

II. Introducción a la Programación

Un **algoritmo**, se define como “*un conjunto ordenado y finito de operaciones que permite hallar la solución de un problema*”¹. Si se lee con atención la definición, no se mencionan computadores, lenguajes de programación o dispositivos de procesamiento alguno y esto es debido a que el énfasis de un algoritmo no está en la programación de computadores sino en la definición de los pasos o instrucciones, que deben ser claros y completos para permitir alcanzar un objetivo.

Además del proceso definido (pasos), los algoritmos tienen otros dos elementos que los caracterizan: las entradas y la(s) salida(s). Las entradas o elementos de entrada o datos de entrada son aquellas cosas o datos que se requieren para poder realizar el algoritmo a cabalidad. La salida son los elementos o datos que se obtienen una vez se ejecuten los pasos del algoritmo.

La diferencia básica entre un **algoritmo** y un **programa** es que este último está escrito en un lenguaje de programación, de forma que pueda ser interpretado y ejecutado por un computador.

-Escribir un programa, implica necesariamente, hacer un algoritmo-

Cuando vamos a programar, nuestro cerebro ejecuta la misma secuencia de pasos que aplica en cualquier situación problemática: primero, entender el problema con todas sus condiciones y características; segundo, buscar una solución que se ajuste a dichas condiciones y características; tercero, poner en ejecución la solución hallada; cuarto, verificar si la solución cumple con lo que se indicó que debería hacer².

Ejemplo:

Tenemos una situación problema: “Es necesario llegar desde la UAO hasta Chipichape”.

Primero: entendamos el problema, es claro hay que encontrar una forma de llegar desde la UAO hasta Chipichape, pero hay algunas restricciones que tenemos que considerar: ¿hay restricción de dinero para el transporte? ¿hay restricción de tiempo para el transporte? ¿cuántas personas deben transportarse?

Digamos que debemos transportar a 3 personas, no importa cuánto tiempo se demoren pero no deberían gastar más de \$10.000.

¹ Real Academia Española. Diccionario. 2015. Disponible en: <http://www.rae.es/recursos/diccionarios/drae> [consultado: Julio 20 de 2015]

² En desarrollo de software, esta característica se conoce como corrección, un algoritmo correcto, es un algoritmo que efectivamente cumple lo que se supone que debería hacer.

Ahora si, teniendo claro el problema, entramos al paso dos: buscar alternativas de solución.

- **Alternativa 1:** Tomar un taxi. Permite transportar a 3 personas pero con seguridad costaría más de 10.000 pesos. ¡Descartada!
- **Alternativa 2:** Ir en MIO. Permite transportar a 3 personas y cuesta menos de 10.000 pesos. ¡es una alternativa válida!
- **Alternativa 3:** Ir caminando. Permite el transporte de 3 personas y cuesta menos de 10.000 pesos. ¡es una alternativa válida!

Una vez encontramos alternativas válidas podemos seleccionar una, vamos a seleccionar la Alternativa 2, porque aunque la Alternativa 3 es más barata, tardaría mucho tiempo y las personas llegarían muy cansadas.

Una vez seleccionada la alternativa, procedemos al Tercer paso, implementar la solución. En este caso, la implementación consiste en determinar cuáles rutas del MIO se deberán tomar desde la UAO hasta Chipichape y en cuales estaciones se debe hacer cambio de ruta.

Antes de enviar a las tres personas a aventurarse por las calles de Cali, podemos adelantar el paso Cuatro: Probar la solución. Probar es verificar si con las entradas que hemos definido y los pasos que se han establecido logramos el resultado esperado; entonces, se puede hacer una revisión sobre el mapa, para verificar si las rutas seleccionadas con intercambio en las estaciones definidas permiten que las personas lleguen desde la UAO hasta Chipichape.

Una vez se comprueba que la solución diseñada es apropiada, podemos generalizarla, cualquier persona que tenga el mismo problema con las mismas restricciones, podrá utilizar nuestra solución.

Este es un ejemplo bastante simple, pero representa nuestra forma de pensar y actuar ante cualquier situación problema, lo que sucede es que no siempre somos concientes de este proceso, nuevamente, porque lo tenemos tan interiorizado y lo realizamos de forma tan automática que no lo pensamos paso por paso; sin embargo, si lo piensas por un minuto, ante cualquier situación actúas de la misma forma, bueno, con algunas variaciones, porque algunas situaciones requieren que la etapa uno sea de mayor duración que las otras, pero otras situaciones requieren que la etapa dos sea la de mayor duración, etc.

