



ENRIQUE
GÓMEZ JIMÉNEZ

PERCY
CAÑIPA VALDEZ

ESTRUCTURA DE DATOS

Un enfoque con Python, Java y C++



ENRIQUE GÓMEZ JIMÉNEZ

Es licenciado en Ingeniería Informática con mención en Desarrollo de Software. Posee una maestría en Gestión de la Innovación Tecnológica de la Universidad Nacional de Costa Rica.

Profesionalmente, el máster Enrique Gómez Jiménez se ha desempeñado como analista programador en varias empresas privadas y en instituciones públicas. Ha fungido como miembro de equipos de proyectos de desarrollo de software nacionales e internacionales, jefe de informática e implementador de infraestructuras tecnológicas. Durante muchos años se desempeñó como trabajador independiente (freelance) con productos de software propios y de terceros para el mercado local. Actualmente se dedica a la docencia de tiempo completo como profesor propietario de la carrera de Ingeniería en Sistemas de Información de la Universidad Nacional de Costa Rica (UNA); y de tiempo parcial trabaja con la Universidad Estatal a Distancia (UNED) de Costa Rica.

PERCY CAÑIPA VALDEZ

Es bachiller en ingeniería en Sistemas. Posee una maestría en Administración de Tecnologías de Información de la Universidad Nacional de Costa Rica.

Profesionalmente, se ha desempeñado como analista de sistemas en varias empresas privadas. Fue parte de equipos de proyectos de desarrollo de software para empresas públicas nacionales, del sector bancario, así como del sector de seguros. Fue profesor de la carrera de Ingeniería de Sistema de Información en la Universidad Nacional de Costa Rica sede Heredia Actualmente, es profesor a tiempo completo de la carrera de Ingeniería de Sistemas de la Universidad Estatal a Distancia (UNED) de Costa Rica.

Estructura de datos

Un enfoque con Python, Java y C++

```
22. #crea enodo
23. raizBebidas
24. #crea dos no
25. bCaliente
26. bFria
27. #agrega los
odo raizBebidas
29. raizBebidas.
30. #crea tres n
31. cafe = nodoA
32. te = nodoArb
33. jugo = nodoA
```

raiz raiz Bebida
= nodoArbol('Be
dos independien
= nodoArbol('Ca

Estructura de datos

Un enfoque con Python, Java y C++

ENRIQUE GÓMEZ JIMÉNEZ
PERCY CAÑIPA VALDEZ

insertarHijo(bF
nodos independie
Arbol('Café')
bol('Té')
Arbol('Jugo nara

alphaeditorial

Catalogación en la publicación – Biblioteca Nacional de Colombia

Gómez Jiménez, Enrique, autor

Estructura de datos : un enfoque con Python, Java y C++ / Enrique Gómez Jiménez, Percy Cañipa Valdez. -- Primera edición. -- Bogotá : Alpha Editorial, 2023.
394 Páginas.

Incluye datos curriculares de los autores.

ISBN 978-958-778-900-3 -- 978-958-778-901-0 (digital)

1. Estructura de datos (Computadores) 2. Procesamiento electrónico de datos 3. Organización de archivos (Computadores) 4. C++ (Lenguaje de programación de computadores) 5. Java (Lenguaje de programación de computadores) 5. Python (Lenguaje de programación de computadores) 6. Programación orientada a objetos (Computación) 7. Cañipa Valdez, Percy, autor

CDD: 005.73 ed. 23

CO-BoBN- a1117625

Alpha Editorial S.A.

Calle 62 20-46 /esquina, Bogotá
Teléfono (601) 746 0102
cliente@alpha-editorial.com
www.alpha-editorial.com

Libros digitales

www.alphaeditorialcloud.com

Primera edición: Bogotá, 2023

© Alpha Editorial S. A.

© Enrique Gómez Jiménez / Percy Cañipa Valdez

Derechos reservados. Esta publicación no puede ser reproducida total ni parcialmente. Ni puede ser registrada por un sistema de recuperación de información, en ninguna forma ni por ningún medio, sea mecánico, fotoquímico, electrónico, magnético, electroóptico, fotocopia o cualquier otro, sin el previo permiso escrito de la editorial.

Edición: Samantha Córdoba Hernández

Portada: Ana Paula Santander Durán

ISBN 978-958-778-900-3

ISBN 978-958-778-901-0 DIGITAL

Impreso en Colombia

Printed and made in Colombia

CONTENIDO

Agradecimientos	xiii
Figuras	xv
Tablas	xxii
Introducción	xxv
Material web	xxvi

1. Introducción a C++

1.1. Conceptos básicos de C++	3
1.2. Tipos de datos en C++	8
1.3. Entrada y salida de datos en C++	9
1.4. Operaciones y operadores matemáticos en C++	10
1.5. Estructuras condicionales y repetitivas en C++	10
1.6. Funciones, métodos y procedimientos en C++	11
1.7. Parámetros en C++	14
1.8. Arreglos en C++	16
Ejercicios propuestos y resueltos	18

2. Programación orientada a objetos (POO) en C++

2.1. Introducción a la programación orientada a objetos en C++	23
2.1.1. Clases	23
2.1.2. Herencia simple en C++	25
2.1.3. Herencia múltiple en C++	30
2.2. Clases abstractas, asociación por composición y por agregación	33
2.2.1. Clases abstractas	33
2.2.2. Clases compuestas (relación de composición)	36
2.2.3. Clases agregadas (relación de agregación)	36
Ejercicios propuestos y resueltos	41

3. Estructuras de datos y recursividad en C++

3.1. Iteración y recursividad en C++	47
3.2. Manejo de cadenas en C++	49
3.3. Punteros en C++	51
3.4. Punteros a clases en C++	56
3.5. Punteros y vectores en C++	57
3.6. Arreglos unidimensionales dinámicos mediante punteros en C++	60
3.7. Arreglos como parámetros por referencia mediante punteros en C++	61
3.8. Matrices dinámicas mediante punteros en C++	63
3.9. Struct mediante punteros en C++	64
Ejercicios propuestos y resueltos	67

