

Introducción a la programación

Escribir programas de computación es un arte que tiene muchísimos años

y que cada día adquiere más adeptos.

En este libro trataremos de introducirnos

en los conceptos básicos para empezar a desarrollar y disfrutar

de este arte. ¿Y qué mejor para empezar

a programar que C y Pascal, dos lenguajes maduros y especialmente

pensados para programadores?

¿Qué es un lenguaje de programación?	16
Lenguaje de máquina	16
Lenguaje de bajo nivel	16
Lenguaje de alto nivel	17
Tipos de programación	18
Programación secuencial	18
Programación estructurada	18
Programación orientada a objetos	19
Programación lógica	19
Resolver problemas con una computadora	20
Análisis del problema	20
Construcción de un algoritmo	21
Codificación del algoritmo	24
Pruebas, ajustes y documentación	24

¿Qué es un lenguaje de programación?

Los lenguajes de programación son un conjunto de reglas, herramientas y condiciones que nos permiten crear programas o aplicaciones dentro de una computadora. Estos programas son los que nos permitirán ordenar distintas acciones a la computadora en un “idioma” comprensible por ella. Como su nombre lo indica, un lenguaje tiene su parte sintáctica y su parte semántica. ¿Qué quiere decir esto? Que todo lenguaje de programación posee reglas acerca de cómo se deben escribir las sentencias y de qué forma. A su vez, los lenguajes de programación se dividen en tres grandes grupos: los lenguajes de máquina, los de bajo nivel y los de alto nivel. A continuación explicaremos brevemente cada uno de ellos.

Lenguaje de máquina

Los lenguajes de máquina son los que entiende una computadora sin la necesidad de realizar ninguna conversión. Escribirlos resulta extremadamente difícil para un programador convencional. Hoy en día, nadie programa en este lenguaje, pero como es necesario para que la computadora entienda lo que tiene que hacer, existen programas que se encargan de transformar el código comprensible por un programador en código comprensible por una computadora.

Las instrucciones en este tipo de lenguaje se componen en la unidad de memoria más pequeña que existe dentro de una computadora, que se llama *bit*. Cada una de estas unidades puede tener sólo dos valores posibles: 1 o 0. Veamos algunos ejemplos de sentencias en este tipo de lenguaje:

```
0101  1001  0000  1101
0000  1111  1010  1011
```

Para facilitar la tarea del programador, se han diseñado otros tipos de lenguajes, que son más comprensibles que el de máquina. Éstos son el de bajo nivel y el de alto nivel.

Lenguaje de bajo nivel

Los lenguajes de bajo nivel, también llamados *ensambladores*, son aquellos cuyas sentencias están formadas por códigos nemotécnicos (abreviaturas de palabras inglesas).

Son lenguajes que, por más complejos que sean, resultan mucho más comprensibles que los lenguajes de máquina.

A continuación vemos un ejemplo de código para este tipo de lenguaje:

```
...
inicio:    mov cx,25
ini:       mov ah,6
           mov dl,13
           int 21h
           mov dl,10
           int 21h
           loop ini
           mov ah,9
           lea dx,nombre
           push cs
           pop ds
           int 21h
...
```

Estos lenguajes son, además, dependientes de la arquitectura de cada procesador, ya que cada procesador ofrece un conjunto de instrucciones distinto para trabajar en este nivel de programación. Una vez escrito el programa en este lenguaje, se necesita otro llamado *programa ensamblador*, para que traduzca las sentencias en instrucciones comprensibles por la máquina.

Lenguaje de alto nivel

Los lenguajes de alto nivel son aquellos que poseen sentencias formadas por palabras similares a las de los lenguajes humanos. Por lo tanto, resulta mucho más sencillo escribir un programa en un lenguaje de alto nivel para luego traducirlo en código comprensible para una computadora.

Algunos ejemplos de este tipo de lenguaje son: Pascal, Delphi, Cobol, FoxPro, JAVA y la mayoría de los lenguajes visuales, como Visual Basic, Visual FoxPro, etc. Además, podemos decir que, dentro de este conjunto de lenguajes de programación, algunos son de más alto nivel que otros, pero, en general, todos entran en esta categoría.