Estas mismas etapas son las que seguiremos en la elaboración de algoritmos, lo que sucede, es que como todo proceso en Ingeniería, tiene sus particularidades en la forma de realizarlo y comunicarlo.

Actividad 1 : El análisis

La primera etapa en el proceso de programación tiene diferentes nombres, en nuestro caso, la denominaremos **Análisis**, en esta etapa se busca comprender la funcionalidad del programa, así como identificar las restricciones que se tengan sobre la operación del mismo.

Otra forma de comprender la etapa de análisis es preguntarse ¿Qué debe hacer el programa? y ¿Qué condiciones se deben cumplir durante la ejecución para lograrlo?

Ahora, ¿De dónde se obtiene esa información?, la respuesta es muuuy simple, de la persona que necesita el programa. Durante el proceso de aprendizaje, es común que los profesores entreguen por escrito la definición del problema, y de allí básicamente se obtiene la información que se requiere. En situaciones reales, debe preguntarse a la(s) persona(s) que requieren el programa sobre todos los aspectos que este deberá incluir.

Para efectos del curso, en la etapa de análisis se establecerán los datos de entrada y los datos de salida.

- **Datos de entrada:** Son aquellos datos que se requiere conocer para que el programa pueda funcionar apropiadamente. Generalmente se asocian con las “cosas” que se preguntará al usuario.

Un buen programador es conciente que al preguntar un dato, el usuario puede contestar de cualquier forma, por ello, debe asegurar que la información ingresada sea apropiada para el desarrollo del programa, para ello se emplea la validación de datos, por ahora solamente nos centraremos en establecer cuales son los valores válidos para un dato de entrada, más adelante veremos cómo se hace la validación.

- **Datos de salida:** Corresponden a los datos o información que el usuario espera que el programa le presente.

Ejemplo:

En la Universidad de la Region se quiere desarrollar un programa que permita determinar rápidamente si un estudiante ha perdido una asignatura por inasistencia y en cualquier caso, su nota definitiva. Para esto debe tenerse en cuenta que un estudiante pierde una asignatura si ha faltado al 20% o más de las horas programadas en un curso, en ese caso, su nota definitiva será 1.5; de otra forma, su nota se calcula como el promedio de cuatro exámenes que se presentan durante el semestre.

Datos de Entrada: Para el desarrollo del programa, es necesario conocer los siguientes datos:

- Número de horas totales programadas en el curso.
- Número de horas que ha faltado el estudiante
- Calificación del primer examen
- Calificación del segundo examen
- Calificación del tercer examen
- Calificación del cuarto examen

Los dos primeros datos permitirán calcular el porcentaje de faltas que ha tenido el estudiante y de esta manera determinar su nota; si no ha perdido la asignatura por faltas, los otros cuatro datos permitirán hacer el cálculo de la definitiva.

Observe que datos como el porcentaje que determina que el estudiante ha perdido la asignatura por faltas o la nota que se asigna en caso de pérdida por faltas, no se incluyen en los datos de entrada porque ya se conocen (20% y 1.5 respectivamente).

Para efectos posteriores de validar los datos, vamos a incluir el rango de valores válidos para cada dato:

- Número de horas totales programadas en el curso (mayor a 0)
- Número de horas que ha faltado el estudiante (mayor o igual a 0)
- Calificación del primer examen (entre 0.0 y 5.0)
- Calificación del segundo examen (entre 0.0 y 5.0)
- Calificación del tercer examen (entre 0.0 y 5.0)
- Calificación del cuarto examen (entre 0.0 y 5.0)

Datos de Salida: Al finalizar el programa se espera que se presente el siguiente dato:

- Nota definitiva del estudiante

OBSERVE QUE: Aunque durante el proceso (programa) deberá calcularse el porcentaje de inasistencias que ha tenido el estudiante, este dato no será mostrado al usuario, por lo tanto no se considera dato de salida.

Actividad 2: El diseño

El diseño de un programa, es realmente el algoritmo del mismo y como tal, contempla las tres secciones mencionadas: entradas, proceso y salidas. Las entradas y salidas se definieron en el paso anterior, así que ahora nos concentraremos en el proceso.