4. Ordenamientos y búsquedas en C++

4.1. Métodos de ordenamientos en C++	73
4.1.1. Método <i>BubbleSort</i> (burbuja)	74
4.1.2. Método <i>Insert</i> (inserción)	76
4.1.3. Método <i>Select</i> (selección)	78
4.1.4. Método <i>Shell</i>	82
4.1.5. Método <i>QuickSort</i> (ordenamiento rápido)	85
4.1.6. Método <i>MergeSort</i> (ordenamiento por combinación)	89
4.2. Búsquedas en arreglos	91
4.2.1. <i>Búsqueda secuencial</i>	92
4.2.2. <i>Búsqueda secuencial (dicotómica)</i>	93
Ejercicios propuestos y resueltos	98

5. Pilas y colas en C++

5.1. Pilas en C++	105
5.2. Colas en C++	107
5.2.1. <i>Inserción de nodos</i>	109

5.2.2. <i>Eliminación de nodos</i>	109
5.2.3. <i>Colas circulares en C++</i>	111
Ejercicios propuestos y resueltos	114

6. Listas enlazadas

6.1. Listas enlazadas en C++	124
6.1.1. <i>Listas simplemente enlazadas en C++</i>	125
6.1.2. <i>Listas doblemente enlazadas en C++</i>	129
6.1.3. <i>Listas circulares enlazadas en C++</i>	137
6.1.4. <i>Listas circulares doblemente enlazadas en C++</i>	139
Ejercicios propuestos y resueltos	140

7. Árboles

7.1. Árboles binarios	149
7.1.1. <i>Árboles binarios distintos, similares y equivalentes</i>	149
7.1.2. <i>Árboles binarios completos</i>	150
7.1.3. <i>Árboles binarios llenos</i>	152
7.1.4. <i>Árboles binarios degenerados</i>	152
7.1.5. <i>Árboles binarios equilibrados</i>	152
7.1.6. <i>Árboles binarios perfectamente equilibrados</i>	152
7.1.7. <i>Árboles multicaminos</i>	153
7.1.8. <i>Árboles binarios de búsqueda (ABB)</i>	153
7.1.9. <i>Recorrido de árboles binarios</i>	154
7.1.10. <i>Árbol binario ordenado</i>	156
7.1.11. <i>Árbol binario AVL</i>	162
Ejercicios propuestos y resueltos	168

8. Grafos

8.1. Aplicabilidad de los grafos	176
8.2. Tipos de grafos	176
8.2.1. <i>Grafo simple</i>	177
8.2.2. <i>Grafo dirigido</i>	177
8.2.3. <i>Grafo no dirigido</i>	178
8.2.4. <i>Grafo etiquetado</i>	178

8.2.5. Grafo aleatorio	179
8.2.6. Hipergrafo	180
8.2.7. Grafos infinitos	181
8.3. Representación de un grafo	181
8.3.1. Matriz de adyacencia	181
8.3.2. Listas de adyacencia	186
8.3.3. Listas de arcos	188
8.4. Algoritmos de recorrido y búsqueda en grafos	188
8.4.1. Búsqueda en amplitud o anchura (BFS: Breadth First Search)	190
8.4.2. Búsqueda primero en profundidad (DFS: Depth First Search)	194
Ejercicios propuestos y resueltos	200

9. STL: Standard Template Library

9.1. Contenedores STL	210
9.1.1. Clase vector	211
9.1.2. Clase deque (doble cola)	211
9.1.3. Clase list (lista)	213
9.2. Iteradores	216
9.3. Algoritmos STL	218
9.3.1. Algorithm	220
9.3.2. Métodos matemáticos	224
9.4. Algoritmos asociativos STL	225
9.4.1. Contenedor asociativo set	225
9.4.2. Contenedor asociativo multiset	227
9.4.3. Contenedor asociativo map	229
Ejercicios propuestos y resueltos	231

10. Manejo de archivos

10.1. Ficheros	237
10.1.1. Tipos de ficheros	237
10.1.2. Memoria intermedia (buffers)	238
10.1.3. Gestión de archivos de texto	239
10.1.4. Gestión de archivos binarios	241
Ejercicios propuestos y resueltos	246

11. Fundamentos de estructura de datos en Java / PARTE I

11.1. Generics collection Java	251
11.1.1. <i>Collection Interface Java</i>	252
11.2. Expresiones Lambda en Java	256
11.3. Colas y pilas en Java	258
11.3.1. <i>Colas simples en Java</i>	258
11.3.2. <i>Colas de prioridad en Java (PriorityQueue)</i>	261
11.3.3. <i>Pilas en Java</i>	264
11.4. Listas enlazadas en Java	267
11.4.1. <i>Listas enlazadas simples con LinkedList Java</i>	269
11.4.2. <i>Listas doblemente enlazadas en Java</i>	271
11.5. Interfaz MAP	272
Ejercicios propuestos y resueltos	276

12. Fundamentos de estructura de datos en Java / PARTE II

12.1. Árboles binarios en Java	285
12.2. Árboles avl () en Java	285
12.3. Grafos en Java	293
12.4. Recorrido de grafos (BFS y DFS)	296
12.4.1. <i>Búsqueda primero en profundidad o Depth-First Search (DFS)</i>	297
12.4.2. <i>Búsqueda primero en amplitud o Breadth-First Search (BFS)</i>	301
12.5. Problema de ruta más corta (algoritmo de Dijkstra)	305
Ejercicios propuestos y resueltos	309