En el caso de C, muchos autores opinan que este lenguaje es de nivel medio, o sea, que posee una escritura sencilla y comprensible por los programadores, pero a la vez ofrece una potente y variada gama de posibilidades para realizar miles de tareas con

una computadora. Yo diría que se puede hacer casi lo mismo que con un lenguaje ensamblador. Fíjese en que hasta es posible insertar código assembler dentro del código de C. En conclusión, C pertenece a un nivel intermedio entre un lenguaje de bajo nivel y uno de alto nivel.

Tipos de programación

No sólo existen varios tipos de lenguajes de programación, sino que también podemos encontrar distintas formas de programar una aplicación. Hay diversos paradigmas que nos permiten encontrar una solución más adecuada a nuestros problemas. La idea es que el programador los conozca y sepa seleccionar el adecuado para cada situación particular. Lo que es verdad es que para cada paradigma conviene utilizar ciertos lenguajes de programación, y no cualquiera. Esto se debe a que la mayoría de los lenguajes fueron creados para ser utilizados en determinados ambientes de programación.

A continuación veremos las formas de programar más conocidas y utilizadas en la actualidad.

Programación secuencial

Este tipo de programación se basa en la creación de programas a partir de un conjunto de sentencias escritas de forma secuencial y cuya ejecución sigue dicha secuencia. Aquí se utiliza la sentencia como **goto** o similar para realizar una bifurcación en la ejecución del programa hacia una etiqueta determinada. Una etiqueta es una marca en el código de un programa para que sea referenciado en algún momento de su ejecución.

Algunos lenguajes que se utilizan para este tipo de programación son Basic, Assembler, Fortran y Cobol, entre otros.

Programación estructurada

La programación estructurada es una de las más conocidas y antiguas que existen. Sus fundamentos los expuso Edsger Dijkstra hace ya mucho tiempo. Hoy en día, más allá de que nuevos tipos de programación están copando el mercado, la programación estructurada perdura en el fondo de los lenguajes modernos.

Este tipo de programación se basa en la modularidad de los programas. Esto quiere decir que los programas se dividen en módulos más pequeños, y cada uno realiza una

tarea específica. Así, la complejidad de un problema determinado también se ve reducida al dividir las tareas, que, en su conjunto, permiten la resolución de éste.

Los programas para este tipo de programación poseen un procedimiento central, que es el encargado de llamar y controlar el correcto funcionamiento de cada módulo que compone la aplicación.

La programación estructurada hace uso de estructuras de control básicas: *secuencia*, *selección* y *repetición*. Además, no permite el uso de sentencias como **goto** o similares que produzcan bifurcaciones en la ejecución de las sentencias de un programa.

Los lenguajes que se utilizan en esta programación son C y Pascal, entre otros.

Programación orientada a objetos

La *programación orientada a objetos* (POO) es una forma de estructurar un programa sobre la base de objetos. Cada elemento o componente en un programa que se base en esta técnica es concebido como un objeto que tiene *propiedades* y *métodos*. La ejecución de un programa depende pura y exclusivamente de una interacción de los objetos que lo componen.

Las propiedades y los métodos de los objetos se especifican en su *clase*. Una clase de objeto vendría a ser el molde de cada instancia particular del objeto. Por lo tanto, cuando uno programa una aplicación orientada a objetos, define clases y luego crea instancias de objetos a través de esas clases para que interaccionen entre sí.

Existen otros conceptos importantes sobre objetos, que son la *herencia* y el *polimorfismo*. Éstos se explicarán con más detalle en el capítulo correspondiente a este tipo de programación.

Los lenguajes que se utilizan en este tipo de programación son muchos, pero los más importantes hoy en día son: JAVA, C++, Delphi, Smalltalk y la gran mayoría de los lenguajes de la nueva plataforma de Microsoft .NET, entre los cuales podemos encontrar Visual Basic.NET, C# y ASP.NET.

Programación lógica

La programación lógica es una forma de programar donde lo más importante es definir un conjunto de hechos, que se conocen con anterioridad, y un conjunto de reglas que nos definen las distintas relaciones que existen entre los componentes del programa. Estos hechos y reglas conforman lo que se llama la base del conocimiento. La programación lógica utiliza lenguajes de alto nivel que se asemejan mucho a los lenguajes humanos. Una de las utilidades más importantes de esta técnica de programación es la inteligencia artificial. Uno de los lenguajes que se utilizan para este tipo de

desarrollos es *Prolog*, un lenguaje de programación lógica antiguo pero potente. También se usa C, que provee más velocidad de procesamiento.