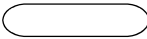


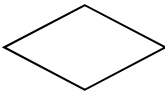


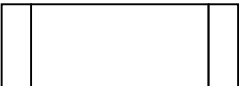

Proceso: Se refiere a todas las instrucciones necesarias para que el computador pueda cumplir el objetivo propuesto para el programa. Este proceso debe describirse en forma


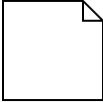
de algoritmo, es decir pasos ordenados, claros y completos que al ser seguidos de forma rigurosa permiten la ejecución del objetivo propuesto.

Existen múltiples formas de representar el proceso de un algoritmo, se puede hacer con lenguaje natural (español o inglés), con lenguaje estructurado (pseudocódigo) o lenguaje gráfico (flujograma). Para nuestro caso se empleará el flujograma, que tiene dos grandes ventajas: es estándar y se gráfico, lo que permite mayor comprensión para los lectores.

FlujoGrama

Durante este curso se empleará la notación de flujograma o diagrama de flujo para presentar el diseño de un programa. El flujograma es una representación gráfica de las instrucciones que se llevarán a cabo en el programa. A continuación se presentan los símbolos a emplear.

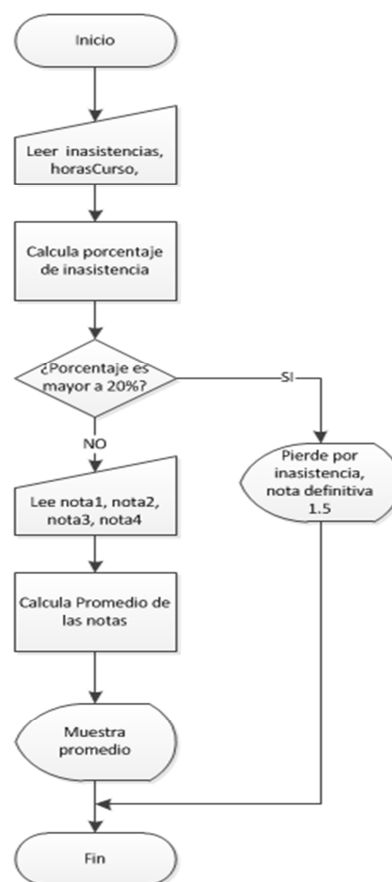
Símbolo	Utilización
	Inicio / Fin Indica los extremos del proceso principal. Dentro del símbolo se deberá escribir la palabra Inicio o Fin. Si este símbolo es utilizado en un subproceso, en el Inicio se deberán escribir las entradas, y en el Fin, se deberá escribir la salida. Solamente deberá aparecer un símbolo de inicio y un símbolo de fin para cada proceso.
	Lectura de Datos Este símbolo se emplea para especificar los datos que se deben leer o ingresar para la realización del algoritmo. Dentro del símbolo se escriben los datos a leer (incluyendo la restricción de los datos, por ejemplo: >0, entre 0 y 20, etc.).
	Proceso Especifica los cálculos requeridos, tales como: evaluación de fórmulas, incrementos, acumulaciones, entre otros. Para cada cálculo se debe especificar la variable donde quedará almacenado el valor.
	Decisiones Este símbolo permite mostrar decisiones que se deben tomar durante la realización del algoritmo, dentro del rombo se escribirá la pregunta o expresión lógica que genera la decisión. Este símbolo se puede emplear para tomar decisiones, en cuyo caso, saldrán del símbolo dos caminos, uno marcado como SI y otro como NO, para indicar las instrucciones que se deben realizar cuando la expresión sea verdadera o falsa, respectivamente.
	Salida Se emplea este símbolo para indicar la información que se va a presentar al usuario.
	Línea de Flujo Indica la secuencia de las instrucciones. Debe tenerse cuidado que las flechas sean rectas (horizontales o verticales) y que no se crucen unas con otras en el diagrama.
	Subproceso Con este símbolo se realiza la invocación o llamado a un subproceso, al interior del cuadro se escribe el nombre del subproceso que se va a invocar. Adicionalmente, se debe especificar los parámetros enviados al subproceso entre paréntesis y la variable en la cual es asignado el valor que éste devuelve.
	Conector Este símbolo permite conectar varias flechas en la misma página.