13. Fundamentos de estructura de datos en Python / PARTE I

13.1. Conceptos básicos de Python	315
13.2. Sintaxis en Python	315
13.3. Bifurcaciones y ciclos en Python	316

CONTENIDO

13.4. Funciones en Python	317
13.5. Recursividad en Python	318
13.6. Arreglos en Python	320
13.7. Colas y pilas en Python	324
13.7.1. Pilas	324
13.7.2. Colas	326
13.8. Listas en Python	329
13.8.1. Lista simplemente enlazada	332
13.8.2. Lista doblemente enlazada	336
13.8.3. Lista circular simplemente enlazada	337
13.8.4. Lista circular doblemente enlazada	342
Ejercicios propuestos y resueltos	343

14. Fundamentos de estructura de datos en Python / PARTE II

14.1. Árboles en Python	349
14.1.1. Árboles binarios de búsqueda	350
14.1.2. Árboles binarios balanceados AVL	352
14.2. Grafos	353
14.3. Recorridos DFS y BFS	361
14.3.1. DFS (Depth First Search o búsqueda primero en profundidad)	361
14.3.2. BFS (Breadth First Search o búsqueda primero en amplitud)	361
Ejercicios propuestos y resueltos	363

AGRADECIMIENTOS

Agradezco profundamente la confianza otorgada por Alpha Editorial que nos permitió a mí y a mi coautor escribir un segundo libro. A Samantha Córdoba Hernández, editora del área de ingenierías, quien ha realizado un excelente trabajo de revisión para la publicación de este libro.

A mi familia quien reconoce el esfuerzo que le he dedicado a esta obra. A mis estudiantes de la UNA y de la UNED quienes son el objetivo y la motivación para escribir esta obra.

ENRIQUE GÓMEZ JIMÉNEZ

Este libro es el resultado de un arduo trabajo, dedicación y pasión por el tema que aborda. Me gustaría expresar mi agradecimiento a todas las personas que me han apoyado durante este proceso.

En primer lugar, quiero agradecer a mi familia por su incondicional apoyo y motivación, a mi madre (Q.E.P.D.) por sus consejos que siempre los recuerdo. Sin su aliento y confianza, este proyecto no habría sido posible.

También quiero agradecer a Enrique que me motivó a emprender este proyecto. Asimismo, me gustaría expresar mi gratitud a Samantha Córdoba editora de Alpha Editorial.

Por último, agradezco a los lectores por su interés en este libro. Espero que encuentren en sus páginas información valiosa y útil que les permita adquirir nuevos conocimientos y perspectivas.

PERCY CAÑIPA VALDEZ

FIGURAS

Figura 1.1	Convenciones para nombrar variables, procedimientos y otros	8
Figura 1.2	Ejecución del programa del listado 1.5	16
Figura 1.3	Arreglo unidimensional de 10 columnas	17
Figura 1.4	Arreglo bidimensional de 3 filas y 8 columnas	18
Figura 2.1	Ejecución del listado 2.4	30
Figura 2.2	Ejecución del listado 2.9	33
Figura 2.3	Ejecución del programa del listado 2.10	35
Figura 2.4	Diagramas de clase que demuestra la relación de agregación	37
Figura 2.5	Diagramas de clase que demuestra la relación de composición	37
Figura 2.6	Diagrama donde se implementa la relación de agregación y composición	39
Figura 2.7	Ejecución del listado 2.11	40
Figura 2.8	Diagrama de clases	42
Figura 2.9	Salida de programa	42
Figura 3.1	Ejecución del listado 3.4	51
Figura 3.2	Ejecución del listado 3.5	53
Figura 3.3	Ejecución del listado 3.7	55
Figura 3.4	Ejecución del listado 3.8	57
Figura 3.5	Ejecución del listado 3.9	58
Figura 3.6	Ejecución del listado 3.10	60
Figura 3.7	Ejecución del listado 3.11	62
Figura 3.8	Ejecución del listado 3.12	63
Figura 3.9	Ejecución del listado 3.13	65
Figura 3.10	Ejecución del listado 3.14	66
Figura 3.11	Ejemplo de ejecución	67
Figura 4.1	Forma de ordenar mediante método de burbuja	74
Figura 4.2	Ejecución del listado 4.1	76
Figura 4.3	Método de ordenamiento Insert	77
Figura 4.4	Ejecución del listado 4.2	79
Figura 4.5	Procedimiento para el ordenamiento por selección	79
Figura 4.6	Ejecución del listado 4.3	81

Figura 4.7	Procedimiento del método de ordenamiento ShellSort	83
Figura 4.8	Ejecución del listado 4.4	85
Figura 4.9	Estado de acomodo de ejecución de ordenamiento QuickSort	86
Figura 4.10	Ejecución del listado 4.5	88
Figura 4.11	Ejecución lógica del ordenamiento MergeSort	89
Figura 4.12	Ejecución del listado 4.6	92
Figura 4.13	Ejecución del listado 4.8	97
Figura 4.14	Ejemplo de interfaz de arreglo generado aleatoriamente y desordenado y de arreglo ordenado con BubbleSort	98
Figura 4.15	Ejemplo de interfaz de países precargados	100
Figura 5.1	Funcionamiento de pilas y colas	105
Figura 5.2	Forma de operar de una pila	106
Figura 5.3	Estructura de datos cola	107
Figura 5.4	Estados de inserciones de elementos en una cola	108
Figura 5.5	Ejecución del listado 5.5	109
Figura 5.6	Ejecución del listado 5.6	111
Figura 5.7	Cola circular, sus datos y sus índices o apuntadores	112
Figura 5.8	Representación de memoria de estructura lineal mediante una cola circular	112
Figura 5.9	Posibles estados de datos de una cola circular	113
Figura 5.10	Ejecución del listado 5.7	114
Figura 5.11	Ejemplo de interfaz que implementa una pila	115
Figura 5.12	Ejemplo de interfaz que implementa una cola	117
Figura 6.1	Implementación lógica de listas enlazadas simples	123
Figura 6.2	Ejemplos de listas doblemente enlazadas	124
Figura 6.3	Ejemplos de listas circulares simplemente enlazadas	125
Figura 6.4	Ejemplos de listas circulares doblemente enlazadas	125
Figura 6.5	Estado inicial para recorrer los nodos de una lista simplemente enlazada	128
Figura 6.6	Ejecución del listado 6.1	129
Figura 6.7	Lista con un solo nodo	131
Figura 6.8	Pasos para agregar un nodo en la primera posición de una lista doblemente enlazada	131
Figura 6.9	Pasos para agregar un nodo en la última posición de una lista doblemente enlazada	132
Figura 6.10	Pasos para agregar un nodo en cualquier posición de una lista doblemente enlazada	133
Figura 6.11	Eliminación del único nodo de una lista doblemente enlazada	135
Figura 6.12	Eliminación del primer nodo de una lista doblemente enlazada	135
Figura 6.13	Eliminación del último nodo de una lista doblemente enlazada	136

