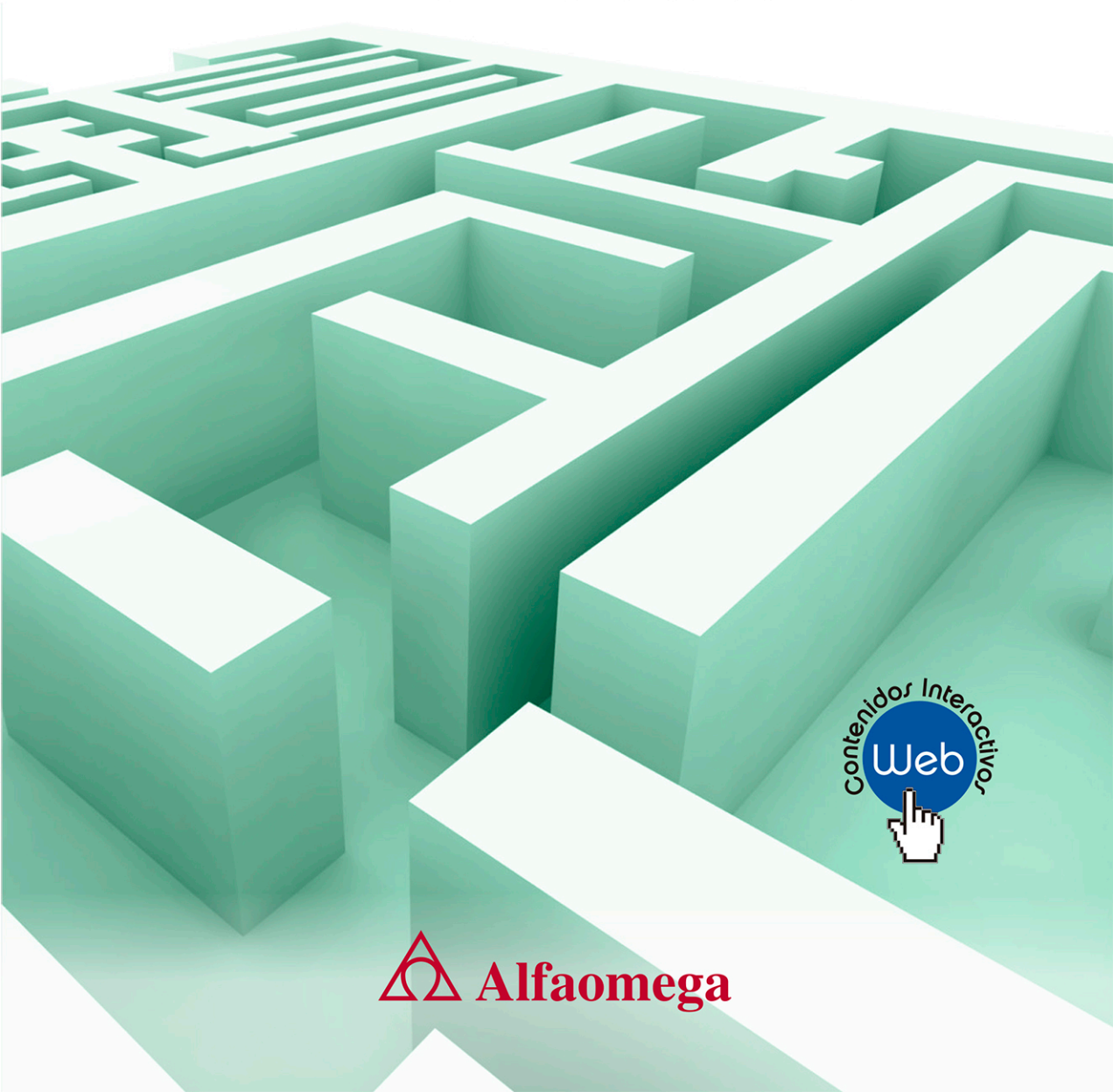
PROGRAMACIÓN ESTRUCTURADA A FONDO

IMPLEMENTACIÓN DE ALGORITMOS EN C

ING. PABLO AUGUSTO SZNAJDLEDER



Alfaomega

Programación estructurada a fondo

Implementación de algoritmos en C

Ing. Pablo Augusto Sznajdleder

Alfaomega

Sznajdleder, Pablo Augusto
Programación estructurada a fondo : Implementación de algoritmos en C / Pablo Augusto Sznajdleder. - 1a ed . - Ciudad Autónoma de Buenos Aires : Alfaomega Grupo Editor Argentino, 2017.
368 p. ; 16.8 x 23 cm.

ISBN 978-987-3832-28-4

1. Programación. I. Título.
CDD 005.3

Edición: Damián Fernández
Revisión de estilo: Vanesa García
Diagramación: Diego Ay
Revisión de armado: Damián Fernández

Internet: http://www.alfaomega.com.co

Av. Córdoba 1215 Piso “10”, Buenos Aires, Argentina, C.P. 1055
ISBN 978-958-778-333-9
Queda hecho el depósito que prevé la ley 11.723

Empresas del grupo:

Argentina: Alfaomega Grupo Editor Argentino, S. A.
Av. Córdoba 1215 Piso “10”, Buenos Aires, Argentina, C.P. 1055
Tel.: (54-11) 4811-7183/0887 / E-mail: ventas@alfaomegaeditor.com.ar

México: Alfaomega Grupo Editor, S. A. de C.V.
Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores. C.P. 06720, Del. Cuauhtemoc. Ciudad de México.
Tel.: (52-55) 5575-5022 - Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396.
E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S. A.
Calle 62 N° 20-46, Bogotá, Colombia
Tel. (57-1)7460102 - Fax: (57-1) 2100122
E-mail: cliente@alfaomegacolombiana.com

Chile: Alfaomega Grupo Editor, S. A.
Av. Providencia 1443, Oficina 24, Santiago de Chile, Chile
Tel.: (56-2) 235-4248/2947-5786 – Fax: (56-2) 235-5786
E-mail: agechile@alfaomega.cl

"El Capitán América"
por Octaviano Sznajdleder

Esta obra no hubiera sido posible sin el apoyo incondicional
de mi esposa Analía y mi hijo Octaviano.

