <http://www.virtual.unal.edu.co/cursos/sedes/manizales/4060024/html/contenido.html>
- Baase, Van Gelder. Algoritmos Computacionales. Ed. Addison Wesley, México. 2002.
- Becerra Santamaría, Cesar. Programación en C.
- Conceptualización general Algoritmos y Programación. Fecha de consulta: 27 de Septiembre 2013. Disponible en: <http://www.capacitateparatriunfar.com/AlgoritmiaII.pdf>
- Definición ABC. Fecha de consulta: 30 de septiembre 2013. Disponible en: <http://www.definicionabc.com/general/dato-2.php#ixzz2g0yVQT6K>
- Diagramas de flujo. Fecha de consulta: 30 de septiembre 2013. Disponible en: <http://www.monografias.com/trabajos59/diagrama-flujo/diagrama-flujo.shtml#ixzz2dOSCErBh>
- Diseño de algoritmos paralelos. Fecha de consulta: 26 de Septiembre 2013. Disponible en: <http://arqui-g3-mel-alo-jc-wil.wikispaces.com/Dise%C3%B1o+de+algoritmos+paralelos>
- Educación básica. Algoritmos y programación. Fecha de consulta: 25 de septiembre 2013. Disponible en: <http://www.eduteka.org/pdfdir/AlgoritmosProgramacion.pdf>
- Ejemplos y ejercicios. Fecha de consulta: 26 de septiembre 2013. Disponible en: <http://proflauracardozo.files.wordpress.com/2013/01/algoejemplos.pdf>
- Fundamentos de la informática. Fecha de consulta: 20 de septiembre 2013. Disponible en:

- <http://www.nebrija.es/~oruano/java/02%20Algoritmos%20diagramas%20de%20flujo%20y%20pseudocodigo.pdf>
- Guía de referencia. Fecha de consulta: 5 de Octubre 2013. Disponible en: <http://www.eduteka.org/ScratchGuiaReferencia.php>
  - Joyanes Aguilar, Luis. Fundamentos de programación: algoritmos y estructuras de datos. - 2ed. Ed. McGrawHill. Madrid 1996.
  - Lenguaje de programación: Qué es, Significado y Concepto. Fecha de consulta: 4 de Octubre 2013. Disponible en: <http://definicion.de/lenguaje-de-programacion/#ixzz2iTU6TA6u>
  - Lenguajes de programación que deberías aprender. Fecha de consulta: 4 de Octubre 2013. Disponible en: <http://www.tufuncion.com/diferentes-lenguajes-programacion>
  - Lenguajes de programación: según su forma de ejecución. Fecha de consulta: 3 de Octubre 2013. Disponible en: <http://jorgesaavedra.wordpress.com/2007/05/05/lenguajes-de-programacion/>
  - Lenguajes de programación. Fecha de consulta: 2 de Octubre 2013. Disponible en: <http://es.kioskea.net/contents/304-lenguajes-de-programacion>
  - Lenguajes de programación. Fecha de consulta: 2 de Octubre 2013. Disponible en: [http://www.elinformatico.org/Profesional\\_informatico\\_programacion\\_web.aspx](http://www.elinformatico.org/Profesional_informatico_programacion_web.aspx)
  - Lozano, Letvin. Programación Estructurada: Básica y Libre. Ed McGrawHill. México.
  - Metodología de la programación y desarrollo de software. Fecha de consulta: 2 de Octubre 2013. Disponible en: [http://www.mhe.es/universidad/informatica/844814645X/archivos/apendice\\_2.pdf](http://www.mhe.es/universidad/informatica/844814645X/archivos/apendice_2.pdf).
  - Norton, Peter. Introducción a la Computación. Ed. McGrawHill. México. 2000.
  - Programa Nacional de Formación en Sistemas e Informática Ministerio de Educación Superior. Fecha de consulta: 3 de Octubre 2013.
  - Tipos de datos. Fecha de consulta: 1 de Octubre 2013. Disponible en: <http://progra.usm.cl/apunte/materia/tipos.html>
  - Verificación de algoritmos. Fecha de consulta: 4 de Octubre 2013. Disponible en: <http://www.infor.uva.es/~felix/datos/prii/verificacion.pdf>
  - Vuelta atrás, Capítulo 6. Fecha de consulta: 4 de Octubre 2013. Disponible en: <http://www.lcc.uma.es/~av/Libro/CAP6.pdf> .
  - Whats JAVA. Fecha de consulta: 4 de Octubre 2013. Disponible en: [http://www.java.com/es/download/whatis\\_java.jsp](http://www.java.com/es/download/whatis_java.jsp)