Figura 6.14	Eliminación de nodo intermedio de una lista doblemente enlazada	136
Figura 6.15	Ejecución del listado 6.2	137
Figura 6.16	Lista circular enlazada	137
Figura 6.17	Agregar un nodo en una lista circular enlazada vacía	138
Figura 6.18	Agregar un nodo en una lista circular enlazada no vacía	138
Figura 6.19	Lista circular doblemente enlazada	139
Figura 6.20	Menú para la ejecución de operaciones sobre una lista circular doblemente enlazada	142
Figura 6.21	Operación de búsqueda en la colección de datos de una lista circular doblemente enlazada	143
Figura 6.22	Operación de inclusión en la colección de datos de una lista circular doblemente enlazada	143
Figura 7.1	Estructura de dato: árbol	147
Figura 7.2	Altura y nivel de un árbol	148
Figura 7.3	Peso de un árbol	149
Figura 7.4	Tipos de árboles	149
Figura 7.5	Tipos de árboles binarios	150
Figura 7.6	Árbol binario completo (totalmente balanceado)	151
Figura 7.7	Árbol binario completo (balanceado a la izquierda)	151
Figura 7.8	Árbol binario incompleto	151
Figura 7.9	Árboles binarios llenos y no llenos	152
Figura 7.10	Árboles binarios	153
Figura 7.11	Árboles binarios equilibrados	154
Figura 7.12	Árbol multicamino	154
Figura 7.13	Árbol binario de búsqueda	154
Figura 7.14	Menú principal del programa de manejo de un árbol binario de búsqueda	157
Figura 7.15	Árbol para procesar según el sitio web de prueba seleccionado	158
Figura 7.16	Salida generada por el programa del listado 7.1	159
Figura 7.17	Propuesta de borrado de nodo 7 en un árbol binario de búsqueda	159
Figura 7.18	Muestra de eliminación de nodo según figura 7.17	160
Figura 7.19	Propuesta de borrado de nodo 15 en un árbol binario de búsqueda	160
Figura 7.20	Muestra de eliminación de nodo según figura 7.19	161
Figura 7.21	Propuesta de borrado de nodo 21 en un árbol binario de búsqueda	162
Figura 7.22	Muestra de eliminación de nodo según figura 7.21	163
Figura 7.23	Factor de equilibrio (FE) de un árbol binario de búsqueda equilibrado (AVL)	164
Figura 7.24	Balanceo a la izquierda	164
Figura 7.25	Combinación de rotaciones para balanceo de un árbol binario	165
Figura 7.26	Solución brindada para los ejercicios 1 y 2	166

Figura 7.27	Ejecución del código del listado 7.2	166
Figura 7.28	Ejemplo a resolver con el programa del listado 7.2	167
Figura 7.29	Ejecución del programa del listado 7.2 para resolver el enunciado de la figura 7.28	167
Figura 7.30	Muestra de conformación de los nodos dentro de un árbol binario según su raíz y hojas	169
Figura 7.31	Muestra de nodos que conforman un árbol binario	169
Figura 8.1	El problema de los puentes de Königsberg	175
Figura 8.2	Conformación de la estructura de datos grafo	176
Figura 8.3	Teoría de grafos y la estructuración de redes sociales	177
Figura 8.4	Grafos simples	177
Figura 8.5	Grafo dirigido	178
Figura 8.6	Representación grafo dirigido de $E = \{(A,B), (A,D), (B,C), (D,B), (C,A)\}$	179
Figura 8.7	Representación grafo no dirigido de $V = \{A,B,C,D\}$	179
Figura 8.8	Representación grafo no dirigido ponderado (etiquetado)	179
Figura 8.9	Representación grafo aleatorio Erdős-Rényi con 50 vértices y 100 aristas.	180
Figura 8.10	Representación de un hipergrafo	180
Figura 8.11	Representación de un grafo infinito	181
Figura 8.12	Representación de un grafo no dirigido mediante una matriz de adyacencia	182
Figura 8.13	Representación de un grafo dirigido con pesos mediante una matriz de adyacencia	182
Figura 8.14	Demostración de grafo no dirigido	186
Figura 8.15	Ejecución del listado 8.1 (propuesta de grafo)	186
Figura 8.16	Ejecución del listado 8.1 (construcción de gráfico manualmente)	187
Figura 8.17	Diagrama de grafos con variables de entrada	187
Figura 8.18	Grafo no dirigido representado mediante una lista de adyacencia	188
Figura 8.19	Grafo dirigido con relación de nodos mediante aristas	189
Figura 8.20	Prueba de ejecución del listado 8.2	189
Figura 8.21	Grafo no dirigido representado mediante una lista de arcos	190
Figura 8.22	Traza de búsqueda por recorrido en amplitud	191
Figura 8.23	Niveles	192
Figura 8.24	Traza	192
Figura 8.25	Ejecución del código del listado 8.3	195
Figura 8.26	Grafo ejemplo	196
Figura 8.27	Traza de búsqueda por recorrido en amplitud (BFS)	197
Figura 9.1	Ejecución del listado 9.1	212
Figura 9.2	Ejecución del listado 9.2	213
Figura 9.3	Ejecución del listado 9.3	215