Por lo general, este tipo de programación hace uso de algoritmos recursivos. Estos algoritmos son explicados en los próximos capítulos.

Resolver problemas con una computadora

Hoy en día, las computadoras están presentes no sólo en ambientes específicos a los que sólo unos pocos tienen acceso, sino que también en muchos hogares podemos encontrar una. Esto hace que, con el paso de los días, más y más personas se interesen por desarrollar aplicaciones que corran en computadoras con el simple objetivo de solucionar problemas, tanto de negocios como cotidianos.

La tarea del programador es indicarle a la computadora un conjunto de instrucciones para que ella solucione su problema. El idioma que el programador utiliza para indicarle todo esto a una computadora es lo que ya conocimos como lenguaje de programación. Para resolver un problema mediante una computadora, se suelen seguir ciertos pasos, que son parte de una metodología. Aclaremos, antes de ver dichos pasos, que con la práctica algunos se pueden omitir, pero ante cualquier duda es recomendable efectuar los siguientes:

- Análisis del problema.
- Construcción de un algoritmo mediante un diagrama de flujo y pseudocódigo.
- Codificación del algoritmo.
- Pruebas, ajustes y documentación.

Debemos aclarar que los pasos de la metodología son más, pero, para centrarnos en los más importantes, sólo explicaremos los recién mencionados.

En este libro detallaremos cada uno de estos pasos, pero en los ejemplos que se plantean en capítulos posteriores sólo se desarrollarán diagramas de flujo y la codificación de los algoritmos.

Análisis del problema

La primera etapa de la metodología para resolver un problema mediante una computadora es el análisis. Esta etapa se basa en recolectar y analizar información que

nos permita identificar tres componentes básicos: los *datos de entrada*, los *datos de salida* deseables y un *proceso* que nos permita obtener dichos datos de salida (**Figura 1**).

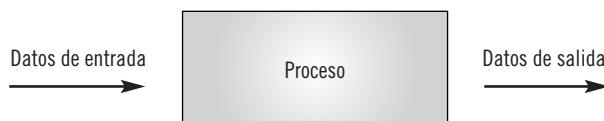


Figura 1. Componentes de la etapa de análisis de un problema.

Los *datos de entrada* son los datos que se ingresarán de alguna forma o mediante algún método en el programa. Éstos pueden ser desde una planilla de alumnos de una facultad hasta un archivo binario con información codificada que contiene los saldos de cuentas bancarias.

Los *datos de salida* son aquellos que resultan de aplicar el algoritmo, y constituyen el objetivo de todo este proceso. Éstos pueden no ser datos propiamente dichos, sino el resultado de aplicar un proceso específico a los datos de entrada. Algunos ejemplos pueden ser un listado de los clientes morosos de un club deportivo, el ordenamiento de alguna estructura de datos determinada, u operaciones matemáticas como por ejemplo la obtención del factorial de un determinado número.

Si analizamos un problema determinado e identificamos que es necesario listar los clientes que deban más de dos cuotas de su crédito, un proceso que se puede aplicar para resolver dicho problema es una simple búsqueda en que la condición de filtro sea “que deba más de dos cuotas”. Entonces, en esta etapa de la metodología se deben analizar los posibles procesos que nos permitan llegar a la solución del problema. De acuerdo con la complejidad del problema, existirá uno o más.

Construcción de un algoritmo

La siguiente etapa de la metodología es el diseño y la construcción del algoritmo que nos permitirá obtener el resultado deseado. Cuando se analizó el problema, se determinó *qué* se debía hacer para llegar a los objetivos buscados, y ahora, en la etapa del diseño del algoritmo, se debe determinar *cómo* se llevará a cabo esto.

Una recomendación a esta altura de la metodología es dividir el proceso en tareas más sencillas y más fáciles de implementar. Por ejemplo, si queremos imprimir las facturas de los clientes que pagaron el último mes, podríamos definir dos tareas específicas: la primera es generar las facturas en alguna estructura de datos acorde con la información requerida, y la segunda es recorrer las estructuras e imprimirlas. Así, el proceso central, que era obtener las facturas de los clientes, se dividió en dos tareas más sencillas e interdependientes entre sí: por un lado, generar las facturas, y por otro, imprimirlas.