Para ellos está dedicada, con todo el amor de mi corazón.

Quiero agradecer:

a Analía Mora, Hernán Mora, Nahuel Mora y Pablo Bergonzi por el permanente apoyo técnico.

a Damián Fernández por confiar e impulsar mis proyectos editoriales, y por la oportunidad de convertirme en autor.

a Octaviano Sznajdleder por las ilustraciones.

A todos ellos... ¡**Gracias**!

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 tiene 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 espera ser su compañera profesional 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, información actualizada, pruebas (test) de autoevaluación, diapositivas y vínculos con otros sitios Web relacionados.

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 metas propuestas. Cada capítulo concluye con diversas actividades pedagógicas para asegurar la asimilación del conocimiento y su extensión y actualización futuras.

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

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

Pablo Augusto Sznajdleder

Es Ingeniero en Sistemas de Información egresado de la Universidad Tecnológica Nacional (UTN.BA, 1999).
Es profesor ordinario con categoría de asociado en la cátedra de “Algoritmos y estructura de datos” y, promotor y profesor en la materia optativa “Algoritmos complejos para estructuras avanzadas”; ambas materias en UTN.BA.

Es autor de los libros *JEE a fondo* (Alfaomega, 2015), *Algoritmos a fondo* (Alfaomega 2013), *Java a fondo* (Alfaomega, 2010/12/15) y *HolaMundo.pascal* (CEIT, 2007).

Se desempeñó como instructor Java para Sun Microsystems y Oracle obteniendo, en 1997, las certificaciones SCJP y SCJD; estas fueron las primeras certificaciones Java acreditadas en Argentina y estuvieron entre las primeras logradas en latinoamérica.

Actualmente, se desempeña en el ámbito profesional como consultor e instructor en tecnologías Java/JEE proveyendo servicios de *coaching* y capacitación en las empresas líderes del país.

En el ámbito académico, es candidato del “Programa de Maestría en Ingeniería de Sistemas de Información” de la Universidad Tecnológica Nacional (UTN.BA); e Investigador Tesista del Laboratorio de Investigación, Desarrollo e Innovación en Espacios Virtuales de Trabajo de la Universidad Nacional de Lanús.

www.thejavalistener.com

Revisor técnico: Alberto Templos Carbajal

Es Ingeniero en computación de la Facultad de Ingeniería de la UNAM y ejerce, desde 1983, como Profesor de dicha facultad en las Divisiones de Ingeniería Eléctrica en distintas asignaturas de la carrera de Ingeniería en Computación y Educación Continua. Desde 1986, es Profesor a tiempo completo y ha ocupado los siguientes cargos: Coordinador de las carreras de Ingeniero en Computación y, de manera temporal, de Ingeniero Eléctrico Electrónico e Ingeniero en Telecomunicaciones, Jefe del Departamento de Ingeniería en Computación, Secretario Académico de las Divisiones de Ingeniería Eléctrica y Estudios de Postgrado y Coordinador de Postgrado en la Secretaría de Postgrado e Investigación de la Facultad de Ingeniería. También ha sido cofundador de dos empresas de computación y asesor en esa misma área de diversas compañías.
Actualmente, es miembro de diferentes comités, evaluador de planes de estudio del área de computación para diferentes organismos nacionales y es responsable de un proyecto de investigación e innovación tecnológica sobre el Diseño de algoritmos para robots. Su productividad, principalmente, está orientada a la docencia.

Contenido

Acceso a los materiales de la Web de apoyo

Para tener acceso al material de la plataforma de contenidos interactivos del libro, siga los siguientes pasos:

1. Ir a la página http://libroweb.alfaomega.com.mx
2. Ir a la sección Catálogo y seleccionar la imagen de la portada del libro, al dar doble clic sobre ella, tendrá acceso al material descargable.

Estimado profesor: Si desea acceder a los contenidos exclusivos para docentes por favor contacte al representante de la editorial que lo suele visitar o envíenos un correo electrónico a webmaster@alfaomega.com.mx

Información del contenido de la página Web

El material marcado con asterisco (*) solo está disponible para docentes.

Capítulo 1

Introducción a los algoritmos y a la programación de computadoras

- Autoevaluación
- Videotutorial:
  - Instalación y uso de Eclipse para C
- Presentaciones*

Capítulo 2

Estructuras básicas de control y lógica algorítmica

- Autoevaluación
- Videotutorial:
  - Uso del debugger para depurar un programa
- Presentaciones*

Capítulo 3

Funciones, modularización y metodología top-down

- Autoevaluación
- Videotutorial:
  - Mantener archivos de funciones separados del programa principal
- Presentaciones*

Capítulo 4

Tipos de datos alfanuméricos

- Autoevaluación
- Presentaciones*

Capítulo 5

Punteros a carácter

- Autoevaluación
- Presentaciones*

Capítulo 6

Punteros, arrays y aritmética de direcciones

- Autoevaluación
- Videotutorial:
  - Pasar argumentos en línea de comandos con Eclipse
- Presentaciones*

Capítulo 7

Tipos de datos estructurados

- Autoevaluación
- Videotutoriales:
  - Algoritmo de la burbuja
  - Algoritmo de la búsqueda binaria
- Presentaciones*

