Programación y desarrollo de algoritmos con C++

Pablo Sznajdleder

Programación y desarrollo de algoritmos con C++

Pablo Sznajdleder

Marcombo

Alfaomega

Programación y desarrollo de algoritmos con C++

Pablo Augusto Sznajdleder

Curso de Algoritmos y Programación a Fondo
Primera edición: 2021
ISBN: 978-987-3832-73-4

Primera edición: MARCOMBO, S.L. 2021

www.marcombo.com

ISBN: 978-84-267-3320-7
D.L.: B 8668-2021

Impreso en Servicepoint
Printed in Spain

A mi esposa e hijo, a mis viejos y a la memoria de mis abuelos;
a mi amigazo Walter y a Graciela Sosiski.

Mensaje del editor

Los conocimientos son esenciales en el desempeño profesional, sin ellos es imposible lograr las habilidades para competir laboralmente. La universidad o las instituciones de formación para el trabajo ofrecen la oportunidad de adquirir conocimientos que serán aprovechados más adelante en beneficio propio y de la sociedad; el avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos. Cuando se toma la decisión de embarcarse en una vida profesional, se adquiere un compromiso de por vida: mantenerse al día en los conocimientos del área u oficio que se ha decidido desempeñar.

Alfaomega y Marcombo tienen por misión ofrecerles a estudiantes y profesionales conocimientos actualizados dentro de lineamientos pedagógicos que faciliten su utilización y permitan desarrollar las competencias requeridas por una profesión determinada. Alfaomega y Marcombo esperan ser sus compañeras profesionales en este viaje de por vida por el mundo del conocimiento.

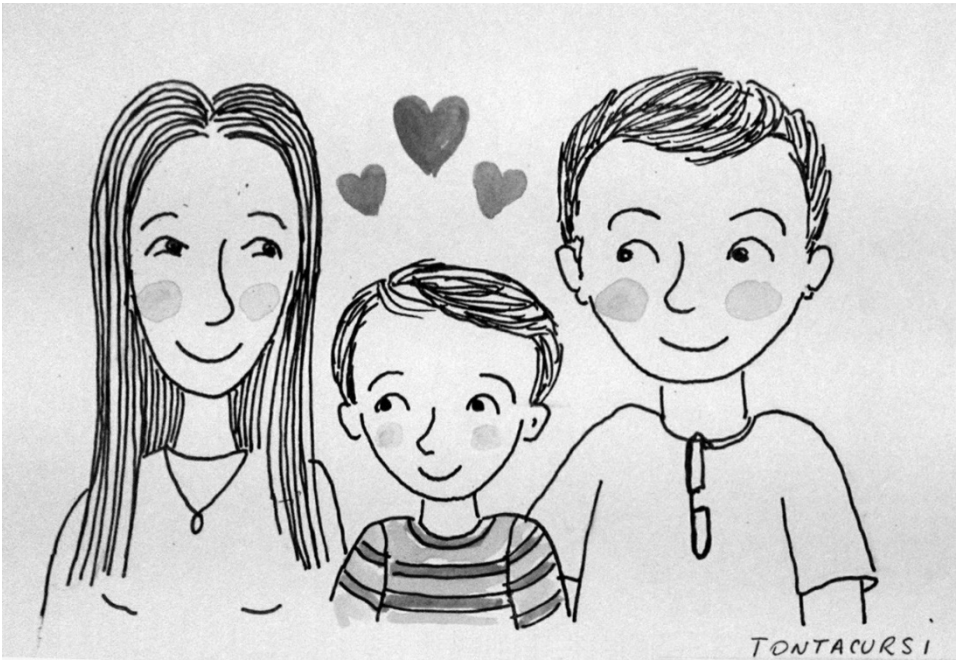
Alfaomega hace uso de los medios impresos tradicionales en combinación con las tecnologías de la información y las comunicaciones (TIC) para facilitar el aprendizaje.

Libros como este tienen su complemento en una página web, en donde el alumno y su profesor encontrarán materiales adicionales.

Esta obra contiene numerosos gráficos, cuadros y otros recursos para despertar el interés del estudiante, y facilitarle la comprensión y apropiación del conocimiento. Cada capítulo se desarrolla con argumentos presentados en forma sencilla y estructurada claramente hacia los objetivos y las metas propuestas.