Símbolo	Utilización
	Conector de página Si el diseño de un algoritmo ocupa varias páginas, se puede emplear este conector para indicar la secuencia de instrucciones, si existen varios conectores, es conveniente numerarlos para evitar alguna confusión.
	Comentario Este símbolo permite incluir comentarios o notas adicionales sobre el algoritmo. Se sugiere utilizar este símbolo (junto al símbolo de Inicio) para incluir el nombre de los procesos.

Ejemplo:

Continuando con nuestro ejemplo y considerando los datos de entrada y salida definidos, pasaremos a desarrollar el diseño del mismo.

A manera de ejemplo, se presenta el siguiente diagrama como una propuesta de solución³:



³ Los flujogramas de ejemplo que se presentan en este documento, así como los que se presentarán posteriormente durante el curso, han sido realizados empleando la herramienta Microsoft Visio 2010. Los estudiantes son libres de emplear esta u otra herramienta e incluso hacer sus diagramas a mano, siempre y cuando respeten los símbolos definidos.

NOTA: como todo proceso de diseño en Ingeniería, pueden presentarse diferentes propuestas para la solución de un problema, sin embargo, debe asegurarse que todos los pasos se cumplan y la salida sea la esperada.

Actividad 3: La implementación

El siguiente paso es realizar la implementación del algoritmo, es decir, traducir las instrucciones definidas a un lenguaje de programación. Si bien cada lenguaje tiene una sintaxis diferente, todos manejan los mismos elementos conceptuales, los cuales si son claros para el programador, le permitirán emplear cualquier lenguaje con un mínimo tiempo de aprendizaje.

Para el desarrollo de nuestro curso, se empleará el lenguaje de programación Visual C#, por lo cual, se presentará la sintaxis específica de este lenguaje al tiempo que se presentan los conceptos generales.

Los elementos básicos de programación se describen a continuación:

1. **Variables**

Las variables en programación son equivalentes a las variables que se emplean en matemáticas, son palabras (a veces de una sola letra) que representan valores que pueden variar a lo largo del programa. Las variables en programación tienen dos características propias: nombre y el tipo.

El nombre de la variable, conocido también como identificador, es el que permite hacer referencia a la variable en cualquier parte del programa. Cada lenguaje define algunos limitantes para la selección del nombre de las variables, sin embargo, algunas normas generales de buena programación son las siguientes:

- Una variable debe nombrarse con una o varias palabras que indiquen el dato que contiene, *por ejemplo: nombre, telefono, salario, temperatura, etc.*
- El nombre de una variable no puede incluir espacios ni caracteres especiales. Si el nombre se compone de varias palabras, pueden enlazarse con subrayado o simplemente poner las palabras seguidas una de otra, *por ejemplo: salario_total, temperaturaMedia, promedio_Semestral.*

- El nombre de la variable debe empezar por una letra, posteriormente puede incluir numeros. Si el nombre de una variable inicia con un número, el computador puede entender que se trata de un número y no de una variable. *Por ejemplo, semestre5, login123, etc.*

El tipo de la variable, se refiere al tipo de dato que la variable podrá almacenar. La mayoría de los lenguajes requieren que se establezca el tipo de la variable y que no se cambie durante toda la ejecución del programa. Adicionalmente, cada lenguaje tiene limitaciones propias para cada tipo de datos (como el rango de datos válidos). Los tipos de datos más generales son los siguientes:

- Entero: indica que las variables guardarán números negativos, positivos o cero SIN parte decimal. En C# existen varios tipos enteros: sbyte, short, int, long; la diferencia entre un tipo y otro es el tamaño disponible para almacenar datos. El tipo de dato entero más común es el **int**, que tiene un tamaño de 4 bytes y permite almacenar números entre -2.147'483.648 y 2.147'483.647
- Reales o decimales: indica que las variables guardarán números negativos, positivos o cero incluyendo parte decimal. En C# existen dos tipo de datos reales: float y double. Al igual que con los enteros, la diferente entre ellos se basa en el tamaño disponible. El tipo más comun en los reales es el **float** que tiene 7 dígitos de precisión.
- Booleano: es un tipo de dato especial que permite almacenar valores de verdad (verdadero o falso). En C# se emplea el tipo **bool** y los únicos valores que puede tomar son True o False.
- Caracteres: indica que la variable almacenará caracteres únicos, que pueden ser letras ('a','F','z'), números como símbolos y no como cantidad ('3', '9','0') o caracteres especiales, que corresponden a esos caracteres que se disponen y que no son letras ni números (')', '#', '['). Cuando se emplea un número como símbolo (carácter) no es posible hacer operaciones aritméticas con él. En C#, este tipo se denomina **char** y sus valores, siempre se presentan entre comillas simples.

