ALGORITMOS

Cátedra de Introducción a la Programación
Facultad de Informática
Universidad Nacional del Comahue

---

1. INTRODUCCIÓN

En este apunte se describe una introducción a la resolución de problemas destacando la importancia de abstracciones y modelos. El apunte tambien define el concepto de algoritmo detallando como los algoritmos nos ayudan en la resolución de problemas por computadora. Así mismo pone énfasis en la tarea de diseño de un algoritmo. Se presentan conceptos teóricos y ejemplos.

2. PROBLEMAS

Definimos **problema** como una discrepancia entre un estado actual y un estado deseado. Resolver un problema consiste entonces en encontrar una secuencia de pasos que nos permitan pasar de un estado a otro.

La resolución de un problema no afecta a un problema particular sino a una **clase de problemas**. Cuando hablamos de resolución de problemas estamos refiriéndonos en realidad a un conjunto de problemas.

Para hallar soluciones a problemas se requiere habilidad, conocimiento y experiencia.

Un problema puede ser simple o complejo según quien tenga que hallar su solución. Las personas tienen distintas capacidades para hallar soluciones para diferentes **clases de problemas**.

Aunque a veces es posible hallar la solución de un problema aparentemente difícil de una forma puramente intuitiva ya que podemos tener raptos de inspiración, en general resulta conveniente entrenarnos en el uso de algunas estrategias y tácticas que nos orienten en la búsqueda de soluciones de problemas.

Estas tácticas y estrategias no necesariamente nos conducirán a la solución, pero es probable que nos ayuden en menor o mayor medida.

La búsqueda de la solución de un problema es una tarea difícil de sistematizar. Cada problema puede presentarse en forma aparentemente aislada y frecuentemente no sabemos cómo encararlo.

Sin embargo existen algunas pautas útiles que, de ser seguidas, pueden ayudarnos para enfrentar el problema:

—Establecer el problema en forma clara y entenderlo completamente.
—Clarificar cualquier redundancia que presente el problema.
—Definir exactamente qué se quiere hacer.
—Especificar con precisión todas las restricciones o condiciones que debe satisfacer la solución.
—Identificar claramente la información disponible.
—Explicitar toda información implícita en el planteo que pueda resultar útil.
—Encontrar una representación adecuada para esta información.
—Retomar el enunciado original ante un callejón sin salida.
—Verificar la solución encontrada.

Estas pautas no se deben seguir necesariamente en el orden antes expuesto; muchas tareas se repiten y a menudo se relee el enunciado con mayor atención, focalizando la atención en algún aspecto que tal vez resultó

inadvertido en un primer momento.

Es fundamental encontrar una buena representación para el problema. Esta representación constituirá un **modelo** del problema que deseamos resolver.

Hallar una solución para un problema específico puede brindarnos pautas para resolver la clase de problemas a la cual pertenece el problema particular.

Si podemos resolver una clase de problemas, estamos en condiciones de hallar una solución para una instancia específica, de acuerdo a esta resolución.

3. ABSTRACCIONES Y MODELOS

El corazón de la tarea de resolver problemas reside en ser capaces de manejar la complejidad inherente a ellos. La economía nacional, el movimiento de las moléculas en un objeto físico simple, son problemas muy complejos y por lo tanto difíciles de resolver. Para disminuir la complejidad de los problemas se utiliza una técnica muy poderosa: la **abstracción**.

El proceso de abstracción permite la construcción de **modelos**. Un modelo describe las propiedades fundamentales de un sistema, descartando los detalles irrelevantes.

Las leyes del movimiento de Newton son un modelo de la realidad física. Este modelo abstrae la enorme complejidad del movimiento de las partículas de los objetos y describe las propiedades globales del comportamiento integrado.

Para muchos propósitos los movimientos detallados de las moléculas en un objeto son irrelevantes, ya que consideramos los movimientos de las moléculas en su conjunto. Por ejemplo, nosotros hablamos de la velocidad de un objeto (que es la velocidad promedio de todas sus moléculas), de su temperatura (que es una medida de la energía cinética del movimiento molecular). En estos casos estamos **abstrayendo** de los detalles irrelevantes para el problema que tenemos entre manos.

Para algunos propósitos estas abstracciones pueden resultar inadecuadas. Cuando intentamos entender las reacciones químicas, por ejemplo, debemos considerar un modelo mas detallado que describa la estructura atómica de los materiales.