Los libros de Alfaomega y Marcombo están diseñados para ser utilizados dentro de los procesos de enseñanza-aprendizaje, y pueden ser usados como textos para diversos cursos o como apoyo para reforzar el desarrollo profesional.

Alfaomega y Marcombo esperan contribuir así a la formación y el desarrollo de profesionales exitosos para beneficio de la sociedad.



La familia de Pablo retratada por Lucia Belén Berro @lavidadetontacursi

Acerca del autor

Pablo Augusto Sznajdleder es ingeniero en sistemas de información, graduado por la Universidad Tecnológica Nacional (UTN.BA, 1999).

Su tesis de maestría (UTN.BA, 2018) describe cómo implementar transferencias de conocimientos asistidas por tecnologías de mediación de interacción.

Profesor concursado en la cátedra de Algoritmos y estructura de datos, y director de cátedra en Patrones algorítmicos para estructuras avanzadas. Ambas materias en UTN.BA.

Autor de diversas obras, entre las que destacan: *Java a fondo* (Alfaomega), *Algoritmos a fondo* (Alfaomega) y *JEE a fondo* (Alfaomega).

Entre 1996 y 2001 trabajó como instructor Java para Sun Microsystems y Oracle Argentina, y obtuvo, en 1997, las certificaciones SCJP y SCJD. Estas fueron las primeras certificaciones Java acreditadas en Argentina, y estuvieron entre las primeras logradas en Latinoamérica.

Hoy, con 25 años de experiencia en tecnología Java, se desempeña en el ámbito profesional como consultor e instructor, proveyendo servicios de *coaching* y capacitación para las empresas líderes del país.

THE JAVA LISTENER

Learning music, enjoying Java

/thejavalistener

Antes de comenzar a leer

En este libro se utiliza la tipografía `Courier` en los casos en los que se hace referencia a código o acciones por realizar en la computadora, ya sea en un ejemplo o cuando se refiere a alguna función mencionada en el texto. También se usa para indicar menús de programas, teclas, URL, grupos de noticias o direcciones de correos electrónicos.

Los términos o definiciones cuyos significados están muy asociados al inglés se expresan en dicho idioma en *cursiva*.

El código fuente de los ejemplos, los tres apéndices (Ejercicios, Especificaciones y Problemas), así como todos los recursos didácticos y de programación que se utilizan en este libro, podrán descargarse a medida que se avanza en la lectura.

Estos recursos también están disponibles en www.marcombo.info con el código ALGORITMOS1.

Contenido

Prólogo

Los algoritmos, cada vez más presentes en nuestras vidas.

Esta frase, la cual escuchamos cada vez con mayor frecuencia, puede dar motivo a una búsqueda en Internet, y encontraríamos una cantidad extensa de resultados que permiten apreciar que es una realidad.

Los buscadores, las redes sociales, las aplicaciones de música y películas, entre otras tecnologías de la información y la comunicación (TIC), los emplean para alcanzar sus objetivos.

Del mismo modo que Nicholas Negroponte, a través de su superventas *Ser Digital* en la década de los 90, llamó la atención sobre los nuevos vientos en la vida humana, hoy podríamos citar el título de una obra como *Vivir en Algoritmos* para representar lo que se derrama sobre las personas como un omnipresente paradigma.

El profesor Pablo Augusto Sznajdleder, apreciado exalumno de la carrera de Ingeniería de Sistemas de Información de la querida UTN Regional Bs. As., viene trabajando desde hace varios años en buscar cómo facilitar y acompañar en el proceso de aprendizaje a los estudiantes de ingeniería.

En este libro Pablo aplica su experiencia en la docencia, implementando una modalidad a través de la cual los interesados pueden tener continuidad con los conocimientos volcados en la obra mediante recursos TIC externos.

Como profesor del profesor, puedo valorar su esfuerzo a la hora de simplificar didácticamente el entramado de conceptos, dando un encadenamiento incremental a las ideas para alcanzar este saber tan importante y con alto grado de abstracción.

No dudo que esta obra tendrá el éxito esperado y será de utilidad especialmente para estudiantes de licenciaturas o ingenierías del área informática y de sistemas.

Mi felicitación a Pablo, que enorgullece a quienes intervinimos humildemente en su formación profesional.

Ing. MSc. Alejandro Luis Echazú
Profesor UTN - BA

Agradecimientos

A Domingo Mandrafina, Damián Fernández y Alejandro Echazú.