Capítulo 8

Operaciones sobre archivos

- Autoevaluación
- Videotutoriales:
  - Leer y escribir un archivo
  - Leer y escribir un archivo de registros
- Presentaciones*

Capítulo 9

Tipo Abstracto de Dato (TAD)

- Autoevaluación
- Presentaciones*

Capítulo 10

Análisis de ejercicios integradores

- Autoevaluación
- Presentaciones*

Capítulo 11

Estructuras de datos dinámicas lineales

- Autoevaluación
- Presentaciones*

Más material

- Código fuente de cada capítulo
- Hipervínculos de interés
- Fe de erratas

	Videotutoriales que complementan la obra.
	Conceptos para recordar: bajo este ícono se encuentran definiciones importantes que refuerzan lo explicado en la página.
	Comentarios o información extra: este ícono ayuda a comprender mejor o ampliar el texto principal.

Prólogo

Siempre he pensado que para que exista entendimiento, se debe tener una visión clara de lo que está sucediendo o de lo que se pretende lograr, eso es crucial para el programador de computadoras, puesto que éste sencillamente no puede programar lo que no entiende. Hoy en día, en el que la informática es algo cotidiano y accesible para todo mundo, es común escuchar a personas que minimizan a los sistemas informáticos y piensan que su desarrollo es un proceso lineal, sin dificultad, atribuyendo el hecho de que la modificación de un programa es algo poco costoso debido al hecho de que se trata de algo "virtual" y no existente.

Hay personas que piensan que modificar la lógica de un sistema es tan sencillo como cambiar el color en un mapa de bits, sin darse cuenta que un cambio en la lógica en una parte, puede comprometer seriamente el funcionamiento de todo el sistema. Regularmente esas personas son las que le llaman "logaritmo" a la parte lógica de un programa.

Los que somos participes día a día en el desarrollo de sistemas y estamos inmersos en el fascinante mundo de la programación sabemos que un programa con buena lógica es aquel que cumple su cometido de la manera más directa y económica posible, reduciendo el universo de validaciones a unas cuantas posibilidades bien identificadas; sin embargo, llegar a tener una buena lógica no es algo que se obtenga de la noche a la mañana, pues ésta es producto de la combinación entre creatividad, conocimiento y experiencia.

Esto último es lo que desalienta a muchos iniciados en el mundo de la programación, el verse cara a cara frente al ordenador e intentar darle instrucciones sin tener una idea clara de lo que pretenden lograr y por consecuencia no poder cumplir con el objetivo previsto. La mayoría de desertores le echan la culpa al lenguaje de programación, cuando el lenguaje es solo eso: un lenguaje; la buena lógica es como los ademanes y expresión corporal que acompañan a las palabras de un buen interlocutor.

El autor del presente libro conoce de la problemática anteriormente mencionada y plasma su conocimiento y mucha de su experiencia en las siguientes páginas, preocupándose porque el lector obtenga un pensamiento analítico y tenga una visión de la solución antes de implementarla con un lenguaje de programación. Algoritmos a fondo, ofrece un enfoque práctico, secuencial e incremental, el más acorde para cualquier iniciado o estudioso promedio. Los estudiantes iniciados lo encontrarán de gran ayuda porque parte desde "cero" y es el mismo lector el que va avanzando conforme va dominando los temas. Los estudiantes intermedios y avanzados lo encontrarán fascinante debido a su organización clara y jerárquica, siempre pensada en el que el lector aprenda, identifique y desarrolle sus competencias y áreas de oportunidad (para lo que es bueno y lo que le falta), por encima de abarcar un universo de tópicos en comparación a materiales similares.

En el presente libro el autor aborda los temas contenidos de manera directa y los desarrolla de manera clara y concisa, sin texto de relleno. El hecho de emplear lenguaje C es una motivación para entender bien conceptos profundos como lo son los arreglos, los apuntadores, las cadenas de caracteres, el contexto de un programa y la asignación dinámica de la memoria. Conceptos que se ven mermados en otros lenguajes de alto nivel, pero que todo programador en busca de adquirir una buena lógica debe tener presentes.

A pesar de que la informática está al alcance de todos, aún sigue siendo reducido el número de personas que pueden dictarle instrucciones a una computadora de manera exitosa, como indicaba al inicio, el problema radica en el hecho de que para poder lograrlo se requiere de un pensamiento ordenado y conciso, que tenga presente que es lo que desea obtener. Sin duda alguna, el presente libro ayudará al lector a conseguir ese tipo de pensamiento.

Ingeniero Erick Huerta Valdepeña
DOCENTE DE NIVEL SUPERIOR EN EL INSTITUTO POLITÉCNICO NACIONAL
Y EN LA UNIVERSIDAD TECNOLÓGICA DE MÉXICO

Ciudad de México, octubre de 2016

Introducción

Programación estructurada a fondo, Implementación de algoritmos en C es un libro de nivel universitario pensado para cubrir las necesidades de los alumnos que cursan las asignaturas de Algoritmos y Estructura de Datos.

La obra comienza desde “cero” explicando los conceptos de “algoritmo”, “estructuras de control”, “variables”, “lenguajes de programación”, etcétera; y llega hasta el análisis y resolución de problemas complejos planteados en ejercicios integradores.

Se estudian estructuras de datos estáticas y dinámicas, lineales y no lineales; trabajando con programación estructurada e implementaciones en lenguaje C.

El libro se desarrolla como un “curso de programación” donde se guía al alumno en un proceso de aprendizaje durante el cual podrá adquirir la lógica necesaria para diseñar e implementar algoritmos.

Cada capítulo introduce un mayor nivel de dificultad, ya sea incorporando nuevos conceptos y recursos, o bien profundizando en técnicas de programación más complejas.