Cada modelo contiene el detalle necesario para explicar el fenómeno bajo estudio. Para analizar el movimiento o la temperatura de un objeto podemos ignorar completamente su estructura molecular.

Notemos que en cada caso hemos definido un modelo adecuado para representar el fenómeno a estudiar.**Estos modelos son distintos aún cuando la realidad es una sola y brindan una visión destacada del aspecto que se desea resaltar**.

Por ejemplo, consideremos la construcción de un edificio de departamentos. Existirán varios modelos de acuerdo a diferentes puntos de vista, aunque está claro que el edificio es uno solo. Para el ingeniero civil, una carta de la estructura general será suficiente representación, mientras que el arquitecto requerirá una serie de planos de los diferentes tipos de planta. Por otro lado, para la inmobiliaria encargada de la venta, probablemente será necesaria una maqueta con las distintas plantas y algunos otros detalles que permitan al comprador tener una visión global y dinámica del conjunto edilicio.

En la ciencia, la complejidad de la realidad puede ser entendida comprendiendo una cantidad de modelos -abstracciones- que la describen desde distintos puntos de vista.

Es necesario no ahondar demasiado en el proceso de abstracción a riesgo de caer en un **modelo mentiroso**, es decir, en nuestro afán por comprender fácilmente la realidad, tratamos de simplificarla a tal extremo que desechamos propiedades fundamentales. Por ejemplo, cuando se estudia la estructura de un átomo, se cae siempre en un gráfico que sitúa al núcleo en el centro rodeado por electrones que giran en órbitas perfectas. Hoy sabemos que este modelo es completamente irreal, pues los electrones intercambian constantemente energía saltando de órbita en órbita.

Los modelos cambian para darnos mejores aproximaciones de cómo se comporta el mundo real. Nuestros modelos aproximan una realidad física pero los modelos en sí nunca son **verdad**, siempre están sujetos al cambio al incorporar nuevo conocimiento.

El nivel de detalle requerido en un modelo depende de lo que pensemos hacer con él. Un modelo adecuado para tener un primer contacto con una cierta área de conocimiento, puede ser totalmente inadecuado o insuficiente si vamos a profundizar el estudio de problemas dentro de esa área. Es más, cierto modelo de la realidad puede ser adecuado para resolver algunos problemas y no para otros.

Es importante preservar siempre la calidad del modelo, es decir, respetar la realidad que se trata de representar. Así podemos hacer una serie de suposiciones iniciales y limitar nuestro modelo. Luego se podrá trabajar sobre el modelo y ver si la realidad se adapta al comportamiento previsto. Si así fuera, hemos hallado un buen **modelo** que podrá servir de base para realizar estudios más profundos.

4. ALGORITMOS

Queremos encontrar métodos generales capaces de resolver una **clase de problemas**. Un algoritmo es un **modelo** de la resolución de un problema, o mejor aún, de una clase de problemas. [1]

Informalmente un algoritmo es un método para resolver un problema, o un conjunto de directivas o una fórmula que indicará exactamente cómo se obtendrán los resultados deseados. De acuerdo a esta definición, todos trabajamos continuamente con algoritmos, sólo que no somos tan formales como para llamarlos así.

El conjunto de instrucciones dadas para armar un triciclo a partir de sus componentes, el procedimiento para registrarse en un colegio, una receta para preparar una torta, son ejemplos válidos de algoritmos. Describamos un algoritmo que llamaremos **shampoo**:

1. Mojar el cabello.
2. Aplicar shampoo.
3. Masajear.
4. Enjuagar.
5. Repetir.

Existen dos puntos de vista desde los cuales debe ser considerado un algoritmo:

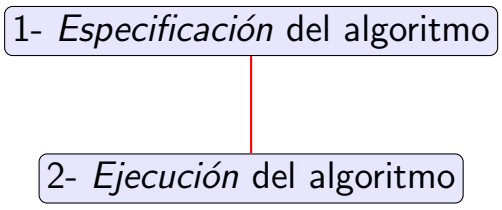

Fig. 1. Puntos de vista de un algoritmo

—Por un lado esta la **especificación** del algoritmo. La especificación comprende precisamente al conjunto de acciones que permiten resolver una clase de problemas. Desde este punto de vista un algoritmo se puede pensar como una entidad **estática**.