CAPÍTULO 1
INTRODUCCIÓN AL DISEÑO DE ALGORITMOS Y PROGRAMACIÓN

1.1. Lección 1

Durante esta lección analizaremos conceptos básicos sobre algoritmos: qué es un algoritmo, para qué sirve y cuál es la relación que existe entre *algoritmo* y *programa*.

Estudiaremos los recursos iniciales de programación: variables, tipos de datos, operadores aritméticos, lógicos y relacionales, y las estructuras de control de flujo de datos que describe el *teorema de la programación estructurada*.

Veremos también cómo usar la herramienta de programación Eclipse para codificar, compilar, depurar y ejecutar nuestro primer programa.

1.1.1. ¿Qué es un algoritmo?

Es un conjunto finito y ordenado de pasos o acciones cuya ejecución nos permite resolver un problema.

Dicho de otro modo, dado un determinado problema, cuáles serían las acciones y en qué orden deberíamos ejecutarlas para lograr una resolución satisfactoria.

Lección 1

Ejecutamos algoritmos para resolver los problemas cotidianos que se nos presentan día a día, independientemente de cuál sea su nivel de complejidad.

Por ejemplo, si estamos parados en la acera y queremos cruzar la calle, el siguiente algoritmo resolverá nuestra situación problemática:

```
Miramos hacia la izquierda.
Si no viene ningún coche, entonces:
        Miramos hacia la derecha.
        Si no viene ningún coche, entonces:
                Cruzamos.
        Si no:
                Esperamos.
        Fin Si no
Si no:
        Esperamos.
Fin Si no.
```

Lo anterior es solo una mínima parte del algoritmo que ejecutamos cada vez que tenemos que cruzar una calle.

Podría suceder que, al mirar hacia la izquierda (o hacia la derecha), observemos que sí viene un coche, pero se encuentra a una distancia tal que igual podremos cruzar sin ser atropellados. O tal vez observemos que el coche no está tan lejos, pero se aproxima a baja velocidad, así que podremos cruzar de todos modos.

La observación sobre la distancia y velocidad del coche, así como la determinación del grado de peligrosidad que esto representa para nosotros (si es que decidimos cruzar la calle), implica resolver una serie de cálculos de relativa complejidad que realizamos mentalmente y sin darnos cuenta.

Por ejemplo, llamemos *p* a nuestra ubicación, *q* a la ubicación del coche, *v* a su velocidad, *z* a nuestra velocidad (cuán rápido caminamos) y *d* a la distancia que debemos recorrer para llegar a la acera de enfrente.

Entonces, el tiempo en que el coche llegará hacia nosotros lo calculamos como $|p-q|/v$, y debe ser mayor que el tiempo que nos llevará cruzar la calle, que podemos calcular como $|p-d|/z$. Incluso inconscientemente manejamos un nivel de riesgo, pues no nos gustaría sentir el viento del coche pasando a 1 milímetro de nuestro cuerpo. Así que, si llamamos *r* al riesgo que estamos dispuestos a asumir, solo cruzaremos la calle si se comprueba que $| |p-q|/v - |p-d|/z |>r$.

Esta decisión, que tomaremos en función de los cálculos precedentes, la podemos representar algorítmicamente con el siguiente fragmento de pseudocódigo:

```
Si | |p-q|/v - |p-d|/z | > r entonces:
        Cruzamos.
Fin Si.
```

Sin embargo, no solo los ingenieros podemos cruzar las calles de la ciudad. Todas las personas lo hacen a diario, porque el cerebro humano tiene la capacidad de realizar estas operaciones matemáticas en una milésima de segundo.

Pero hay algo más: *cruzamos*. ¿Qué significa esto? Todos comprendemos el significado de esta expresión, porque tenemos capacidad de abstracción. Pero, si nos detenemos a pensar qué implica *cruzar*, veremos que también requiere una serie de acciones. Se trata de un algoritmo en sí mismo.

Cruzar implica:

```
Bajar el paso de la acera.
Andar hasta llegar a la acera de enfrente.
Subir el paso de la otra acera.
```

Aún hay más: *andar* también es un algoritmo en sí mismo, que implica:

```
Levantar un pie.
Adelantarlo.
Bajarlo.
```

Razonando así, podemos entrar en detalle tanto como queramos. Sin embargo, esto no es necesario, porque nuestra capacidad de abstracción nos permite comprender qué representa un único término y cuáles acciones están involucradas.