El autor remarca la idea de “curso de programación” ya que esta es totalmente opuesta al concepto de “libro tradicional” que, generalmente, presenta una estructura mucho más rígida, tipo “diccionario” o “enciclopedia”.

Cualquier alumno universitario debería poder leer el libro y aprender a programar por sí solo sin experimentar ningún tipo de dificultad ya que este es uno de los principales objetivos del libro.

La obra se complementa con una serie tutoriales grabados en video en los que el autor explica temas que dada su naturaleza resultarían extremadamente tediosos de leer.

Introducción a los algoritmos y a la programación de computadoras

1

Contenido

Objetivos del capítulo

- Entender los conceptos básicos sobre algoritmos y programación.
- Identificar los principales recursos lógicos y físicos que utilizan los programas.
- Estudiar el teorema de la programación estructurada y las estructuras de control que este teorema describe.
- Conocer las herramientas imprescindibles de programación: lenguaje de programación, compilador, entorno de desarrollo (IDE), etcétera.
- Desarrollar, compilar y ejecutar nuestro primer programa de computación.

1.1 Introducción

Este capítulo introduce al lector en los conceptos iniciales sobre algoritmos y programación. Aquellos lectores que nunca han programado encontrarán aquí los conocimientos básicos para hacerlo.

Los contenidos son netamente introductorios ya que pretenden brindarle al lector las pautas, las expresiones y la jerga que necesitará conocer para poder leer y comprender los capítulos sucesivos.

Por lo anterior, la profundidad con la que se tratan los temas aquí expuestos es de un nivel inicial, ya que cada uno de estos temas será analizado en detalle llegado el momento, en los capítulos que así lo requieran.

1.2 Concepto de algoritmo

1.2.1 Definición de algoritmo y problema

Llamamos “algoritmo” al conjunto finito y ordenado de acciones con las que podemos resolver un determinado problema.

Llamamos “problema” a una situación que se nos presenta y que, mediante la aplicación de un algoritmo, pretendemos resolver.

Los algoritmos están presentes en nuestra vida cotidiana y, aún sin saberlo, aplicamos algoritmos cada vez que se nos presenta un problema sin importar cuál sea su grado de complejidad.

Para ejemplificar esto imaginemos que estamos dando un paseo por la ciudad y que, al llegar a una esquina, tenemos que cruzar la calle. Intuitivamente, aplicaremos el siguiente algoritmo:

- Esperar a que la luz del semáforo peatonal esté en verde (`esperarSemaforo`);
- Cruzar la calle (`cruzarCalle`);

Este algoritmo está compuesto por las acciones: `esperarSemaforo` y `cruzarCalle`, en ese orden, y es extremadamente simple gracias a que cada una de estas engloba a otro conjunto de acciones más puntuales y específicas. Por ejemplo:

La acción `esperarSemaforo` implica:

- Observar la luz del semáforo (`observarLuz`);
- **Si** la luz está en “rojo” o en “verde intermitente” **entonces**
  - esperar un momento (`esperar`);
  - **ir a**: `observarLuz`;

Por otro lado, la acción `cruzarCalle` implica:

- Bajar la calzada (`bajarCalzada`);
- Avanzar un paso (`avanzarPaso`);
- **Si** la distancia hacia la otra calzada es mayor que la distancia que podemos avanzar dando un nuevo paso **entonces**

- **ir a**: `avanzarPaso`;
- Subir la calzada de la vereda de enfrente (`subirCalzada`);

Como vemos, cada acción puede descomponerse en acciones más puntuales y específicas. Por lo tanto, cada una de las acciones del algoritmo se resuelve con su propio algoritmo o secuencia finita y ordenada de acciones.

> **Módulo**
> Cuando una acción se descompone en acciones más puntuales y específicas la llamamos "módulo".

Podemos seguir profundizando cada acción tanto como queramos. Por ejemplo, analicemos la acción (o el módulo) `bajarCalzada`: esta acción se resuelve:

- levantando un pie,
- adelantándolo,
- inclinando el cuerpo hacia adelante,
- y apoyando el pie en la calle,
- luego levantando el otro pie,
- adelantándolo y
- apoyándolo en la calle.

Claro que “levantar un pie” implica tensionar un conjunto de músculos, etcétera.

1.2.2 Análisis del enunciado de un problema

Resulta evidente que, si vamos a diseñar un algoritmo para resolver un determinado problema, tenemos que tener totalmente estudiado y analizado el contexto de dicho problema. Esto implica:

- Comprender el alcance.
- Identificar los datos de entrada.
- Identificar los datos de salida o resultados.

El análisis anterior es fundamental para poder diseñar una estrategia de solución que será la piedra fundamental para el desarrollo del algoritmo.

1.2.2.1 Análisis del problema

Repasemos el contexto del problema de “cruzar la calle” que formulamos más arriba:

Caminando por la ciudad llegamos a una esquina y pretendemos cruzar la calle. Para poder cruzar con la mayor seguridad posible, tenemos que esperar que el semáforo habilite el paso peatonal. Hasta que esto ocurra nos encontraremos detenidos al lado del semáforo y una vez que este lo permita avanzaremos, paso a paso, hasta llegar a la vereda de enfrente. Se considera que el semáforo habilita el paso peatonal cuando emite una luz “verde fija”. En cambio si el semáforo emite una luz “roja” o “verde intermitente” se recomienda esperar.

1.2.2.2 Datos de entrada

En este contexto podemos identificar los siguientes datos de entrada:

- Nuestra posición inicial - posición en donde nos detuvimos a esperar a que el semáforo habilite el paso peatonal. La llamaremos `posInicial`.
- Luz del semáforo - el color de la luz que el semáforo está emitiendo. Lo llamaremos `luz`.
- Distancia de avance de paso - que distancia avanzamos cada vez que damos un nuevo paso. La llamaremos `distPaso`.
- Posición de la vereda de enfrente. La llamaremos `posFinal` ya que es allí hacia donde nos dirigimos.