—Por otro lado debemos considerar la **ejecución** del algoritmo. Cuando alguien lleva a cabo las directivas indicadas por el algoritmo, hablamos de ejecución. Durante la ejecución estamos hallando la solución de un problema específico o de una instancia de la clase de problemas. La ejecución es un proceso **dinámico**.

[1] La palabra algoritmo deriva del nombre del matemático Mohammed ibnMusa alKhwarizmi (780-850), que era parte de la corte de Bagdad, Irak. Posiblemente su nombre también sea el origen de la palabra álgebra.

En adelante, cuando hablemos de algoritmos, nos estaremos refiriendo a la especificación y cuando nos referamos a la dinámica de un algoritmo, diremos ejecución de algoritmos.

Formalmente, definiremos entonces el concepto de **algoritmo** como:

Definición: `Algoritmo`

Un algoritmo es una secuencia de operaciones precisas (i.e. no ambigüas) para resolver un problema en un número finito de pasos.

[2]

Por **secuencia de operaciones**, entendemos que para cada paso del algoritmo, el siguiente está definido sin ambigüedad. Esto es, luego de completar un paso debemos saber siempre cuál será ejecutado a continuación. El orden de ejecución puede indicarse en la especificación, numerando los pasos con enteros positivos. Alternativamente el orden puede asumirse por la posición de las operaciones. Sin embargo, en muchas ocasiones podemos desear interrumpir esta secuencia.

En nuestro ejemplo el paso 5 presenta ambigüedades, al no indicar qué pasos deben ser repetidos. Sin embargo, para cualquier persona es obvio que no debe volver a mojar el cabello y no tendría sentido enjuagar nuevamente si no colocamos shampoo otra vez.

Inherente al concepto de secuencia de operaciones, subyace la idea de **principio** y **fin**. Todo algoritmo debe tener un punto inicial y por lo menos uno final.

Las **propiedades** de un algoritmo incluyen:

—**Entrada / Salida**: Recibe Entrada y genera Salida.
—**Precisión**: Cada paso esta precisamente explicitado.
—**Determinismo**: Los resultados intermedios de cada paso de ejecución son únicos y determinados solo por las entradas y resultados del paso anterior.
—**Correctitud**: Un algoritmo es correcto, si cumple con los tres puntos siguientes:
  —El algoritmo resuelve el problema para el cual fue diseñado: **efectividad**.
  —Para cada **entrada**, el algoritmo produce la **salida** deseada.
  —El algoritmo termina en un tiempo finito: **finitud**.

[3]

—**Generalización**: Un algoritmo es aplicable a un conjunto de entradas.

Las operaciones que componen un algoritmo son llamadas **primitivas**, y deben ser entendibles para la persona o máquina que las ejecutará. Por lo tanto el conjunto de primitivas posibles para escribir un algoritmo depende de quién o qué lo ejecutará.

Ejemplo 4.1 *Por ejemplo, la siguiente receta es un método posible para preparar una tarta de manzanas:*

1. *Prepare la masa.*
2. *Prepare el dulce de manzanas.*
3. *Rellene la masa con el dulce.*
4. *Coloque la tarta en horno moderado por 30 minutos.*

Para una persona familiarizada con la cocina esta receta puede ser aceptable. Sin embargo los pasos 1 y 2 presentarán ambigüedades para alguien sin experiencia en la cocina y deberán ser desarrollados con más detalle.

Para que un algoritmo pueda ser interpretado y ejecutado debe estar expresado entonces en función de acciones comprensibles. La clave está pues en definir el conjunto de acciones en términos de las cuales va a poder ser expresado el algoritmo.

[2] 'A finite set of rules which gives a sequence of operations for solving a specific type of problem' Donald Knuth The art of computing programming Fundamental algorithms Vol 1 Addison Wesley 1968
[3] En otras materias se estudiará a los algoritmos como 'Conjunto de pasos computable para lograr un resultado' y se los evaluará desde el punto de vista de su **tiempo de ejecución, eficiencia y complejidad**

Uno de los puntos fundamentales para decidir si un algoritmo es válido, es pues el hecho de que las acciones que lo componen sean **primitivas**.

Ahora bien, ¿cuándo un algoritmo puede ser interpretado y ejecutado por una computadora?

5. RESOLUCIÓN DE PROBLEMAS POR COMPUTADORA

Hemos dicho que estamos en presencia de un problema cuando existe una discrepancia entre la situación actual y la situación deseada.