Una vez determinadas las tareas o los módulos que componen el algoritmo, lo que debemos hacer es escribirlo utilizando tanto diagramas de flujo como pseudocódigo. Cualquiera de estas dos herramientas son válidas para crear algoritmos; se puede usar una u otra, o bien ambas. En la práctica, solían usarse más los diagramas de flujo, pero últimamente se está utilizando con más frecuencia el pseudocódigo. Con la experiencia, esta etapa se suele saltar, y se escribe el algoritmo directamente en algún lenguaje de programación. Pero lo recomendable en todo momento es no dejar estos diagramas (o el pseudocódigo) a un lado, porque constituyen una fuente muy importante en la documentación del sistema que estamos desarrollando. Pero ¿qué son un diagrama de flujo y un pseudocódigo? Bueno, en los siguientes párrafos explicaremos un poco mejor cada uno de ellos.

Diagramas de flujo

Los diagramas de flujo sirven para indicar cómo es el flujo de ejecución de las acciones que debe realizar el programa, más allá del lenguaje de programación que se utilice. Existen diversas formas de hacerlos, pero se han fijado algunas pautas generales para este tipo de diagramas. A continuación, en la **Tabla 1** veremos cada una de las figuras que se pueden utilizar en un diagrama de flujo, como también una breve descripción.

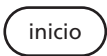
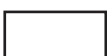
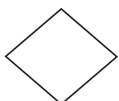
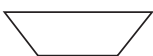
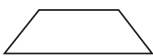

	Este símbolo se utiliza tanto para indicar el comienzo de un programa como su fin. Si en su interior dice INICIO, indica el comienzo del algoritmo. Si, al contrario, en su interior figura FIN, indica que es el fin del programa.
	Este símbolo sirve para representar un proceso. Este proceso puede contener una o más sentencias. Las asignaciones se representan con una flecha ("<-").
	Para representar las decisiones en un punto determinado, se puede utilizar este símbolo, así como para armar una estructura cíclica.
	Este símbolo se utiliza para leer datos tanto del teclado como de un archivo.
	Para imprimir por pantalla se utiliza este símbolo.
	Para unir todos los elementos de un diagrama indicando el flujo de ejecución.

Tabla 1. Figuras básicas para crear un diagrama de flujo.

Ahora veamos un pequeño ejemplo de un diagrama de flujo que incrementa un número de unidad en unidad hasta llegar a 100, y va mostrando dicho número en cada iteración. Este ejemplo se muestra en la **Figura 2**.