1.2.2.3 Datos de salida

Si bien en este problema no se identifican datos de salida resulta evidente que luego de ejecutar la secuencia de acciones del algoritmo nuestra posición actual deberá ser la misma que la posición de la vereda de enfrente. Es decir, si llamamos `p` a nuestra posición actual resultará que su valor al inicio del algoritmo será `posInicial` y al final del mismo deberá ser `posFinal`.

1.2.3 Memoria y operaciones aritméticas y lógicas

En la vida real, cuando observamos la luz del semáforo almacenamos este dato en algún lugar de nuestra memoria para poder evaluar si corresponde cruzar la calle o no.

Generalmente, los datos de entrada ingresan al algoritmo a través de una acción que llamamos “lectura” o “ingreso de datos” y permanecen en la memoria el tiempo durante el cual el algoritmo se está ejecutando.

Por otro lado, el hecho de evaluar si corresponde o no cruzar la calle implica realizar una operación lógica. Esto es: determinar si la proposición “el semáforo emite luz roja o verde intermitente” resulta ser verdadera o falsa.

También realizaremos una operación lógica cuando estemos cruzando la calle y tengamos que determinar si corresponde dar un nuevo paso o no. Recordemos que llamamos `p` a nuestra posición actual, `posFinal` a la posición de la vereda de enfrente y `distPaso` a la distancia que avanzamos dando un nuevo paso. Por lo tanto, corresponde dar otro paso si se verifica que `posFinal > p+distPaso`. Aquí además de la operación lógica estamos realizando una operación aritmética: la suma `p+distPaso`.

En resumen, estos son los recursos que tenemos disponibles para diseñar y desarrollar algoritmos:

- La memoria.
- La posibilidad de realizar operaciones aritméticas.
- La posibilidad de realizar operaciones lógicas.

1.2.4 Teorema de la programación estructurada

Este teorema establece que todo algoritmo puede resolverse mediante el uso de tres estructuras básicas llamadas “estructuras de control”:

- La “estructura secuencial” o “acción simple”.
- La “estructura de decisión” o “acción condicional”.
- La “estructura iterativa” o “acción de repetición”.

Dos de estas estructuras las hemos utilizado en nuestro ejemplo de cruzar la calle. La estructura secuencial es la más sencilla y simplemente implica ejecutar una acción, luego otra y así sucesivamente.

A Corrado Böhm y Giuseppe Jacopini se les atribuye el teorema de la programación estructurada, debido a un artículo de 1966. David Harel rastreó los orígenes de la programación estructurada hasta la descripción de 1946 de la arquitectura de von Neumann y el teorema de la forma normal de Kleene. La carta escrita en 1968 por Dijkstra, titulada "*Go To Considered Harmful*" reavivó el debate. En la Web de apoyo, encontrará el vínculo a la página Web personal de Corrado Böhm.

La estructura de decisión la utilizamos para evaluar si correspondía dar un nuevo paso en función de nuestra ubicación actual y la ubicación de la vereda de enfrente.

Sin embargo, para resolver el problema hemos utilizado una estructura tabú: la estructura "**ir a**" o, en inglés, "*go to*" o "*goto*". Esta estructura quedó descartada luego de que el teorema de la programación estructurada demostrara que con una adecuada combinación de las tres estructuras de control antes mencionadas es posible resolver cualquier algoritmo sin tener que recurrir al "*goto*" (o estructura "ir a").

Para ejemplificarlo vamos a replantear el desarrollo de los algoritmos anteriores y reemplazaremos la estructura "**ir a**" (*goto*) por una estructura iterativa: la estructura "**repetir mientras que**".

Veamos el desarrollo del módulo `esperarSemaforo`:

Con "**ir a**"	Sin ~~"**ir a**"~~ (solución correcta)
• `observarLuz`; • **Si** la luz está en "rojo" o en "verde intermitente" **entonces** - `esperar`; - **ir a**: `observarLuz`;	• `observarLuz`; • **Repetir mientras que** la luz esté en "rojo" o en "verde intermitente" **hacer** - `esperar`; - `observarLuz`;

Veamos ahora el desarrollo del módulo `cruzarCalle`:

Con "**ir a**"	Sin ~~"**ir a**"~~ (solución correcta)
• `bajarCalzada`; • `avanzarPaso`; • **Si** la distancia hacia la otra calzada es mayor que la distancia que podemos avanzar dando un nuevo paso **entonces** - **ir a**: `avanzarPaso`; • `subirCalzada`;	• `bajarCalzada`; • `avanzarPaso`; • **Repetir mientras que** la distancia hacia la otra calzada sea mayor que la distancia que podemos avanzar dando un nuevo paso **hacer** - `avanzarPaso`; • `subirCalzada`;

Como vemos, la combinación "acción condicional + *goto*" se puede reemplazar por una estructura de repetición. Si bien ambos desarrollos son equivalentes la nueva versión es mejor porque evita el uso del *goto*, estructura que quedó en desuso porque trae grandes problemas de mantenibilidad.

1.3 Conceptos de programación

En general, estudiamos algoritmos para aplicarlos a la resolución de problemas mediante el uso de la computadora. Las computadoras tienen memoria y tienen la capacidad de resolver operaciones aritméticas y lógicas. Por lo tanto, son la herramienta fundamental para ejecutar los algoritmos que vamos a desarrollar.

Para que una computadora pueda ejecutar las acciones que componen un algoritmo tendremos que especificarlas o describirlas de forma tal que las pueda comprender.