- Cadenas: algunos lenguajes disponene un tipo de dato especial, denominado cadena, que corresponde a un conjunto de caracteres, por ejemplo “perro”, “mi casa es bonita”, “Maria Martinez”, “ el promedio es 3.45”. En C# este tipo de dato, se denomina **string** y los datos que almacena siempre se representan entre comillas dobles.

2. Constantes

Las constantes se refieren a datos que no cambiarán su valor durante la ejecución del programa, no requieren nombre y solamente se trabajan con el dato correspondiente (3, 6.7, “hola”).

3. Operador de asignación

La definición de una variable supone que en alguna parte del proceso establecido, la variable debe tomar un valor o cambiar el que tiene, una forma de realizar esta operación es asignandole dicho valor, para ello se emplea un operador específico. El operador de asignación empleado por cada lenguaje puede diferir pero el más común es el operador =.

A través del operador de asignación se asigna el valor a la derecha del operador (que podría ser una variable o constante) a la variable que se encuentra a la izquierda del operador.

Ejemplo:

Total = 35.6	asigna el valor 35.6 a la variable total
Nombre = “Miguel”	asigna la cadena Miguel a la variable nombre.
Promedio = nota	asigna a la variable promedio el valor que tenga (en ese momento) la variable nota.
3 = x	es una expresión no válida, porque no se puede asignar un valor a otro valor.

4. Expresiones

Como se explicó en la unidad anterior, las expresiones son sentencias que toman un valor particular; en programación se distinguen las expresiones aritméticas, que dan como resultado un valor numérico y las expresiones lógicas que dan como resultado un valor de verdad (dentro de las expresiones lógicas se incluyen las relacionales).

5. Operaciones

El último elemento constituyente de un programa de computadora son efectivamente las operaciones o instrucciones que se pueden emplear.

Operaciones de lectura: son las operaciones con las que se permite al usuario hacer el ingreso de los datos para que el programa funcione.

Operaciones de escritura: son las operaciones que permiten presentar información en pantalla.

Decisiones: son operaciones que permiten definir un conjunto de instrucciones que se ejecutarán bajo una condición específica.

Repeticiones o ciclos: Permiten definir un conjunto de instrucciones que se repetirán mientras se cumpla una condición establecida.

6. Método

Un método o procedimiento se puede entender como una porción de programa que realiza una tarea específica. Un método puede llamarse o invocarse en cualquier momento dentro de un programa, en ese momento se ejecutan las instrucciones del método y una vez termine se regresa a continuar la ejecución del programa.

Al realizar el llamado o invocación al método, es posible enviar datos que este requiera para realizar su operación, de forma similar, al terminar la ejecución del método, se puede retornar un dato al programa principal para que continúe su labor.

NOTA: Aunque pueden existir variaciones, en general, los métodos (procedimientos, rutinas o funciones) en todos los lenguajes de programación tienen la posibilidad de retornar un solo dato.

Actividad 4: Las pruebas

La última actividad del proceso de desarrollo de software es la de pruebas, en esta parte se ejecuta el programa bajo condiciones controladas para verificar su corrección, es decir, verificar que cumple los requerimientos definidos previamente.

Contrario a lo que se puede pensar, las pruebas no están diseñadas para probar que el programa funciona bien, por el contrario, las pruebas se diseñan intentando que el

programa falle. Si al ejecutar las pruebas, el programa no falla, se puede asegurar que bajo esas condiciones, el programa funciona bien, pero no se puede asegurar que el programa funcionará bien bajo todas las circunstancias.

Cada prueba incluye:

- Una descripción de la característica o funcionalidad que se desea probar.
- Los datos y valores que se deben emplear como entrada al programa.
- Los datos de salida que se deberían obtener.
- Para aplicaciones más complejas, deben indicarse las condiciones del sistema al momento de la prueba.

Cada prueba se ejecuta verificando si la salida obtenida es la esperada, si es así, se indica que la prueba es aprobada, de no ser así, debe empezar un proceso de revisión del programa para encontrar la causa del error y corregirla, una vez realizado este proceso se ejecutan nuevamente todas las pruebas para verificar que se corrigió efectivamente el error y que no se insertó otro error en el proceso.