Figura 9.4	Ejecución del listado 9.4	219
Figura 9.5	Ejemplo de template	219
Figura 9.6	Ejemplo de template genérico	220
Figura 9.7	Salida de la ejecución del listado 9.5	224
Figura 9.8	Salida de la ejecución del listado 9.6	226
Figura 9.9	Salida de la ejecución del listado 9.7	226
Figura 9.10	Salida de la ejecución del listado 9.8	228
Figura 9.11	Salida de la ejecución del listado 9.9	229
Figura 9.12	Salida de la ejecución del listado 9.10	231
Figura 9.13	Eliminación de elementos de una lista mediante STL	232
Figura 10.1	Ejecución del listado 10.1 que gestiona un archivo de texto	241
Figura 10.2	Salida generada por la ejecución de la opción 2 del listado 10.1	241
Figura 10.3	Ejecución del listado 10.2 (menú principal)	242
Figura 10.4	Ejecución del listado 10.2 (agregación de estudiante)	242
Figura 10.5	Ejecución del listado 10.2 (modificación de estudiante)	242
Figura 10.6	Ejecución del listado 10.2 (eliminación de estudiante)	243
Figura 10.7	Ejecución del listado 10.2 (listado de estudiante)	243
Figura 10.8	Ejecución del listado 10.3 (menú principal)	245
Figura 10.9	Ejecución del listado 10.3 (agregación de registro)	245
Figura 10.10	Ejecución del listado 10.4 (modificación de un registro)	245
Figura 10.11	Ejecución del listado 10.3 (listado de registros)	246
Figura 10.12	Ejecución del listado 10.3 (eliminación de registro)	246
Figura 10.13	Búsqueda de un nodo entre los elementos de un árbol binario	247
Figura 10.14	Búsqueda de estudiante	247
Figura 11.1	Jerarquía Collection Interface	253
Figura 11.2	Ejecución del código del listado 11.1	255
Figura 11.3	Ejecución del código del listado 11.2	259
Figura 11.4	Ejecución del listado 11.4	261
Figura 11.5	Ejecución del listado 11.7	265
Figura 11.6	Ejecución del listado 11.9	268
Figura 11.7	Procedimiento para inclusión y eliminación de nodos en una lista enlazada	269
Figura 11.8	Implementación de listas enlazadas mediante ArrayList	270
Figura 11.9	Implementación de listas enlazadas con apuntadores	271
Figura 11.10	Ejecución del programa del listado 11.15	272
Figura 11.11	Listas simple y doblemente enlazada	273
Figura 11.12	Ejecución del código del listado 11.18	274

FIGURAS

Figura 11.13	Resultado de la ejecución del código del listado 11.19	276
Figura 11.14	Listado de datos de todos los nodos de una árbol binario	282
Figura 12.1	Ejecución del código del listado 12.3	286
Figura 12.2	Recorrido In Order, según resultados de ejecución del listado 12.3	287
Figura 12.3	Árbol binario balanceado (AVL)	287
Figura 12.4	Inserción de los valores iniciales del árbol binario balanceado (AVL)	288
Figura 12.5	Inserción del valor 16 y balanceo del árbol binario AVL	288
Figura 12.6	Inserción del valor 6 y balanceo del árbol binario AVL	289
Figura 12.7	Inserción del valor 11 y balanceo del árbol binario AVL	289
Figura 12.8	Inserción del valor 51 y 61 en el árbol binario AVL	290
Figura 12.9	Balanceo del árbol binario AVL después de insertar los valores 51 y 61	290
Figura 12.10	Balanceo del árbol binario AVL después de insertar el valor 71	291
Figura 12.11	Balanceo del árbol binario AVL después de insertar el valor 66	291
Figura 12.12	Árbol binario AVL totalmente balanceado	292
Figura 12.13	Ejecución del código del listado 12.6.	292
Figura 12.14	Mapeo de algunos cantones de Costa Rica	293
Figura 12.15	Ejecución del código del listado 12.10	295
Figura 12.16	Grafo dirigido que muestra la navegación entre nodos	296
Figura 12.17	Grafo dirigido que muestra la navegación entre rutas de ciudades	297
Figura 12.18	Búsqueda primero en profundidad (DFS)	297
Figura 12.19	Ejecución del código del listado 12.14 para DFS	301
Figura 12.20	Búsqueda primero en amplitud (BFS)	302
Figura 12.21	Ejecución del código del listado 12.17. Implementación de BFS	305
Figura 12.22	Grafo de distancias entre cantones de Costa Rica	306
Figura 12.23	Ejecución del código del listado 12.20	309
Figura 13.1	Ejecución del código del listado 13.9	327
Figura 13.2	Ejecución del código del listado 13.10	329
Figura 13.3	Ejemplo de lista simplemente enlazada	332
Figura 13.4	Ejemplo de inserción al inicio de la lista simplemente enlazada	333
Figura 13.5	Ejemplo de inserción después de un nodo de la lista simplemente enlazada	336
Figura 13.6	Ejemplo de inserción al final de la lista simplemente enlazada	337
Figura 13.7	Ejecución del código del listado 13.12	337
Figura 13.8	Ejecución del código del listado 13.13	338
Figura 13.9	Recorridos hacia atrás y hacia adelante de lista doblemente enlazada del listado 14.13	338
Figura 13.10	Recorridos hacia atrás y hacia adelante de lista doblemente enlazada del listado 13.13 después de ser ordenada	338
Figura 13.11	Ejecución del código del listado 13.15	343

Figura 14.1	Ejecución del código del listado 14.1	351
Figura 14.2	Ejecución del código del listado 14.2	352
Figura 14.3	Ejecución del código del listado 14.3	353
Figura 14.4	Árbol binario balanceado AVL con los datos de entrada 4,5,7,2,1,3,6	353
Figura 14.5	Grafo sin pesos	355
Figura 14.6	Ejecución del código del listado 14.4	360
Figura 14.7	Grafo con pesos ponderados	360
Figura 14.8	Ejecución del código del listado 14.5	360
Figura 14.9	Grafo de cantones de Costa Rica con rutas	361
Figura 14.10	Recorrido DFS sobre el grafo de la figura 14.9	364
Figura 14.11	Recorrido BFS sobre el grafo de la figura 14.9	364
Figura 14.12	Ejecución del código del listado 14.6	365
Figura 14.13	Rutas entre ciudades en Colombia	366

TABLAS

Tabla 1.1	Tipos de datos utilizados en C++	9
Tabla 1.2	Operadores aritméticos y funciones predeterminadas en C++	10
Tabla 1.3	Operadores relacionales en C++	11
Tabla 1.4	Operadores lógicos en C++	11
Tabla 1.5	Estructuras condicionales o de control en C++	12
Tabla 1.6	Estructuras repetitivas en C++	13
Tabla 3.1	Funciones de <i>string</i> en C++	49
Tabla 3.2	Uso básico de punteros en C++	55
Tabla 6.1	Comportamiento de inserción de nodos en listas enlazadas en C++	127
Tabla 6.2	Proceso de borrado de un nodo en listas enlazadas en C++	130
Tabla 9.1	Funcionalidades de la clase <i>vector</i> de STL	211
Tabla 9.2	Funcionalidades de la clase <i>algorithm</i>	221
Tabla 9.3	Algunos métodos para contenedores asociativos	226
Tabla 10.1	Modos de gestión de archivos (banderas o <i>flags</i>)	243
Tabla 11.1	Programación imperativa en Java vs programación funcional en Java	256
Tabla 13.1	Proceso de creación e inserción de nodos	334
Tabla 13.2	Tipos de eliminación de nodos	335
Tabla 14.1	Recorridos según árbol avl figura 14.4	354
Tabla 14.2	Estado del árbol AVL luego de la inserción de datos: 8, 11, 10, 21 y 13	354
Tabla 14.3	Recorridos según árbol AVL después de la inserción de datos	355
Tabla 14.4	Estado del árbol AVL luego de eliminar los nodos con valores 10, 21 y 13	356

Estructura de datos

Un enfoque con Python, Java y C++

INTRODUCCIÓN

La estructura de datos es uno de los pilares fundamentales de la ciencia de la computación y es esencial para el diseño y la implementación de algoritmos eficientes y escalables. La eficacia de un programa depende en gran medida de la elección correcta de las estructuras de datos adecuadas para el problema a resolver.

Este libro es una guía práctica para el aprendizaje y la comprensión de las estructuras de datos esenciales, desde las más básicas, como arreglos y listas, hasta estructuras más avanzadas, como árboles, grafos y algoritmos de búsqueda y ordenamiento. Se explican los conceptos fundamentales de cada estructura de datos y se proporcionan ejemplos de implementaciones en lenguajes de programación comunes.

Este libro está dirigido a estudiantes de ciencias de la computación, ingenieros de software y desarrolladores que desean mejorar sus habilidades en estructuras de datos. El objetivo es proporcionar una base sólida para el diseño y la implementación eficientes de algoritmos en la vida real.

MATERIAL WEB

Este libro se fortalece con diversos códigos que se encuentran en la web y facilitan la comprensión y práctica de los contenidos expuestos. En el transcurso del libro, las referencias directas a estos recursos están señaladas mediante el siguiente ícono:



Para acceder a estos recursos utilice el vínculo:

<https://www.alpha-editorial.com/Papel/9789587789003/Estructura+De+Datos>

También puede seguir los siguientes pasos:

1. Acceder a www.alphaeditorialcloud.com
2. Escribir el nombre del libro en el buscador.
3. En la página del libro encontrará el vínculo al material web.

Para los ejemplos programados en C++ en este libro se utilizará el compilador Dev C++, descargado desde <https://sourceforge.net/projects/orwelldevcpp/>, pero existen muchas otras opciones que se pueden utilizar para la escritura y ejecución del código.

Por ejemplo, se pueden utilizar cualquiera de los que se recomiendan en el sitio

<https://geekflare.com/es/best-cpp-ide/>.

Por tanto, el lector deberá seleccionar el IDE y compilador que más se ajuste a sus necesidades y preferencias.

1

Introducción a C++

1.1. Conceptos básicos de C++	3
1.2. Tipos de datos en C++	8
1.3. Entrada y salida de datos en C++	9
1.4. Operaciones y operadores matemáticos en C++	10
1.5. Estructuras condicionales y repetitivas en C++	10
1.6. Funciones, métodos y procedimientos en C++	11
1.7. Parámetros en C++	14
1.8. Arreglos en C++	17
Ejercicios propuestos y resueltos	18

A PRINCIPIOS de los años 70 Dennis Ritchie trabajaba con los laboratorios AT&T utilizando un lenguaje llamado BCPL (*Basic Combined Programming Language*) o lenguaje de programación combinado inventado por Martin Richards. Ritchie no se sentía cómodo trabajando con ese lenguaje y escribió un lenguaje compilador denominado C que permitía manejar el hardware del computador de la misma manera que el programa *Ensamblador*, a la vez que permitía una programación estructurada de alto nivel. Este lenguaje se ejecutaba sobre máquinas con sistemas operativos UNIX por lo que se volvía limitado. El verdadero logro de Ritchie sucedió cuando reescribió el compilador de C en el propio C y luego Ken Thompson modificó UNIX completamente en C y no en *Ensamblador*. Para esa época se inició la comercialización de las computadoras personales y aparecieron versiones de compiladores de C que lo convirtieron en el lenguaje favorito de los programadores de aplicaciones.