Todos escuchamos alguna vez que "las computadoras solo entienden 1 (unos) y 0 (ceros)" y, efectivamente, esto es así, pero para nosotros (simples humanos) especificar las acciones de nuestros algoritmos como diferentes combinaciones de unos y ceros sería una tarea verdaderamente difícil. Afortunadamente, existen los lenguajes de programación que proveen una solución a este problema.

1.3.1 Lenguajes de programación

Las computadoras entienden el lenguaje binario (unos y ceros) y nosotros, los humanos, entendemos lenguajes naturales (español, inglés, portugués, etc.).

Los lenguajes naturales son los que hablamos y escribimos los seres humanos: inglés, español, italiano, etc. Son dinámicos ya que, constantemente, incorporan nuevas variaciones, palabras y significados. Por el contrario, los lenguajes formales son rígidos y se construyen a partir de un conjunto de símbolos (alfabeto) unidos por un conjunto de reglas (gramática). Los lenguajes de programación son lenguajes formales.

Los lenguajes de programación son lenguajes formales que se componen de un conjunto de palabras, generalmente en inglés, y reglas sintácticas y semánticas.

Podemos utilizar un lenguaje de programación para escribir o codificar nuestro algoritmo y luego, con un programa especial llamado "compilador", podremos generar los "unos y ceros" que representan sus acciones. De esta manera, la computadora será capaz de comprender y convertir al algoritmo en un programa de computación.

Existen muchos lenguajes de programación: Pascal, C, Java, COBOL, Basic, Smalltalk, etc., y también existen muchos lenguajes derivados de los anteriores: Delphi (derivado de Pascal), C++ (derivado de C), C# (derivado de C++), Visual Basic (derivado de Basic), etcétera.

El lenguaje de programación C fue creado por Dennis Ritchie (1941-2011).
En 1967 Ritchie comenzó a trabajar para los laboratorios Bell, donde se ocupó, entre otros, del desarrollo del lenguaje B. Creó el lenguaje de programación C en 1972, junto con Ken Thompson. Ritchie también participó en el desarrollo del sistema operativo Unix.

En este libro utilizaremos el lenguaje de programación C.

1.3.2 Codificación de un algoritmo

Cuando escribimos las acciones de un algoritmo en algún lenguaje de programación decimos que lo estamos "codificando". Generalmente, cada acción se codifica en una línea de código.

Al conjunto de líneas de código, que obtenemos luego de codificar el algoritmo, lo llamamos "código fuente".

El código fuente debe estar contenido en un archivo de texto cuyo nombre debe tener una extensión determinada que dependerá del lenguaje de programación que hayamos utilizado. Por ejemplo, si codificamos en Pascal entonces el nombre del archivo debe tener la extensión ".pas". Si codificamos en Java entonces la extensión del nombre del archivo deberá ser ".java" y si el algoritmo fue codificado en C entonces el nombre del archivo deberá tener la extensión ".c".

1.3.3 Bibliotecas de funciones

Los lenguajes de programación proveen bibliotecas o conjuntos de funciones a través de las cuales se ofrece cierta funcionalidad básica que permite, por ejemplo, leer y escribir datos sobre cualquier dispositivo de entrada/salida como podría ser la consola.

Es decir, gracias a que los lenguajes proveen estos conjuntos de funciones los programadores estamos exentos de programarlas y simplemente nos limitaremos a utilizarlas.

Programando en C, por ejemplo, cuando necesitemos mostrar un mensaje en la pantalla utilizaremos la función `printf` y cuando queramos leer datos a través del teclado utilizaremos la función `scanf`. Ambas funciones forman parte de la biblioteca estándar de entrada/salida de C, también conocida como "stdio.h".

1.3.4 Programas de computación

Un programa es un algoritmo que ha sido codificado y compilado y que, por lo tanto, puede ser ejecutado en una computadora.

El algoritmo constituye la lógica del programa. El programa se limita a ejecutar cada una de las acciones del algoritmo. Por esto, si el algoritmo tiene algún error entonces el programa también lo tendrá y si el algoritmo es lógicamente perfecto entonces el programa también lo será.

El proceso para crear un nuevo programa es el siguiente:

- Diseñar y desarrollar su algoritmo.
- Codificar el algoritmo utilizando un lenguaje de programación.
- Compilarlo para obtener el código de máquina o archivo ejecutable (los "unos y ceros").

Dado que el algoritmo es la parte lógica del programa muchas veces utilizaremos ambos términos como sinónimos ya que para hacer un programa primero necesitaremos diseñar su algoritmo. Por otro lado, si desarrollamos un algoritmo seguramente será para, luego, codificarlo y compilarlo, es decir, programarlo.

1.3.5 Consola

Llamamos "consola" al conjunto compuesto por el teclado y la pantalla de la computadora en modo texto. Cuando hablemos de ingreso de datos por consola nos estaremos refiriendo al teclado y cuando hablemos de mostrar datos por consola estaremos hablando de la pantalla, siempre en modo texto.

1.3.6 Entrada y salida de datos

Llamamos “entrada” al conjunto de datos externos que ingresan al algoritmo. Por ejemplo, el ingreso de datos por teclado, la información que se lee a través de algún dispositivo como podría ser un lector de código de barras, un *scanner* de huellas digitales, etcétera.

Llamamos “salida” a la información que el algoritmo emite sobre algún dispositivo como ser la consola, una impresora, un archivo, etcétera.

La consola es el dispositivo de entrada y salida por omisión. Es decir, si hablamos de ingreso de datos y no especificamos nada más será porque el ingreso de datos lo haremos a través del teclado. Análogamente, si hablamos de mostrar cierta información y no especificamos más detalles nos estaremos refiriendo a mostrarla por la pantalla de la computadora, en modo texto.