La resolución de un problema consiste en hallar un algoritmo o un método que nos permita pasar de la situación actual a la deseada. Este método permite en realidad resolver una clase de problemas, mas que un problema particular.

Existen clases de problemas, para los cuales es muy difícil hallar un método de resolución. Tomemos por ejemplo, un detective cuyo problema es hallar el asesino de un crimen. El detective por supuesto se basará en sus experiencias anteriores y seguramente considerará algunos casos similares ya resueltos. Es más, es posible que los primeros pasos sean siempre los mismos: averiguar cómo, cuándo y dónde se produjo el crimen, quiénes fueron los últimos que vieron a la víctima con vida... De acuerdo a las respuestas el caso tomará rumbos diferentes. Resulta muy difícil encontrar un algoritmo para resolver crímenes. En el mejor de los casos el detective podrá hallar la solución para cada caso particular.

Existen otras clases de problemas, sin embargo, para los que una vez hallada una solución para una **instancia** especifica, es fácil inferir un método de resolución para toda la clase de problemas.

Supongamos que nos enfrentamos al problema de buscar en una guía telefónica el número de teléfono de Pérez Juan. Para realizar esta tarea tenemos que partir de dos supuestos: sabemos leer y contamos con una guía.

A partir de estos dos hechos podemos hallar una solución al problema de una forma muy rudimentaria y tediosa:

Podemos comparar el primer nombre con Pérez Juan, si coinciden (saber leer implica poder decidir si dos nombres son iguales) hemos hallado la solución. Si no, consideramos el segundo nombre y así siguiendo hasta encontrar a Pérez Juan o hasta que se agotó la guía (la persona no figura).

Este método es muy **ineficiente** porque partimos de premisas muy simples. Si disponemos de más información el problema se simplifica.

Si sabemos que la guía esta ordenada alfabéticamente y conocemos el alfabeto, nuestro algoritmo podría ser **descartar por mitades**:

Podemos abrir la guía por la mitad, aproximadamente, y comparar Pérez Juan con un nombre cualquiera. Pueden darse tres circunstancias:

1. El nombre es igual a Pérez Juan: ÉXITO
2. El nombre es menor a Pérez Juan: DESCARTAR LA PRIMERA MITAD DE LA GUÍA.
3. El nombre es mayor a Pérez Juan: DESCARTAR LA SEGUNDA MITAD DE LA GUÍA.

El proceso se repite nuevamente hasta hallar a Pérez Juan o agotar la guía.

Si tenemos conocimiento de la distribución de los apellidos en la guía, podríamos descartar una parte de la misma de distinta manera:

Podemos abrir la guía considerando la posición de P en el alfabeto y luego buscar secuencialmente por hojas y luego, una vez quehemos localizado la página, avanzar por nombre.

En ningún idioma existe la misma cantidad de apellidos para cada letra. En castellano, si buscamos un apellido que comienza con P, descartaremos bastante más que las primeras 17/27 partes de la guía, aún cuando

la posición de P en el alfabeto es 17, provisto que buena parte de los nombres comienzan con las primeras letras del alfabeto. Diferente situación se hubiera presentado en otro idioma.

Cada método ha sido una mejora respecto al anterior, pero requiere más Información, **supone que sabemos hacer más cosas**. En cada caso, además, hacemos uso de mayores conocimientos y de operaciones más complejas.

Cualquiera de los algoritmos sirve no sólo para intentar localizar a Pérez Juan en la guía, sino a cualquier persona. No sólo podemos hallar la solución de un problema, sino resolver toda una clase de problemas. Es más, el mismo método podría ser utilizado para hallar solución a problemas similares, como por ejemplo buscar una palabra en el diccionario.

De hecho, la confección de una guía presupone que la tarea de buscar un número telefónico se va a llevar a cabo frecuentemente. El esfuerzo empleado en encontrar un algoritmo se ve compensado por el hecho de que va a ser utilizado muchas veces.

Tomemos otro ejemplo. Cuando deseamos calcular el cociente q de un división entera, la forma más sencilla es hallarlo por restas sucesivas. El método es sencillo pero lento. Si esta tarea va a ser llevada a cabo muchas veces conviene encontrar un método más eficiente, aunque menos simple. En cualquier caso saber dividir, conocer el método, no implica **entender** el concepto.

Notemos que la acción de **dividir**, por definición, produce dos valores, el cociente q y el resto r. Aunque frecuentemente nos referimos indistintamente a hallar el cociente o a dividir, no son acciones equivalentes. La división entera de dos números a y b implica hallar otros dos números q y r tales que $a = q*b + r$.

Dos métodos para resolver un mismo problema normalmente difieren en el tipo de cosas que hay que saber hacer para llevarlos a cabo.

Para que una computadora pueda ser utilizada para hallar la **solución** de un problema, es necesario que sepa **resolver el problema**. Es decir, es necesario que se le brinde un método para resolver dicho problema. ¿Cómo explicitaremos dicho método?

La tarea de **programar** implica precisamente la obtención de un modelo computacional para la resolución del problema. Esto es, lograr un conjunto de instrucciones comprensibles para la computadora que permitan alcanzar la solución deseada para un conjunto de problemas.

Podemos pensar que:

—No tiene sentido escribir programas puntuales que permitan hallar soluciones para problemas específicos como por ejemplo, buscar el número de teléfono de Pérez Juan o resolver la ecuación $x^2 + 3x - 1 = 0$.
—Estaríamos buscando un método para resolver una clase de problemas con una única instancia. Recordemos que, de acuerdo a lo dicho en la sección anterior, un algoritmo es un ente estático que describe una serie de pasos a realizar para resolver una clase de problemas. Cuando el algoritmo se ejecuta se resuelve una instancia particular de dicha clase.
—Para que un algoritmo, destinado a resolver un problema, pueda ser transformado en un programa, es necesario que seamos capaces de definirlo en términos de operaciones que la computadora **sepa** hacer. Por ejemplo podemos encontrar un método para **cambiar la cubierta de un auto**, pero difícilmente podremos describir este método en términos de acciones que una computadora común pueda ejecutar.

Existe otra diferencia fundamental entre los problemas cotidianos planteados en la sección anterior y aquellos que pueden ser resueltos de manera tal de poder ser interpretados y ejecutados por una computadora.

Vimos que un algoritmo es una secuencia de pasos llevada a cabo sobre ciertos datos y que luego de un número finito de pasos produce un resultado. En los algoritmos vistos para **lavar el cabello** o **preparar una tarta** era un poco difícil distinguir **datos** y **acciones**.

En estos ejemplos no podemos hablar de **datos**, sino únicamente de estado inicial y estado final. Para el primer ejemplo el estado inicial era **cabello sucio** y el estado final **cabello limpio**. Sin embargo, esta especificación es ambigüa y por lo tanto la **ejecución** del algoritmo provocará variaciones en el estado final,

dependientes del estado inicial y de la interpretación de las acciones.

Cuando pensamos en llevar un algoritmo a la computadora, estas ambigüedades no son admisibles. Los datos de entrada y de salida, así como las acciones, deben estar perfectamente definidos.

Los datos de entrada pueden considerarse como la descripción del estado inicial de un problema. Sin embargo en algunos problemas es difícil identificar el estado inicial con un conjunto de datos de entrada.

Los datos de salida pueden considerarse como la descripción del estado final de un problema. Sin embargo en algunos problemas es difícil identificar el estado final con un conjunto de datos de salida.

Un **dato** representa a un conjunto de valores. En la ejecución del algoritmo cada dato de entrada tomará un valor dentro del conjunto. Cada instancia diferirá de otra en función de los valores que tomen los datos. Los algoritmos que trataremos producirán los mismos resultados cuando se ejecuten para los mismos valores de los datos de entrada. La ejecución de una misma receta no siempre provocará como resultado la misma tarta, porque hay una serie de factores que determinan el estado final y son difíciles de especificar.

Aunque los puntos anteriores son muy importantes, la clave fundamental reside en ser capaces de **hallar el método de resolución del problema**.

Resolver un problema, o mejor dicho una clase de problemas, consiste precisamente en encontrar un método que permita hallar la solución de cualquier problema de la clase.

En ocasiones el mismo planteo del problema incluye la descripción del método para resolverlo. En otros casos, el enunciado no indica cómo resolver el problema, pero existen métodos para resolverlo. Sin embargo con frecuencia, el problema no especifica el método y el diseño del algoritmo es entonces nuestra responsabilidad.

Definición: Programa

Cuando el algoritmo puede ser transformado en una secuencia de instrucciones ejecutables por una computadora, se transforma en un **programa**.
Un programa estará escrito en un **lenguaje de programación**, es decir, en una notación formal y estricta, que podrá ser interpretada por la computadora.

Si intentamos obtener un algoritmo como una secuencia de acciones primitivas, y nos interesa transformar el algoritmo en un programa, el conjunto de acciones primitivas, para el programa queda definido en función del lenguaje de programación.

Un problema se puede resolver por métodos distintos, con distintos grados de eficiencia, como mostrábamos en el ejemplo de la búsqueda de un número en una guía telefónica.

Cuando se usa una computadora es deseable minimizar el consumo de recursos. Nos referimos principalmente al costo en tiempo y espacio. Frecuentemente, en ciencias de la computación, la pregunta no es cómo resolver un problema sino cómo resolverlo de modo **eficiente**. Distintos algoritmos para resolver el mismo problema son evaluados en función de su eficiencia relativa y se busca el más rápido, el que consume menos espacio o soluciones de compromiso entre ambos costos.

El desarrollo de un programa requiere atravesar una serie de etapas que resumiremos en: Análisis, Diseño, implementación y testing. El **Análisis** es el proceso de definir claramente el problema que se desea resolver. El **Diseño** es el proceso de utilizar la información recolectada en la etapa de análisis para desarrollar un modelo para la solución del problema. la **Implementación** consiste en utilizar el modelo para crear el programa. **Testing** consiste en asegurar que el programa cumple con el modelo diseñado, resuelve el problema.
Debemos distinguir siempre la etapa de DISEÑO de la etapa de IMPLEMENTACIÓN. Si entendemos los conceptos de programación y como estos se emplean en la etapa de ANÁLISIS y DISE NO, luego los podremos aplicar a cualquier lenguaje de programación (a cualquier IMPLEMENTACIÓN).
En nuestra materia utilizaremos pseudocódigo en la etapa de diseño de un algoritmo, y una vez que el algoritmo esté especificado en pseudocódigo, lo implementaremos utilizando un lenguaje de programación (en nuestro caso PHP). Podemos utilizar cualquier lenguaje de programación a partir de un algoritmo escrito

en pseudocódigo ya que la herramienta de diseño es independiente de cualquier lenguaje de programación. Para realizar la verificación del programa utilizaremos Trazas. En este apunte explicaremos como construir algoritmos con pseudocódigo, como realizar trazas, y veremos algunos conceptos asociados del lenguaje de programación que emplearemos.

6. DISEÑO DE ALGORITMOS

Los componentes mas significativos de un algoritmo son:

Variable:

> Nombre o etiqueta usado para identificar una ubicación en donde puede almacenarse un dato. También se puede definir como un nombre o etiqueta que representa un contenedor de datos cuyo valor puede variar durante la ejecución del programa.

Tipo de Dato:

> Determina los valores que puede tomar una variable y las operaciones que pueden realizarse con ella. Ejemplos de tipo de datos son: entero, real, lógico, carácter.

El lenguaje de programación PHP provee diversos tipos de datos primitivos, por ejemplo el tipo de dato int para enteros, float para reales, str para caracteres y bool para valores booleanos ó lógicos. Los tipos de datos de PHP se analizarán en detalle en próximos apuntes [4].

Adicionalmente un Dato es un Objeto sobre el cual opera un algoritmo. Para resolver un problema mediante computadora, es necesario identificar datos e incógnitas para luego diseñar el algoritmo, que implementado en un lenguaje de programación resolverá el problema.

Como se explicó en la definición, una variable es un nombre que representa un contenedor de datos cuyo valor puede variar durante la ejecución del programa. En cambio una constante es un valor que se asigna una vez y se mantiene fijo para todo el algoritmo (o en el programa). Ambos son un espacio reservado en memoria para almacenar un valor. Ambos tienen un tipo de dato y un identificador.

Constante:

> Representa un valor que es asignado una vez, y se mantiene fijo para todo el algoritmo o programa.[a]

[a] En PHP se utiliza la palabra reservada **CONST** al inicio del programa.

En un algoritmo, los **datos de entrada** son los que se van a procesar. Los **datos de salida** son datos derivados, es decir, obtenidos a partir de los datos de entrada.

A partir del enunciado de un problema identificamos los datos de entrada y las incógnitas.