El desafío consiste en diseñar algoritmos para que sean ejecutados por un ordenador, el cual no tiene nuestra capacidad de abstracción.

1.1.2. Lenguaje algorítmico

Para documentar un algoritmo, se utiliza un lenguaje algorítmico. Existen diversos lenguajes algorítmicos, pero los más utilizados son:

1. Diagramas
2. Pseudocódigo
3. Lenguajes de programación

En este curso utilizaremos los tres, pero principalmente trabajaremos con el lenguaje de programación C++.

1.1.3. Recursos de programación

1.1.3.1. El ordenador

El ordenador solo puede realizar operaciones aritméticas y lógicas, y tiene mucha memoria. Pero, al no tener nuestra capacidad de abstracción, tendremos que indicarle, hasta el más mínimo detalle, cómo debe hacer cada una de las cosas que queremos que haga.

1.1.3.2. Operaciones aritméticas y expresiones lógicas

Una operación aritmética consiste en un cálculo matemático primario: suma, resta, multiplicación, división y módulo (o valor residual). Todos los lenguajes de programación proveen estos operadores matemáticos.

Una *expresión lógica* es un enunciado susceptible de ser verdadero o falso. “2 es mayor que 5” es una expresión lógica cuyo valor de verdad es *falso*.

Más adelante veremos que existen operadores lógicos que usaremos para realizar operaciones entre expresiones lógicas, y obtendremos como resultado una nueva expresión lógica con su correspondiente valor de verdad.

1.1.3.3. Lenguaje de programación

Para que un ordenador pueda entender y ejecutar un algoritmo, debemos escribirlo en algún lenguaje de programación. En este curso utilizaremos el lenguaje C++.

Los lenguajes de programación se componen de un conjunto de palabras en inglés y un conjunto de reglas sintácticas. La acción de escribir un algoritmo en C++ (o en cualquier otro lenguaje) se llama *codificar*. El algoritmo codificado se llama *código fuente* o simplemente *programa*.

1.1.3.4. Programa

Un programa es un algoritmo codificado en algún lenguaje de programación. También podríamos decir que *el algoritmo es la lógica de un programa de ordenador*.

Por ejemplo, el siguiente algoritmo nos permite preguntarle a una persona cómo se llama y luego saludarla por su nombre:

Preguntar su nombre (¿Cómo te llamas?).

Escuchar y memorizar su nombre (Preparar la memoria y esperar la respuesta).

Saludarlo por su nombre ("Hola," seguido del nombre que memorizamos).

Si el pseudocódigo del algoritmo anterior lo codificamos en C++, lo convertiremos en un programa de computación:

```
int main()
{
   // mensaje para el usuario
   cout << "Ingrese su nombre: ";

   // preparamos la memoria para guardar su nombre
   string nom;

   // el usuario ingresa su nombre y lo guardamos en memoria
   cin >> nom;

   // salida del programa: un saludo para el usuario
   cout << "Hola, " << nom << endl;

   return 0;
}
```

Las líneas que comienzan con `//` (doble barra) son comentarios o acotaciones que nos ayudarán a comprender qué hace la siguiente línea de código.

Las instrucciones `cout` y `cin`, que significan *console out* y *console in*, permiten respectivamente mostrar un mensaje (salida) e introducir un dato (entrada).

Para introducir un dato (que en este caso será el nombre de la persona que interactúa con el programa) y recordarlo posteriormente, debemos almacenarlo en la memoria del ordenador. Esto requiere que anticipadamente dispongamos de un espacio de memoria donde, en este caso, vamos a guardar una cadena de caracteres (conjunto de caracteres).

La línea que dice `string nom` reserva dicho espacio de memoria identificándolo con la palabra `nom`. En adelante, dentro del programa, usaremos `nom` para tener acceso al dato que introdujo el usuario.

Por el momento no analizaremos las palabras `main` y `return`.

1.1.3.5. Variables y tipos de dato

Los identificadores que utilizamos para representar espacios de memoria se llaman variables. Por ejemplo, la variable `nom` del programa anterior.

Las variables permiten guardar datos en la memoria del ordenador. El espacio de memoria que representa una variable debe estar preparado especialmente en función de cuál sea el tipo de dato que allí queremos guardar.

En el ejemplo anterior, el tipo de dato de la variable `nom` es `string`, lo que nos permite guardar valores alfanuméricos (*cadenas de caracteres*).