1.3.7 Lenguajes algorítmicos

Llamamos “lenguaje algorítmico” a todo recurso que permita describir con mayor o menor nivel de detalle los pasos que componen un algoritmo.

Los lenguajes de programación, por ejemplo, son lenguajes algorítmicos ya que, como veremos más adelante, la codificación del algoritmo es en si misma una descripción detallada de los pasos que lo componen. Sin embargo, para describir un algoritmo no siempre será necesario llegar al nivel de detalle que implica codificarlo. Una opción válida es utilizar un “pseudocódigo”.

1.3.8 Pseudocódigo

El pseudocódigo surge de mezclar un lenguaje natural (por ejemplo, el español) con ciertas convenciones sintácticas y semánticas propias de un lenguaje de programación. Justamente, el algoritmo que resuelve el problema de “cruzar la calle” fue desarrollado utilizando un pseudocódigo.

Los diagramas también permiten detallar los pasos que componen un algoritmo; por lo tanto, son lenguajes algorítmicos. En este libro profundizaremos en el uso de diagramas para luego codificarlos con el lenguaje de programación C.

1.4 Representación gráfica de algoritmos

Los algoritmos pueden representarse mediante el uso de diagramas. Los diagramas proveen una visión simplificada de la lógica del algoritmo y son una herramienta importantísima que utilizaremos para analizar y documentar los algoritmos o programas que vamos a desarrollar.

En este libro, utilizaremos una versión modificada de los diagramas de *Nassi-Shneiderman*, también conocidos como diagramas *Chapin*. Estos diagramas se componen de un conjunto de símbolos que permiten representar cada una de las estructuras de control que describe el teorema de la programación estructurada: la estructura secuencial, la estructura de decisión y la estructura de repetición.

Los diagramas Nassi-Shneiderman, también conocidos como "diagramas de Chapin", fueron publicados en 1973 por Ben Shneiderman e Isaac Nassi en el artículo llamado "A short history of strutctured flowcharts", con el objetivo de eliminar las líneas de los diagramas tradicionales y así reforzar la idea de estructuras de “única entrada y única salida”.
Si en la programación estructurada se elimina la sentencia GOTO entonces se deben eliminar las líneas en los diagramas que la representan.

1.4.1 Representación gráfica de la estructura secuencial o acción simple

La estructura secuencial se representa en un recuadro o, si se trata de un ingreso o egreso de datos se utiliza un trapecio como observamos a continuación:

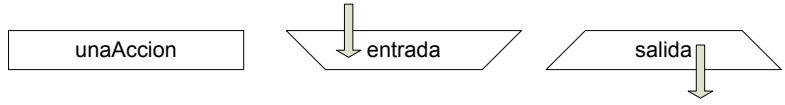


Fig. 1.1 Representación de estructuras secuenciales.

Nota: las flechas grises no son parte de los símbolos de entrada y salida, simplemente son ilustrativas.

1.4.2 Representación gráfica de la estructura de decisión

Esta estructura decide entre ejecutar un conjunto de acciones u otro en función de que se cumpla o no una determinada condición o expresión lógica.

Informalmente, diremos que una “expresión lógica” es un enunciado susceptible de ser verdadero o falso. Por ejemplo, “2 es par” o “5 es mayor que 8”. Ambas expresiones tienen valor de verdad, la primera es verdadera y la segunda es falsa.

La estructura se representa dentro de una “casa con dos habitaciones y techo a dos aguas”. En “el altillo” indicamos la expresión lógica que la estructura debe evaluar. Si esta expresión resulta ser verdadera entonces se ejecutarán las acciones ubicadas en la sección izquierda. En cambio, si la expresión resulta ser falsa se ejecutarán las acciones que estén ubicadas en la sección derecha.

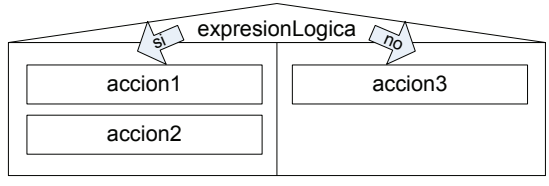


Fig. 1.2 Representación gráfica de la estructura condicional.

En este caso, si se verifica `expresionLogica` se ejecutarán las acciones `accion1` y `accion2`. En cambio, si `expresionLogica` resulta ser falsa se ejecutará únicamente la acción `accion3`. Las flechas grises son ilustrativas y no son parte del diagrama.

1.4.3 Representación gráfica de la estructura de repetición

La estructura de repetición se representa en una “caja” con una cabecera que indica la expresión lógica que se debe cumplir para seguir ejecutando las acciones contenidas en la sección principal.

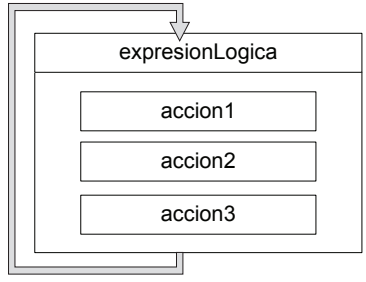


Fig. 1.3 Representación gráfica de la estructura de repetición.

En este caso, mientras `expresionLogica` resulte ser verdadera se repetirán una y otra vez las acciones `accion1`, `accion2` y `accion3`. Por lo tanto, dentro de alguna de estas acciones deberá suceder algo que haga que la condición del ciclo se deje de cumplir. De lo contrario, el algoritmo se quedará iterando dentro de este ciclo de repeticiones.

Con lo estudiado hasta aquí, podemos representar gráficamente el algoritmo que resuelve el problema de cruzar la calle.