En 1983, el Instituto Americano de Normalización (ANSI, *American National Standard Institute*, por sus siglas en inglés) estandarizó C, y en 1989 conjuntamente con la Organización Internacional de Normalización (ISO, *International Standard Organization*, por sus siglas en inglés) lo formalizaron. De ahí fue que empezó la evolución de C estándar, convirtiéndose en uno de los compiladores más potentes de la industria. En 1998, impulsado por Bjarnes Stroustrup a C se le incluyeron algunas librerías enfocadas a la programación orientada a objetos, creando lo que se llegó a conocer como C++. El cambio de paradigma que se dio con la creación de C++ fue la adición del paradigma de programación orientada a objetos a la programación estructurada existente en C. Esto permitió a los desarrolladores escribir código más modular y reutilizable y también dio lugar a la creación de muchos marcos y bibliotecas de software importantes basados en C++. Por ejemplo, para desarrollar controladores de dispositivos o sistemas empotrados, generalmente en dispositivos con recursos limitados, se requiere más de la simplicidad y el poder de C estándar. Otro tipo de aplicaciones más robustas y que requieren el uso de programación orientada a objetos entonces se recurrirá a C++.

1.1. Conceptos básicos de C++

Cuando empezamos a aprender un lenguaje de programación debemos iniciar con sus conceptos básicos. Entre esos conceptos podemos enumerar el sistema de tipos utilizados en C++, el ámbito de ejecución, las unidades de traducción y vinculación, la función *main* y los argumentos que se procesan en la línea de comando, objetos temporales, entre otros. A continuación se explican estos conceptos que son necesarios para comprender el lenguaje C++.

- **Sistema de tipado en C++.** En C++ el concepto de tipado es muy importante. Esto es porque cada variable, argumento de función o valor devuelto por esta debe tener un tipo de dato válido para ser compilado.

En este contexto, C++ asigna, de manera implícita, un tipo a todas las expresiones construidas en el lenguaje antes de ser evaluadas. Entre este tipado se tienen datos primitivos como *int* (para almacenar valores enteros), *double* (para almacenar valores de punto flotante), *string* (para almacenar texto en forma de literales), entre otros tipos. También es posible crear estructuras propias mediante la declaración de clases o el uso del objeto *struct*. Además de establecer el tipo de dato que se generará mediante el tipado, C++ especifica la cantidad de memoria que el sistema operativo debe asignar para la variable (o resultado de una expresión). Dentro de esta categorización se puede determinar la terminología relacionada con la caracterización de tipos en C++.

- a. **Tipo escalar.** Este tipo representa un valor único definido mediante un intervalo. Los tipos escalares incluyen tipos aritméticos (enteros o de punto flotante), enumeradores, punteros, entre otros. Los tipos de datos fundamentales o primitivos (enteros, flotantes, *char*, *long*, *float* y de cadena, principalmente).
 - b. **Tipo compuesto.** Es un tipo que no representa un valor escalar. En esta categoría se tienen arreglos (unidimensionales y multidimensionales), funciones, clases, objetos *struct*, uniones, enumeradores, referencias y punteros a miembros de clase no estáticos.
 - c. **Variable.** Es el nombre simbólico que se le asigna a un espacio de memoria de algún tipo de dato. En ese espacio de memoria puede almacenar el valor de acuerdo con su tipo y también puede recuperarse.
 - d. **Objeto.** Significa crear la instancia de una estructura que puede ser de una clase o de un objeto *struct*. Inclusive, en un contexto amplio, los tipos escalares representan objetos.
- **Ámbito.** Es conocido como el alcance que puede tener un elemento de programa (clase, función, variable). Es decir, el nombre y el contenido de una variable, por ejemplo, tiene un alcance (donde se puede ver y utilizar) dentro de un programa. Pueden coexistir nombres de variables o funciones dentro de un programa que se llamen igual, pero gracias a ese alcance puede que no haya infracciones entre ellos dado que comparten diferente contexto. En cuanto a las variables no estáticas automáticas, este ámbito es el encargado de crearlas y destruirlas de la memoria del programa. Algunos tipos de ámbitos de un programa son:
- a. **Ámbito global.** Este ámbito cubre cualquier lugar fuera de una clase, función o espacio de nombres. Este espacio se extiende desde el punto de declaración hasta el final del archivo de programa declarado. En caso de nombres de estructuras globales, esta visibilidad se rige por las reglas de vinculación que determinan

si es visible o no en otros archivos de programas. Una función o una variable declarada global en un programa, incluido en uno generado en C++, puede ser accedida desde cualquier punto de dicho programa.

- b. **Ámbito del espacio de nombres.** Un espacio de nombres puede contener clases, variables, funciones o métodos que son visibles desde su punto de declaración hasta el final del espacio de nombres. Un espacio de nombres puede ser definido en varios bloques en distintos archivos de programa. Estos espacios de nombres constituyen regiones declarativas que proporcionan un ámbito a identificadores tales como variables, funciones, entre otros. Se utilizan principalmente para organizar el código en grupos lógicos que generan bibliotecas y así evitar conflictos de nombres. Por ejemplo se puede crear un espacio de nombres *Utilidades* y *Documentos* y ambos contener un nombre de función de igual nombre, por ejemplo *Imprimir*. No habría problemas de nombre dado que a una se le llamaría como *Utilidades.Imprimir()* y a la otra *Documentos.Imprimir()*.
- c. **Ámbito local.** En este ámbito se recurre a lo declarado dentro de una función o *lambda*. Se incluyen los parámetros que estas funciones pudieran contener. Solo son visibles desde su declaración dentro de la función hasta el final de dicha función o cuerpo *lambda*. Este es un tipo de ámbito de bloque que implementa una lógica muy concreta. Por ejemplo, una función que realice los cálculos de una planilla y que genere los datos. Dentro de ella podría existir un parámetro *totalPlanilla* que totalice la planilla, pero solo tenga visibilidad dentro de esa función.
- d. **Ámbito de clase.** Todo lo declarado dentro de una clase tiene visibilidad dentro de la misma, independientemente del punto de declaración. Es posible que la clase al ser pública pueda ser derivada en otra clase, por lo que sus propiedades, atributos y métodos hereden en la instancia ese alcance también. La accesibilidad de los miembros de la clase (atributos, métodos y propiedades, principalmente) se rigen por las palabras clave *public*, *private* y *protected*. Cuando un miembro de clase es declarado *private*, su ámbito o visibilidad está limitado a los demás miembros de la clase. Cuando es *protected* solo es accesible o visible desde los miembros definidos en la clase que pertenece a un mismo paquete (*package*) y también desde cualquier función de cualquier subclase que hereda. Este tipo de acceso es apropiado para una subclase de una clase, pero no para una clase sin relación. Cuando es *public* se puede acceder al miembro de la clase desde cualquier lugar del código, desde el interior de la propia clase o desde cualquier objeto derivado de esta.

- e. **Ámbito de instrucción.** Cuando se utilizan *for*, *if*, *while* o *switch* por ejemplo, los nombres declarados dentro de estas instrucciones son visibles desde el inicio hasta el final de la declaración de bloque. En este caso, todo bloque después de un *for*, por ejemplo lo que esté entre llaves será visible a nivel de instrucción.

- **Función *main* y sus argumentos.** Un programa en C++ debe tener siempre una función *main*. No se puede compilar un programa que no lo tenga. De no tenerlo se generará un error de compilación. Solo las bibliotecas *static* y las bibliotecas de vínculos dinámicos carecen de esta declarativa. Una función *main* indica que es el inicio de ejecución de un programa en C++, estableciendo en cero todos los miembros *static* del programa. La signatura de una función *main* en C++ tiene la siguiente composición:

```
int main();  
int main(int argc, char *argv[]);
```

- **Finalización de un programa en C++.** En el lenguaje C++ es posible finalizar un programa mediante las instrucciones *abort* *exit* o *terminate*. En los siguientes literales se dan ejemplos de fragmentos de código en donde se explica cómo terminar un programa mediante estas tres instrucciones:

- a. En este programa todo se termina en la línea 5.

```
1.     include<iostream>  
2.     using namespace std;  
3.     int main(){  
4.         cout << "Estoy en la línea 1 !" <<endl;  
5.         abort();  
6.         cout << "Estoy en la línea 2 !";  
7.     }
```

Como se puede observar en el listado anterior, en las líneas 1 y 2 se incluyen dos librerías propias de C++ que son *iostream* y *std*. La primera (*iostream*) es un componente de la biblioteca estándar (STL) de C++ que se utiliza para procesar las entradas (*input*) y salidas (*output*). Es un acrónimo de *Input/Output*. La librería (*std*) es la invocación a *std-lib standard library* o biblioteca estándar, la cual tiene como propósito gestionar la memoria dinámica de la computadora donde se ejecutan los programas en C++, el control de procesos, entre otros. Finalmente, las líneas 3 a 7 muestran el cuerpo principal de un programa en C++, iniciando con *int main()* seguido de la apertura de llave y culminando con el cierre de llaves respectivo (línea 7).

b. En este programa todo se termina en la línea 5.

```
#include<iostream>
1.     using namespace std;
2.     int main(){
3.         cout << "Estoy en la línea 1 !" <<endl;
4.         exit(0);
5.         cout << "Estoy en la línea 2 !";
6.     }
```

c. En este programa todo se termina en la línea 5.

```
#include<iostream>
2.     using namespace std;
3.     int main() {
4.         cout << "Estoy en la línea 1 !" <<endl;
5.         terminate();
6.         cout << "Estoy en la línea 2 !";
7.     }
```

- **Elementos del lenguaje.** Igual que en todos los lenguajes de programación existentes, C++ cuenta con elementos esenciales que lo hacen funcionar. Estos elementos son identificadores, palabras reservadas, literales, delimitadores, separadores, entre otros. A continuación se hace un resumen de algunos de estos elementos del lenguaje.
 - a. **Identificadores.** Representan entidades (variables, tipos, constantes, procedimientos u otros) que son asignados por el desarrollador de *software* en un programa. Este desarrollador puede elegir cualquier nombre de identificador a excepción de los que constituyen palabras reservadas en el lenguaje de programación. Se debe recordar que C++ discrimina entre mayúsculas y minúsculas, por lo que *Hola* es diferente a *hola* en el nombramiento de alguna variable. No existe un límite de longitud en la asignación de un identificador, sin embargo, una buena práctica es que el nombre sea corto y bien representativo. Existen algunas convenciones para el nombramiento de variables, propiedades, atributos, funciones, entre otras, tales como *Camel Case*, *Snake Case* o *Pascal Case*. La figura 1.1 muestra estas convenciones de nombramientos y ejemplos.
 - *Camel Case*: es usualmente utilizado para nombrar variables, propiedades, atributos, métodos y funciones.
 - *Snake Case*: es utilizado para asignar nombres en lenguajes de scripting como Python y para constantes en otros lenguajes como C.
 - *Pascal Case*: se identifica con el nombramiento de clases, interfaces y espacios de nombres.
 - b. **Palabras reservadas.** Las palabras reservadas en C++ tienen un significado predeterminado para el compilador, dado que debe ser sometido al análisis sintáctico y semántico de este. Existe una gran