Sin embargo, si en el programa le hubiésemos pedido al usuario que introdujera su edad, habríamos necesitado disponer de una variable tipo numérico entero: `int`. Y si le hubiéramos preguntado por su altura, probablemente la variable debiera haber sido tipo numérico real: `double`.

Observemos que llamamos `nom` a la variable, justamente porque la utilizaremos para guardar el nombre de una persona. Podríamos haberla llamado `n`, `x`, `pepe` o `theWalrusWasPaul`.

No importa cuál sea el nombre de la variable, aunque sí es muy importante escoger un nombre que describa cuál es el dato que la variable va a contener, pues eso nos ayudará a incrementar la *legibilidad del programa*.

Por ejemplo, si en el programa vamos a pedir introducir: nombre, edad y altura, deberíamos declarar tres variables cuyos nombres podrían ser los siguientes:

Opción 1	Opción 2	Opción 3	Opción 4

`string nombre;`
`int edad;`
`double altura;`
	`string nom;`
	`int ed;`
	`double alt;`
		`string n;`
		`int e;`
		`double a;`
			`string x;`
			`int y;`
			`double z;`

Tabla 1.1. Nombres de variables

Claramente, la opción 1 es la más apropiada, y la opción 4 sería la peor elección. Sin embargo, funcionalmente todas son correctas.

Las variables deben comenzar con un carácter alfabético o un guion bajo (`'_'`). No pueden contener símbolos de puntuación ni operadores de ningún tipo.

Nombres correctos	Nombres incorrectos

`string fecha;`
`string fecha_nacimiento;`
`string fechaNacimiento;`
`string fecha3;`
`string f;`
`string _fec;`
	`string fecha-nacimiento;`
	`string fecha+nacimiento;`
	`string 3fecha;`
	`string fecha.nacimiento;`

Tabla 1.2. Nombres de variables correctos e incorrectos

1.1.3.6. Convención de nombres de variables

Más allá de las diferentes posibilidades que mostramos en la tabla anterior, existe una convención de nombres que restringe y ordena el modo en que debemos llamar a las variables que declaramos en nuestros programas.

Según dicha convención, que aceptaremos y respetaremos a lo largo del curso, las variables deben escribirse en minúscula. Si su nombre se compone de dos o más palabras, cada inicial (excepto la primera) debe colocarse en mayúscula.

Por ejemplo:

```
int codigoPostal;
string nombre;
double alturaPromedio;
```

Por supuesto, lo anterior no impide que utilicemos nombres triviales como `i`, `j`, `k`, `p`, `q` y `aux` para nombrar aquellas variables que resultan poco relevantes. Pero sí debemos nombrarlas con letras minúsculas. Nunca en mayúscula.

1.1.3.7. Consola

Más arriba mencionamos la *consola*. Simplemente diremos que se trata de un dispositivo de entrada/salida compuesto por el teclado físico del ordenador (entrada), y una ventana terminal (salida) a la que accedemos mediante la siguiente combinación de teclas: WIN + R → CMD (en Windows).

1.1.3.8. Compilador

Para que el ordenador pueda ejecutar un programa, debemos convertir su código fuente en lo que llamaremos *código de máquina*, esto es, *unos y ceros*. Es importante tener en cuenta que el ordenador solo comprende y ejecuta instrucciones codificadas de este modo.

Los lenguajes de programación incluyen un programa llamado *compilador* que realiza esa tarea por nosotros. Como se ilustra en la siguiente imagen:

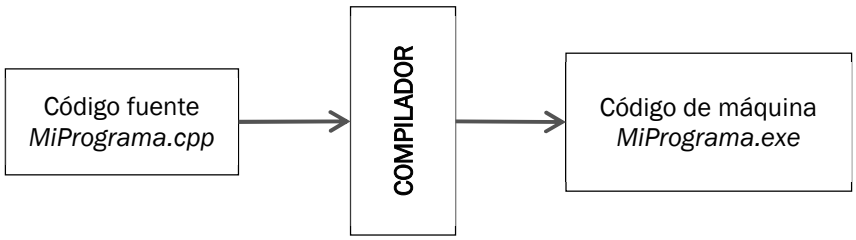


Figura 1.1. Proceso de compilación

El compilador toma el código fuente de un programa (que debe estar contenido en un archivo con extensión `.cpp`), y genera un archivo con extensión `.exe` que contiene el código de máquina (*unos* y *ceros*) que sí podrá ser ejecutado por el ordenador.