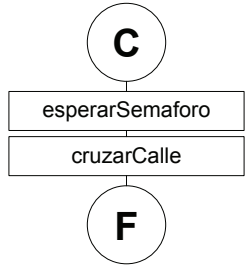


Fig. 1.4 Diagrama del algoritmo principal.

El diagrama principal comienza con una **C** y finaliza con una **F**, iniciales de "Comienzo" y "Fin", cada una encerrada dentro de un círculo. Luego, las acciones simples `esperarSemaforo` y `cruzarCalle` se representan dentro de rectángulos, una detrás de la otra. Como cada una de estas acciones corresponde a un módulo tendremos que proveer también sus propios diagramas.

1.4.4 Representación gráfica de módulos o funciones

Como estudiamos anteriormente, un módulo representa un algoritmo que resuelve un problema específico y puntual. Para representar, gráficamente, las acciones que componen a un módulo, utilizaremos un paralelogramo con el nombre del módulo como encabezado y una **R** (inicial de "Retorno") encerrada dentro de un círculo para indicar que el módulo finalizó y que se retornará el control al algoritmo o programa principal.

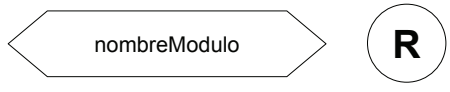


Fig. 1.5 Encabezado y finalización de un módulo.

Con esto, podemos desarrollar los diagramas de los módulos `esperarSemaforo` y `cruzarCalle`.

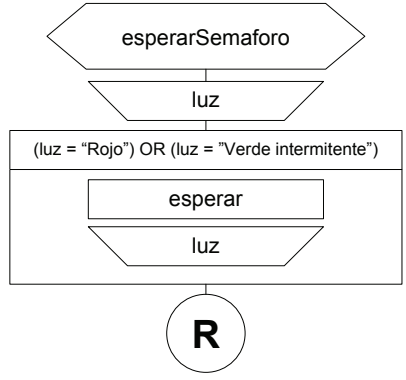


Fig. 1.6 Módulo `esperarSemaforo`.

En el diagrama del módulo `esperarSemaforo`, leemos el color de la luz que actualmente está emitiendo el semáforo y lo “mantenemos en memoria” asociándolo al identificador `luz`. Esto lo hacemos indicando el nombre del identificador dentro del símbolo de lectura o ingreso de datos.

Luego ingresamos a una estructura de repetición que iterará mientras que la expresión lógica `(luz = "Rojo") OR (luz = "Verde intermitente")` resulte verdadera. Las acciones encerradas dentro de esta estructura se repetirán una y otra vez mientras la condición anterior continúe verificándose. Estas acciones son: `esperar` y volver a leer la luz que emite el semáforo.

Evidentemente, en algún momento, el semáforo emitirá la luz “verde fija”, la condición de la cabecera ya no se verificará y el ciclo de repeticiones dejará de iterar.

Veamos ahora el diagrama del módulo `cruzarCalle`.

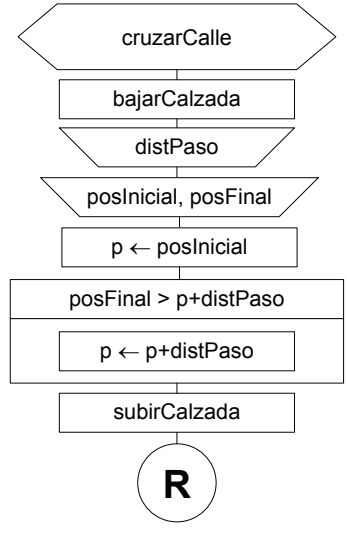


Fig. 1.7 Módulo `cruzarCalle`.

El análisis de este módulo es similar al anterior. Aquí primero invocamos al módulo `bajarCalzada` y luego, ingresamos los datos `distPaso` (distancia que avanzamos al dar un nuevo paso), `posInicial` y `posFinal` (la posición donde estamos parados y la posición de la vereda de enfrente respectivamente). A continuación, "asignamos" al identificador `p` el valor de `posInicial` y luego, ingresamos a una estructura de repetición que iterará mientras que se verifique la expresión lógica:

```
posFinal > p+distPaso
```

Dentro de la estructura de repetición, tenemos que avanzar un paso. Si bien en el primer análisis del algoritmo habíamos definido un módulo `avanzarPaso`, en este caso, preferí o reemplazarlo directamente por la acción:

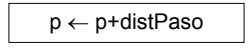


Fig. 1.8 Asignación y acumulador.

Esta acción indica que a `p` (identificador que contiene nuestra posición) le asignamos la suma de su valor actual más el valor de `distPaso` (distancia que avanzamos dando un nuevo paso). Luego de esto estaremos más cerca de `posFinal`.
Para finalizar invocamos al módulo `subirCalzada` y retornamos al diagrama del programa principal.

1.5 Nuestro primer programa

Vamos a analizar, diseñar y programar nuestro primer algoritmo. Se trata de un programa trivial que simplemente emite en la consola la frase "Hola Mundo".

La representación gráfica de este algoritmo es la siguiente:

El algoritmo comienza, emite un mensaje en la consola diciendo "Hola Mundo" y finaliza.

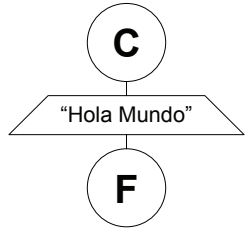


Fig. 1.9 Representación gráfica del algoritmo "Hola Mundo".

1.5.1 Codificación del algoritmo utilizando el lenguaje C

El siguiente paso es codificar el algoritmo. Con el diagrama resuelto, la codificación se limita a transcribirlo en el `lenguaje` de programación elegido que, en este caso, es C.

Veamos el código fuente y luego lo analizaremos: