Programación

Unidad 3: Algoritmos

“Hay dos formas de diseñar software: una es hacer que sea tan simple que obviamente no haya deficiencias, y la otra es hacer que sea tan complicado que no haya deficiencias obvias. El primer método es mucho más difícil”

- C. A. R. Hoare

Algoritmos

Definición

Antes de adentrarnos en la definición de qué es un algoritmo, cabe mencionar que una persona que se desempeña como programador es antes una persona que resuelve problemas, por lo que para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo metódico y sistemático. Previamente hablamos sobre la metodología necesaria para resolver un problema mediante el uso de una computadora. El eje central de esta metodología es el concepto de *algoritmo.*

Ahora sí y sin más preámbulo, la definición de **algoritmo** es: un método para resolver un problema.

Si bien la anterior definición es la más simple que podemos encontrar, la definición de algoritmo que más se aproxima a nuestro propósito es la siguiente:

Definición	Secuencia finita de instrucciones, reglas o pasos que describen en forma precisa las operaciones que una computadora debe realizar para llevar a cabo una tarea en tiempo finito [Knuth, 1968].

Los algoritmos son independientes tanto del lenguaje de programación en el que se expresan como de la computadora que los ejecuta, así un algoritmo puede ser escrito en diferentes lenguajes y ser ejecutado por distintas máquinas y sin embargo el algoritmo será el mismo.

Tratando de hacer una analogía, una receta de cocina se puede escribir en diferentes idiomas (español, inglés, portugués), pero los pasos de la receta seguirán siendo los mismos.

Tener en cuenta que los algoritmos son mucho más importantes que los lenguajes de programación que se utilizan para expresarlos y que las computadoras en las que son ejecutados; es por esto que el proceso de diseño de un algoritmo es uno de los aspectos más importantes y nos va a demandar creatividad, conocimientos y técnicas para lograr una buena solución.

Características de un algoritmo

Las características fundamentales de todo algoritmo son:

- Debe ser *preciso*. Indicando el orden de realización de cada paso.

- Debe estar bien definido y no ser ambiguo. Si se sigue un algoritmo 2 veces debemos alcanzar el mismo resultado.
- Debe ser finito. Debe tener un número finito de pasos. Debe terminar en algún momento.

La definición de todo algoritmo debe describir 3 partes:

1. Los datos de entrada (volver a leer la Unidad 1)
2. El proceso: los pasos a ejecutar
3. La salida esperada

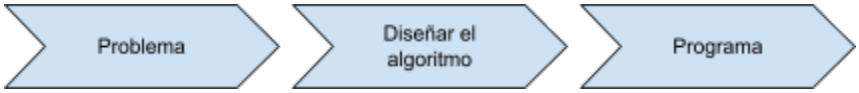
Ejemplo:
Problema: “Un cliente realiza un pedido a una fábrica. La fábrica examina los datos del cliente para saber si tiene deuda o no. Si el cliente no posee deuda, la empresa acepta el pedido, en caso contrario lo rechaza”

Algoritmo

```
1. Inicio
2. Leer el pedido
3. Analizar estado del cliente
4. Si el cliente no es deudor
       a. escribir acepta el pedido
5. Caso contrario
       a. escribir rechaza el pedido.
6. Fin.
```

Diseño de un algoritmo

La resolución de un problema demanda el diseño de un algoritmo que resuelva el problema propuesto.



Cualquier problema se puede resolver más eficazmente cuando se lo divide en problemas más pequeños que sean obviamente más fáciles de solucionar. A esta técnica se la conoce como “*divide y vencerás”*.

La descomposición de un problema en subproblemas más simples y a su vez la división de estos subproblemas en otros más simples que puedan ser implementados para su solución en la computadora se la denomina **diseño descendente**.

Normalmente, lo que suele suceder es que, nuestro primer planteo del algoritmo tendrá pocos pasos (que agrupen grandes tareas) y esté incompleto o sea ambiguo, por lo que debemos

tomar cada uno de esos pasos y volver a dividirlos, en pasos más pequeños y detallados, hasta alcanzar un grado de descomposición de las tareas que se asemeje a instrucciones que puedan ser volcadas a una computadora. A este último paso lo denominamos *refinamiento*.

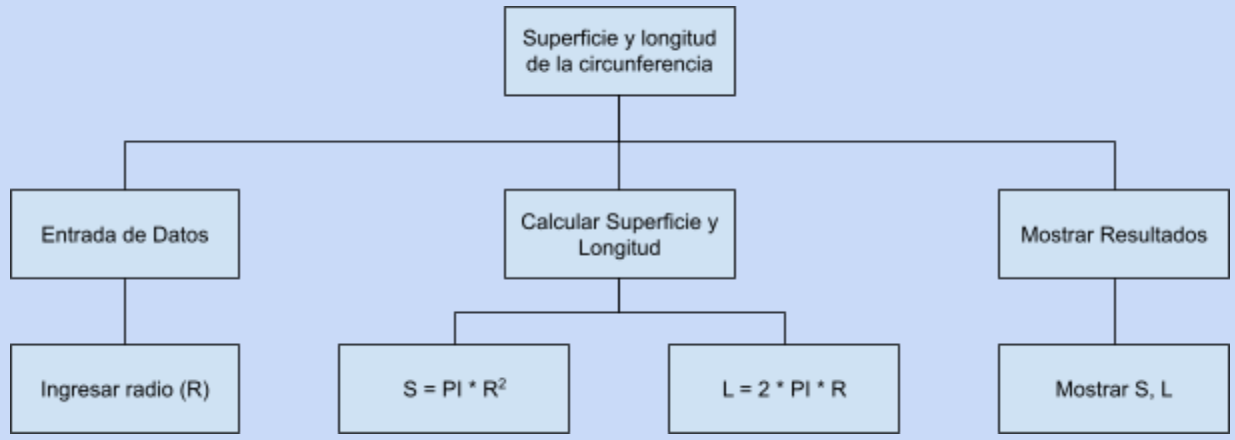
Entonces, el algoritmo para el diseño de un algoritmo podría ser algo como:

```
1. Inicio.
2. Descomponer el problema en problemas menores
3. Si tengo un grado de refinamiento que me permite volcarlo a la computadora entonces:
       a.   entonces => Tengo mi algoritmo!! :). Fin.
4. Sino
       a.  volver al punto 2.
5. Fin.
```

Vamos a verlo con un ejemplo sencillo:

Problema: “Dado el radio de una circunferencia, mostrar la superficie y el perímetro de la misma”

Diseño del Algoritmo



En este ejemplo simple, vemos como en un primer paso, dividimos el problema en 3 (grandes) tareas: “Entrada de Datos”, “Calcular Superficie y Longitud”, “Mostrar Resultados”. Pero estas tareas siguen siendo muy grandes y complejas como para poder implementarlas, por lo que nos conviene realizar una iteración más, un refinamiento de cada una de estas tareas para poder lograr un nivel de simpleza en cada una que nos permita volcarlo a la computadora. Es ahí donde surge el segundo nivel.

Representación de un algoritmo

Previamente mencionamos que lo más importante es lograr diseñar el algoritmo que nos resuelve el problema planteado, debido a que puede tener varias representaciones según el lenguaje elegido, la computadora donde se ejecute, etc. Bien en este punto vamos a describir 2 herramientas que nos permitirán representar un algoritmo de forma precisa sin preocuparnos, todavía, por la elección de un lenguaje de programación.

Estas herramientas son el pseudocódigo y el diagrama de flujo.

Pseudocódigo

El pseudocódigo, podríamos decir que es un lenguaje (pero no un lenguaje de programación) para especificar un algoritmo. Este lenguaje hace que el paso a un lenguaje de programación sea relativamente sencillo.

El pseudocódigo nació como un medio para representar las estructuras de control de la programación estructurada (cosas que vamos a ver en las próximas unidades) . Se lo considera un primer borrador ya que el pseudocódigo debe ser traducido a un lenguaje de programación específico. El pseudocódigo no puede ser ejecutado por una computadora.

La ventaja que nos proporciona el uso de esta herramienta, es poder concentrarnos en cómo resolver el problema y no preocuparnos por las reglas del lenguaje de programación. Otra de las ventajas es que al ser escrito en el idioma nativo de la persona que programa puede ser utilizado como una herramienta de comunicación para con otros programadores. Es fácil de modificar.

Todo algoritmo en pseudocódigo tiene la siguiente estructura general:

```
Algoritmo MiPrimerAlgoritmo
     acción 1;
     acción 2;
     ...
     acción n;
FinAlgoritmo
```

Comienza con la palabra clave *Algoritmo* (o alternativamente *Proceso*, son sinónimos) seguida del nombre del programa, luego le sigue una secuencia de instrucciones y finaliza con la palabra *FinAlgoritmo* (o *FinProceso*). Una secuencia de instrucciones es una lista de una o más instrucciones y/o estructuras de control.

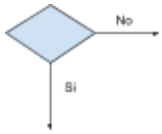
Ejemplo

```
Proceso Suma

        Definir A,B,C como Reales;

        // para cargar un dato, se le muestra un mensaje al usuario
        // con la instrucción Escribir, y luego se lee el dato en
        // una variable (A para el primero, B para el segundo) con
        // la instrucción Leer

        Escribir "Ingrese el primer numero:";
        Leer A;

        Escribir "Ingrese el segundo numero:";
        Leer B;

        // ahora se calcula la suma y se guarda el resultado en la
        // variable C mediante la asignación (=)

        C = A+B;

        // finalmente, se muestra el resultado, precedido de un
        // mensaje para avisar al usuario, todo en una sola
        // instrucción Escribir

        Escribir "El resultado es: ",C;

FinProceso
```

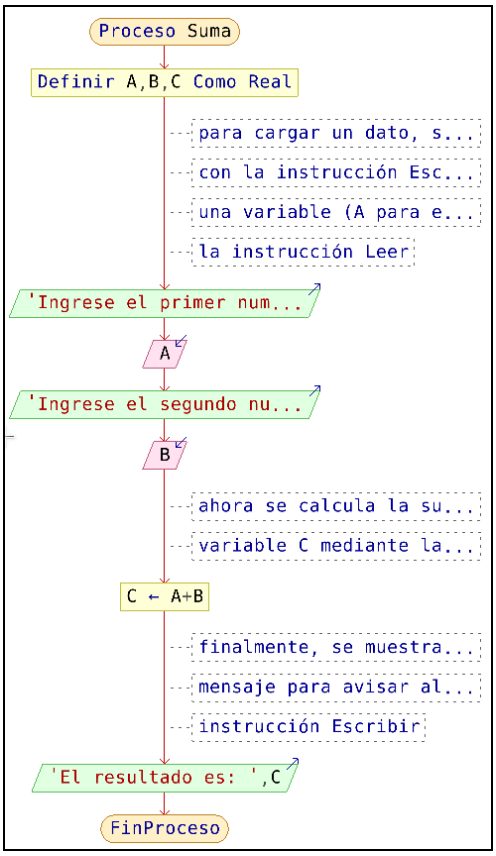
Diagrama de Flujo

Un diagrama de flujo es un diagrama que utiliza símbolos estandarizados que nos permiten representar los pasos de un algoritmo. Estos símbolos se encuentran unidos por líneas o flechas, denominadas líneas de flujo, que indican la secuencia en que se debe ejecutar.

Símbolo	Descripción
	Terminal. Permite representar el principio y el fin del algoritmo.
	Entrada / Salida. Utilizado para ingreso de datos y también para mostrar la salida de información.
	Proceso. Cualquier tipo de operación que pueda originar cambio de valor.

	Decisión. Evalúa una expresión lógica y en función de su resultado Verdadero o Falso, determina el camino a seguir.
	Decisión múltiple. De acuerdo al resultado de comparación se seguirá una opción u otra.
	Conector a otra página.

Ejemplo



Entradas y Salidas Estándar

Los programas interactúan con el exterior a través de los datos de entrada y salida. Los datos de entrada, en su mayoría, serán ingresados por un usuario a través del teclado, mouse, pantalla

táctil mientras que las salidas será información que el programa ofrecerá al usuario a través de la pantalla o en su defecto una impresora.

Obtención de entrada del usuario.

Para obtener los datos de entrada que nos proporcionará el usuario, debemos en primera instancia definir todas las variables necesarias para poder almacenar dichos datos. En segundo lugar debemos ejecutar una sentencia, una instrucción, que nos permita pedirle al usuario cada uno de los datos.

Veamos un ejemplo de esto en pseudocódigo:

Problema: “Pedirle al usuario que ingrese su nombre y mostrar un mensaje por pantalla saludándolo.”

```
1  Proceso Saludar
2     Definir nombreUsuario como Cadena;
3     Mostrar “Ingrese su nombre por favor: “;
4     Leer nombreUsuario;
5     Mostrar “Hola “, nombreUsuario;
6  FinProceso
```

Del ejemplo anterior podemos destacar algunas cuestiones importantes:

- En primer lugar en la línea #2 estamos definiendo la variable *nombreUsuario* para poder guardar los datos que el usuario vaya a ingresar
- En la línea #3 estamos utilizando la instrucción *Mostrar* que nos permite imprimir un mensaje en la pantalla.
- En la línea #4 estamos utilizando la instrucción *Leer* que nos permite obtener los datos ingresados por el usuario y guardarlos en la variable *nombreUsuario*.

Impresión de salida en pantalla.

Cuando desde el programa necesitamos mostrarle un mensaje al usuario o bien presentar los resultados de nuestro algoritmo, vamos a utilizar la pantalla de la computadora o como dijimos al comienzo una impresora.

Para poder mostrar algo por pantalla debemos ejecutar una sentencia que imprima por pantalla el mensaje y/o dato en cuestión.

Veamos un ejemplo de esto en pseudocódigo:

Problema: “Pedirle al usuario que ingrese 2 números enteros y mostrar su suma.”

```
1  Proceso Sumar
2     Definir numero1 como Entero;
3     Definir numero2 como Entero;
4     Definir suma como Entero;
5
6     Mostrar “Ingrese el primer número por favor: “;
7     Leer numero1;
8
9     Mostrar “Ingrese el segundo número por favor: “;
10    Leer numero2;
11
12    suma = numero1 + numero2;
13
14    Mostrar “La suma es: “, suma;
15 FinProceso
```

Del ejemplo podemos ver (además de la definición de variables necesaria para poder guardar los datos que ingresa el usuario) que la instrucción *Mostrar* se puede utilizar tanto para imprimir un mensaje por pantalla, como en las líneas #6 y #9, como para imprimir un mensaje y valores como en la línea #14.

Contadores y Acumuladores

¿Qué es un contador?

En primer lugar es una variable, de tipo Entero, que como su nombre lo indica su función es contar cosas.

¿Y un acumulador?

También es una variable, sólo que su tipo dependerá de qué valor necesito acumular. Por ejemplo si necesito acumular valores de tipo Real, mi acumulador deberá ser de tipo Real. Un acumulador irá sumando valores a los valores que tenía guardado previamente.

Veamos un ejemplo para clarificar:

Problema: “Obtener el promedio de notas de un alumno y mostrarlo por pantalla”

```
1  Proceso Promedio
2
3      Definir nota_1 como Real;
4      Definir nota_2 como Real;
5      Definir nota_3 como Real;
6
7      Definir sumaParcial como Real; // Defino mi acumulador
8
```

```
 9      Definir cantidadDeNotas como Entero; // Defino mi contador
10
11      Mostrar “Ingrese la primer nota: “;
12      Leer nota_1;
13      sumaParcial = sumaParcial + nota_1; // acumulo el valor de la nota 1
14      cantidadDeNotas = cantidadDeNotas + 1; // cuento una nota más
15
16
17      Mostrar “Ingrese la segunda nota: “;
18      Leer nota_2;
19      sumaParcial = sumaParcial + nota_2; // acumulo el valor de la nota 2
20      cantidadDeNotas = cantidadDeNotas + 1;  // cuento una nota más
21
22      Mostrar “Ingrese la tercer nota: “;
23      Leer nota_3;
24      sumaParcial = sumaParcial + nota_3; // acumulo el valor de la nota 3
25      cantidadDeNotas = cantidadDeNotas + 1;  // cuento una nota más
26
27      Mostrar “El promedio es: “, sumaParcial / cantidadDeNotas;
28  FinProceso
```

En este ejemplo nuestro contador es la variable *cantidadDeNotas* y nuestro acumulador es la variable *sumaParcial*.

Vemos, como mencionamos en su definición, que el contador es una variable de tipo Entero y además, como nos sucederá en la mayoría de los casos, incrementa su valor de a 1, cada vez que me ingresan una nota.

Por otra parte vemos que la variable *sumaParcial*, nuestro acumulador es de tipo Real ya que acumula valores de tipo Real como son las notas de los alumnos. También vemos que a diferencia del contador, el acumulador va sumando valores, por lo general ingresados por el usuario, a los valores que ya tenía guardados.

Bibliografía utilizada

Luis Joyanes Aguilar, Ignacio Zahonero Martínez. Programación en C. Segunda Edición. España: McGRAW-HILL/INTERAMERICANA DE ESPAÑA. S.A.U., 2005. ISBN-84: 481-9844-1.

● Brian W. Kernighan, Rob Pike. La práctica de la programación. Pearson Educación. Méjico (2000)

● Debugging with CodeBlocks: http://wiki.codeblocks.org/index.php?title=Debugging_with_Code::Blocks