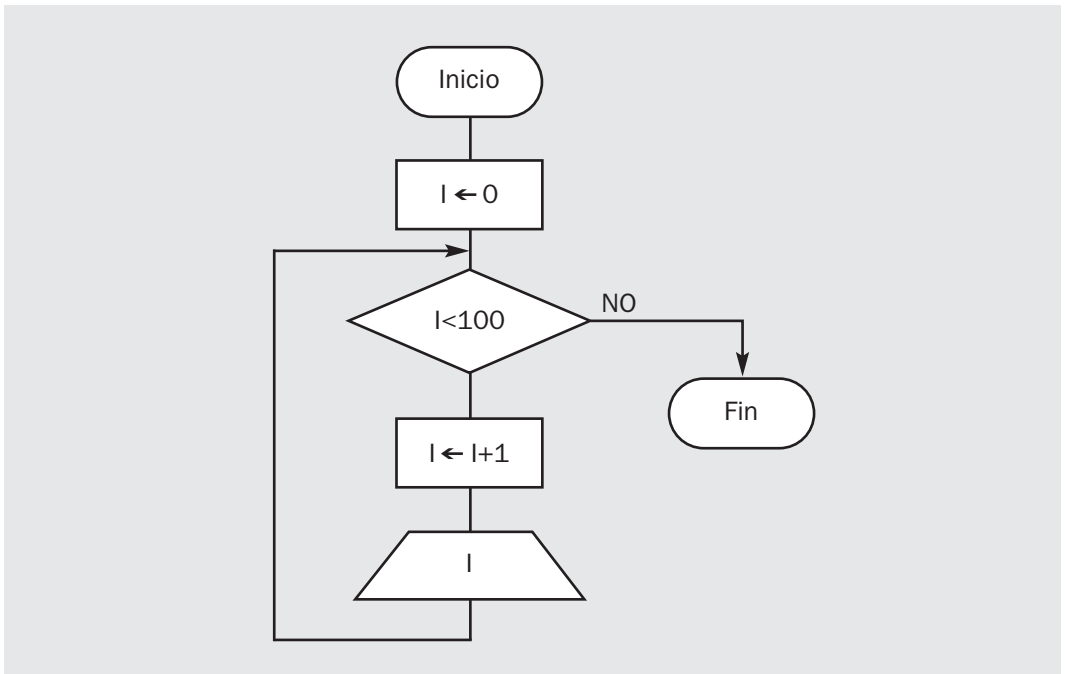


Figura 2. Simple ejemplo de un diagrama de flujo.

Los diagramas de flujo suelen estar acompañados de pseudocódigo, pero no siempre es así. Para empezar a programar es recomendable tenerlos en cuenta, porque son de gran utilidad, ya que nos fijan muchos conceptos.

Pseudocódigo

El pseudocódigo es otra forma de diagramar algoritmos o, mejor dicho, especificar las acciones que debe ejecutar un programa. La forma de hacerlo es mediante un lenguaje muy sencillo y similar al nuestro. La idea es ir escribiendo con palabras sencillas las acciones que debe seguir el programa para alcanzar los objetivos. Por ejemplo, escribamos el pseudocódigo del programa que se muestra en el diagrama de flujo de la **Figura 2**.

```

Inicio
Numero = 0
Si Numero es menor o igual que 100 entonces
    Mostrar por pantalla: Numero
    Incrementar Numero
Fin del si
Fin del programa
  
```

Como vemos en el ejemplo anterior, escribir un programa en pseudocódigo es muy sencillo. Esto se puede hacer antes o después de realizar el diagrama de flujo. Es más, uno se puede basar en el otro para realizarlo. Con la práctica notará que teniendo uno, el otro sale como si fuese una traducción.

Codificación del algoritmo

La etapa siguiente a la construcción del algoritmo es su codificación. En este punto de la metodología es donde podemos optar por infinidad de alternativas, dado que existen miles y miles de lenguajes de programación para crear programas. Algunos se adaptan más que otros a determinadas necesidades, todo depende de lo que deseemos hacer.

En este libro proponemos C y Pascal, dos lenguajes que tienen mucha historia y que hoy en día están presentes en la mayoría de los libros de enseñanza en el campo de la programación. Pero aclaramos que muchas de las técnicas que presentaremos en este libro son aplicables a otros lenguajes de programación.

En los próximos capítulos explicaremos los fundamentos de la programación que nos permitirán escribir algoritmos rápidos, eficientes, claros y concisos.

Pruebas, ajustes y documentación

Una vez escrito el algoritmo, no terminamos nuestra tarea como programadores. Falta probar que todo funcione bien y, en el caso de que esto no sea así, realizar los ajustes necesarios para el correcto funcionamiento de nuestro programa.

En esta etapa es donde se profundizan tareas como la depuración, una técnica para encontrar errores y seguir la ejecución de un programa paso a paso. Esto se explicará mejor en el próximo capítulo.

Por último, cuando tenemos nuestra aplicación funcionando correctamente, debemos documentar todo. Lo importante de esto es que, muchas veces, debemos retocar código fuente antiguo, o que hace mucho que no vemos, y no recordamos qué es lo que hacía dicho programa. Cuando son muchas líneas de código y las tareas del programa no son sencillas, nos costará mucho tener que leer de nuevo todo el código para entenderlo. Por lo tanto, es recomendable comentar el código: dejar escrito, en un archivo o en un soporte de papel, un diagrama de flujo o pseudocódigo del programa, de modo que podamos analizar la aplicación más rápidamente.

La documentación también incluye, si es necesario, los manuales de usuario. Estos documentos son instrucciones acerca de cómo se debe usar la aplicación en caso de que deba utilizarla una persona que no conoce de programación.