Ejemplo 6.1 *El perímetro de un jardín rectangular es de 58 m. Si el lado mayor mide 11 m más que el lado menor. Cuánto miden los lados del jardín?*

Datos de entrada	P = Perímetro
Incógnitas	I = lado menor , L = lado mayor
Relaciones entre los datos y las incógnitas	P = 2L + 2I , L = I +11

Table I. Datos

Recurriendo a la abstracción matemática de que nos encontramos con un sistema de dos ecuaciones con dos incógnitas, podemos resolver que:

$$58 = 2L + 2I$$
$$L = I + 11$$

[4] Los tipos de datos que asumen las variables en PHP se definen de forma dinámica, lo que significa que no se tiene que especificar cuál es su tipo de antemano y puede tomar distintos valores en otro momento. Se usa el símbolo = para asignar valores.

Si reemplazamos:
$58 = 2(I + 11) + 2I$
$I = \frac{58-22}{4}$

Finalmente los resultados obtenidos: $I = 9$ y $L = 20$

Para poder resolver el problema mediante un algoritmo necesitaremos recurrir a instrucciones que luego serán implementadas en el lenguaje de programación. Una **instrucción** es una orden o acción clara y precisa.

6.1. Asignación

La **Asignación** es la instrucción que nos permite asociar un valor a una variable, es decir, nos permite almacenar un dato en una variable. Es la instrucción fundamental, dado que todo algoritmo puede verse como una composición de asignaciones. Su sintaxis es:

X ← E donde X es una variable y E puede ser:

—Un valor;

—Una expresión que luego de ser evaluada arroja un resultado;

—Otra variable a la cual le fue asignado un valor con anterioridad.

Por ejemplo, si X es una variable, entonces las siguientes asignaciones serían válidas:

X ← 5.

X ← (8+6*3 69).

X ← Y (donde Y es una variable con valor asignado, p.ej., Y ← 7).

Asignación

En la evaluación de una asignación primero se evalúa la expresión que está en el lado derecho de la asignación, el valor obtenido se asigna a la variable que está en el lado izquierdo

Debe quedar claro que **la asignación NO es una igualdad**. El planteo de una igualdad, por ejemplo letra = 'A' (donde letra es una variable de tipo carácter) representa una expresión lógica, que a la hora de ser evaluada, resultará verdadera, si el valor almacenado en la variable letra es el carácter 'A'; de lo contrario resultará falsa.

En cambio una asignación implica almacenar información en una variable, p. ej., si *entero* es una variable entera en la cual deseamos almacenar el valor 8, podremos hacerlo con la instrucción:

entero ← 8

Y si posteriormente necesitamos obtener el sucesor del valor que actualmente contiene la variable *entero*, deberemos plantear la asignación:

entero ← *entero* + 1

En primer lugar se evalúa la expresión *entero + 1* (con el valor actual de la variable entero, que es 8), y luego se produce la asignación. Esta instrucción logrará que el valor de entero luego de su ejecución sea 9.

Observar que *entero* = *entero* + 1 es una expresión lógica cuyo valor de verdad es falso, cualquiera sea el contenido de la variable entero, razón por la cual no tiene sentido utilizarla. En cambio, *entero* ← *entero* + 1 no sólo tiene sentido, sino que es imprescindible en algunas situaciones.

6.2. Instrucciones de Entrada/Salida

Las **Instrucciones de Entrada/Salida** permiten la comunicación con el mundo exterior; permiten el ingreso de datos a un programa y la visualización de resultados del mismo.

Entrada

Las instrucciones de **entrada**, al igual que la asignación, permiten asociar un valor a una variable, pero en el caso de la entrada, este valor es tomado desde el exterior.

La sintaxis correspondiente es: LEER (nv) donde:nv es el nombre de una variable a la que deseamos asignarle un valor.

Por ejemplo, si *edad* es una variable entera, podemos asignarle un valor de la siguiente forma:
LEER (*edad*)
De esta forma, el valor que será asociado a la variable *edad* podrá ser distinto cada vez que se ejecute esta instrucción,mientras que si usamos una asignación nuestro programa obliga a la variable *edad* a tener el mismo valor (el que aparece en dicha asignación) cada vez que se ejecute la instrucción.

Salida

Las instrucciones de **salida** permiten mostrar los resultados obtenidos por nuestro programa. De no existir este tipo de instrucciones, la solución del problema quedaría almacenada en alguna variable y no sería posible conocer su valor.

