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 r
31. cafe = nodoA
32. te = nodoArb
33. jugo = nodoA
```

Estructura de datos

Un enfoque con Python, Java y C++

ENRIQUE GÓMEZ JIMÉNEZ
PERCY CAÑIPA VALDEZ

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

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

3. Estructuras de datos y recursividad en C++

4. Ordenamientos y búsquedas en C++

5. Pilas y colas en C++

6. Listas enlazadas

7. Árboles

8. Grafos

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

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

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

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

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

TABLAS

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++

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 Stroutstrup 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 tipo *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.

```
1.      #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