La sintaxis es: ESCRIBIR (nv) donde: nv es el nombre de la variable que almacena la información que deseamos conocer.

Por ejemplo,
ESCRIBIR(*edad*)

Ejemplo 6.2 *Diseñar un algoritmo para calcular el perímetro de un rectángulo dados los lados.*

Para el algoritmo cálculo_perímetro_de_rectángulo tendremos:

Datos de entrada	ladoMenor - ladoMayor
Datos de Salida	perimetro
Relaciones entre los datos	perimetro = 2 * ladoMenor + 2 * ladoMayor

Table II. Datos

Nuestro algoritmo permitirá entonces que el usuario ingrese los lados ladoMenor y ladoMayor y le mostrará el perimetro del rectángulo. Por lo tanto se leerán los datos como parte del algoritmo. Podemos resumirlo en:

```
calcularPerimetro()
        | Ingresar Datos de Entrada
        | Realizar Calculos
        | Mostrar Datos de Salida
FIN calcularPerimetro()
```

Más específicamente, el ALGORITMO escrito en pseudocódigo sería:

```
ALGORITMO calcularPerimetro
    | (* lee los lados de un rectangulo, valores mayores a 0, calc.Perimetro *)
    | FLOAT ladoMenor, ladoMayor, perimetro
    | ESCRIBIR("Ingrese el valor del lado menor: ")
    | LEER(ladoMenor)
    | ESCRIBIR("Ingrese el valor del lado mayor: ")
    | LEER(ladoMayor)
    | perimetro ← 2*ladoMenor + 2*ladoMayor
    | ESCRIBIR("El perimetro del rectangulo de lados ",ladoMenor, " y ",ladoMayor, " es ", perimetro)
FIN ALGORITMO calcularPerimetro
```

Comentarios en Pseudocódigo

Utilizaremos los símbolos (*) para realizar comentarios en pseudocódigo. Los comentarios son útiles para los programadores, hace que el código sea legible para otros programadores*

Traza

La **traza** *de un algoritmo se puede definir como la ejecución manual de forma secuencial de los pasos que lo componen. Así, la traza del algoritmo es el valor que van adoptando las* **variables** *a medida que se va ejecutando. En una tabla representamos los sucesivos estados de todas las variables*

Para nuestro ejemplo:

ladoMenor	*ladoMayor*	*perimetro*

Si el usuario ingresa los datos 2 y 3 para ladoMenor y ladoMayor respectivamente, tendremos:

ladoMenor	*ladoMayor*	*perimetro*
2	3	

*Luego de la asignación perimetro ← 2*ladoMenor+ 2*ladoMayor el valor de P se habrá modificado:*

ladoMenor	*ladoMayor*	*perimetro*
2	3	10

6.3. Comentarios

Para realizar comentarios en pseudocódigo, utilizaremos los símbolos (* y *) para comenzar y cerrar el comentario respectivamente.

Cuando escribimos algoritmos, en general, es útil realizar comentarios explicativos. Los comentarios no son instrucciones, simplemente sirven para que, cuando un programador lea el código, pueda comprender mejor lo que lee. Los comentarios son anotaciones, que son legibles para el programador, y tienen el propósito de hacer el código fuente más fácil de entender, mejorando su mantenimiento y/o reuso.

Todos los lenguajes de programación tienen símbolos para delimitar comentarios.

Veremos que en el lenguaje de programación PHP los delimitadores de comentarios son:
// para comentarios de una línea.
/* y */ para comentarios de múltiples líneas.

6.4. Implentación en PHP del algoritmo de Perímetro

En PHP la implementación sería:

```
1  <?php
2  /* Algoritmo perimetro rectangulo */
3  function main(){
4      /*variables: float $ladoMenor, float $ladoMayor, float $perimetro*/
5      echo "Calcula el perimetro de un rectangulo";
6
7      echo "Ingrese el valor del lado menor";
8      /*leer lo que ingresa el usuario*/
9      $ladoMenor = trim(fgets(STDIN));
10
11     echo "Ingrese el valor del lado mayor ";
12     /*leer lo que ingresa el usuario*/
13     $ladoMayor = trim(fgets(STDIN));
14
15     $perimetro = 2*$ladoMenor + 2*$ladoMayor
```

```
16      echo "El perimetro de lados ".$ladoMenor." y ".$ladoMayor." es ".$perimetro
17  }
18  main();
19  ?>
```